

**eXplainable Artificial Intelligence
(XAI) for the Measurement of
 $Br(K^+ \rightarrow \pi^+ \nu \bar{\nu})$ with NA62
Experiment at CERN**



Joseph Carmignani, BSc, MSc
Experimental Particle Physics Group

Lancaster University

A thesis submitted for the degree of
Doctor of Philosophy

May, 2022

I dedicate this thesis to Joey, Vali & Angie

Declaration

I declare that the contents of this thesis are, to the best of my knowledge and belief, original and my sole scientific contribution. The material has not been submitted, either in whole or in part, for a degree at this, or any other university. This thesis is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the Preface chapter at the beginning of this thesis. This thesis does not exceed the maximum permitted word length: it contains fewer than 80,000 words including appendices and footnotes, but excluding the bibliography. A rough estimate of the word count is: 49431

Joseph Carmignani
May, 2022

**eXplainable Artificial Intelligence (XAI) for the Measurement of
 $Br(K^+ \rightarrow \pi^+\nu\bar{\nu})$ with NA62 Experiment at CERN**

Joseph Carmignani, BSc, MSc.

Experimental Particle Physics Group, Lancaster University

A thesis submitted for the degree of *Doctor of Philosophy*. May, 2022

Abstract

In this thesis a Neural Net (NN) code is first presented from scratch and applied to the Kaon-Pion matching in the rare Kaon decay ($K^+ \rightarrow \pi^+\nu\bar{\nu}$) analysis of NA62 at CERN. The NN code showed increased efficiency in Kaon decay identification with respect to the standard algorithm based on statistical analysis. It is designed and trained on $K^+ \rightarrow \pi^+\pi^+\pi^-$ decay channel to optimize the statistical significance of $K^+ - \pi^+$ matching by amplifying the association between parent Kaons and downstream Pions over accidental beam particles (“Pileup”) and final state Pions. Essential enhancement and evaluation processes using state-of-the-art techniques of XAI (eXplainable Artificial Intelligence) are presented in the context of choosing the optimal NN-discriminant that fits in the framework of $\pi\nu\nu$ analysis in NA62 based on necessary physics-related metrics. Another XAI application of an innovative Calorimetric “Virtual Bubble Chamber” technique, called NNODA (Neural Net Object Detection Approach), for NA62’s LKr (Liquid Krypton Calorimeter) is constructed to analyze images of clusters using DL (Deep Learning) Computer Vision (CV) techniques. The idea is to use color tags on the cluster timing to veto random activities and unwanted decay products (mainly π^0 background) allowing an unusual and flexible event selection time window of ± 10 ns around the arrival time of the charged single particle in the final state. NNODA efficiently increased signal acceptance by controlling random cuts. Additionally, practical data science skills in Robotics are presented, by training algorithms that would help a drone to identify and locate end-effectors in unusual environments. Then, An AI-based vision system is proposed for an embedded device and presented in its full facets, and specifically uses DL CV in image classification and object detection. These XAI tools and others have been successfully transferred to NA62’s most precise measurement of $Br(K^+ \rightarrow \pi^+\nu\bar{\nu})$ in a cross-disciplinary fashion.

Publications

Only one publication, shown below, has been created directly from the thesis, this (soon-to-be) published work is a solo contribution and presented within chapter 4:

[57]

The following publications have been generated while developing this thesis, these are the general NA62 Collaboration analysis papers and to an extent has inspired the thesis into what it has accomplished (Please check Preface for contributions):

[1][2][5][3][4][6][7][8]

Acknowledgements

I'm ever more grateful for the people who pushed me forward and provided me with all needed support. My life partner and best friend Joelle, and our two amazing daughters Valentina and Angelina. Mum, dad and loving sister Joelle. My adviser Prof Giuseppe Ruggiero and Prof Cristina Lazzeroni head of NA62. Dr. Allahyar Montazeri for supervising my internship. Dr. Laura Kormos and Dr. Ian Bailey for their timely advice, help and guidance. Prof Louis Lyons, Em P John Dainton and Em P Italo Mannelli for their gigantic influence and inspiration. Prof Aneta Stefanovska for her friendship to my family and the positive spirit that she is. Prof Roger Jones (HoD) for backing me up and believing in me, and Lancaster University being my second home for more than four consecutive years.

Contents

1	Kaon Physics of NA62	1
1.1	Current Status of Kaon Physics	1
1.2	Experimental Status of $K^+ \rightarrow \pi^+ \nu \bar{\nu}$	8
1.3	The beam and detector	10
1.4	Analysis method	12
1.5	Event selection	13
1.6	Single event sensitivity	16
1.7	Background evaluation and validation	17
2	Introduction to AI	22
2.1	Artificial Intelligence: Aristotle to COVID-19	22
2.2	Supervised Neural Networks	25
2.2.1	Convolutional Neural Nets CNN	34
2.2.1.1	Convolutional layers	36
2.2.1.2	ReLU Activation Layer	40
2.2.1.3	Pooling Layers	41
2.2.1.4	Final Output	43
2.3	Practical Computer Vision	44
2.3.1	Feature Extractors	45
2.3.1.1	Residual Networks (ResNets)	46
2.3.1.2	Inception Network	48
2.3.1.3	MobileNets	51
2.3.2	Object Detectors OD	52
3	NN and K-pi Matching	59
3.1	Inputs and Design	59
3.1.1	Data Preparation	59
3.1.2	Basic Development	62
3.1.3	Architecture/model and Hyper-parameters Tuning	64
3.2	Results	70

3.3	XAI Analysis	73
3.3.1	Redefinition of the training sample	76
3.3.2	Upgrade of the discriminant	78
4	NNODA for LKr Calorimeter	85
4.1	Physics Review	85
4.2	Data Preparation	89
4.3	Training	92
4.3.1	YOLO model	92
4.3.2	SSD models	94
4.3.3	Faster-RCNN model	94
4.3.4	Fine Tuning in Transfer Learning	95
4.3.5	Configurations	98
4.3.5.1	Charged Tracks Clusters using Darknet API	98
4.3.5.2	Charged Tracks Clusters using Tensorflow Object Detection API	99
4.4	XAI in Performance Checks	100
4.4.1	Metrics	101
4.4.2	Technical Analysis & Model Competition	104
4.5	XAI in Calorimetric Implications	111
5	Summary and Conclusions	121
5.1	$K^+ - \pi^+$ Track Matching	121
5.2	NNODA: LKr Calorimetric Study	124
5.3	Robotics	125
	Appendix A Standard Notations for DL	129
A.1	Deep Learning Representations	129
A.2	Fundamental CNN Notations	130
	Appendix B Technical Configurations	132
B.1	Clusters in Darknet API Setups	132
B.2	Clusters in Tensorflow API Setups	133
	Appendix C Robotics Applications	139
C.1	Introduction	139
C.1.1	End-Effector Data Box Annotations	140
C.2	Mobile Activity project	140
C.3	End-effector detection using Darknet API	147
C.4	DetectNet using Jetson-Inference API	152
C.5	Facebook's Detectron2 API	159

Appendix D Robotics Setups	166
D.1 Mobile Activity	166
D.2 End-effector in Darknet API Setups	168
D.3 End-effector in DIGITS' DetectNet Setups	173
References	178

List of Acronyms

NNODA	Neural Net Object Detection Approach
MDA	Mean Decrease of Accuracy
XAI	eXplainable Artificial Intelligence
NN	Neural Network
LKr	Liquid Krypton
NN	Neural Network
MIP	Minimal Ionised Particle
CNN	Convolutional Neural Network
YOLO	You Only Look Once
FPN	Feature Pyramid Network
DL	Deep Learning
FCNC	Flavor Changing Neutral Current
CKM	Cabibbo-Kobayashi-Masakawa
CP	Charge Parity
NP	New Physics
SM	Standard Model
BSM	Beyond Standard Model
CMFV	Constrained Minimal Flavor Violation
SES	Single Event Sensitivity
CDA	Closest Distance of Approach
GPU	Graphical Processing Unit
CPU	Central Processing Unit
LR	Logistic Regression

KNN	K-Nearest Neighbours
BDT	Boosted Decision Trees
RF	Random Forest
NB	Naive Bayes
SVM	Support Vector Machine
SGD	Stochastic Gradient Descent
ReLU	Rectified Linear Unit
SeLU	Scaled Exponential Linear Unit
FC	Fully Connected
OD	Object Detector
RoI	Region of Interest
SSD	Single Shot Detector
API	Application Programming Interface
RV	Random Veto
PV	Photon Veto
SA	Standard Algorithm
PNN	$\pi\nu\nu$
IoU	Intersection over Union
mAP	mean Average Precision
FP	False Positive
FN	False Negative
TP	True Positive
TN	True Negative

List of Figures

1.1	A typical CKM unitarity triangle.	4
1.2	The penguin and box diagrams of $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ [52]	5
1.3	Relations between the unitarity triangle and the $K \rightarrow \pi \nu \bar{\nu}$ decays [42].	6
1.4	Plots of popular correlations in the $Br(K_L \rightarrow \pi^0 \nu \bar{\nu})$ versus $Br(K^+ \rightarrow \pi^+ \nu \bar{\nu})$ plane. The expanding red region represents lack of correlation for models with general LH and/or RH NP couplings with Z' . Green region is the correlation zone of models conforming to CMFV. While blue region illustrates the correlation induced by ε_k constraints in models with exclusively LH or RH couplings [51].	7
1.5	Expected theoretical distributions of the m_{miss}^2 variable relevant to the $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ measurement. The m_{miss}^2 is computed under the hypothesis that the charged particle in the final state is a π^+ . The $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ signal (red line) is multiplied by 10^{10} for visibility. The hashed areas include the signal regions.	9
1.6	Schematic top view of the NA62 beamline and detector.	10
1.7	Reconstructed m_{miss}^2 as a function of p_{π^+} for minimum-bias events selected without applying π^+ identification and photon rejection, assuming the K^+ and π^+ mass for the parent and decay particle, respectively. Signal regions 1 and 2 (hatched areas), as well as 3π , $\pi^+\pi^0$, and $\mu\nu$ background regions (solid thick contours) are shown. The control regions are located between the signal and background regions.	13
1.8	Evaluation of the $\pi^+\pi^0$ background. Left: Distribution, in the (p_{π^+}, m_{miss}^2) plane of events in the $\pi^+\pi^0$ region and in the adjacent control regions after the complete signal selection, is applied to the S1 and S2 subsets. The intensity of the grey shaded area reflects the variation of the SM signal acceptance in the plane. Right: Data/MC comparison of the m_{miss}^2 distribution of minimum-bias $K^+ \rightarrow \pi^+\pi^0$ events selected by tagging the $\pi^0 \rightarrow \gamma\gamma$ decay. This data sample is used to measure the $K^+ \rightarrow \pi^+\pi^0$ kinematic factor f_{kin}	18

1.9	Properties of upstream background events. Left: Extrapolation of π^+ tracks of the upstream sample described in the text to the $(X_{\text{COL}}, Y_{\text{COL}})$ plane in the S2 subset. The small (large) red rectangles correspond to the inner (outer) borders of the new collimator. The outline of the last dipole of the beam achromat is shown with blue solid lines. Right: CDA distribution of the events in the upstream sample shown on the left plot (black markers with error bars), compared to the CDA distribution extracted from data and its uncertainty (brown shaded area) and to the same distribution of K^+ decaying in the FV (grey shaded area).	20
1.10	Background predictions. Left: Reconstructed m_{miss}^2 as a function of π^+ momentum after applying the signal selection to the S1 and S2 subsets. Events in the background regions are displayed as light grey dots. The control regions, populated by the solid black markers, are adjacent to the background regions. The numbers next to these regions are the expected numbers of background events (in brackets) and the observed numbers (without brackets). Right: Expected numbers of background events summed over Regions 1 and 2 in the 2018 subsets.	21
2.1	The shuffling must be synchronised between the X and the Y matrices.	29
2.2	The last mini batch might be smaller than the mini-batch-size.	30
2.3	A mind-map of the general L-layered NN Dynamics [95]. As we notice here for every forward function, there is a corresponding backward function. That is why at every step of our forward module we must store some values in a cache. Cached values are useful for computing gradients. In the back-propagation module we can then use the cache to calculate the gradients.	32
2.4	An instance of a classic architecture of CNN [95]. Here the hidden layers of the convolution block called “filter” (<i>within the red square dotted line</i>) are repeated two times ($\times 2$).	35
2.5	CONV layer Operation: (Left) input image in visual (bottom) and pixel (top) forms, (Middle) CONV layer in visual (bottom) and pixel (top) forms, (Right) output image image in visual (bottom) and pixel (top) forms.	36
2.6	CONV operation output after applying Zero-padding	38
2.7	Convolution operation with a filter 3x3 and a stride of 1 (stride = amount you move the window each time you slide) [95]	39
2.8	The 3D final output shape of the convolution step (please check Appendix for the classic notation used in this image)[95]	40
2.9	After a ReLU activation pass	41

2.10	Pooling Layers	42
2.11	Output after a 3×3 Average-pooling with a stride=1	42
2.12	A ResNet block showing a “skip-connection”	46
2.13	(Up) in red “ID BLOCK” with skip connection over the main path: 3 convolutional layers, here “CONV2D”, 3 batch normalization passes, 2 ReLU activation layers in between, shortcut and input of main path are added together and ReLU activation finally applied on the sum, (Middle) in blue “CONV BLOCK” same but adding a CONV2D with Batch Norm filter on the skip connection over the main path, (Bottom) general structure of ResNet50 [95].	47
2.14	“Networks in Networks” mechanism [95]	49
2.15	Inception module [95]	50
2.16	Left: Standard convolutional layer with batch-norm and ReLU. Right: Depth-wise Separable convolutions with Depth-wise and Point-wise layers followed by batch-norm and ReLU [19].	51
2.17	(Left) MobileNetv2 main blocks, (Right) MobileNetv2 body architecture [25]	52
2.18	An output vector y_{car} is showing and associated with the car’s bounding box in this 3x3 grided image. The proper input values are stated relative to the central grid (<i>red square</i>) [95].	54
2.19	Each of the 19x19 grid cells colored according to which class has the largest predicted probability in that cell [95].	55
2.20	In a) Ground Truth (GT) boxes are showing, one for the dog in the image and one for the cat. b) SSD default boxes at 8x8 where it is more likely to find central cells, and in c) 4×4 feature maps include both prediction vectors CONF and LOC with a much higher chance to pick the box with the right shape.	57
2.21	The Faster R-CNN architecture [27].	58
2.22	The FPN method with predictions made independently at all levels [28].	58
3.1	The (9,530,200,1) model diagram.	65
3.2	The following graph is usually called the “happy face”!	66
3.3	The following graph shows the ROC curves and AUC scores of the different models used in comparison tests.	68
3.4	The (18,530,230,128,200,1) model diagram.	69
3.5	The probability distribution of the NN Discriminant output.	70
3.6	By varying the threshold on NN Discriminant (plot in red) we got 5% more Efficiency for current mistag and 40% less Mistag for current efficiency with a Likelihood-based Discriminant (in black).	71
3.7	Best (red) VS Worst (blue) test Runs	72

3.8	Clear pattern recognition on variables such as the time difference and CDA.	73
3.9	Performance plot without CDA.	74
3.10	Lambda of Fake VS Real Kaons predicted from test data.	75
3.11	time difference between RICH and GTK of Fake VS Real Kaons predicted from test data.	75
3.12	time difference between KTAG and GTK of Fake VS Real Kaons predicted from test data.	76
3.13	CDA of Fake vs. Real Kaons predicted from test data.	76
3.14	The final fraction of $K^+ \rightarrow \pi^+\pi^0$ and other similar events entering signal regions.	77
3.15	Efficiency comparisons on one test run 8255 of 2017A data. Performance plot of NN predictions (in red) before and (in blue) after data redefinition. (In grey) The performance plot for classic Discriminant.	78
3.16	Tails versus the fractional $K^+ \rightarrow \pi^+\nu\bar{\nu}$ acceptance variation for different sets of cuts applied on the Single NN discriminant for the tight (left) and the wide (right) primary cuts.	81
3.17	Tails versus the fractional $K^+ \rightarrow \pi^+\nu\bar{\nu}$ acceptance variation for different sets of cuts applied on the 2-D NN discriminant (black dots). The performance of the Single NN discriminant cut (grey dots) and of the standard likelihood cuts (blue square) are also shown for comparison. Grey dots are the same as those of figure 3.16 (right). Lines connecting dots and the blue straight lines are for visualization only.	83
4.1	LKr's Calorimetric Visual Data: (top) energy-based visualization, (bottom) time-based visualization. Black hole showing in middle of picture is for the beam pipe of the experiment.	87
4.2	This figure taken from [6] shows the range of PV inefficiency in LKr that is prioritized for improvement.	88
4.3	This figure shows an instance where clusters are labelled and box-style annotated using CVAT tool. Single clusters are in light-blue colored boxes while Merged ones in brown-orange and negative clusters are left without annotation.	91
4.4	Left: The original network architecture that outputs probabilities for 1000 different class labels. Middle: Removing the FC layers from the network and the output of the final pooling layer will serve as the extracted features. Right: Removing the original FC layers and replacing them with a brand-new FC head. Now these can be fine-tuned to a specific dataset [102].	96

4.5	Left: At the start, all layers are frozen, and the gradient is only allowed to back-propagate through the FC layers to achieve a “warm up”. Right: Afterwards, one choice might be to unfreeze all the layers and allow each of them to be fine-tuned as well [102].	97
4.6	obj.data file	98
4.7	Computing the Intersection over Union is simply dividing the area of overlap between the bounding boxes by the area of union.	101
4.8	https://commons.wikimedia.org/wiki/File:Precisionrecall.svg	102
4.9	Precision/Recall curve for the example in hand. The black circles are showing the wiggles in precision that are avoided by using interpolation instead [111].	103
4.10	Results of initial training showing the loss (<i>blue</i>) and mAP (<i>red</i>) plots.	104
4.11	An instance of initial training predictions. True identifications in (<i>green</i>) boxes first. Then circled in (<i>blue</i>) is a FP (mid-bottom) while 2 FNs are highlighted in (<i>red</i>) (upper-right).	105
4.12	Results of second training showing the loss (<i>blue</i>) and mAP (<i>red</i>) plots	107
4.13	Circled in (<i>red</i>) is a benign case of FP (upper-middle)	108
4.14	YOLO’s second run showing the loss (<i>blue</i>) and mAP (<i>red</i>) plots. . .	109
4.15	Results showing the individual AP for classes and total mAP plots (Best peak at 15k iteration) for SSD_MobileNetV2. The vertical axis is the AP while the horizontal one is for training epochs.	110
4.16	Results showing the individual AP for classes and total mAP plots (Best peak at 16k iteration) for FasterRCNN_InceptionResNet_V2. The vertical axis is the AP while the horizontal one is for training epochs.	111
4.17	Results showing the individual AP for classes and total mAP plots (Best peak at 5k iteration) for SSD_ResNet_FPN. The vertical axis is the AP while the horizontal one is for training epochs.	112
4.18	GT: Only two detections of Single clusters should exist	113
4.19	YOLO: Circled in (<i>red</i>) is a case of FP (upper-middle)	114
4.20	SSD_ResNet_FPN: Predicted correctly	115
4.21	GT: All these detections should exist	116
4.22	YOLO: Circled in (<i>red</i>) is a case of FN (upper-middle)	117
4.23	SSD_ResNet_FPN: Predicted correctly	118
4.24	(Circled in <i>red</i>) is the MIP, here μ^+ . (Circled in <i>brown</i>) two insignificant in-time clusters with $E < 30$ MeV. (In a <i>red</i> square) is the in-time time window of $\pm 10ns$, setting the limit for all greenish gradients in the time palette of the time visualization.	118
4.25	(Circled in <i>red</i>) is the MIP, here μ^+ . (In a <i>red</i> square) is the in-time time window of $\pm 10ns$, setting the limit for all greenish gradients in the time palette of the time visualization.	119

4.26	(Circled in <i>red</i>) is the MIP, here π^+ . (In <i>red</i> ovals) two significant in-time clusters. (In a <i>red</i> square) is the in-time time window of $\pm 10ns$, setting the limit for all yellowish (exceptionally instead of greenish) gradients in the time palette for a clearer time visualization.	119
4.27	(Circled in <i>red</i>) is the MIP, here π^+ . (Circled in <i>pink</i> ovals) two insignificant out-of-time clusters at $-60ns$ and $-20ns$ out of the MIP time. (In a <i>red</i> square) is the in-time time window of $\pm 10ns$, setting the limit for all greenish gradients in the time palette of the time visualization.	120
4.28	Zoomed in look at the $-20ns$ out-of-time cluster. (In a <i>red</i> square) is the in-time time window of $\pm 10ns$, setting the limit for all greenish gradients in the time palette of the time visualization. (In a <i>black</i> square) the bulk of energy is clearly out-of-time. (In <i>pink</i> ovals) peculiar uni-cells or pixels appear on the cluster's rim and are found to be in-time with the π^+	120
A.1	A typical Neural Net graph with proper representations[95]	129
B.1	The label map for the two classes.	134
C.1	Boxed annotation of Brokk40 in CVAT	141
C.2	Classification of the Brokk40 at rest.	144
C.3	Classification of the Brokk40 whilst grabbing.	145
C.4	Correct classification of a different end-effector and its activity.	146
C.5	Correct classification of the Brokk40 and the activity in poor lighting conditions.	147
C.6	Classification showing transition in activity - (left) "Rest" and (right) "Grab"	148
C.7	Results of initial training showing the loss (<i>blue</i>) and mAP (<i>red</i>) plots	149
C.8	Correct classification of a strange kind of end-effector that is grasping an object	150
C.9	Correct (<i>left</i>) against incorrect (<i>right</i>) identifications	151
C.10	Boxed annotation of Brokk40	152
C.11	Coordinate feedback of the object detected shown within a terminal	153
C.12	Network display board	155
C.13	First training round for 30 epochs and 2000x1500 padding	156
C.14	Second training round for 200 epochs and 2500x2500 padding	157
C.15	Some successful detections from inference of the DetectNet model	158
C.16	Some failed detections from inference of the DetectNet model, a FP (<i>left</i>) and a FN (<i>right</i>)	159
C.17	Structure of the MS_COCO_endeffector dataset	160
C.18	Tensorboards showing components of training	161

C.19	The loss function plot for the test and real run	162
C.20	From left to right the corresponding plots for Accuracy, FN and FP .	163
C.21	Performance checks list for the Mask R-CNN model	164
C.22	Two successful detections (<i>top</i>) along with false positives (<i>bottom</i>) . .	165
D.1	The GPU would appear clearly in the terminal if it is running. <i>Underlined in red</i> are the GPU's name, unit number and computing capability.	166
D.2	Loading the model and ground-truth labels of the dataset.	167
D.3	A part of MobileNetV2 graph.	168
D.4	Showing above are the customised head and the training's starting point.	169
D.5	Text file displaying annotation and labelling information	169
D.6	obj.data file	170
D.7	A section of the train.txt file.	171
D.8	The obj.names file	171
D.9	DIGITS New Object Detection Data-set board	174
D.10	Job Status window showing "Done" (in <i>green</i>) or "Error" (in <i>red</i>) . .	175
D.11	New Image Model parameters board	176

List of Tables

1.1	Inputs to the <i>SES</i> evaluation, <i>SES</i> values and numbers of expected SM signal events in the S1 and S2 subsets.	16
2.1	Used Hybrid models' components	45
3.1	List of Feature Importance weights from the “permutation importance” MDA sensitivity analysis.	82
4.1	Performance checks on AP for all models	109
4.2	Selection/Rejection Efficiencies	113

Preface

The main goal of this thesis is to develop and apply Machine Learning techniques based on Neural Networks (NN) in the analysis of NA62 experiment. The methods followed were far from established in comparison with the mainstream NA62 $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ decay analysis and been carried on as a solo effort under the supervision of Giuseppe Ruggiero. NA62 analysis is founded on a cut-based traditional approach which is expected to present important limitations for the future of the experiment when the beam's intensity will reach its highest levels. This would require some innovative approaches to overcome the problem of high intensity. This goal has been achieved from at least three different perspectives: (i) it proved the feasibility and applicability of such machine learning tools in the general $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ analysis, and to show how transferable such techniques are in a multi-disciplinary context, similar applications in Robotics are also mentioned in parallel with results, (ii) presented the possibility of innovative and efficient computer vision NN-based approaches to the Liquid Krypton (LKr) Electromagnetic Calorimeter and NA62 photon veto, and, (iii) paved the way for better future implementations of such techniques. So, to say that in that sense my contribution appeared and earned a place in authorship of the main published work of NA62 Collaboration during my PhD (Papers [1][2] [5] [3] [4] [6] [7] [8]).

I have written a Neural Net (NN) code from scratch and applied it to the Kaon-Pion matching in the pion neutrino-neutrino analysis of particle physics data of NA62 at CERN. It is a difficult problem due to the wide diversity of parameters and the correlation between them. My NN code showed increased efficiency in Kaon decay identification with respect to the standard algorithm based on statistical analysis. by picking up the right dependencies and putting more weight on the most influential variables, the end results appeared very promising. It was the first Neural Net code intended to work with NA62 framework and inspired other members in the collaboration to follow on the path and convinced them of the power of such algorithms.

My other project was an implementation of a new “Virtual Bubble Chamber” technique for NA62's LKr to analyze calorimetric images of clusters in energy deposit using a NN object detector code we called NNODA (Neural Net Object Detection Approach). It is innovative unique idea to use color tags on the cluster timing to filter out and reduce random activities plaguing particle physics fixed target on-flight decay experiments like NA62. This is also a first attempt to implement a NN object detector using the fine granularity and cell time resolution of LKr, a technique rarely used in HEP calorimetry. The exceptional structure of NA62's Liquid Krypton Calorimeter

made this project possible, and we took full advantage of its granular features. The project has been presented to the Collaboration and soon to be published in an independent CERN note then a scientific paper [57]. In more details, (NNODA) is proposed to improve LKr's Photon Veto while reducing the time window selection from $\pm 40\text{ns}$ to $\pm 10\text{ns}$ for calorimetric activities around the main charged particle (MIP).

As part of my Doctorate studies, a data science internship was a requirement also. I followed through one in collaboration with the engineering department at Lancaster University and NNL (National Nuclear Laboratory). The focus of my training was on applying advanced Machine Learning algorithms in the field of robotics. I have gained skills using Computer Vision techniques in overly complicated real-life scenarios, like training an algorithm that would help a drone to identify and locate robotic end-effectors in an unusual environment. I learned to create and generate synthetic data to boost the performance of the algorithm and add volume and quality to datasets used in training. These skills have been effectively transported to NA62 particle data analysis in a multi-disciplinary fashion.

I have analyzed the data of NA62 for the $K^0 \rightarrow \pi^0 \nu \bar{\nu}$ and two neutrinos (The exceedingly rare Standard Model Kaon decay to a pion and two neutrinos) analysis. My focus was on the association between K^0 and π^0 , and on clusters in the LKr (Liquid Krypton) electromagnetic calorimeter for photon rejection. I have gained skills in reducing, selecting, organizing, and arranging generalized and well varied data in efficient datasets. As well as labeling and annotating them to be used in training Machine Learning and Computer vision algorithms to serve the main goal of the general analysis. Also developed a lot of experience in choosing the right machine learning techniques that best suit the type of data in hand and the main objective of the case studied. I have gained high-level skills in A.I. analysis algorithms, CNN (Convolutional Neural Nets) (e.g., MobilNet, ResNet, Inception Net etc...) applied to computer vision for image recognition and Object Detectors in object detection problems. I developed expertise level of using API platforms like Darknet and TensorFlow to make full use of these powerful algorithms (e.g., YOLO (You Only Look Once), FPN (Feature Pyramid Network), mask/Faster-RCNN, SSD etc...) on a wide spectrum of datasets.

My contributions lead to positive research results through effective organization, prioritization, and follow-through of key physics data analysis projects. While I was independently motivated, I appreciated collective efforts and collaborated productively within group settings. I have delivered talks and made several presentations at the NA62 meetings and at Lancaster University Data Science group

events. I have interacted and cooperated with physics coordinators and senior members of the Physics Department and NA62 in various occasions and on many subjects. I also taught 2nd year labs in a PGTA (Postgraduate Teaching Assistant) roles for three consecutive years and tutored physics for Further Education, Higher Education and Under-Grad level Physics & Engineering Degrees. I have also helped in the training and mentoring of inexperienced staff members and am knowledgeable about hazard safety and lab procedure (e.g., I helped and taught Master students Linux operation and machine learning in computer vision on many occasions). Examples of experiments I mastered and taught are Michelson Interferometer, Bridges in Electrical Circuits, Field Effect Transistor, Blackbody Radiation, Waves on a Coaxial Line, Fourier Analysis Using a Filter, Spectroscopic Study of One Electron Atoms, Optical Absorption, Brewster Angles, and the Polarization-Dependence of Reflection Coefficients, Hall Effect, Solar Cells, and others similar. I did also my required full share of shifts in monitoring the experiment and handling extremely complicated control and data-taking software. These shifts have provided me with an opportunity to get to know my colleagues at the collaboration since we worked together for long hours. The diversity of NA62 and the Physics personnel enriched my sense of cultural understanding and acceptance of differences as strengths in cooperation. On the other hand, I have proven my ability to taking on responsibility independently by developing Deep Learning algorithms that proved efficient which qualified me as one of the few DL specialists in NA62 Collaboration.

For all this collected experience through my PhD years, I am very much grateful and thankful. I could not have made it through the pandemic and all without the people that helped me and guided me through it all.

Chapter 1

Kaon Physics of NA62

1.1 Current Status of Kaon Physics

Since their discovery, in 1947 [100], kaons played a crucial role in the understanding of fundamental interactions. For example, the first CP symmetry violation evidence came with the famous kaon decay experiment performed by Christenson et al. in 1964 [63].

At the LHC now direct and indirect searches for New Physics (NP) are running at the highest intensities. The flavor physics sector is continuously investigated, with processes involving strange and beauty quarks. Heavy flavors are explored with the LHCb [85] and Belle II [33] experiments and the kaon physics programme complements such studies involving the strange quark, however, on a lower mass scale.

The fundamental role of quark mixing in weak interactions became well known after the insights proposed by Cabibbo [56]. He realized that the weak eigenstates of quarks are not exactly the mass eigenstates but a linear combination, of them, rotated by an angle θ named after him.

After Cabibbo, and using his model, Glashow, Iliopoulos and Maiani [70] discovered the so-called GIM mechanism that explains the suppression of Flavor Changing Neutral Current (FCNC) processes. One of which is the $K_L \rightarrow \mu^+\mu^-$ decay, whose branching ratio is recently observed to be smaller than 10^{-8} [98]. They postulated the existence of a new up-like quark, the charm, following the experimental evidence that shows heavy suppression of decay amplitudes with $\Delta S=2$ transitions such as $K_L \rightarrow \mu^+\mu^-$. Their intuition was that decay modes with up-like quarks should cancel out modes with up quarks leaving a near 0 residual amplitude.

The $K^+ \rightarrow \pi^+\nu\bar{\nu}$ decay, on the other hand, is strongly related to the Cabibbo-Kobayashi-Masakawa (CKM) matrix of weak interactions, introduced for the first time in 1973 [81]. Generalizing the Cabibbo theory, CKM matrix defines the probability of

transition between quark flavors by constructing weak interaction eigenstates in the following flavor mix:

$$\begin{pmatrix} d' \\ s' \\ b' \end{pmatrix}_{weak} = \begin{pmatrix} V_{ud} & V_{us} & V_{ub} \\ V_{cd} & V_{cs} & V_{cb} \\ V_{td} & V_{ts} & V_{tb} \end{pmatrix} \times \begin{pmatrix} d \\ s \\ b \end{pmatrix}_{flavor} = V_{CKM} \times \begin{pmatrix} d \\ s \\ b \end{pmatrix}_{flavor} \quad (1.1)$$

An arbitrary and non-unique choice is to exclusively mix the down-like quarks as is commonly used in standard formalism. A natural consequence of the weak interaction universality is the unitarity of the CKM matrix, providing a powerful experimental test for the Standard Model. The unitarity requirement and gauge invariance reduce the free parameters of CKM matrix to three mixing angles and a complex phase that leads to CP symmetry violation.

Another useful way to represent the CKM matrix is by using the Wolfenstein parametrization [115], defined as follows:

$$\lambda = \frac{|V_{us}|}{\sqrt{|V_{ud}|^2 + |V_{us}|^2}} \quad (1.2)$$

$$A = \frac{|V_{cb}|}{\lambda^2 \sqrt{|V_{ud}|^2 + |V_{us}|^2}} \quad (1.3)$$

$$\rho = \frac{Re(V_{ub})}{A\lambda^3} \quad (1.4)$$

$$\eta = \frac{Im(V_{ub})}{A\lambda^3} \quad (1.5)$$

and since,

$$V_{ub}^* = A\lambda^3(\rho + i\eta) = \frac{A\lambda^3(\bar{\rho} + i\bar{\eta})\sqrt{1 - A^2\lambda^4}}{\sqrt{1 - \lambda^2[1 - A^2\lambda^4(\bar{\rho} + i\bar{\eta})]}} \quad (1.6)$$

these relations led to a final phase convention independent relation:

$$\bar{\rho} + i\bar{\eta} = \frac{-(V_{ud}V_{ub}^*)}{(V_{cd}V_{cb}^*)} \quad (1.7)$$

that ensures the CKM matrix written in terms of λ , A , $\bar{\rho}$, and $\bar{\eta}$ is unitary to all orders in λ . Moreover, being experimentally proven to take a small value ($\lambda \simeq |V_{us}| = 0.2243 \pm 0.0005$ [98]), λ allows a power series expansion of the matrix terms that

highlights the hierarchy of the CKM elements and is traditionally approximated to third order in the following manner:

$$V_{CKM} = \begin{pmatrix} 1 - \frac{\lambda^2}{2} & \lambda & A\lambda^3(\rho - i\eta) \\ -\lambda & 1 - \frac{\lambda^2}{2} & A\lambda^2 \\ A\lambda^3(1 - \rho - i\eta) & -A\lambda^2 & 1 \end{pmatrix} + O(\lambda^4) \quad (1.8)$$

While $\bar{\rho}$ and $\bar{\eta}$ can also be approximated to leading orders in λ as:

$$\bar{\rho} = \rho(1 - \lambda^2/2 + \dots) \quad (1.9)$$

and

$$\bar{\eta} = \eta(1 - \lambda^2/2 + \dots) \quad (1.10)$$

The CKM matrix elements are amongst free and fundamental parameters of the SM. To add experimental constraints on them is crucial to our understanding of SM and searches for NP. The key to achieving that is unitarity which imposes the following natural conditions:

$$\sum_i V_{ij}V_{ik}^* = \delta_{jk}, \quad \sum_j V_{ij}V_{kj}^* = \delta_{ik} \quad (1.11)$$

Six triangles in a complex plane are obtained from the vanishing combinations. Out of which the ones obtained by taking the scalar products of neighboring rows or columns are of particular interest and used to test CP violation of Jarlskog invariant [80]. Hence, the one triangle most commonly used is derived from

$$V_{ud}V_{ub}^* + V_{cd}V_{cb}^* + V_{td}V_{tb}^* = 0 \quad (1.12)$$

and have a well-defined geometric representation in the $(\bar{\rho}, \bar{\eta})$ plane when dividing by the best measured elements (i.e., $V_{cd}V_{cb}^*$) as figure 1.1 shows. The current measured values and global fit results of the Wolfenstein parameters can be found in ‘‘Review of Particle Physics’’ [98]. No experimental deviation from the unitarity condition has emerged so far.

The kaon sector provides a test of the SM by measuring the branching ratios of $K^+ \rightarrow \pi^+\nu\bar{\nu}$ and $K_L \rightarrow \pi^0\nu\bar{\nu}$. These rare decay channels allow a measurement of the unitarity triangle parameter β , which makes it sensitive to NP in an independent domain of parameter space and complementary to B decay measurements [48].

The *FCNC* decay $K^+ \rightarrow \pi^+\nu\bar{\nu}$ involves the transition $s \rightarrow d\nu\nu$ which, in the SM, is only allowed via Z^0 penguin and box diagrams (See figure 1.2). The amplitude of each diagram is a sum of three contributions to the loops from all three up-like quarks.

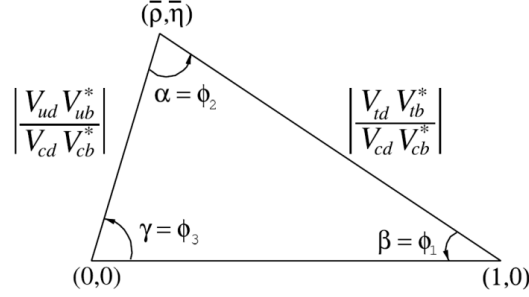


Figure 1.1: A typical CKM unitarity triangle.

The amplitude of such contributions can be approximated, due to GIM mechanism, as:

$$A_q \simeq \frac{m_q^2}{m_W^2} V_{qs}^* V_{qd} \quad (1.13)$$

The amplitude will apparently become dominated by top quark contribution due to large mass squared. The quadratic term of GIM mechanism multiplied by an extremely small value of the CKM element V_{td} makes the transition from a top into a down quark, therefore, $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ decay, extremely rare.

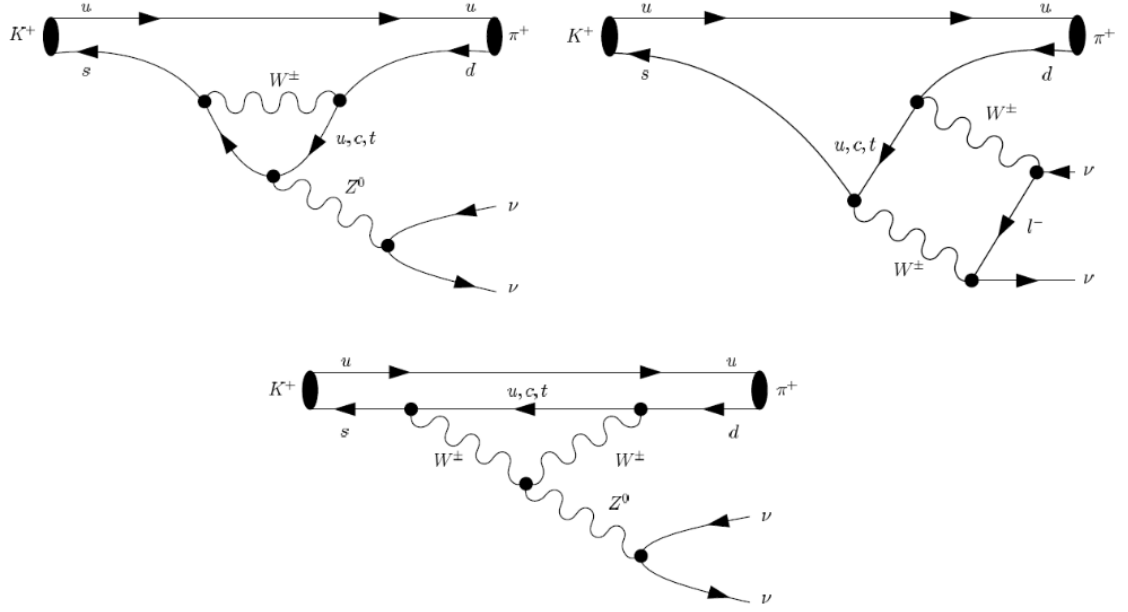
Summing up over all three neutrino flavors, while considering small but non-negligible contribution of charm quark, gives a final form for the $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ branching ratio [53] [50]:

$$Br(K^+ \rightarrow \pi^+ \nu \bar{\nu}) = k_+ (1 + \Delta_{EM}) \left[\left(\frac{Im \lambda_t}{\lambda^5} X(x_t) \right)^2 + \left(\frac{Re \lambda_c}{\lambda} P_c(X) + \frac{Re \lambda_t}{\lambda^5} X(x_t) \right)^2 \right] \quad (1.14)$$

$$k_+ = (5.173 \pm 0.025) \cdot 10^{-11} \left[\frac{\lambda}{0.225} \right]^8, \Delta_{EM} = -0.003 \quad (1.15)$$

where:

- k_+ (and its electromagnetic correction Δ_{EM}) summarizes the hadronic contributions, that can be extracted experimentally from semi-leptonic decays $K^+ \rightarrow \pi^0 l^+ \nu$ ($l = e, \mu$) using Isospin symmetry between π^+ and π^0 [91]. This is one of the crucial aspects that strongly improves the theoretical uncertainty.
- $X(x_t)$ is the loop function of top quark contribution, with $x_t = m_t^2/m_W^2$. Its most recent value, including two-loop electroweak corrections [44] and NLO QCD corrections [49], is $X(x_t) = 1.481 \pm 0.005 \pm 0.008$ [53].

Figure 1.2: The penguin and box diagrams of $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ [52]

- $P_c(X)$ is the loop function of charm quark contribution. It is computed as sum of short-distance (0.365 ± 0.012) and long-distance (0.04 ± 0.02) contributions [53][43][76].
- $\lambda = |V_{us}|$ and $\lambda_q = V_{qs}^* V_{qd}$ are factors from CKM matrix.

The final value for $Br(K^+ \rightarrow \pi^+ \nu \bar{\nu})$ in the SM context is computed in [53] to be:

$$Br^{SM}(K^+ \rightarrow \pi^+ \nu \bar{\nu}) = (0.84 \pm 0.10) \cdot 10^{-10} \quad (1.16)$$

To better understand the 10% global uncertainty, it is practical to factorize as follows[53]:

$$Br^{SM}(K^+ \rightarrow \pi^+ \nu \bar{\nu}) = (0.839 \pm 0.030) \cdot 10^{-10} \cdot [|V_{cb}| / 40.7 \cdot 10^{-3}]^{2.8} \cdot [\gamma / 73.2^\circ]^{0.74} \quad (1.17)$$

where γ is the angle from unitarity triangle illustrated in figure 1.1. Obviously, from the factorized form of equation 1.17, the uncertainty is dominated by the knowledge of the CKM parameters V_{cb} and γ . Similarly, the branching ratio of $K_L \rightarrow \pi^0 \nu \bar{\nu}$ is computed in [53] as:

$$Br^{SM}(K^L \rightarrow \pi^0 \nu \bar{\nu}) = (0.34 \pm 0.06) \cdot 10^{-10} \quad (1.18)$$

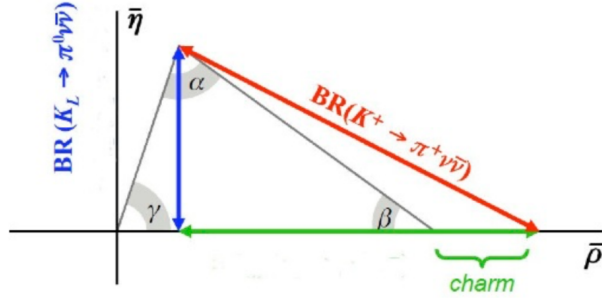


Figure 1.3: Relations between the unitarity triangle and the $K \rightarrow \pi \nu \bar{\nu}$ decays [42].

The combined BR measurements of both decay modes would allow a precise estimation of two elements of the unitarity triangle as shown in figure 1.3. The measurement of the neutral channel allows us to directly extract the height of the unitarity triangle: it is a CP-violating process and only the imaginary part of the decay amplitude is involved. The charged channel, instead, receives contributions by both the imaginary and the real component of the decay amplitude. The imaginary part is the same as for the neutral channel, while the real part is provided by the top, within the unitarity triangle, and by the charm quark. The $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ decay provides, therefore, the measurement of one side of the unitarity triangle, after the correction for the charm contribution which is mainly dependant on the charged lepton mass included in the box diagrams. Additionally, only electron and muon effects are counted while the τ lepton contribution is negligible [42].

Beyond Standard Model (BSM) physics predicts a whole new breed of particle contributions to $K \rightarrow \pi \nu \bar{\nu}$ loopy “Penguinology” and, therefore, large deviations in the branching fractions of both charged and neutral channels. The largest deviations would be derived from new sources of flavor violation [34][35]. Also, the experimental value of the CP violation parameter ϵ_k limits the expected $Br(K^+ \rightarrow \pi^+ \nu \bar{\nu})$ range within models with currents of defined chirality. An instance amongst models reviewed in [51] is one with a new heavy gauge boson Z' that generates FCNC processes at tree level. Such models can predict correlations between $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ and $K_L \rightarrow \pi^0 \nu \bar{\nu}$ decays for different masses of Z' and Left/Right-Handed (LH/RH) chiral couplings $\Delta_{L,R}$. Other similar correlations are predicted by Constrained Minimal Flavor Violation (CMFV) models[84] (See figure 1.4). Additionally, present experimental constraints limit the range of variation of $Br(K^+ \rightarrow \pi^+ \nu \bar{\nu})$ in supersymmetric models as well [77][36][112]. The $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ decay is also affected by lepton flavor non-universality models, which assume that NP particles are only coupled to third generation quarks and leptons [39][54]. $Br(K^+ \rightarrow \pi^+ \nu \bar{\nu})$

measurement can also constrain models with leptoquarks [37][65]. Finally, it is important to note that, under the assumptions of lepton flavor conservation and fixed Isospin jumps between underlying interactions ($\Delta I = 1/2$), the Grossman-Nir [73] bound is defined as:

$$Br(K^L \rightarrow \pi^0 \nu \bar{\nu}) < 4.4 \cdot Br(K^+ \rightarrow \pi^+ \nu \bar{\nu}) \quad (1.19)$$

and respected by the SM and most scenarios beyond it. However, a most recent study showed a possibility for its violation [74].

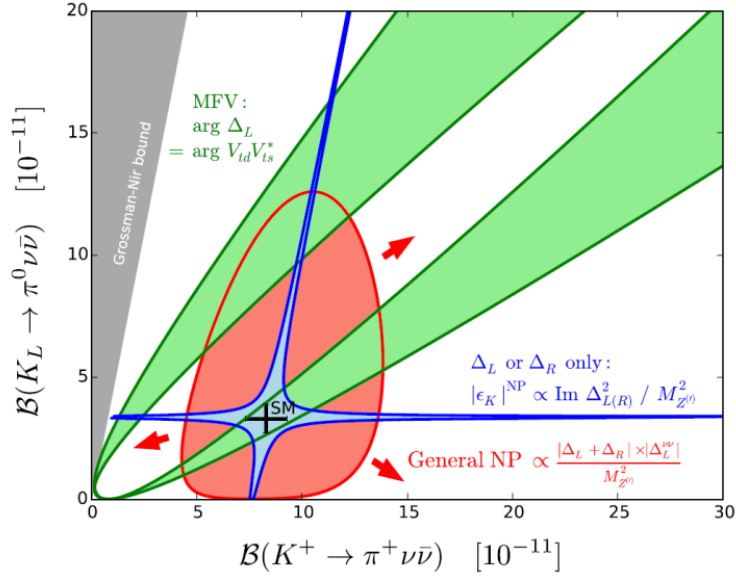


Figure 1.4: Plots of popular correlations in the $Br(K_L \rightarrow \pi^0 \nu \bar{\nu})$ versus $Br(K^+ \rightarrow \pi^+ \nu \bar{\nu})$ plane. The expanding red region represents lack of correlation for models with general LH and/or RH NP couplings with Z' . Green region is the correlation zone of models conforming to CMFV. While blue region illustrates the correlation induced by ϵ_k constraints in models with exclusively LH or RH couplings [51].

In summary, extreme SM suppression makes these decays particularly sensitive to NP. In model independent terms the $K \rightarrow \pi \nu \nu$ decays probe NP at the highest mass scales [67, 10, 79, 68, 64]. Existing experimental constraints on NP affect the $K \rightarrow \pi \nu \nu$ weakly, especially those on minimal flavour violating models. Model dependent scenarios predict deviation of the branching ratios from the SM as large as 30-40% and correlations between the charged and neutral modes, depending on the model [87, 88, 9, 78, 69, 90, 110, 89, 55, 106]. Therefore, measurements of the

$K \rightarrow \pi \nu \bar{\nu}$ branching ratios with $O(10\%)$ precision at least are needed to pin down NP effects.

1.2 Experimental Status of $K^+ \rightarrow \pi^+ \nu \bar{\nu}$

NA62 is a fixed target experiment running at CERN SPS designed to study K^+ physics. The primary goal of NA62 is to measure precisely the $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ branching ratio.

The experiments E787 and E949 at Brookhaven National Laboratory [61] studied the $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ decay using a kaon decay-at-rest technique in the early 2000 and measured $BR = (17.3^{+11.5}_{-10.5}) \times 10^{-11}$.

The NA62 experiment makes use of a novel kaon decay-in-flight technique and took data in 2016, 2017 and 2018. The analysis of the 2016 data demonstrated the validity of the new technique in terms of background rejection power and provided the observation of 1 candidate event, anyway compatible with the expected background.

With the 2017 data analysis NA62 surpassed the sensitivity of the $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ SM and observed two candidates' events. This observation was compatible either with the signal plus background or with the background only hypothesis, leading to the upper limit on the branching ratio $BR < 1.78 \times 10^{-10}$ at 90% C.L.

The 2018 data analysis provided 17 candidates observed. The combination of this analysis together with 2016 and 2017 analysis led to a 3.5σ evidence of the $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ decay and to the most precise up-to-date measurement of the branching ratio

$$BR = (10.6^{+4.0}_{-3.4}|_{stat} \pm 0.9_{syst}) \times 10^{-11}. \quad (1.20)$$

After the long shutdown 2 of CERN, the experiment has resumed data taking in 2021 and is scheduled to take data until 2024. The aim is to increase the beam intensity of 30-40% with respect to the previous running conditions, to achieve enough statistics to measure the $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ branching ratio with 10% precision.

The signature of the $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ decay is a single π^+ and missing energy. The squared missing mass, $m_{miss}^2 = (P_K - P_{\pi^+})^2$, where P_K and P_{π^+} indicate the 4-momenta of the K^+ and π^+ , describes the kinematics of the one-track final state. In particular, the presence of two neutrinos makes the signal broadly distributed over the m_{miss}^2 range, as illustrated in Figure 1.5. The dominant K^+ decay modes $K^+ \rightarrow \mu^+ \nu$, $K^+ \rightarrow \pi^+ \pi^0$ and $K^+ \rightarrow \pi^+ \pi^{+(0)} \pi^{-(0)}$ have different m_{miss}^2 distributions; it is therefore

possible to define regions, either side of the $K^+ \rightarrow \pi^+ \pi^0$ peak, qualitatively indicated in Figure 1.5, where the search for the signal is performed, also known as *signal regions* [1].

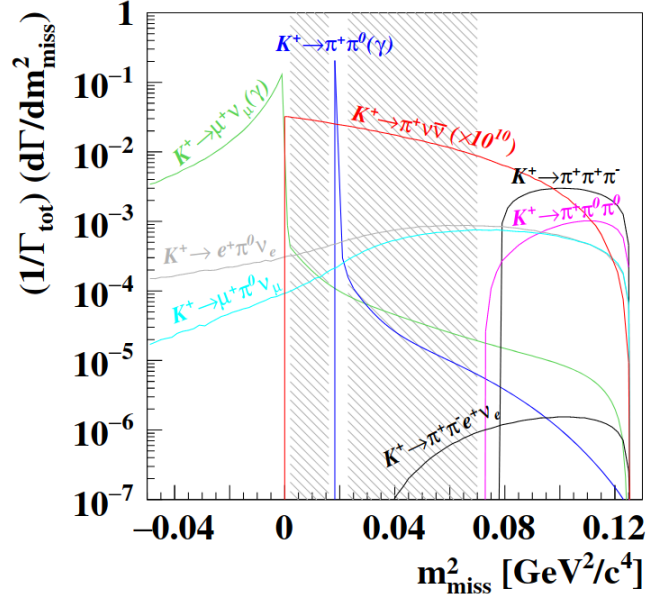


Figure 1.5: Expected theoretical distributions of the m_{miss}^2 variable relevant to the $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ measurement. The m_{miss}^2 is computed under the hypothesis that the charged particle in the final state is a π^+ . The $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ signal (red line) is multiplied by 10^{10} for visibility. The hashed areas include the signal regions.

The $K^+ \rightarrow \mu^+ \nu$, $K^+ \rightarrow \pi^+ \pi^0$ and $K^+ \rightarrow \pi^+ \pi^{+(0)} \pi^{-(0)}$ decays enter the signal regions through radiative and/or resolution tails of the reconstructed m_{miss}^2 . The signal selection, based on kinematics only, relies on the accurate measurement of the m_{miss}^2 quantity, i.e. of the K^+ and π^+ momenta and directions. In contrast, $K^+ \rightarrow \pi^0 l^+ \nu$ or rarer decays, like $K^+ \rightarrow \pi^+ \pi^- l^+ \nu$, span over the signal regions because of the presence of undetected neutrinos; however, these background decays modes include a lepton in the final state and exhibit extra activity in the form of photons or charged particles. A particle identification system must therefore separate π^+ from μ^+ and e^+ . A track matching of the highest form should be performed between the K^+ and π^+ in the final state to insure a suppression of background by at least three orders of magnitude. Moreover, photons and additional charged particles in final state must be vetoed as efficiently as possible. One of the critical points of the analysis of the old data was the intensity dependence, that lead to a decrease of

signal efficiency with intensity. The work done for this thesis aims to tackle this issue through the application of advanced neural network techniques to specific aspects of the $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ data analysis, namely the $K^+ - \pi^+$ matching and the veto of the photons using the electromagnetic calorimeter.

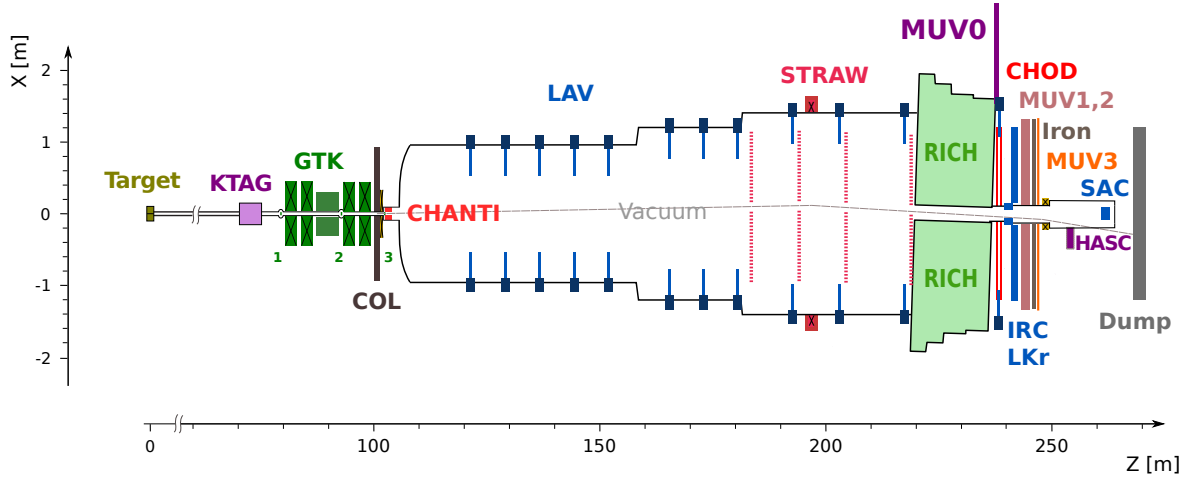


Figure 1.6: Schematic top view of the NA62 beamline and detector.

1.3 The beam and detector

The layout of the NA62 beamline and detector [14] is shown schematically in Figure 1.6. A secondary hadron beam of positive charge with a nominal momentum of $75 \text{ GeV}/c$, extracted from the CERN SPS, reaches a beryllium Target in extraction-pulses (spills) each lasting about 5s (3s effectively). The beam is made of π^+ (70%), protons (23%) and K^+ (6%). A differential Cherenkov counter (KTAG) using a nitrogen radiator tags the K^+ beam with 70 ps rms time resolution. Three silicon pixel stations (GTK) measure momentum, direction, and time of beam particles with resolutions of about $0.15 \text{ GeV}/c$, $16 \mu\text{rad}$ and 100 ps, respectively.

A collimator between the second and third GTK stations (final collimator) defines the beginning of a 120 m long vacuum decay region. Six stations of plastic scintillation bars (CHANTI) detect the extra activity produced in inelastic interactions of beam particles with the last GTK station. At the entrance of the vacuum decay region, the particle rate was about 500 MHz during the 2018 data taking period. The first 80 m

of the decay region defines a fiducial volume (FV) where about 13% of the K^+ decays.

A magnetic spectrometer (STRAW) made of four straw chambers located in the vacuum region downstream of the FV traces charged K^+ decay products, measuring their momenta thanks to a dipole magnet placed between the second and the third chamber, providing a horizontal momentum kick of about 270 MeV/ c .

A ring-imaging Cherenkov detector (RICH) for charged particle identification measures the time of particles above threshold to better than 100 ps resolution.

A matrix of scintillation tiles readout by two SiPMs each (CHOD) and two planes of horizontal and vertical scintillation slabs built for the NA48 experiment [17] (NA48-CHOD) provide a 99% efficient trigger and a 200 ps resolution time measurement for charged particles.

A quasi-homogeneous electromagnetic calorimeter at liquid Krypton (LKr) detects photons emitted forward in K^+ decays with angles between 1 and 10 mrad and complements RICH for particle identification.

A two-module hadronic iron/scintillator-strip sampling calorimeter (MUV1,2) supplements RICH and LKr for π^+ identification. Behind an 80 cm thick iron wall, a matrix of 148 scintillation tiles (MUV3), each of them read-out by two PMs, detects μ^+ with 400 ps time resolution.

Twelve stations of annular electromagnetic calorimeters made of lead glass crystals (LAV) surround the FV and the downstream regions to achieve hermetic acceptance for photons emitted in K^+ decays in the FV at polar angles up to 50 mrad. A lead/scintillator sampling calorimeter (IRC) located in front of the LKr, covers an annular region between 65 and 135 mm from the beam axis. The IRC and a similar detector (SAC) placed on the beam axis at the downstream end of the set-up ensures the detection of photons down to 0 angle. A dipole magnet between MUV3 and SAC bends the beam of charged particles out of the SAC acceptance.

Two off-acceptance detectors placed laterally close to the CHOD (MUV0) and to the beam pipe before the SAC (HASC) ensure additional detection of charged particles emitted in decays like $K^+ \rightarrow \pi^+\pi^+\pi^-$ and directed at large and small angles, respectively.

NA62 collected the data for the $K^+ \rightarrow \pi^+\nu\bar{\nu}$ analysis through a dedicated two-level trigger algorithm (PNN). The uppermost level (L0) required a signal in RICH to tag a

charged particle in coincidence within ± 10 ns with at least one hit tile in CHOD and no signals in diagonally opposed CHOD quadrants to reduce $K^+ \rightarrow \pi^+ \pi^+ \pi^-$ decays; no particle detected in MUV3 against $K^+ \rightarrow \mu^+ \nu$; less than 30 GeV in LKr and no more than 1 isolated in-time cluster (L0Calo) to suppress $K^+ \rightarrow \pi^+ \pi^0$ decays. The higher trigger level (L1) required: a kaon identified in KTAG and no energy deposited in more than two crystals of any LAV station, within ± 10 ns of the L0 trigger RICH time; at least one STRAW track corresponding to a particle with momentum below 50 GeV/c and forming a vertex with the nominal beam direction upstream of the first STRAW chamber. Additional data stream has been collected using a minimum-bias trigger based on the presence of CHOD signals and a trigger specific for 3-track topology.

The work presented in this thesis made use of the data from 2017 both from PNN, minimum-bias and 3-track trigger streams.

1.4 Analysis method

The experimental signature of the $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ decay consists of a K^+ with 4-momentum P_K in the initial state and a π^+ with 4-momentum P_π and missing energy in the final state. The kinematic variable used to discriminate between the signal and background from K^+ decays is the squared missing mass $m_{\text{miss}}^2 = (P_K - P_\pi)^2$. This variable defines two regions where to search for the signal (Region 1 and Region 2, as shown in Figure 2), and to separate it from the other K^+ decay backgrounds.

The measurement of $\text{BR}(K^+ \rightarrow \pi^+ \nu \bar{\nu})$ relies on the calculation of the single event sensitivity (SES) and the background evaluation. The SES is defined as $1/(N_{K^+} \cdot \epsilon_{\pi \nu \bar{\nu}})$, where N_{K^+} is the effective number of K^+ decays occurring in a pre-defined decay region and $\epsilon_{\pi \nu \bar{\nu}}$ is the signal efficiency. The $K^+ \rightarrow \pi^+ \pi^0$ decays are used as normalization to compute N_{K^+} . Signal and normalization decays share the same selection defined by the presence of a single π^+ forming a vertex with a parent K^+ inside the decay region. The rejection of extra activity from photons or charged particles is applied only to the signal selection. Control regions (Figure 1.7) are used to validate the background estimates. Control and signal regions are masked until the completion of the analysis to avoid bias during the optimization of the selection conditions. The $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ branching ratio is obtained from a binned log-likelihood fit using the signal acceptance and background expectation.

The analysis is optimized separately in six π^+ momentum bin, 5 GeV/c wide, spanning from 15 to 45 GeV/c.

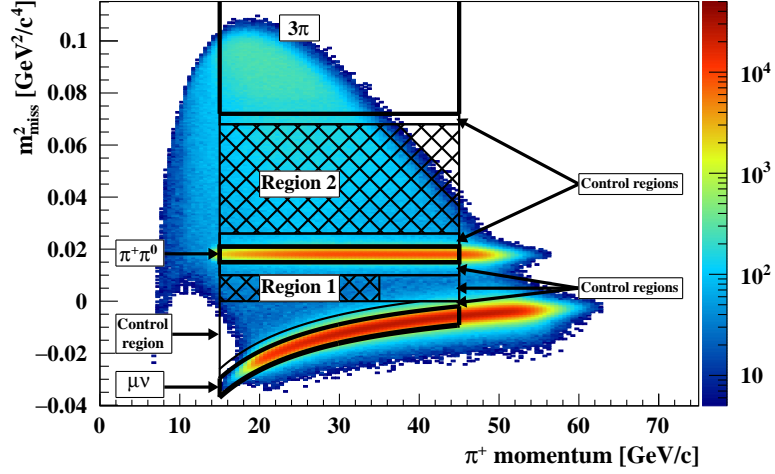


Figure 1.7: Reconstructed m_{miss}^2 as a function of p_{π^+} for minimum-bias events selected without applying π^+ identification and photon rejection, assuming the K^+ and π^+ mass for the parent and decay particle, respectively. Signal regions 1 and 2 (hatched areas), as well as 3π , $\pi^+\pi^0$, and $\mu\nu$ background regions (solid thick contours) are shown. The control regions are located between the signal and background regions.

1.5 Event selection

The signal and normalization channels both require the presence of a track identified as a π^+ in the detector downstream (“Downstream charged particle”) and of a parent K^+ track in the detectors upstream that form a vertex in the main decay volume together with the π^+ track. Additional specific selection requirements distinguish normalization from signal events.

Downstream charged particle: One or two isolated STRAW tracks are allowed in an event. If two STRAW tracks are present, the one closest to the trigger time is selected. Events with a negatively charged STRAW track are rejected to remove $K^+ \rightarrow \pi^+\pi^+\pi^-$ and $K^+ \rightarrow \pi^+\pi^-e^+\nu$ decays. The selected track must be within the detector-sensitive regions and spatially associated to signals in the RICH, CHOD, NA48-CHOD, and LKr. The track angle measured after the spectrometer magnet must be geometrically compatible with the center of the reconstructed RICH ring. Time constraints are imposed on the associated signals in the RICH, CHOD and LKr using the NA48-CHOD time as a reference.

Parent K^+ : The parent K^+ of a selected downstream charged particle is defined

by the signal in KTAG with time T_{KTAG} closest in time and within 2 ns of the downstream particle, and a beam track in GTK with time T_{GTK} within 600 ps of the KTAG signal and associated in space with the downstream track in the STRAW. The association between the GTK, KTAG and STRAW signals relies on a discriminant built from the time difference $\Delta T = T_{\text{GTK}} - T_{\text{KTAG}}$ and the closest distance of approach (CDA) of the downstream charged particle to the GTK track. The templates for the ΔT and CDA distributions of K^+ decays are obtained from a dedicated sample of $K^+ \rightarrow \pi^+\pi^+\pi^-$ decays, where the K^+ is fully reconstructed using the pion momenta and directions measured by the STRAW. The GTK track with the largest value of the discriminant is then identified as the parent K^+ . The selected K^+ must be consistent with the nominal beam momentum and direction and the CDA, which has a resolution of about 1 mm, must be less than 4 mm. No more than five reconstructed GTK tracks are allowed.

Kaon decay: The mid-point of the segment at the CDA of the downstream charged particle to the parent K^+ defines the kaon decay vertex. The Z position of the kaon decay vertex (Z_{vertex}) must be inside the region 110–165 m referred to as the fiducial volume (FV) in the following. In addition, the FV of the first p_{π^+} bin (15–20 GeV/c) is limited to 110–155 m to reduce the background from $K^+ \rightarrow \pi^+\pi^0$ decays. In the last two momentum bins (35–45 GeV/c) the FV is limited to 110–160 m to suppress $K^+ \rightarrow \pi^+\pi^-e^+\nu$ decays and upstream background, which dominate at high momentum. Further Z_{vertex} -dependent constraints are imposed on the angle of the downstream charged particle. The analysis of the 2016, 2017 and part of the data employ a cut on the charged particle backward-extrapolated position at the exit of the final collimator (COL) that must be outside a rectangular box with transverse dimensions $100 \times 500 \text{ mm}^2$. This requirement was needed to avoid background of pions coming from the beam line and passing through cracks of the final collimator. A more hermetic final collimator was installed in 2018. For the sample collected thereafter, a Boosted Decision Tree (BDT) algorithm is trained on an out-of-time data sample enriched in upstream K^+ decays. A cut on the resulting BDT output value is chosen to provide the same background rejection as a cut-based selection using the same variables, while increasing the signal acceptance by 8%.

Pion identification: The π^+ identification uses information from the calorimeters and the RICH and requires that no signal is reconstructed in MUV3 within 7 ns of the π^+ time. A BDT algorithm combines 13 variables describing the energy associated with the π^+ in the calorimeters, the shape of the clusters and the energy sharing between LKr, MUV1 and MUV2. Samples of π^+ , μ^+ and e^+ selected from 2017 data not included in the present analysis are used for training. The π^+ identification by the RICH uses two different approaches to reconstruct a Cherenkov ring. In the first

approach, the track direction is used to predict the position of the ring center, and the expected ring radius is calculated using the track momentum. In the second approach, the ring center and radius are determined by a χ^2 fit to the hit positions, and the charged particle mass is derived using the track momentum. A cut on the measured mass is then applied to distinguish Pions from muons. Particle identification criteria with the calorimeters and RICH are optimized separately to achieve the best signal sensitivity.

Normalization selection: The selection of the normalization $K^+ \rightarrow \pi^+\pi^0$ events is applied to minimum-bias data and requires $0.010 < m_{\text{miss}}^2 < 0.026 \text{ GeV}^2/c^4$ and $15 < p_{\pi^+} < 45 \text{ GeV}/c$. The width of the *normalization region* is defined to be $\pm 8\sigma$, where σ is the m_{miss}^2 peak resolution.

Signal selection: The selection of the signal events is applied only to PNN data and requires that no in-time photons or additional charged particles are present. An in-time photon in the LKr is defined as an energy cluster located at least 100 mm away from the π^+ impact point and coincident in time with the π^+ . The size of the time coincidence window varies with the amount of deposited energy and ranges from ± 5 ns below 1 GeV to ± 50 ns above 15 GeV. In-time photons are identified if a signal is found within 3 ns of the π^+ time.

Multi-charged particle rejection prevents interactions of photons or charged particles in the RICH mirrors, and $K^+ \rightarrow \pi^+\pi^+\pi^-$ or $K^+ \rightarrow \pi^+\pi^-e^+\nu$ decays with partially reconstructed STRAW tracks.

The two $K^+ \rightarrow \pi^+\nu\bar{\nu}$ signal regions are defined in the $(p_{\pi^+}, m_{\text{miss}}^2)$ plane as:

Region 1: $0 < m_{\text{miss}}^2 < 0.010 \text{ GeV}^2/c^4$ and $15 \text{ GeV}/c < p_{\pi^+} < 35 \text{ GeV}/c$;

Region 2: $0.026 < m_{\text{miss}}^2 < 0.068 \text{ GeV}^2/c^4$ and $15 \text{ GeV}/c < p_{\pi^+} < 45 \text{ GeV}/c$.

Additional constraints are also imposed on the m_{miss}^2 value to reduce the kinematic tails due to multiple scattering in the STRAW or wrong K/π association.

The minimum momentum value is fixed at $15 \text{ GeV}/c$ by the RICH threshold for efficient pion detection. The maximum value is fixed at $35 \text{ GeV}/c$ in Region 1, because the $K^+ \rightarrow \mu^+\nu$ decay distribution approaches the signal region at high momenta, and at $45 \text{ GeV}/c$ in Region 2 to remove $K^+ \rightarrow \pi^+\pi^-e^+\nu$ and upstream backgrounds (see Figure 1.7).

	Subset S1	Subset S2
$N_{\pi\pi} \times 10^{-7}$	3.14	11.6
$A_{\pi\pi} \times 10^2$	7.62 ± 0.77	11.77 ± 1.18
$A_{\pi\nu\bar{\nu}} \times 10^2$	3.95 ± 0.40	6.37 ± 0.64
$\epsilon_{\text{trig}}^{\text{PNN}}$	0.89 ± 0.05	0.89 ± 0.05
ϵ_{RV}	0.66 ± 0.01	0.66 ± 0.01
$SES \times 10^{10}$	0.54 ± 0.04	0.14 ± 0.01
$N_{\pi\nu\bar{\nu}}^{\text{exp}}$	$1.56 \pm 0.10 \pm 0.19_{\text{ext}}$	$6.02 \pm 0.39 \pm 0.72_{\text{ext}}$

Table 1.1: Inputs to the SES evaluation, SES values and numbers of expected SM signal events in the S1 and S2 subsets.

1.6 Single event sensitivity

The following expression is used to compute the SES value:

$$SES = \frac{\text{BR}(K^+ \rightarrow \pi^+\pi^0) \cdot A_{\pi\pi}}{D \cdot N_{\pi\pi} \cdot A_{\pi\nu\bar{\nu}} \cdot \epsilon_{\text{RV}} \cdot \epsilon_{\text{trig}}^{\text{PNN}}}. \quad (1.21)$$

Here $N_{\pi\pi}$ is the number of selected $K^+ \rightarrow \pi^+\pi^0$ normalization events. $\text{BR}(K^+ \rightarrow \pi^+\pi^0)$ is the $K^+ \rightarrow \pi^+\pi^0$ branching ratio [62]; $D = 400$ is the down-scaling factor of the minimum-bias trigger: $A_{\pi\nu\bar{\nu}}$ and $A_{\pi\pi}$ are the signal and normalization acceptances, respectively, evaluated with simulations. $1 - \epsilon_{\text{RV}}$ is the inefficiency resulting from the random veto induced by the photon and multi-charged particle rejection due to the presence of accidental activity in the detectors. $\epsilon_{\text{trig}}^{\text{PNN}}$ is the efficiency of the PNN trigger stream. The inputs to the SES computation, the resulting SES values, and the corresponding numbers of expected SM $K^+ \rightarrow \pi^+\nu\bar{\nu}$ events for the S1 and S2 data subsets, integrated over p_{π^+} , are summarized in Table 1.1. The $K^+ \rightarrow \pi^+\nu\bar{\nu}$ decays are simulated using form factors derived from the $K^+ \rightarrow e^+\pi^0\nu$ decay. The accuracy of the description of particles identification and K/π association dominates the uncertainties of $A_{\pi\nu\bar{\nu}}$ and $A_{\pi\pi}$ in Table 1.1, but these effects cancel to first order in the ratio $A_{\pi\pi}/A_{\pi\nu\bar{\nu}}$. The relative contribution of π^0 Dalitz decays, $\pi^0 \rightarrow e^+e^-\gamma$, to the SES result is estimated to be 0.7% and is assigned as a systematic uncertainty to $A_{\pi\pi}$. A systematic uncertainty of 3.5% is propagated to the SES value to consider the quality of the description of π^+ interactions with the material upstream of the LKr, as well as of the m_{miss}^2 distribution. The 2% uncertainty of PNN trigger efficiency is propagated to the SES measurement. The observed discrepancy of up to 5%, takes into consideration the minor correlation between L0 and L1 trigger efficiencies, and being stable across the whole 2018 period, it is propagated to the SES measurement.

The random-veto parameter ϵ_{RV} is measured using a sample of $K^+ \rightarrow \mu^+\nu$ decays from minimum-bias data, selected similarly to $K^+ \rightarrow \pi^+\nu\bar{\nu}$ decays but with inverted particle identification criteria (μ^+ instead of π^+) and in the $\mu\nu$ m_{miss}^2 region. The fraction of events left after applying the photon and multi-charged particle rejection is measured to be $\epsilon_{\text{RV}} = 0.66 \pm 0.01$, including a correction of +0.02 to account for activity in LAV and CHOD induced by the δ -rays produced by muons in the RICH mirrors, as calculated from simulation. The value of ϵ_{RV} depends on the instantaneous beam intensity, and its uncertainty is evaluated by extrapolating ϵ_{RV} to zero intensity and comparing with a MC simulation of $K^+ \rightarrow \mu^+\nu$ decays.

1.7 Background evaluation and validation

Background contributions to the $K^+ \rightarrow \pi^+\nu\bar{\nu}$ final state can be identified from two processes: K^+ decays inside the FV to a final state different from the signal; *upstream events* where a π^+ originates either from a K^+ early decay (before FV) or from an interaction between a beam K^+ and the material upstream of the FV.

The four main K^+ decay backgrounds are $K^+ \rightarrow \pi^+\pi^0$, $K^+ \rightarrow \mu^+\nu$, $K^+ \rightarrow \pi^+\pi^+\pi^-$ and $K^+ \rightarrow \pi^+\pi^-e^+\nu$. The first three enter the signal regions if m_{miss}^2 is misreconstructed. The estimation of these backgrounds rely on the assumption that π^0 rejection for $K^+ \rightarrow \pi^+\pi^0$, particle identification for $K^+ \rightarrow \mu^+\nu$, and multi-charged particle rejection for $K^+ \rightarrow \pi^+\pi^+\pi^-$ are independent of the m_{miss}^2 variable defining the signal regions. After the $K^+ \rightarrow \pi^+\nu\bar{\nu}$ selection is applied, the expected number of events in the signal or control regions is computed for each category as:

$$N_{\text{decay}}^{\text{exp}} = N_{\text{bkg}} \cdot f_{\text{kin}}(\text{region}). \quad (1.22)$$

Here N_{bkg} is the number of PNN-triggered events in the $\pi^+\pi^0$, $\mu\nu$ or 3π background region and $f_{\text{kin}}(\text{region})$ is the fraction of events reconstructed in the signal or control region for each decay mode. The values of the kinematic factor $f_{\text{kin}}(\text{region})$ are obtained: for $K^+ \rightarrow \pi^+\pi^0$ and $K^+ \rightarrow \mu^+\nu$ by using minimum-bias data samples with dedicated selections; for $K^+ \rightarrow \pi^+\pi^+\pi^-$ by using simulated events. Backgrounds from $K^+ \rightarrow \pi^+\pi^-e^+\nu$, $K^+ \rightarrow \pi^+\gamma\gamma$ and semileptonic decays $K^+ \rightarrow \pi^0 l^+\nu$ ($l = e, \mu$) are evaluated only with simulations.

$K^+ \rightarrow \pi^+\pi^0$: After the $K^+ \rightarrow \pi^+\nu\bar{\nu}$ selection, 471 events are observed in the $\pi^+\pi^0$ region (Figure 1.8, left). The kinematic factor is measured using a minimum-bias data sample with a PNN-like selection applied. The m_{miss}^2 distribution is shown in Figure 1.8, right. The disagreement with the assumption that the π^0 tagging is independent of the kinematics is of 3% and assigned as a systematic

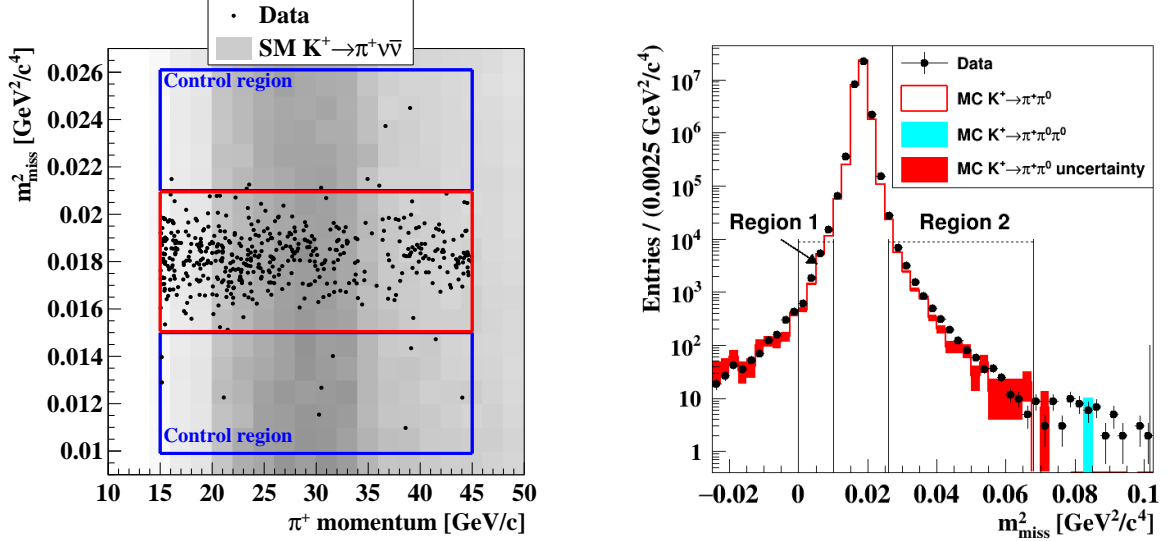


Figure 1.8: Evaluation of the $\pi^+\pi^0$ background. **Left:** Distribution, in the $(p_\pi^+, m_{\text{miss}}^2)$ plane of events in the $\pi^+\pi^0$ region and in the adjacent control regions after the complete signal selection, is applied to the S1 and S2 subsets. The intensity of the grey shaded area reflects the variation of the SM signal acceptance in the plane. **Right:** Data/MC comparison of the m_{miss}^2 distribution of minimum-bias $K^+ \rightarrow \pi^+\pi^0$ events selected by tagging the $\pi^0 \rightarrow \gamma\gamma$ decay. This data sample is used to measure the $K^+ \rightarrow \pi^+\pi^0$ kinematic factor f_{kin} .

uncertainty. The background estimates in the signal and control regions are obtained using equation 1.22 and are validated by comparing expected and observed numbers of events within the control regions. The presence of radiative $K^+ \rightarrow \pi^+\pi^0\gamma$ decays from inner bremsstrahlung increases the fraction of $K^+ \rightarrow \pi^+\pi^0$ background. The value of the correction applied to account for this effect represents 8% of the $K^+ \rightarrow \pi^+\pi^0$ background [6] and a 100% systematic uncertainty is assigned for the precision of the simulated estimation.

$K^+ \rightarrow \mu^+\nu$: After the $K^+ \rightarrow \pi^+\nu\bar{\nu}$ selection, 14112 events are observed in the $\mu\nu$ region. To consider the correlations between the RICH particle identification and the kinematic selection criteria, $f_{\text{kin}}(\text{region})$ is measured using a minimum-bias data sample. The $K^+ \rightarrow \mu^+\nu$ decays are selected applying signal-like conditions, requiring the charged particle to satisfy the μ^+ identification criteria in the calorimeters and the π^+ identification criteria in the RICH. A simulation-driven correction of +3% (relative) is applied to the $K^+ \rightarrow \mu^+\nu$ background estimation in Region 1, to account for muons decaying in flight as $\mu^+ \rightarrow e^+\nu_e\bar{\nu}_\mu$.

$K^+ \rightarrow \pi^+\pi^+\pi^-$: The m_{miss}^2 distribution of the three-body decay spans over a wide kinematic region. A selection requiring only a match between a π^+ in the final state and the parent K^+ is applied to a sample of simulated $K^+ \rightarrow \pi^+\pi^+\pi^-$ decays. To account for the resolution of the m_{miss}^2 variable, the factor $f_{\text{kin}}(\text{region})$ is computed in bins of m_{miss}^2 . The $K^+ \rightarrow \pi^+\pi^+\pi^-$ background is obtained after integrating the background estimates in each m_{miss}^2 bin.

$K^+ \rightarrow \pi^+\pi^-e^+\nu$: This decay is characterized by large values of m_{miss}^2 and contributes only to Region 2. The contribution is suppressed by the $\mathcal{O}(10^{-5})$ branching ratio, multi-charged particle rejection, particle identification and kinematics. A sample of 2×10^9 MC simulated $K^+ \rightarrow \pi^+\pi^-e^+\nu$ decays is used to estimate the background.

Other backgrounds from K^+ decays: The contributions from $K^+ \rightarrow \pi^0 l^+ \nu$ ($l = \mu, e$) and $K^+ \rightarrow \pi^+ \gamma \gamma$ decays are found to be negligible given the particle identification and photon rejection criteria applied to the simulated samples. Upper limits of $\mathcal{O}(10^{-3})$ and $\mathcal{O}(10^{-2})$ events are obtained for the $K^+ \rightarrow \pi^0 l^+ \nu$ and $K^+ \rightarrow \pi^+ \gamma \gamma$ contributions, respectively.

Upstream background: The background from upstream events receives contributions from two types of processes: a π^+ from K^+ decays occurring between GTK stations 2 and 3, matched to an accidental beam particle; a π^+ from interactions of a K^+ with the material in the beam line, produced either promptly or as a decay product of a neutral kaon and matched to the in-time K^+ . Studies on data and MC simulations validate the above classification of upstream events. The evaluation of the background from upstream events follows a data-driven approach. A sample of PNN data is selected with all $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ criteria applied, but requiring: $\text{CDA} > 4$ mm; no K/π association: m_{miss}^2 value inside Regions 1 or 2. The events selected define the *upstream sample*: the distribution of the π^+ tracks at the $(X_{\text{COL}}, Y_{\text{COL}})$ plane (Section 1.5) is shown in Figure 1.9, left. Contamination from K^+ decays in the FV is at the per cent level and therefore negligible. The upstream background is computed as the product of the number of events in the upstream sample, N_{ups} , and the probability, P_{mistag} , that an upstream event has $\text{CDA} < 4$ mm and satisfies the K/π association criteria. The probability P_{mistag} depends on the shapes of the distributions of CDA (Figure 1.9, right) and of ΔT . The probability P_{mistag} is evaluated as a function of ΔT by generating upstream-like events in the $(\text{CDA}, \Delta T)$ plane and applying the K/π association with the $\text{CDA} < 4$ mm condition. The expected

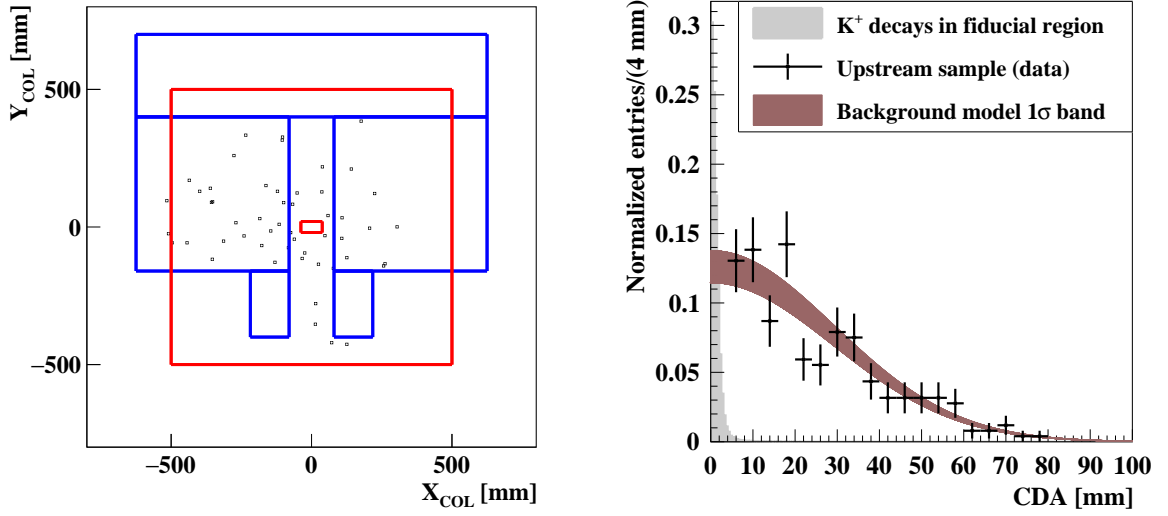


Figure 1.9: Properties of upstream background events. **Left:** Extrapolation of π^+ tracks of the upstream sample described in the text to the (X_{COL}, Y_{COL}) plane in the S2 subset. The small (large) red rectangles correspond to the inner (outer) borders of the new collimator. The outline of the last dipole of the beam achromat is shown with blue solid lines. **Right:** CDA distribution of the events in the upstream sample shown on the left plot (black markers with error bars), compared to the CDA distribution extracted from data and its uncertainty (brown shaded area) and to the same distribution of K^+ decaying in the FV (grey shaded area).

background is computed as

$$N_{\text{upstream}}^{\text{exp}} = \sum_{i=1}^{12} N_{\text{ups}}(|\Delta T_i|) \cdot P_{\text{mistag}}(|\Delta T_i|) \cdot f_{\text{scale}}. \quad (1.23)$$

The sum runs over the twelve 100 ps wide bins covering the $(-600, +600)$ ps region used to reconstruct the tracks in the GTK; $N_{\text{ups}}(|\Delta T_i|)$ is the number of events in the upstream sample in the ΔT bin i ; $P_{\text{mistag}}(|\Delta T_i|)$ is the mis tagging probability. $f_{\text{scale}} = 1.15$ is a scaling factor that accounts for upstream events with $\text{CDA} < 4$ mm not included in N_{ups} . In total, $N_{\text{ups}} = 9$ events are selected in S1 and $N_{\text{ups}} = 38$ in S2, leading to an upstream background of $N_{\text{upstream}}^{\text{exp}} = 3.3_{-0.73}^{+0.98}$ combining the S1 and S2 subsets. The uncertainty is dominated by statistical uncertainty of N_{ups} . A systematic uncertainty of 20% is added, related to the modelling of the CDA shape below 4 mm. A 15% systematic uncertainty is assigned to the value of f_{scale} .

Background summary: The background prediction for the sum of all contributions described above is validated in the six control regions located between the signal

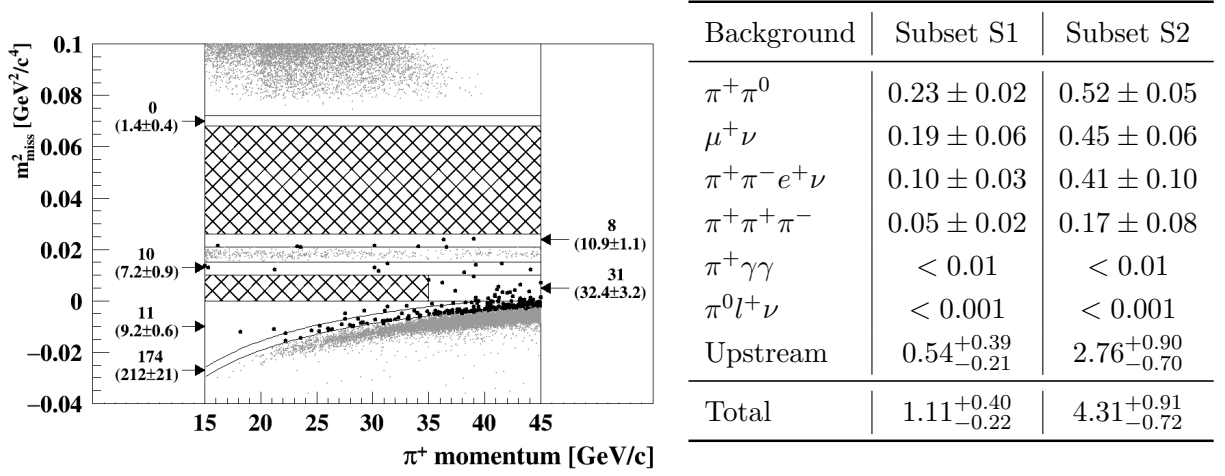


Figure 1.10: Background predictions. **Left:** Reconstructed m_{miss}^2 as a function of π^+ momentum after applying the signal selection to the S1 and S2 subsets. Events in the background regions are displayed as light grey dots. The control regions, populated by the solid black markers, are adjacent to the background regions. The numbers next to these regions are the expected numbers of background events (in brackets) and the observed numbers (without brackets). **Right:** Expected numbers of background events summed over Regions 1 and 2 in the 2018 subsets.

and the $\pi^+\pi^0$, $\mu\nu$ and 3π regions. After unmasking the control regions, the observed and expected numbers of events are found to be statistically compatible across all control regions (Figure 1.10, left). A summary of the background estimates summed over Region 1 and Region 2 is presented in Figure 1.10, right for the two subsets S1 and S2 of the 2018 data.

Finally, despite the highest precision performed for $K^+ \rightarrow \pi^+\nu\bar{\nu}$ measurement, limitations were present due to kaon-pion association and Photon Veto contributions to lower acceptance. In addition, more drastic signal losses are expected in the future with High Intensity Runs. Especially that inefficiency, due to those requirements, is intensity dependent. In this thesis, an attempt is presented, while exploiting innovative Artificial Intelligence technology, to explore ways for overcoming these issues.

Chapter 2

Introduction to AI

2.1 Artificial Intelligence: Aristotle to COVID-19

In 1950, Alan Turing saw the need to define the intelligence level of any artificial agent based on human mental performance. He proposed a test (Turing Test), to provide a satisfactory operational definition. The agent (e.g., a computer) passes the test if a human investigator interacting remotely with it, through written messages, cannot tell the difference whether the responses come from a person or not. This approach provoked and still inciting scientific and philosophical debates regarding the possibility of achieving and realizing such a comparison in an objective way. One main reason for such skepticism is our large gap in understanding of human rationality itself, let aside comparing all its functionalities with artificially intelligent agents. Since it is outside the scope of my thesis to discuss the main dogmatic definitions of Artificial Intelligence (AI), I will stick with S. Russell's [105] pragmatic approach to this topic that, in my opinion, led to the blooming of learning machines and later ushered a new era of technological advancement through one of toughest pages in human history, during COVID-19 world pandemic.

S. Russell's alternative practical approach to AI presented in his book, can be regarded as similar to Richard Feynman's to Quantum Mechanics. In other words, instead of losing focus while delving into ontological controversies, he is satisfied with a productive frame of first principles that works and clearly gives results. N. David Mermin, a semi-retired professor of physics at Cornell University, was behind the famous motto to this approach (i.e., "Shut up and calculate!"). So then, when judging intelligence in the processes of thinking and acting, it would be because of rationality and logic.

The first to attempt to demystify rationality and undeniable reason is the Greek

philosopher Aristotle. From his first principles of logic, any rational agent can be considered “thinking” if it can follow logical laws in structures and patterns to present correct conclusions when given correct information. The other attributes that the rational agent should possess are ones that allow it to “act” accordingly. The main engine that would guide these attributes, among other secondary ones, is the principle of inference. The correct inferences would lead the rational agent to reason logically to (e.g., operate computations based on) the conclusion that a given action will achieve one’s goals and then to act on that conclusion. In addition to logic and computing inference, the great contribution of mathematics to AI came with the theory of probability and Bayesian analysis. When classical inference fails, computed conclusions become ambiguous, hence the need of probability. Thomas Bayes (1702-1761) introduced a statistical rule for updating probabilities in the light of new evidence, to become the basis of most modern approaches to uncertain reasoning in AI systems.

From the two basic philosophical pillars founded by the Greek expert in logic and Bayesian analysis in probability theory, most rational agents are empirically constructed and described in concrete instances, from early Neural Nets to state-of-the-art Deep Learning-based intelligent systems. In the mid-19th century, computers blueprints started laying foundations for efficient computing machines. Charles Babbage (1792-1871) designed the “Analytical Engine” which included addressable memory, stored programs, and conditional jumps. It did not materialize until modern computers started emerging in the modern era. AI also owes a debt to the software side of computer science, which has supplied operating systems, high-level languages and platforms, libraries, and tools to make modern programming widely accessible. Finally, Norbert Wiener (1894-1964) laid the foundations to control theory, especially the branch known as stochastic optimal control which has as its goal the design of systems that maximize an **objective function** over time. The tools of control theory (i.e., calculus and matrix algebra) inspired a remarkably similar concept in optimizing intelligent systems in what is called stochastic gradient descent, which has as its goal to minimize a loss function over time. That concept came to its best use when in the mid-1980s, four distinct groups reinvented the back-propagation learning algorithm first found by [47]. That came at the same time when parallel distributed processing [104] started gaining momentum and caused great excitement. Until recently, multi-threading on Graphical Processing Units GPU started making the long-revered Moore’s law¹ itself irrelevant. The reason for that is that computing efficiency is not merely measured by transistors size or number anymore.

The above-mentioned overview is a swift review of the developments of theoretical

¹Moore’s Law says that the number of transistors per square inch doubles every 1 to 1.5 years.

rational agents into actual intelligent systems that paved the way to the optimal blooming of recent AI reemergence in every aspect of sciences and more of every-day's life (e.g., page 27 of [105]). Finally, and most recently, an unusual world pandemic started its deadly exponential spread in 2019. COVID-19 incited governments and private sectors to provide AI with plenty more freedom and resources to evolve even faster, out of the urgency and necessity to save lives. Also, public trust and confidence are regained, due to the results AI presented in the fields of medical diagnosis and epidemiology that helped wage the war against COVID-19. In at least 7 ways [114], AI gave humanity the upper hand over the pandemic:

1. Early detection and diagnosis of the infection: AI can quickly analyze irregular symptom and other 'red flags' and thus alarm the patients and the healthcare authorities.
2. Monitoring the treatment: AI can build an intelligent platform for automatic monitoring and prediction of the spread of this virus.
3. Contact tracing of the individuals: AI can help analyze the level of infection by this virus identifying the clusters and 'hot spots' and can successfully do the contact tracing of the individuals and to monitor them.
4. Projection of cases and mortality: This technology can track and forecast the nature of the virus from the available data, social media, and media platforms, about the risks of the infection and its spread.
5. Development of drugs and vaccines: AI is used for drug research by analyzing the available data on COVID-19. This technology is used in speeding up drug testing in real-time.
6. Reducing the workload of healthcare workers: It helps in early diagnosis and providing treatment at an early stage using digital approaches and decision science, offers the best training to students and doctors regarding this new disease.
7. Prevention of the disease: With the help of real-time data analysis, AI can provide updated information which is helpful in the prevention of this disease.

Next the detailed functionalities of Neural Nets in what is known as Supervised learning will be presented, then Deep Learning DL in Computer Vision, to end up this introductory chapter with basic concepts of AI-related applications in image recognition and more-so in object detection.

2.2 Supervised Neural Networks

An algorithm is *learning* if it improves its performance on future tasks after making interactions with data. Learning is a broad philosophical and scientific topic with many categories and levels. Hence, we will concentrate on one class of learning problem: from a collection of input-output pairs, learn a function that predicts the output for new inputs.

Any component of an agent can be improved by learning from data. The improvements, and the techniques used to make them, depend on three major factors:

1. What *prior knowledge* the agent already has.
2. What *representation* is used for the data and the component.
3. What *feedback* is available to learn from.

The *type of feedback* available for learning is usually the most principal factor in determining the nature of the learning problem that the algorithm faces [105]. There are three types of feedback that the field of machine learning usually distinguishes as three cases of learning:

In *unsupervised learning* the algorithm learns patterns in the input even though no explicit feedback is supplied. The most common unsupervised learning task is *clustering*: detecting potentially useful clusters of input examples.

In *reinforcement learning* the algorithm learns from a series of reinforcements-rewards or punishments. It is up to the algorithm to decide which of the actions prior to the reinforcement was most responsible for it.

In *supervised learning* the algorithm observes some example input-output pairs and learns a function that maps from input to output. In this case, the output value is available directly from the algorithm's perceptions (after the fact); the environment (example and function) is a *teacher*.

We will focus on the last case and try to develop a supervised learning algorithm. The vertebra of such an algorithm is the learning or mapping objective function. Let us for now call this function h for *hypothesis* in *hypothesis space* H and h will be called a *consistent* hypothesis if it agrees with the input-output data. The problem that appears here is: *how do we choose from among multiple consistent hypotheses?* One answer is *Ockham's*² *razor*: prefer the *simplest* hypothesis consistent with the

²Named after the 14th century English philosopher, William of Ockham.

data [105]. Intuitively, this makes sense, because hypotheses that are not simpler than the data themselves are failing to extract any *pattern* from the data. However, we should always keep in mind that *there is a trade-off between the expressiveness of a hypothesis and the complexity of finding a simple, consistent one*. Therefore, the general successful strategy of a machine learning project is to start with the simplest h . A learning algorithm takes as input an object or situation described by a set of *attributes* and returns a “decision”—the predicted output value for the input. The input values for the attributes can be continuous or discrete. The output value can also be continuous or discrete; learning a discrete-valued function is called *classification* learning; learning a continuous function is called *regression*. We will concentrate on *Boolean* classification currently, wherein each example is classified as true(*positive*) or false(*negative*).

Finally, we define a theoretical frame for *performance* and *training*. A learning algorithm is good if it produces hypotheses that do a decent job of predicting the classifications of unseen examples. We will look at a methodology for assessing prediction quality after the fact.

Obviously, a prediction is good if it turns out to be true, so we can assess the quality of a hypothesis by checking its predictions against the correct classification once we know it. We do this on a set of unseen examples known as the *test set*. If we train on all our available examples, then we will have to go out and get some more to test on, so often it is more convenient to adopt the following methodology:

1. Collect a large Control Sample of data.
2. Divide it into two disjoint sets: the *training set* (90%) and the *test set* (10%); a strategy known as *cross-validation*.
3. Apply the learning algorithm to the training set, generating a hypothesis h .
4. Measure the *accuracy* or percentage of examples in the test set that are correctly classified by h , as a *single-number evaluation metric* [96].
5. Repeat step 2 to 4 for varied sizes of training sets and different randomly selected training sets of each size³.

Unfortunately, this is far from the whole story. It is quite possible, and in fact even when vital information is missing, the learning algorithm will find a hypothesis that is consistent with all the examples. This is because the algorithm can use *irrelevant* attributes to make insignificant distinctions among examples. This problem is called

³An easier approach to this step, assuming we have a large enough control sample of data, is to use mini-batches and random shuffling, in addition to hyper-parameters tuning that will be explained in the “Optimization” section

over-fitting. It is out of the scope for now to delve into probability theory and explain the mathematical tools used to deal with such a problem. We will be satisfied by the practical strategy mentioned in [96] which is:

1. Make sure that our data control sample is as general and random as it can be (the larger the better).
2. We can measure the deviation by comparing the actual number miss-classified positive and negative examples.
3. Notice any illogical increase in performance in the test set over the training set predictions.

In the case of *supervised learning*, the quality of a binary classifier is typically described by a measure that quantifies how well the machine learning algorithm separates signal from background⁴. On a sample of data, where we know the true class labels, there are 2×2 categories formed by the true and the estimated labels both for signal and background classes. The matrix of entries is known as the input matrix and a sample of data in these categories can be used to quantify the performance of the machine learning algorithm. Typical performance measures are the *accuracy* of the mean as the % of true predictions [86], or more specifically the percentage of true positive and true negative labels [40]. These numbers can be used as evaluation metrics either during or *after* the training of a neural network, while the training itself uses differentiate function to allow for updating through cycles of iterations. The function is called “loss function” because it keeps a record of the errors from one cycle to another. The cross-entropy is the mathematical function chosen for classification algorithms in general (see for example [93]). The most common function used in general for binary classifications is the vanilla logistic loss function or cross entropy that will also be described next. Obviously, a prediction is good if it turns out to be true, so we can assess the quality of a hypothesis by checking its predictions against the correct classification once we know it. We do this on a set of unseen examples known as the *test set*. It is important to note that a dimensional consistency check must be carried out throughout the construction process. Variables with consistent dimensions should be stacked together to form the main input matrices.

All preparation steps described so far applies to all supervised learning methods not exclusively including: Logistic Regression (LR)[75], K-Nearest Neighbours (KNN)[60], Decision and Boosted Decision Trees (BDT)[32][101], Random Forest (RF)[41], Naive Bayes (NB)[99], Support Vector Machine (SVM) with or without Kernel functions modification (e.g., Radial Basis Function (RBF))[113][30], Boosting

⁴Signal and background here are mentioned in a simplistic binary meaning, merely to differentiate two labels nothing more.

Classifiers (i.e., Gradient Boosting machines, Ada Boost, etc...)[94] and Neural Networks (NN)[107] which will be the main method adopted in this thesis for reasons will be stated in Chapter 3.

For now, the next step is to describe the development of a *Supervised* model using NN (Neural Network) *Algorithm* with the following ingredients:

- The mathematical definition of the Sigmoid and ReLU *activation* functions. (Check the Appendix for more details)
- The *initialisation* of the parameters with “He” method⁵. The initialisation consists of zero initial values for the biases $b^{[l]}$ and random values for the weights $w^{[l]}$. Random weights initialisation is used to break the symmetry and make sure different hidden units can learn different things. The initial values of the weights should be multiplied by a scaling factor of $\sqrt{\frac{2}{\text{dimension-of-the-previous-layer}}}$ to keep them reduced.
- A function that selects the mini batches randomly according to the size selected, then does the *shuffling* and the associated dimensional *partition*. In Stochastic Gradient Descent, we use only 1 training example before updating the gradients. When the training set is large, SGD can be faster, but the parameters will “oscillate” toward the minimum rather than converge smoothly. Note also that implementing SGD requires 3 for-loops in total:
 1. Over the number of iterations
 2. Over the m training examples
 3. Over the layers (to update all parameters, from $(W^{[1]}, b^{[1]})$ to $(W^{[L]}, b^{[L]})$)

In practice, we’ll often get faster results if we do not use either the whole training set, or only one example, to perform each update. Mini-batch gradient descent uses an intermediate number of examples for each step. With mini-batch gradient descent, we loop over the mini-batches instead of looping over individual training examples. The difference between gradient descent, mini-batch gradient descent and stochastic gradient descent is the number of examples we use to perform one update step. A crucial point though is that we always must tune a learning rate (i.e., as a hyper-parameter) α . To build mini batches from the training set (X, Y) , there are two steps:

1. Create a shuffled version of the training set (X, Y) as shown in Figure 2.1. Each column of X and Y represents a training example. Note that the

⁵Named after the first author of (He et al., 2015).

random shuffling is done synchronously between X and Y. Such that after the shuffling the i^{th} column of X is the example corresponding to the i^{th} label in Y. The shuffling step ensures that examples will be split randomly into different mini batches.

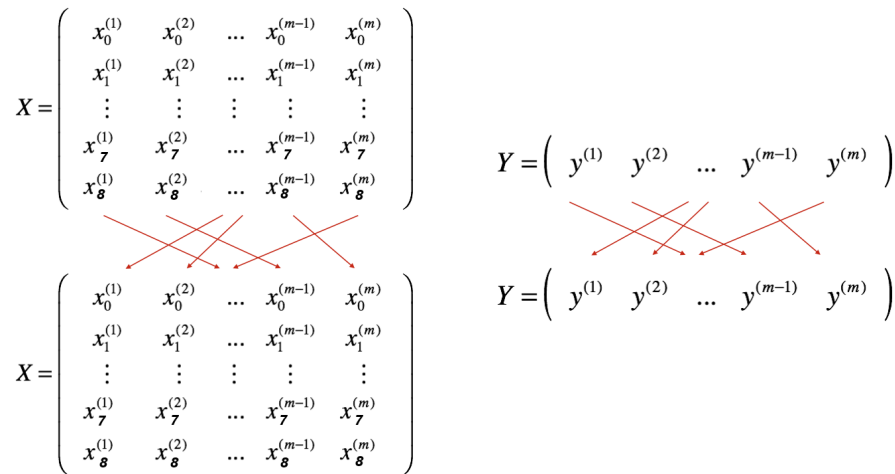


Figure 2.1: The shuffling must be synchronised between the X and the Y matrices.

2. Partition the shuffled (X, Y) into mini batches of size `mini_batch_size` (here 256 in Figure 2.2). Note that the number of training examples is not always divisible by `mini_batch_size` so the last mini batch might be smaller. Let $\lfloor s \rfloor$ represents s rounded down to the nearest integer (this is `math.floor(s)` in Python). If the total number of examples is not a multiple of `mini_batch_size=256` then there will be $\lfloor \frac{m}{\text{mini_batch_size}} \rfloor$ mini-batches with a full 256 examples, and the number of examples in the final mini-batch will be $(m - \text{mini_batch_size} \times \lfloor \frac{m}{\text{mini_batch_size}} \rfloor)$.

- The *Forward Propagation* computes the outcome in the forward direction (shown in purple in Figure 2.3). It completes the LINEAR part of a layer's forward propagation step (it calculates the output of the following function: $Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$ where $A^0 = X$)⁶, combines the previous two steps into a new [LINEAR→ACTIVATION] forward function (as in: $A^{[l]} = g(Z^{[l]})$ where the activation “g” can be sigmoid or ReLU), stacks the [LINEAR→RELU] forward function ($L - 2$) time (through layers 1 to $L - 2$) and finally adds a [LINEAR→SIGMOID] activation at (the $L - 1$ th layer). This gives a new

⁶We will use capital letters now to give indications that we are dealing with matrices and matrix operations.

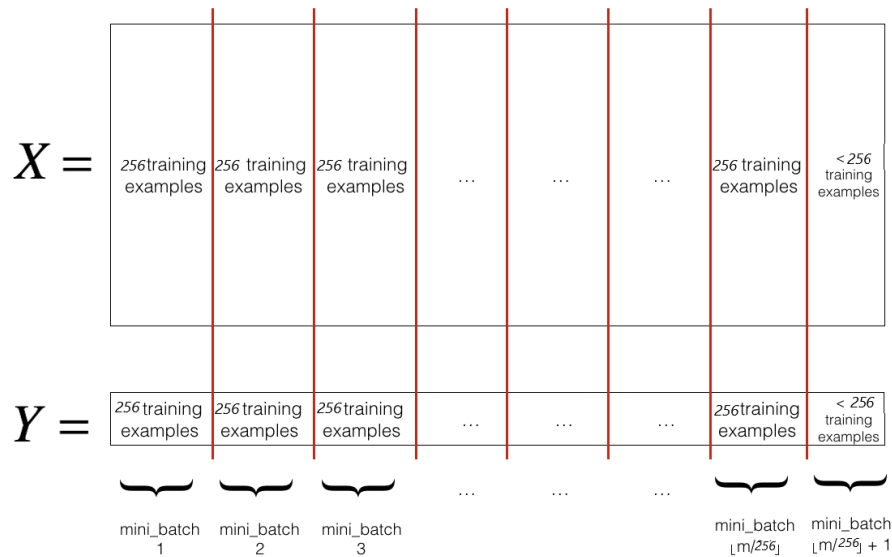


Figure 2.2: The last mini batch might be smaller than the mini-batch-size.

L -model-forward function with a single node final output layer for binary classifications (where L is the total number of layers and l is the current layer index that runs from 0 to $L - 1$).

- The definition and computation of the *cross-entropy* cost of the vanilla logistic loss. The goal of logistic regression is to minimize the error between its predictions and training data. Given an input represented by a feature vector X , the algorithm will evaluate the probability of a class. Z is a linear function “Matrix”, but since we are looking for a probability constraint between $[0,1]$, the sigmoid function is used at the final hidden layer. We can think of the NN as a 2-dimensional graph formed of many layers and every layer has many nodes. Each node is learning the weights of this linear function and mapping it to all the other nodes in the next layer. When the training is done the functions on the $L - 1$ layer should converge to give an output probability-classification⁷. However, to train the parameters “Matrices” W and b , we need to define a *Cost Function*. The loss function first, measures the discrepancy between the prediction ($\hat{y}^{(i)}$) and the desired output ($y^{(i)}$). One most used is **cross entropy** loss function which computes the error for a single training example in the following manner:

$$j(\hat{y}^{(i)}, y^{(i)}) = -[(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})] \quad (2.1)$$

⁷Check the DL mathematical notations in the index at the end-page.

where,

- If $y^{(i)} = 1$: $j(\hat{y}^{(i)}, y^{(i)}) = -\log(\hat{y}^{(i)})$ where $\log(\hat{y}^{(i)})$ and $\hat{y}^{(i)}$ should be close to one
- If $y^{(i)} = 0$: $j(\hat{y}^{(i)}, y^{(i)}) = -\log(1 - \hat{y}^{(i)})$ where $\log(1 - \hat{y}^{(i)})$ and $\hat{y}^{(i)}$ should be close to zero

The cost function is the average loss function of the entire training set. The goal is to find the parameters w and b that minimize the overall cross entropy cost function, defined in the following:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m j(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m [(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})) \quad (2.2)$$

- The *Backward Propagation* that computes the gradients in the backward direction (denoted in red in Figure 2.3). Complete the LINEAR part of a layer's backward propagation step (resulting in

$$dW^{[l]} = \frac{\partial J}{\partial W^{[l]}} = \frac{1}{m} dZ^{[l]} A^{[l-1]T},$$

$$db^{[l]} = \frac{\partial J}{\partial b^{[l]}} = \frac{1}{m} \sum_{i=1}^m dZ^{[l](i)}$$

$$\text{and } dA^{[l-1]} = \frac{\partial J}{\partial A^{[l-1]}} = W^{[l]T} dZ^{[l]}.$$

Define the gradient of the ACTIVATION function (relu-backward/sigmoid-backward). Combine the previous two steps into a new [LINEAR→ACTIVATION] backward function (resulting in

$$dZ^{[l]} = dA^{[l]} * g'(Z^{[l]})$$

where $g'=0$ (for $Z < 0$) or $dA^{[l]}$ (for $Z > 0$) if g is relu()

and $g'=\sigma(1 - \sigma)$ if g is sigmoid()).

Stack [LINEAR→RELU] backward L-1 times and add [LINEAR→SIGMOID] backward in a new L-model-backward function (resulting in

$$dA^L = \frac{\partial J}{\partial A^L} = -\left(\frac{Y}{A^L} - \frac{1-Y}{1-A^L}\right) \text{ where } A^L = \sigma(Z^L).$$

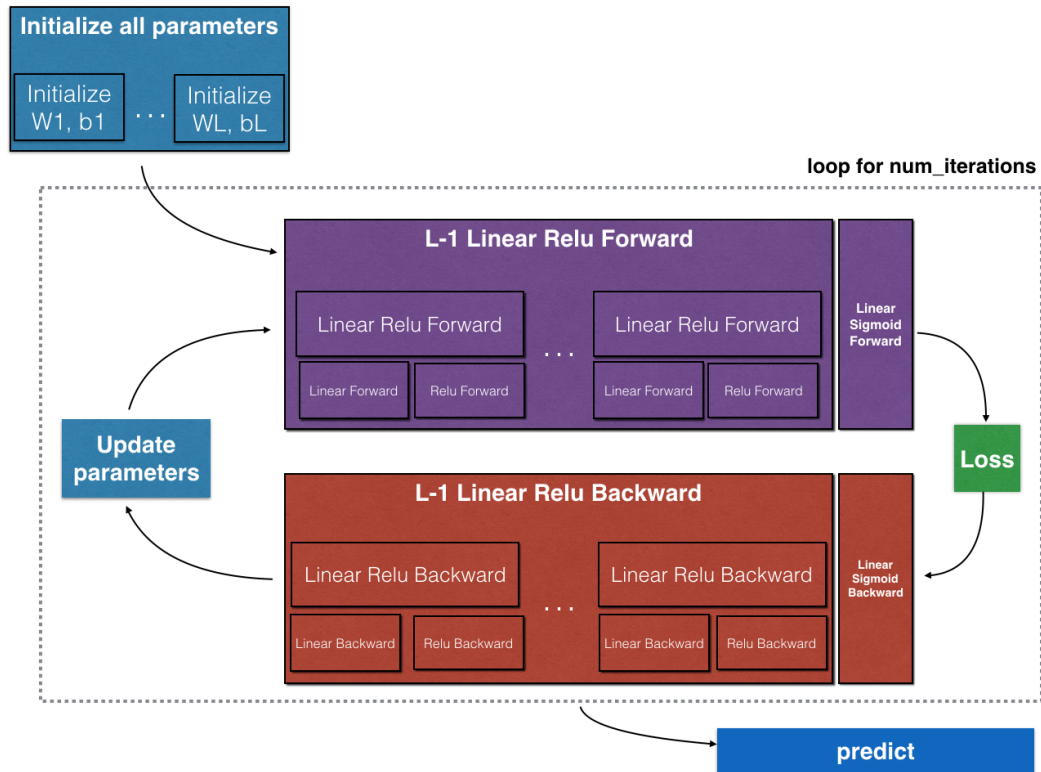


Figure 2.3: A mind-map of the general L-layered NN Dynamics [95]. As we notice here for every forward function, there is a corresponding backward function. That is why at every step of our forward module we must store some values in a cache. Cached values are useful for computing gradients. In the back-propagation module we can then use the cache to calculate the gradients.

- A function that *updates* the main parameters following the Adam optimization main formulation. Adam is one of the most effective optimization algorithms for training neural networks. Adam works in the following way:
 1. It calculates an exponentially weighted average of past gradients and stores it in variables v (before bias correction) and $v^{corrected}$ (with bias correction).
 2. It calculates an exponentially weighted average of the squares of the past gradients and stores it in variables s (before bias correction) and $s^{corrected}$ (with bias correction).
 3. It updates parameters in a direction based on combining information from “1” and “2”.

The update rule is, ($l = 1, \dots, L$)

For the weights:

$$v_{W^{[l]}} = \beta_1 v_{W^{[l]}} + (1 - \beta_1) \frac{\partial J}{\partial W^{[l]}}$$

$$v_{W^{[l]}}^{corrected} = \frac{v_{W^{[l]}}}{1 - (\beta_1)^t}$$

$$s_{W^{[l]}} = \beta_2 s_{W^{[l]}} + (1 - \beta_2) \left(\frac{\partial J}{\partial W^{[l]}} \right)^2$$

$$s_{W^{[l]}}^{corrected} = \frac{s_{W^{[l]}}}{1 - (\beta_2)^t}$$

$$W^{[l]} = W^{[l]} - \alpha \frac{v_{W^{[l]}}^{corrected}}{\sqrt{s_{W^{[l]}}^{corrected} + \varepsilon}}$$

and for the biases:

$$v_{b^{[l]}} = \beta_1 v_{b^{[l]}} + (1 - \beta_1) \frac{\partial J}{\partial b^{[l]}}$$

$$v_{b^{[l]}}^{corrected} = \frac{v_{b^{[l]}}}{1 - (\beta_1)^t}$$

$$s_{b^{[l]}} = \beta_2 s_{b^{[l]}} + (1 - \beta_2) \left(\frac{\partial J}{\partial b^{[l]}} \right)^2$$

$$s_{b^{[l]}}^{corrected} = \frac{s_{b^{[l]}}}{1 - (\beta_2)^t}$$

$$b^{[l]} = b^{[l]} - \alpha \frac{v_{b^{[l]}}^{corrected}}{\sqrt{s_{b^{[l]}}^{corrected} + \varepsilon}}$$

where:

- t counts the number of steps taken of Adam
- L is the number of layers
- β_1 and β_2 are hyper-parameters that control the two exponentially weighted averages.
- α is the learning rate
- ε is a ridiculously small number to avoid dividing by zero

We store all the parameters in the parameters dictionary. The variables v, s are python dictionaries that need to be initialized with arrays of zeros [13].

- The *prediction* function that takes the output of the soft-max and transforms it into a probability of (0 if ≤ 0.5 and 1 otherwise). By counting the mean of the true predictions, we determine the *accuracy* that evaluates our model.

The decisive step is the *model* function that compiles and calls all the “helper functions”. In summary, the model:

- Takes as input the data and a handful of manually selected hyper-parameters (The X(data) and Y(labels) matrices, the layers dimensions array, the learning rate α , the mini batch size, the momenta of Adam optimization β_1 and β_2 , a small number ϵ to avoid division by zero and the number of epochs which are the iterations over the mini batches).
- Initializes the main and Adam parameters.
- Starts iterating over the epochs, for every mini-batch and over all the layers it:
 - Calls the Forward Propagation.
 - Computes the cost.
 - Calls the Backward Propagation.
 - Update the main parameters with Adam optimization.
 - Returns a dictionary with the updated main parameters.

The same mechanism of forward and back-propagation for updating the weights also applies to a more general and wider range of NNs called Convolutional Neural Nets (CNN). Next, CNNs background and functionalities will be introduced.

2.2.1 Convolutional Neural Nets CNN

CNNs are accurate and efficient and had led to a revolutionary breakthrough in the field of machine learning. They are formed of deep layers and numerous connections between them, that made them computationally expensive. Although CNNs have been around since the 1980s, only recently computers have become powerful enough to handle them. A greater advantage of CNNs was due to the development of GPUs and multi-threading with GPUs and CPU clusters [66].

All NNs have a subtle nature of nodal connectivity that is represented by a mapping of the weights. However, CNNs count on multi-layered architectures and these so-called hidden layers are of various kinds. Each kind serves a specific purpose, also they can be repetitive and overall cyclic. For instance, a convolutional layer or CONV is a peculiar type of hidden layer with the purpose of exploiting image features. The general pipeline would include the input layer, the hidden layers in between and the output layer. The hidden layers are made up of:

- Convolutional layers (CONV)
- Activation layers, and these always come right after CONV in the order mentioned. The most common is the Rectified Linear Unit (ReLU) layers which will be focused on later. Other activation layers that exist are tanh, Scaled Exponential Linear Unit (SeLU) and many more.
- Pooling layers (POOL)
- A fully connected layer (FC)
- An output layer or prediction layer for the final activation's output which would in that case be either sigmoid or Soft-max

A CNN's architecture as shown in Figure 2.4 would include a combination of the above-mentioned layers.

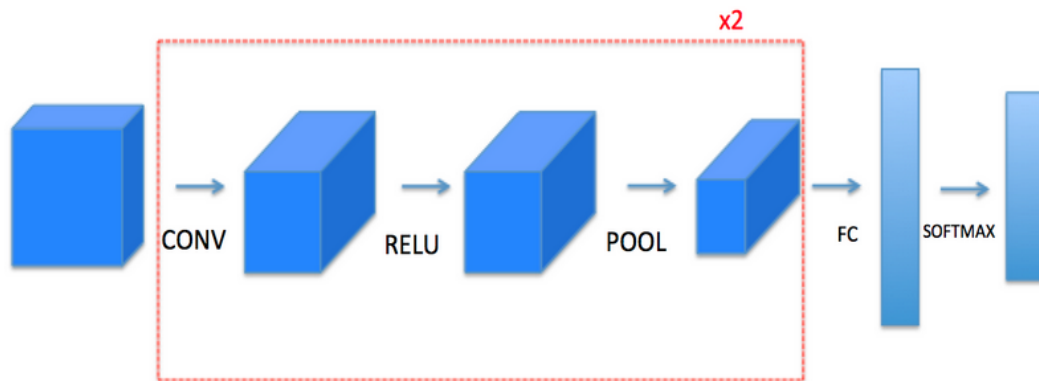


Figure 2.4: An instance of a classic architecture of CNN [95]. Here the hidden layers of the convolution block called “filter” (*within the red square dotted line*) are repeated two times ($\times 2$).

2.2.1.1 Convolutional layers

A convolution layer transforms a certain input into an output volume of different size.

A convolutional layer CONV, as mentioned before, is one of many hidden layers and is the basis of CNN. Many types of CONV exist and at the start of pipeline they can be quite basic, but they increase in complexity or degree of non-linearity throughout the network. The first convolutional layer works to identify simple features within the image. This can include edge detection. Depending on the filter applied, certain edges will be detected such as vertical lines or horizontal lines.

These convolutional layers consist of two stages:

- Zero Padding
- Convolve window

Figure 2.5 shows the application of a vertical edge CONV. On the left, an input image is represented by several pixels. For lighter regions, the pixel number is higher and the opposite for darker regions. Input images, CONV layers and output images can be thought of as matrices containing pixel values. The input shows the pixel composition of an image with a light area in the left half and a dark area on the right. This is split by a line in the middle which can be noted by the change in pixel values.

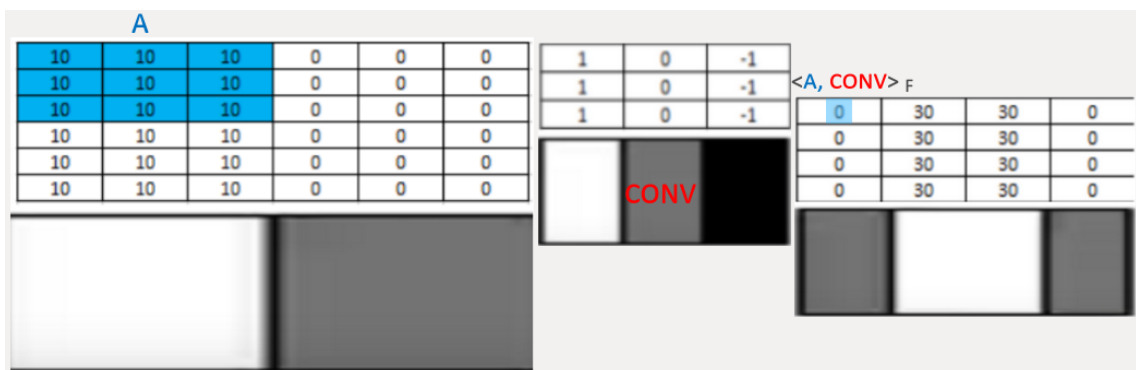


Figure 2.5: CONV layer Operation: (Left) input image in visual (bottom) and pixel (top) forms, (Middle) CONV layer in visual (bottom) and pixel (top) forms, (Right) output image image in visual (bottom) and pixel (top) forms.

Figure 2.5 (middle) is a vertical edge CONV in pixel form. To apply this filter to the image and calculate the first element of the resulting image, the 3 by 3 section

highlighted in blue as a matrix A in Figure 2.5 (right) is elementwise (“Frobenius” inner product) multiplied by the convolution matrix and the sum of elements taken.

Computation of this operation yields a value of 0 as the first element in the output image (Highlighted in blue in figure 2.5 (right)). To calculate the value of the second element, the vertical edge filter is shifted once to the right on the pixels of the input image and the same matrix multiplication occurs. This is known as a stride of 1⁸ and we will consider this case in the current example discussed. However, in some cases, there can be a greater stride which results in a greater shift in pixels since it is a free parameter of the architecture’s layer. It is a choice of tuning for computation/accuracy expense. Larger strides are less computationally expensive, for instance, Stride 2 will reduce the output resolution by half. This vertical edge filter is applied now to the rest of the input image and the process of the filter sliding across the image is known as convolving. Once complete, the output image is created which is shown in figure 2.5 (bottom right). As previously mentioned, the larger numbers indicate lighter regions. There appears a bright line passing down the middle and has therefore, detected the vertical edge from the input image.

This shows that a 6×6 matrix convolved by a 3×3 matrix yields a 4×4 matrix. The input image size can be denoted with n and the filter size an f . Equation 2.3 shows the formula used to calculate the size of the output image. The input size now has been reduced and the edge feature has been emphasized. This is the advantage of applying these CONV layers. As the image goes through the network, more complicated filters can be applied to extract features more advanced than edges. The output image of a CONV becomes feature map. Now it has a reduced size compared to the original image and the computational expense is decreased. The convolution does result in some loss of detail whilst extracting the key features for the next part of the network.

$$n - f + 1 = \text{output image size} \quad (2.3)$$

This can be an issue as mentioned before, especially with very deep neural networks of 100 or more layers. If the input image has reduced dimensionality with each layer, very soon it will become too small to be significant. With the 6×6 input image shown in Figure 2.5, after it has passed through two layers, the output is significantly smaller therefore causing an issue. Zero padding would help by increasing the initial size of the image. Additionally, when applying a CONV, there is a loss of some information from the contour of input image. This is because side and corner pixels are being cropped out in bits, resulting in them being used to compute the output image multiple times less than central pixels are. Zero padding would help saving

⁸stride 1 means moving in steps of ones in the input images pixel by pixel left to right and top to bottom row by row. It is set according to the type of problem studied.

the features from the relevant pixels of the image by adding zeros around the frame. A result of the convolution layer multiplying zeros instead of the input image edge pixels. Since a CONV window of fixed size is moving in steps from one corner to another, padding with zeros will help avoid turning this reduction into information loss and keeps the dimensions balanced. While the significant pixels are kept away from the frame and towards the center, their dimensions would only get compressed. Figure 2.6 is a continuation of our first example, showing the output of an operation with same 3×3 CONV (figure 2.5) after a $p = 1$ zero padding is applied to the original output.

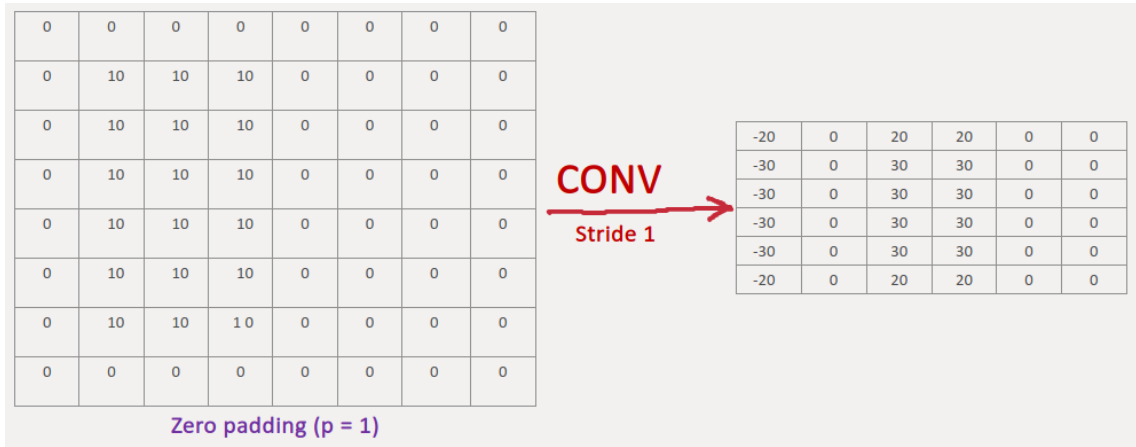


Figure 2.6: CONV operation output after applying Zero-padding

The equation, for the size of output image when padding is applied, becomes equation 2.4 below.

$$n + 2p - f + 1 = \text{output image size} \quad (2.4)$$

The application of padding that results in the output image to be of the same size as the in-padded input image is known as same convolutions. To ensure a same convolution, the formula to determine the size of the padding is shown below in equation 2.5.

$$p = \frac{f - 1}{2} \quad (2.5)$$

f is often chosen of an odd size to keep padding symmetrical.

So, to summarize, the zero padding adds zeroes to the image input⁹ borders and its main two contributions are the following:

- Allowing the use of convolutional layers without necessarily shrinking the height and width of the volumes. This is important for building deeper networks, since otherwise the height/width would shrink as we go to deeper layers. An important special case is the “same” convolution, in which the height/width is exactly preserved after one layer.
- Helping keep more of the information at the border of an image. Without padding, very few values from one layer to another would be affected by pixels at the edges of an image while the convolution window is swiping through.

On the other hand the convolution step, as shown in Figure 2.7, would:

- Take an input volume
- Applies a filter at every position of the input
- Outputs another volume (usually of distinct size)

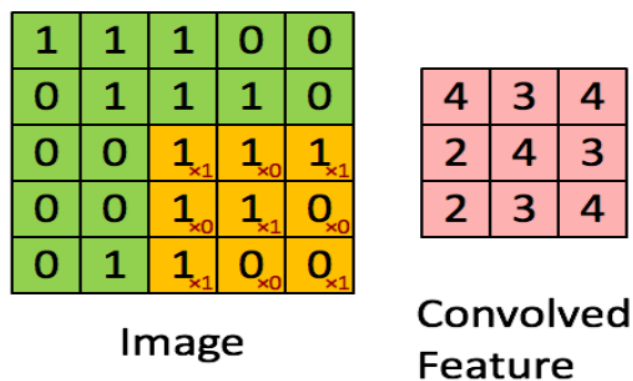


Figure 2.7: Convolution operation with a filter 3x3 and a stride of 1 (stride = amount you move the window each time you slide) [95]

This operation would proceed with a simple “element-wise” matrix multiplication between a filter and slices of pixel values of the input image with the same size as the filter applied. Then summing up all the values multiplied in one slice to form the

⁹represented by a pixelized matrix encoded form

weights at one place in the output matrix. As explained previously, a random bias should be added to the weights as well. The output matrix should have a reduced size in the 2D plane. Then as shown in Figure 2.8 stacking up all the output matrices for the different CONVs used we get the final output volume which is a 3D cuboid shape.

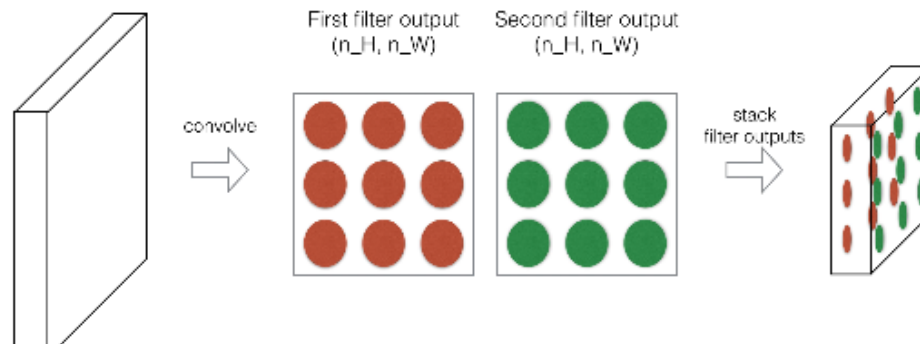


Figure 2.8: The 3D final output shape of the convolution step (please check Appendix for the classic notation used in this image)[95]

2.2.1.2 ReLU Activation Layer

The ReLU layer is a type of activation function within a CNN that is applied to the feature map output of the convolution layers. An activation layer is used to increase the non-linearity of the CONV transformed pixel values. An activation function will take an input and apply a transformation that yields an output that is limited by two fixed values either 0 and 1, or -1 and 1. ReLU however differs from this, being a *max* function between 0 and any real number z , if it receives an input that is negative it will output a 0. However, if the input is positive, the output will be the value of the input itself.

$$\text{ReLU}(z) = \max(0, z) \quad (2.6)$$

ReLU is a popular linear-like activation function as this approach trains the network at a high speed without a major loss of accuracy ([108]).

To use stochastic gradient descent with back-propagation of errors to train deep neural networks, an activation function is needed that looks and acts like a linear one, but is, in fact, a nonlinear function allowing complex relationships in the data to be learned. A practical example showing the power of such functions can be explored in

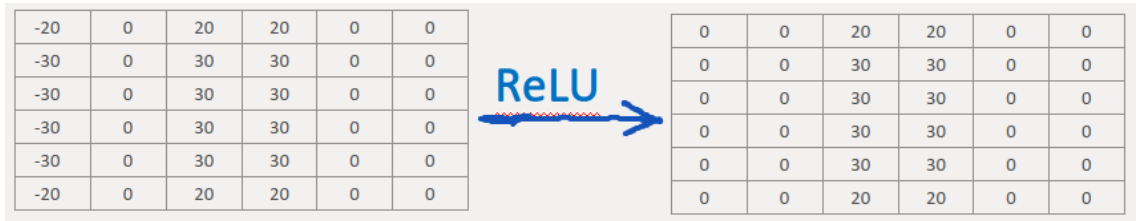


Figure 2.9: After a ReLU activation pass

[92]¹⁰. The function should also provide more sensitivity to the activation sum input and avoid easy saturation¹¹. The only function, that was discovered to do all that, was the ReLU function [71].

Figure 2.9 is a continuation of the first example, showing the output after activating with a ReLU function. There it can be seen clearly how the negative values were simply turned into zeros.

2.2.1.3 Pooling Layers

After the convoluted output gets activated by the rectifier linear unit (figure 2.9), a further reduction of dimensionality happens due to the pooling layers in the height and width of the transformed matrix. This would further increase the speed of the model and improve the robustness of the network. An extra advantage of the pooling stage is that it helps reduce over-fitting which is highly likely to happen in the case of over training the model. It does that by making any feature detector more invariant to the location of the Region of Interest in input images: Pooling works by applying an additional transformation to the inputs in sparse matrix form after being convoluted (CONV) and activated (ReLU), but now the only pixels left in the pooled images are the ones fully saturated with most essential features.

The two types of pooling layers are:

- Max-pooling layer slides an $f \times f$ window over the input and stores the max value of the window in the output.
- Average-pooling layer slides an $f \times f$ window over the input and stores the average value of the window in the output [95].

¹⁰In Chapter 3 p.100, the authors present an instance of nonlinear coordinate transform which can go a long way into making the function act linearly and easily separate features in transformed parameter space.

¹¹Saturation is where the unnecessary values and their operations are occupying memory or computational power fully without real benefit.

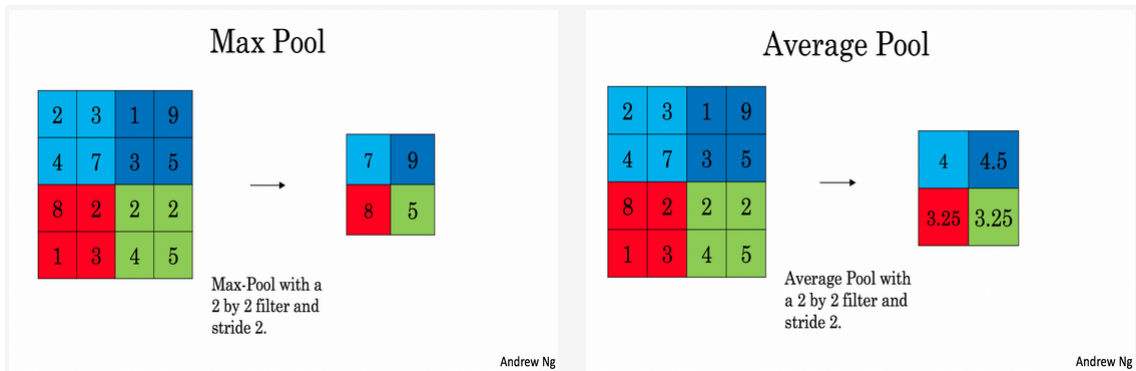


Figure 2.10: Pooling Layers

Figure 2.11: Output after a 3×3 Average-pooling with a stride=1

The formula binding the output size of the pooling to the input size is:

$$\frac{n - f}{stride} + 1 = output\ size \quad (2.7)$$

For example, a maximum pooling layer that is 2 by 2 with a stride of 2 works by the following:

- First, the pooling selection window is applied to the first square and a 2 by 2 section is chosen from the input matrix
- Of this section, the max pixel value is taken and transferred to the output layer
- For a stride of 2, the pooling window is stepped forward by two intervals and the previous step is repeated
- This continues until the whole image has been pooled and a “max pooled” feature map has been generated

The effect of this is that the dimension of the pooled image is reduced by a factor of 2 that has preserved the most “activated” pixels. Figure 2.10 shows this. Average

pooling works similarly however, instead of taking the maximum value, the average value is carried instead. Figure 2.11 is a continuation of the first example showing the output matrix after a 3×3 average pooling is applied with a stride of 1. Both approaches result in a pooled feature map with a reduced number of pixels which decreases computational expense.

2.2.1.4 Final Output

The maximally reduced feature map produced by the pooling layer enters the final stage of the pipeline. Just before prediction, inputs are fed into a Fully Connected Neural Net (FCNN) or what earlier we simply called NN and thoroughly described in Supervised NN section 2.2. Finally, a sigmoid logistic regression is used for binary classifications.

It is the case when the vision system is shown an image and needs to make decision regarding whether an object “x” is within the image or not. The binary outputs are 0 and 1. 0 in the case that object “x” is not within the image and 1 for when it does appear. The sigmoid function, however, is not useful in the case when several objects can be detected within an image and a probability of each object’s certainty is desired. In this case, a Soft-Max layer can be added. A Soft-Max layer can determine multi-class probabilities and is often the final layer in a network. The Soft-Max is a form of logistic regression that normalizes an input value into a vector of values that follows a probability distribution whose total sums up to 1 [38]. The Soft-Max can be seen as a generalization of the sigmoid function which was used to represent a probability distribution over a binary variable [72]. While preparing the data for a multi-class model, a specific encoding should take place to make The Soft-Max layer feasible. An integer variable would be allocated for each class label from 0 to N-1, where N is the number of classes. Nevertheless, these variables, instead of being single integers, are transformed into one-hot format encoded vectors. It is a probabilistic representation of the class label to fit with the soft-max output and keep the dimensions aligned in matrix operations. A vector of zeros for each class is created with dimension N, where only position of the suitable label is highlighted by 1. For instance, three class labels will be integer encoded as 0, 1, and 2. Then transformed into vectors in the following manner:

- Class 0:[1,0,0]
- Class 1:[0,1,0]
- Class 2:[0,0,1]

This is called “one-hot” encoding.

The soft-max function with equation showing below:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^N e^{z_j}} \quad \text{for } i = 1, \dots, N \quad \text{and } z = (z_1, \dots, z_N) \in \mathbb{R}^N \quad (2.8)$$

where z_i are the input elements.

The Soft-Max will output a probability of class membership for each class label, for every given input. So, if the expected target vector for one example would be:

- Ideal output: [0,1,0]

It will approximate the ideal case by putting most of the weight on class 1 and less on the other classes. In our instance, it would be something like:

- Expected output: [0.09003057 0.66524096 0.24472847]

The error between ground-truth values and predicted multinomial¹² probability distribution is often calculated using categorical “cross-entropy”¹³ loss function, and this error is then used to back-propagate and update the model [45].

2.3 Practical Computer Vision

The use of computer vision gained momentous success and popularity in HEP and Data Science in general from the potential it offered to improve and speed up visual data analysis. The purpose of this is to classify, detect and identify coordinates of objects in images from basic RGB data. This section will describe most used CNN architectures, how they evolved to become feature extractors or “Backbones” in computer vision algorithms. Afterwards, a detailed description of the workings of Object Detection will be presented along with some basic vision algorithms and how they function harmoniously with the backbones as hybrid models. In each section detailed explanation and examples of both feature extractors (i.e MobileNetv2, ResNet and Inception) and basic vision algorithms (i.e YOLO, SSD, FPN and Faster-RCNN), will be offered. Afterwards, the four state-of-the-art hybrid computer vision models to be used after training will be introduced. These are: YOLO, SSD-MobileNetv2, SSD-ResNet50-FPN and Faster-RCNN-Inception-ResNet. Table 2.1 shows in what

¹²Multinomial because there are multi-values for every class that result in a probability distribution.

¹³which is like the “cross-entropy” loss function, only this one is categorical instead of binary.

Model	Part	
	Head (Neck)	Backbone
YOLO	YOLOv3	CSPDarknet53
SSD_MobileNetv2	SSD	MobileNetv2
SSD_ResNet_FPN	SSD(FPN)	ResNet50
FasterRCNN_InceptionResNet_V2	FasterRCNN	Inception+ResNet

Table 2.1: Used Hybrid models' components

order these models are combined. In general Hybrid models are formed with two parts: First, the *Backbone*, extract the features from the input space, more commonly known as CNN but in this study will be referred to as Feature Extractors. Second, the *Head*, which is the object detector algorithm that sometimes contain an extension called *Neck*¹⁴. Darknet API was used for YOLO & Tensorflow2 OD API for all others. All these models and their components will be briefly introduced in what is next.

2.3.1 Feature Extractors

As was mentioned before, CNNs main goal is to understand objects in images through feature extraction with sets of repetitive blocks down the pipeline. We then keep cycling in epochs to improve our weight until we get the right output predictions. The training, validation and evaluation follow the same processes described earlier. ML classification algorithm tries to learn about the nature of an object and predict it by learning from its features. These would have been already prepared by human intervention. However, what comes at a striking difference are Deep Learning (DL) Convolutional Neural Nets (CNN) that learn the features while training simply on images of an object. CNNs are inspired by the human brain where the patterns of connectivity between the nodes can learn features without the need for further intervention. Additionally, CNNs can not only predict the object but as well extract its features for even further use. Performance checks can be done following some specific metrics¹⁵ depending on the case.

¹⁴This study will present such extensions in one OD-extension algorithm used called FPN.

¹⁵Standard measurements known in the community of computer vision that will be explained later.

2.3.1.1 Residual Networks (ResNets)

In theory, very deep networks with multitude of layers should perform better since they can represent overly complex mapping functions; but in practice, they proved to be worse. Residual Networks, introduced by He et al. [24], had as main goal to tame and train much deeper networks than what were previously feasible. The main benefit and importance of a very deep multi-layered network lies in its ability to learn features at many distinct levels of abstraction, from edges (at the lower layers) to extraordinarily complex features (at the deeper ones). However, a huge barrier to training them is *vanishing gradients*: very deep networks often have a gradient signal that goes to zero quickly, thus making training unbearably slow. Moreover, during gradient descent cycles, the gradient might decrease exponentially quickly to zero (or, in rare cases, grow exponentially quickly and “explode” to take exceptionally large values). A Residual Network can solve this problem by introducing “shortcuts” or “skip connections” that allow the gradient to be directly propagated to later layers and vice versa in back-propagation.



Figure 2.12: A ResNet block showing a “skip-connection”

Figure 2.12 left shows main path through a classical NN. Right, a shortcut is added to it which is now called a ResNet block¹⁶. By stacking these blocks together, a solution is presented to make possible the use of a very deep network. The first reason behind that is the ease for any of these blocks to learn an identity function. A number of these ResNet blocks can be stacked with negligible risk of harming performance. While as mentioned before with very deep nets the weights tend to vanish, the shortcuts would allow earlier outputs to be propagated to later layers even if the weights in between vanished. So, since all the activation functions are ReLU, any activated positive value in early layer would remain the same with skip-connections and gets propagated forward. The shortcut would be adding the old, activated value to the weights and biases, then activating the sum with another ReLU as seen in the

¹⁶A block is a package of layers of different kind in a sequence.

equation 2.9 below:

$$a^{[l+2]} = \text{ReLU}(w^{[l+2]}a^{[l+1]} + b^{[l+2]} + a^{[l]}) = a^{[l]} \quad \text{if } w^{[l+2]} = 0 \quad \text{and} \quad b^{[l+2]} = 0 \quad (2.9)$$

Where a is the activation, w is the weight, b is the bias and the superscript l is the layer's index¹⁷. This is what meant by identity function learning.

Identity Block corresponds to the case where the input activation ($a^{[l]}$) has the same dimension as the output activation ($a^{[l+2]}$). Figure 2.13 (Top) shows a schematic of an identity block, the upper trajectory is the “shortcut path” while the lower one is the “main path”. The sum is kept propagating an identity function in case the weights vanished in between the layers.

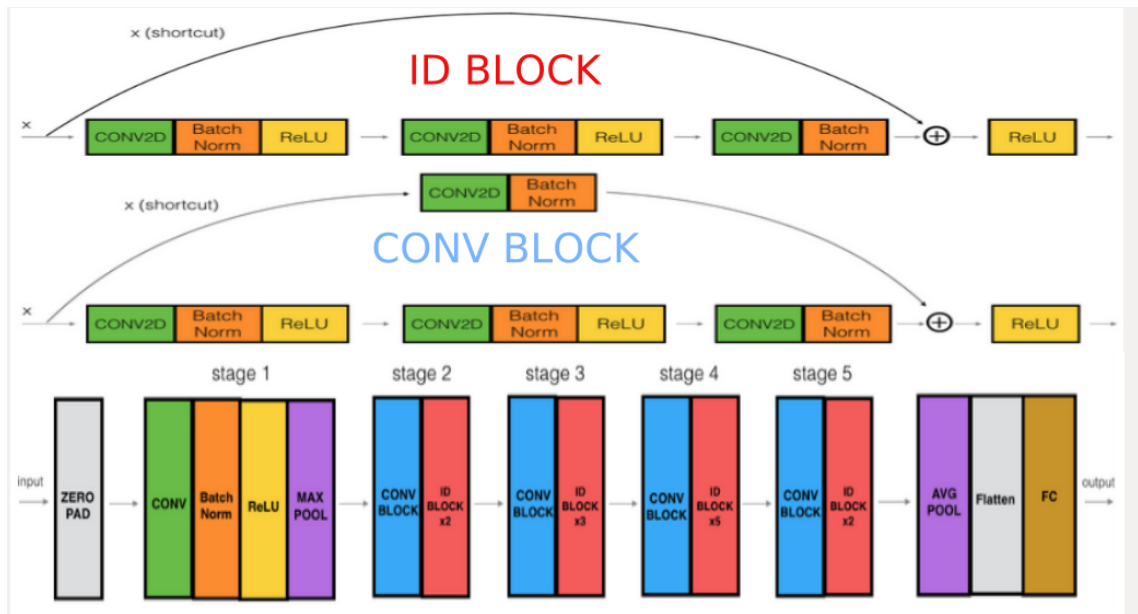


Figure 2.13: (Up) in red “ID BLOCK” with skip connection over the main path: 3 convolutional layers, here “CONV2D”, 3 batch normalization passes, 2 ReLU activation layers in between, shortcut and input of main path are added together and ReLU activation finally applied on the sum, (Middle) in blue “CONV BLOCK” same but adding a CONV2D with Batch Norm filter on the skip connection over the main path, (Bottom) general structure of ResNet50 [95].

¹⁷Check please appendix, where these notations are thoroughly explained

Convolutional Block is the other type of block used when the input and output dimensions do not match up. The only difference from the identity block is that it has a CONV2D layer with a Batch Norm¹⁸ in the shortcut path. Figure 2.13 (Middle) shows its structure.

The CONV2D layer in the shortcut path is used to resize the input x , so that the dimensions match up in the final shortcut addition needed with the main path.

These are the two necessary blocks to understand the structure of a very deep CNN “ResNet” which is formed of 50 (Figure 2.13 (Bottom)) or more layers.

2.3.1.2 Inception Network

Inception Network is the most state-of-the-art of all ConvNets (Convolution Networks). Main motivation behind its design is to build deeper multi-layered networks [21]. Inception is a heuristic approach to go wider in parallel and not only in length just like the movie “Inception”, onion-like layers inside layers but also built in chains.

To understand Inception module¹⁹, we first need to introduce few concepts more or less like the block-components in ResNets. However, in Inception, blocks go in parallel within modules.

The Inception module makes use of the so-called “Networks in Networks and/or Bottlenecks mechanisms” [26].

Networks in Networks: a convolution of 1×1 does not seem particularly useful. As we have seen before in the convolution section it would then be a simple multiplication by some number. For instance, that is the case of $6 \times 6 \times 1$ channel images. So that if we take the 6×6 image and convolve it with a 1×1 filter, we will end up just taking the image pixel values and multiplying them by a number. However, if we take now a $6 \times 6 \times 32$ image (with 32 channels depth-wise), then a convolution by a $1 \times 1 \times 32$ filter can do something that makes much more sense. In figure 2.14 we can see the difference clearly, where in the case of interest a 1×1 convolution will look at each of 36 positions in the image, and it will take the element wise product between 32 numbers on the left and 32 in the filter. Then we will apply a ReLU non-linearity to the sum after that in each 6×6 positions in the output. More generally, we usually have multiple similar filters. So, the same operation is repeated for every filter creating an extra layer or channel depth-wise. The final output dimensions would become $6 \times 6 \times$ “number of filters”.

¹⁸Batch normalization is applied on the input layer by a simple “Keras” function (<https://keras.io/>) For re-centering and re-scaling purposes.

¹⁹Module is defined as a set of fundamental structure that is repeated, it is bigger than blocks and might contain many of them.

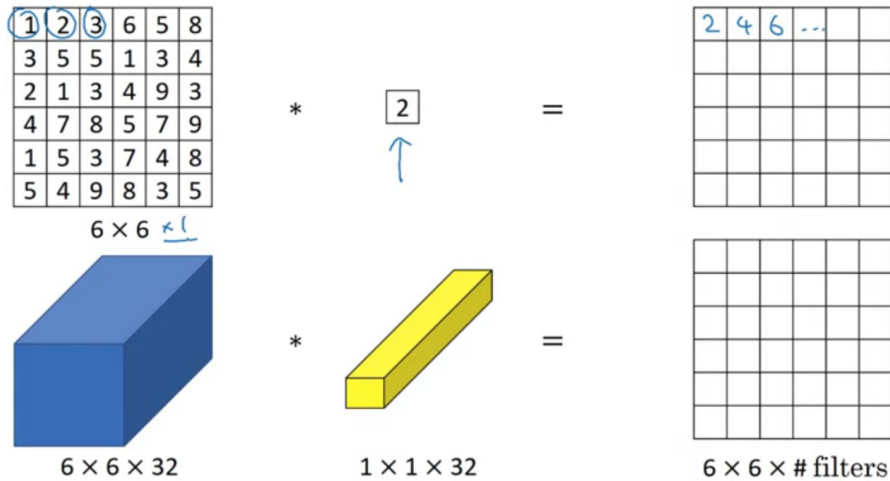


Figure 2.14: “Networks in Networks” mechanism [95]

So, by limiting the quantity of 1x1 convolutional filters used, we can reduce the dimensionality of input volume by having an output with less depth wise channels. We will be doing so while increasing the non-linearity to learn more complicated features as was explained before. The result of that, we would be making full use of a representation that learns advanced features while keeping the computational cost much lower. We will be discussing that in the next part.

Bottlenecks: are filters formed of 1x1 convolutions and other related CONV layers as well. These became quite common for their significant effectiveness in lowering computational costs. The way they do that is by having an intermediary stage that applies a 1x1 convolution to reduce the number of depth-wise channels before increasing them again to realign dimensions in the pipeline. This bottleneck filter is particularly useful as an intermediate step when we need to do large convolutions. An extra 1x1 CONV in the filter can reduce the computational cost to one tenth [95]. Bottlenecks, only inverted, are the core foundation and essential blocks of MobileNets pipeline which we will be discussing next and are effective for reducing computational expenses.

Inception net is a series of repetitive modules with the same structure. One of which, as shown in figure 2.15, consists of:

- Bottleneck block: 1x1 followed by a 5x5 convolution
- Bottleneck block: 1x1 followed by a 3x3 convolution

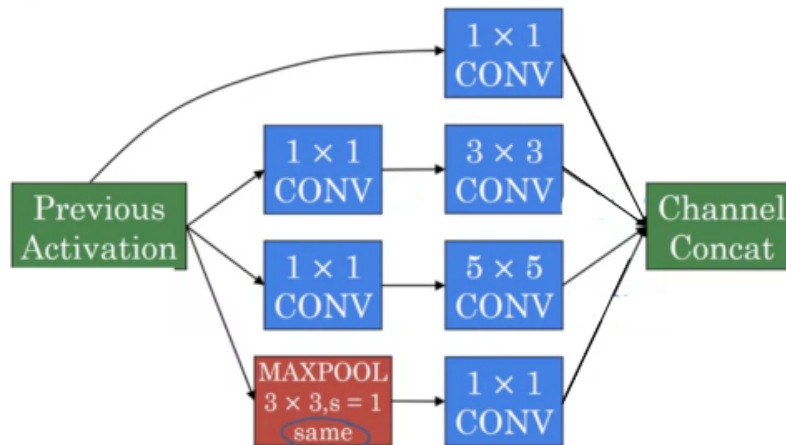


Figure 2.15: Inception module [95]

- Network in Network block: 1x1 convolution
- Pooling block: 3x3 MAX-POOL with stride of 1 and “same” zero padding to align with the previous activation’s height and width. Then, followed by a Network in Network to shrink the number of channels.

The computations for these 4 blocks paths go in parallel within each module. Then the outputs are concatenated (or stacked) together again into one 3D shape of $n_H \times n_W \times n_C$. The notations²⁰ (n_H, n_W, n_C) refer to height, width, and number of channels, respectively. As appearing in [21], Inception Network is a collection of many similar modules put together. One of the blocks would contain the same components as described above, and it is the same for any other block. Only with some extra max pooling added to a few nodes between modules to change the size or dimensions.

Finally, as is the case in all Nets, the last few layers are a fully connected (FC) one followed by a SOFT-MAX to make a prediction. A couple of these predictive layers are attached to other nodes as well just to check predictions on earlier stages and keep it from over-fitting. Inspection Network are an active field of research [95]. While hybrid nets combining blocks of ResNets and Inception Nets are also common and will be discussed and used later.

²⁰See Appendix

2.3.1.3 MobileNets

The problem with nets like ResNets and Inception is that they can be exceptionally large in the order of 200-500MB or even much more depending on the architecture. Network architectures such as these are unsuitable for resource constrained devices due to their sheer size and resulting number of computations [103]. MobileNets, first proposed by Howard et al. in 2017 [19], can be used instead since they are designed for such purposes. MobileNets differ from traditional CNNs through the usage of inverted *depth-wise bottlenecks* described previously (Figure 2.16).

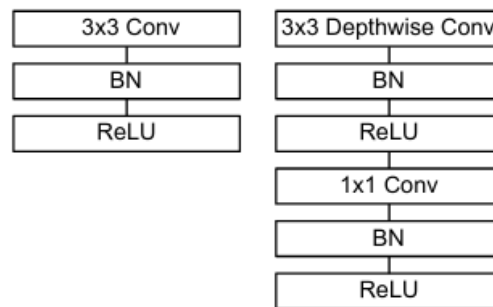


Figure 2.16: Left: Standard convolutional layer with batch-norm and ReLU. Right: Depth-wise Separable convolutions with Depth-wise and Point-wise layers followed by batch-norm and ReLU [19].

Classical convolutions (Conv) have the dimensions $n_H \times n_W \times n_F \times n_C$ where the additional n_F here represents the number of filters used. While depth-wise separable convolutions (Conv dw) are the same concept described earlier as **Bottlenecks** with $n_H \times n_W \times n_C$ dimensions. So that the architecture of MobileNets would be consisting of alternating (Conv) and (Conv dw) blocks finishing by the usual predictive part (FC + SOFT-MAX). Where the (Conv dw) block consist of an inverted Bottleneck as in Figure 2.16 (Right):

- 3×3 convolution, batch normalization and ReLU (named by the authors: “Depth-wise Convolution” block)
- 1×1 convolution, batch normalization and ReLU (named by the authors: “Point-wise Convolution” block)

However, MobileNet (version 2) or MobileNetv2 [25] will be the one adopted instead in our later work. It adds an extra Conv dw block with a skip connection

which is like the ones described for ResNets. Plus, it drops the non-linearity factor in all the blocks' output nodes, while keeping a simple conv 1×1 with linear pass.

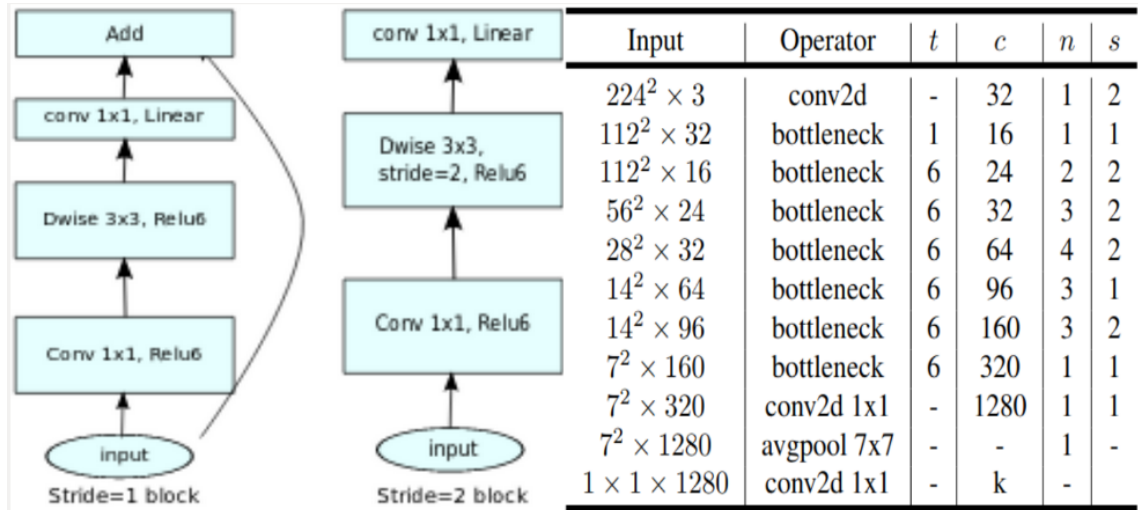


Figure 2.17: (Left) MobileNetv2 main blocks, (Right) MobileNetv2 body architecture [25]

The 2 different blocks used in this version as shown in Figure 2.17 (Left) are:

- A stride=2 linear bottleneck block for downsizing
- A stride=1 residual and linear bottleneck block to avoid losing information along the MobileNetv2 deep architecture

Finally, the overall architecture of MobileNetv2 is showing in Figure 2.17 (Right). Where following the original paper's annotations, each line describes a sequence of 1 or more identical blocks, repeated n times. All blocks in the same sequence have the same number of output channels c . The first block of each sequence has a stride s and all others use a stride of 1. The first classical conv2d is a 3×3 one and others in the final part of the pipeline are 1×1 . All modules in bottleneck sequences are either of the first type if $stride = 2$ (Linear Bottleneck) or of the second if $stride = 1$ (Residual Linear Bottleneck). Moreover, the expansion factor t is a simple multiplicative applied on the input channels number ($output\ channels = t \times input\ channels$).

2.3.2 Object Detectors OD

There is much more to object detection than to simple image classification. In the latter, given an input image, passing it through a CNN (i.e., any trained feature

extractor), leads to a single prediction output with a certain class probability. This is meant to evaluate the whole image or the main part of its visible contents. So, it is a one-to-one correspondence between input and output. On the other hand, object detection complements image classification and adds more features. It can detect exactly where or in what bounded region of the image each required object is located. This detection should ideally be made independently of the size of the object or scale of the image. So, to say, it is a one-to-many correspondence between input and output. When an input image is presented to an object detector it is expected to get in return:

1. A list of bounding boxes, i.e., their (x,y) coordinates (*top left and bottom right corners*) for each object in the image
2. A Class label associated with each of the bounding boxes
3. A probability (i.e., confidence score) associated with each bounding box and its class label.

The above-mentioned output details are encoded in a prediction vector corresponding to each individual detection of every image from the input space. In principle, these prediction vectors consist of 5 basic inputs which are the “existence” flag p_c which signals the existence of an object within a region of the image²¹, the (x, y) coordinates of the center point of the bounding box b_x and b_y within and relative to the RoI or grid cell dimensions, its height b_h and its width b_w within and relative to the RoI or grid cell dimensions as well. While b_x and b_y can only be between 0 and 1, b_h and b_w can be larger than 1, simply because an object with its bounding box could cover more than one RoI or grid cell (Figure 2.18). Other inputs would be added to these depending on the classes’ number included in the dataset. So, if it contains three classes, three inputs would be added as c_1 , c_2 and c_3 , each corresponding to a different class label. These are binary indicators that will only flag 1, the corresponding class of the object, while the others would stay 0.

These binary indicators are made so after being induced by a Softmax function as a multinomial probability score distribution. As a direct consequence of this probabilistic score competition, the highest scorer class would win the flag “1” and the losers would stay “0”. As a direct application, figure 2.18 is clearly showing the prediction vector of the car appearing in the image with its bounding box. The car’s center (*Black Dot*) is in a specific grid out the 9, which makes it the central grid (*Red Square*). All measurements now made for the vector’s inputs are made relative to

²¹this region could be one part “cell” of the input image selected out of an equally divided grid or voted through a more complicated mechanism and hereby called Region of Interest “RoI”.

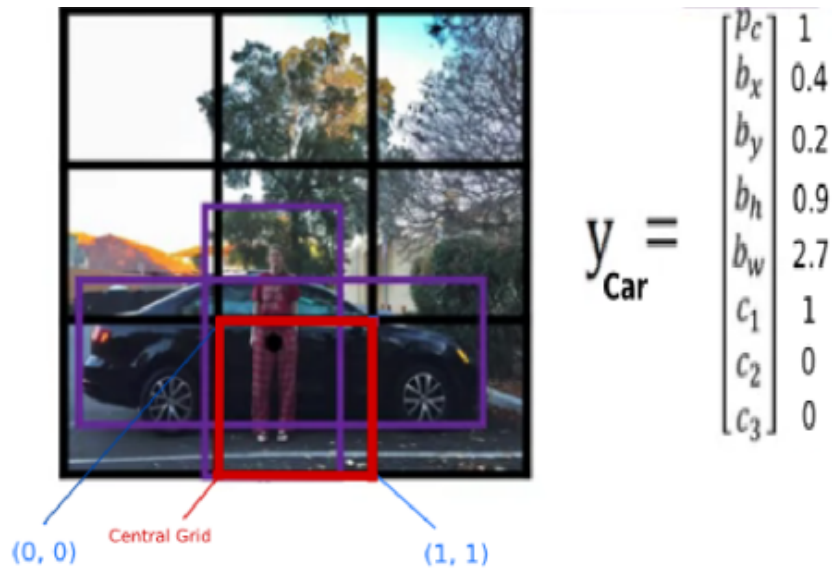


Figure 2.18: An output vector y_{car} is showing and associated with the car’s bounding box in this 3x3 grided image. The proper input values are stated relative to the central grid (*red square*) [95].

that specific grid. The upper left corner being $(0, 0)$ and the lower right $(1, 1)$. Now in y_{car} we have:

- $p_c = 1$ because we found an object in that specific grid which center is within the grid’s boundary
- $b_x \approx 0.4$ because the car’s center is at 40 % of the central grid’s horizontal origin.
- $b_y \approx 0.2$ because the car’s center is at 20 % of the central grid’s vertical origin.
- $b_h \approx 0.9$ because the car’s bounding box height is 90 % of the central grid’s height.
- $b_w \approx 2.7$ because the car’s bounding box width is 2.7 times the central grid’s width.
- $c_1 = 1$ because the winning class label is “Car” and c_1 corresponds to it.
- If c_2 and c_3 , for instance, correspond respectively to “Person” and “Motorcycle”. For the bounding box we are dealing with here, these two would be the losing class labels and consequently c_2 and c_3 would stay 0.

In what's next, YOLO, SSD, Faster-RCNN and FPN OD algorithms are briefly presented.

“You Only Look Once” or **YOLO** [23] [22] would take a batch of images as an input, and outputs a list of prediction vectors. A single pass would consist of the following path: IMAGE→ DEEP CNN→ ENCODING.

If the midpoint of an object falls into a grid cell, that grid cell becomes “central” and is responsible for detecting that object. For each output vector of each cell a probability is computed to determine how likely it is in this cell to find a certain class. Furthermore, an intermediate visualization step in YOLO is practiced. It consists of assigning a color to every grid cell that might contain an object of interest. That color is associated with the class that this specific grid cell is most likely detecting (see figure 2.19). Now for each colored cell there would be many bounding boxes associated with the same or different objects. A filtering process would then take place and the main objectives of such filtering are:

- Getting rid of boxes with a low fidelity score.
- Selecting only the best box when several ones are associated with the same object and overlapping.

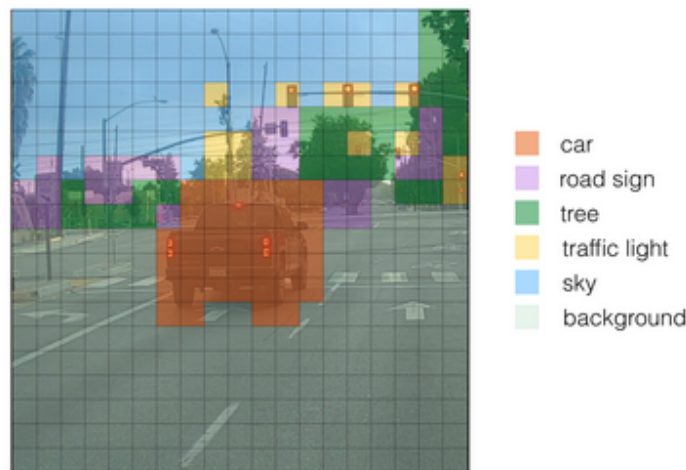


Figure 2.19: Each of the 19x19 grid cells colored according to which class has the largest predicted probability in that cell [95].

SSD or “Single Shot Detector” [29] is another fast OD that generates bounding box coordinate proposals (i.e., prediction vectors). SSD algorithm extracts features at

multiple scales while decreasing the input size progressively through the final stages of the pipeline. Bounding box regression technique called “MultiBox” used in SSD is inspired by Szegedy et al. paper [20]. It is a fast method to generate bounding box coordinate proposals in prediction vectors similarly to what YOLO does but through a different process. However, instead of a single vector it outputs 2 (LOC and CONF). While reducing input scale the detection of such vectors becomes more accurate and selective (Figure 2.20).

The two prediction vectors LOC and CONF consist of the following:

- CONF: this encodes how confident the network is that an object exists within the boundaries of a computed bounding box. The vector is related to the prediction confidence for each class, and it would consist of as many coordinates as the number of classes in the dataset. So, for a dataset of p number of classes one gets:

$$CONF = (c_1, c_2, \dots, c_p) \quad (2.10)$$

- LOC: this output indicates how far away the network’s predicted bounding boxes are from the ground-truth ones in the training set, because it would predict the coordinates and dimensions of the bounding box around the object of interest:

$$LOC = \Delta(cx, cy, w, h) \quad (2.11)$$

Where the 4 coordinates are the difference in the dimensions between the generated and ground-truth bounding box, cx and cy represent the upper left and lower right corners’ location while w and h refer to the width and the height.

Faster R-CNN OD works by learning to localize with a deep neural net “Region Proposal Network” (RPN) [27]. RPN infuses Regions of Interest (RoI) proposal into the architecture directly, allowing the Backbone CNN to produce prediction mono-vectors remarkably like the ones used by YOLO, whereas the difference here is that RPN replaces the grid cells technique by selecting and passing the RoIs into the architecture itself. Figure 2.21 shows how Faster R-CNN works on two stages using two different NNs, one to generate RoIs and the other to encode prediction vectors.

FPN [28] is an OD-extension or Neck, if built on top of any OD, it would extract prediction vectors out of every independent level of the feature pyramid (i.e., hierarchy from simplest to most complicated features) taking into consideration every scaling of the input space from the largest to the smallest. Figure 2.22 shows a feature pyramid

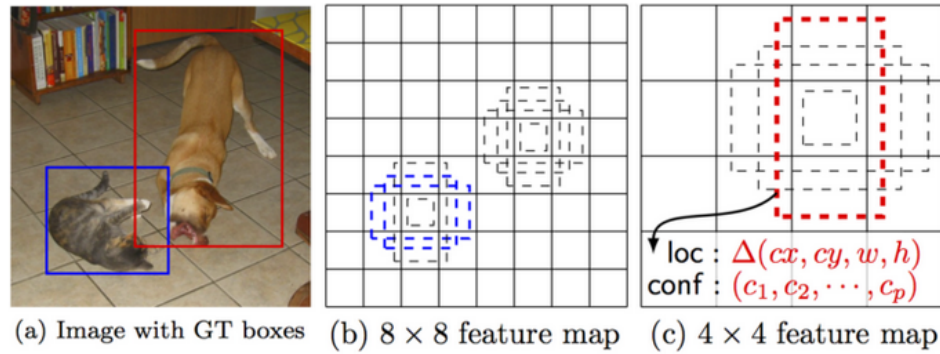


Figure 2.20: In a) Ground Truth (GT) boxes are showing, one for the dog in the image and one for the cat. b) SSD default boxes at 8×8 where it is more likely to find central cells, and in c) 4×4 feature maps include both prediction vectors CONF and LOC with a much higher chance to pick the box with the right shape.

where prediction vectors in object detection are independently made on each scale level of the pyramid.

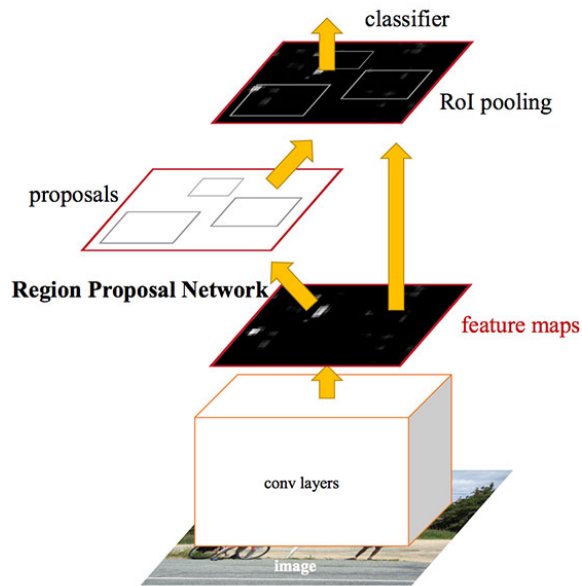


Figure 2.21: The Faster R-CNN architecture [27].

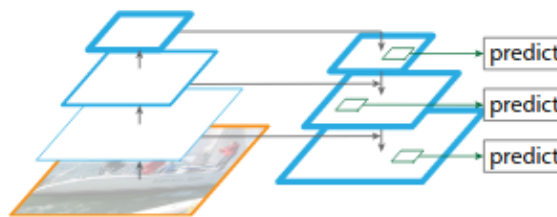


Figure 2.22: The FPN method with predictions made independently at all levels [28].

Chapter 3

NN and K-pi Matching

3.1 Inputs and Design

In this chapter, the focus is on developing a supervised learning algorithm. The vertebra of such an algorithm is the learning function which is the “cost function” itself. The most common one used in general for binary classifications is the cross entropy cost function. The interest is in a *Boolean* classification, wherein each example is classified as true(1) or false(0). Obviously, a prediction is good if it turns out to be true, so we can assess the quality of a hypothesis by checking its predictions against the correct classification once we know it. We do this on a set of unseen examples known as the *test set* (See Chapter 2 for more details).

3.1.1 Data Preparation

The *data preparation* process has been done on stages. Events $K^+ \rightarrow \pi^+\pi^+\pi^-$ were selected from the data collected by NA62 during the run of 2017 to form the training sample. The selection proceeded in 2 main steps: firstly, events with at least 3 reconstructed tracks were filtered at processing level; secondly, an analysis algorithm selected events with at least a track triplet consistent with the $K^+ \rightarrow \pi^+\pi^+\pi^-$.

To this extent each track had to be in the geometrical acceptance of the sub-detectors downstream to the last STRAW chamber. In addition, each track must match signals in CHOD detectors, a cluster in the LKr calorimeter, and should not have signals associated in MUV3. The last requirement rejects accidental muons, and pions decaying to muons. The decay of pions occurring upstream to the STRAW spectrometer spoils the kinematics of the pions, affecting the quality of the K^+ tagging used to flag the training events. Internal consistency between the time of the tracks of a candidate triplet measured with the CHOD signals is required against accidental

triplets. A simple pion identification is applied to the tracks based on the ratio between the energy E of the LKr cluster associated to the track and the magnitude of the track 3-momentum p measured by the STRAW spectrometer that must be consistent with the pion hypothesis defined by $E/p < 0.8$.

Once the 3-pions candidates are selected, quality requirements are applied to the whole event to identify $K^+ \rightarrow \pi^+\pi^+\pi^-$ decays. A time matching between the candidate 3-pions and a signal in KTAG allows the identification of a K^+ in the initial state. This association exploits the average 3-track time measured with CHOD that must be within ± 2 ns of the closest in time KTAG signal. The decay vertex is defined as the mid-point of the 3-tracks at the closest distance of approach. The position of the vertex along the beam axis must be within the decay region defined from 110 to 170 meters downstream to the target. In addition, the 3 tracks must be closer than 10 mm one from the other at the vertex position. Events with signals in LAV within ± 3 ns of the average 3-pions time are rejected to further reduce possible accidental backgrounds.

A final set of additional requirements are applied as kinematics constraints to the selected events. The total 3-pions momentum must be consistent with the nominal 75 GeV/c beam momentum within ± 2.5 GeV/c. The invariant mass of the 3-pions must be within ± 2 MeV/c² of the nominal K^+ mass. The expected K^+ position at the decay vertex and slopes must be consistent with the expected beam envelope defined by the optics of the beam line.

The track of the parent K^+ track in the Gigatracker detector (GTK-track) was defined based on a χ^2 -like variable. The inputs to this variable were the difference of the GTK-track slopes and position with respect to the K^+ slopes and position expected from the 3-pions tracks reconstructed downstream. A broad upper cut at 40 on the χ^2 was applied to reject events without K^+ reconstructed. This cut was chosen broad enough to minimize the bias to the training sample induced by the selection criteria, which has been verified with a posterior to be negligible and decoupled from kinematics in $K^+ \rightarrow \pi^+\pi^+\pi^-$ with all charged pions in final state¹. In the same spirit no time constraints were used to label the K^+ as the time matching, between GTK-track and downstream tracks, is a key input of the NN algorithm. A total of 16 million events were selected, to make the sample as general as possible and accurately covering all features of the $K^+ \rightarrow \pi^+\pi^+\pi^-$ decay channel for π^+ tracks. Then ROOT Trees were built from already labelled Tracks (either from a K^+ parent or from pileup), following the selection process described extensively in 1.5. These

¹Please review figure 1.7 which shows the clean top region of highest squared missing mass with the 3π peak showing around the expected nominal momentum of the kaon beam

Trees contained the following variables:

- The average number of hits in the GTK stations (nHits).
- The three times in sub-detectors (dtKTAG, dtRICH, dtCHOD²) respectively. Considering the GTK time as a reference for the problem in hand a cut of ± 1 ns was applied on the time differences.
- The individual time of hits in CHOD (ftCHOD).
- The individual time of hits in KTAG (ftKTAG).
- The individual time of hits in RICH (ftRICH).
- The average individual time of hits in all three GTKs (ftGTK).
- The estimated X and Y track slopes in GTK (fSlopeX and fSlopeY).
- The estimated X and Y track slopes in STRAW (fSXBM and fSYBM).
- The X and Y average hit positions in GTK (fPositionX and fPositionY).
- The X and Y average hit positions in STRAW (fPBMX and fPBY).
- The Closest Distance of Approach of GTK track with one Pion in the final state (CDA).
- The χ^2 of the track (Chi2).
- The χ^2 of the associated π^+ track (Chi2Pion).
- The instantaneous intensity in the offset time of the K^+ tracks (fLambda).
- The time of individual hits in GTK1 (fTrackTimeS0).
- The time of individual hits in GTK2 (fTrackTimeS1).
- The time of individual hits in GTK3 (fTrackTimeS2).

$K^+ \rightarrow \pi^+\pi^0\pi^0$ is a super clear signal sample with a distinguished squared missing mass distribution that serves very well to do all kinds of tests on the kaon-pion matching algorithm.

For the sample we picked, 23 variables are arranged and labelled, the ones coming from a K^+ parent track with “1” and the ones from accidental detector activity with “0”. A loose cut based on an educated guess was also applied fixing a range of ± 1 ns

²dt refers to the time difference of the hits with respect to GTK reference time.

on time differences and a CDA $\leq 15\text{mm}$. Then the sample is divided into a Training set (95%) and a Validation set (5%) following the strategy of “cross-validation” [96] to prevent and monitor *Over-fitting*. Finally, a synchronized initial Shuffle on the input matrices is performed to avoid biased learning. It is important to note that dimensional consistency check has been performed on the variables’ construction process. The ones with consistent dimensions should be stacked together to form finally the main input matrices.

3.1.2 Basic Development

The second step was the development of a *Supervised NN Algorithm* that uses all the parts detailed in introduction, which is built on many Python-based helper functions. An easier way is to directly use any API to call the functions needed for the model. However, in this case, implementing all the functions (inspired by [95] online course) was primarily educational and deeply insightful in expanding the understanding of the basics of NNs programming functionalities and general practice for the following parts of this thesis.

To start, the mathematical definition of the sigmoid and ReLU as *activations* are prepared in functions³. Then, *initialization* of the parameters with “He” method coded (as explained in Chapter 2). After that, a function that selects the mini batches randomly according to the size selected (1024 in our case), then does the *shuffling* and the associated dimensional *partition*. Also, the *Forward Propagation* that computes the outcome in the forward direction. The definition and computation of the *cross-entropy* cost which is the average of the loss function of the entire training set. The *Backward Propagation* that computes the gradients in the backward direction as well. Then, another function that *updates* the main parameters following Adam optimization main formulation. Then, *prediction* function that takes the output of the final layer and transforms it into a probability of 0 if ≤ 0.5 and 1 otherwise. By counting the Mean of the true predictions we determine the *accuracy* that evaluates the model. Also, this probability threshold could be varied to get the best efficiency vs accuracy, a method to calibrate will also be presented later in the chapter.

The decisive step is the *model* function that compiles and calls all the “helper functions”. The model works in the following way:

- Takes as input the data and a handful of manually selected hyper-parameters (The X(data) and Y(labels) matrices, the layers dimensions array, the learning

³This step could be avoided since these activations are so popular now and they can be called from any commonly used platform (i.e., Keras-Tensorflow, Pytorch etc..)

rate α , the mini batch size, the two momenta of Adam optimization β_1 and β_2 , a small number ϵ to avoid division by zero and the number of epochs which are the iterations over the mini batches).

- Initializes the main and Adam parameters.
- Starts iterating over the epochs, for every mini-batch and over all the layers it:
 - Calls Forward Propagation.
 - Computes the cost.
 - Calls Backward Propagation.
 - Update the main parameters with Adam optimization.
 - Returns a dictionary with the updated main parameters.

As a first test model, the input preparations were performed in the following steps:

- Choose a control data sample (A whole run of 2017 data with a total of 812443 events)
- Build a ROOT Tree from already labelled Tracks ($\pi\nu\nu$ -like environment reconstructed from pure kinematics in channel $K^+ \rightarrow \pi^+\pi^+\pi^-$ or $K^+ \rightarrow \pi^+\pi^0$).
- Arrange 9 test variables and label the ones coming from a K^+ with 1, and the ones from accidental detector activity with 0. These basic 9 variables were:
 - The three time differences between sub-detectors (KTAG, RICH, CHOD) and (GTK) respectively. Considering the GTK time as a reference for the problem in hand.
 - The CDA which is the closest distance of approach between two tracks.
 - The χ^2 of the K^+ .
 - The χ^2 of the π^+ .
 - The instantaneous intensity Lambda (in the offset time of the K^+ tracks).
 - The two transverse projections (VTRx and VTRy) of the normal vector VTR common to the two tracks. The magnitude of such a vector is what we call CDA or closest distance of Approach.
- Apply a loose cut on the events and selecting a range of ± 1 ns on time differences and a CDA ≤ 15 mm.
- Stack the data variables together to finally form the X matrix (where the dimensions are [9, 731083] as in 9 variables and 731083 input events for $X_{training}$ and [9, 81360] as in 9 variables and 81360 events for X_{test}).

- Stack the labels together to form the Y matrix (where the dimensions are [1, 731083] as in 1 label and 731083 input events for $Y_{training}$ and [1, 81360] as in 1 label and 81360 events for Y_{test}).
- Do a Cross-validation by dividing the sample into a Training set (90%) and a Test set (10%).
- Apply a synchronized initial Shuffle on the input matrices to avoid biased learning.
- Run training rounds.

This test model adopted is shown in Figure 3.1 and has the following components:

- A 3 layered NN with 530 nodes on the first layer, 200 nodes on the second and 1 prediction node as output.
- A learning rate α of 10^{-7} .
- A mini batch size of 256.
- Adam momenta $\beta_1=0.9$ and $\beta_2=0.99$.
- Number of training epochs 7700.
- $\epsilon = 10^{-8}$.

It is important to note here that the selection and tuning of the hyper-parameters is an exhaustive process and completely empirical (Details and techniques will be presented next section). The most important signal that was followed so far to estimate the progress of the training, is the quick decrease of the cost value and usually during the first 100 epochs. When the cost function value is dropping, it is a clear sign that the model is learning. We usually notice graphs like the one in Figure 3.2.

3.1.3 Architecture/model and Hyper-parameters Tuning

The main functions to train a NN algorithm were prepared and a first working test-model was done from scratch as shown in previous section, but the challenge remained in finding the proper architecture for the NN and tuning all the parameters required or needed in such machine learning models to acquire the best performance. Also, a proper adaptation of the model's inference with the general NA62 analysis was necessary. Additionally, we had to check and compare performance with other common machine learning models for the same case-study. After the educational

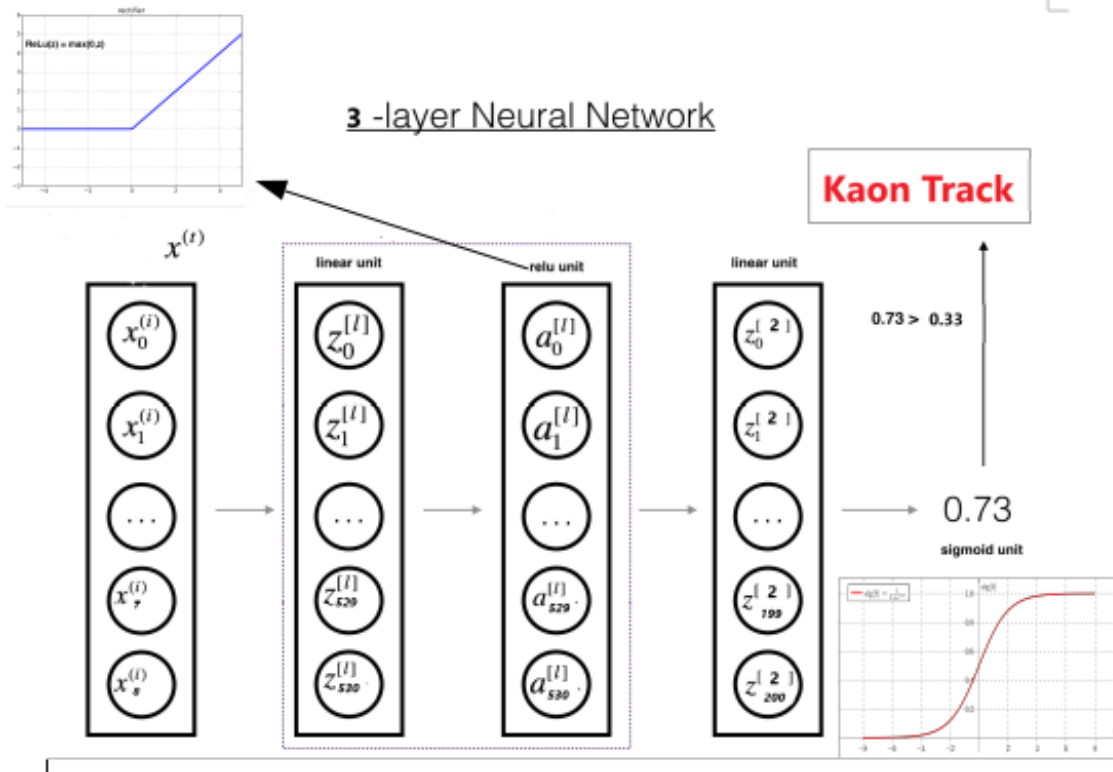


Figure 3.1: The (9,530,200,1) model diagram.

practice done on from-scratch NN code, a more practical implementation took place to save and use trained models in a general context (i.e., for prediction or inference). A second version of the code was reshaped to use some functionalities of renowned ML libraries. The NN algorithm encompasses all the above-mentioned details was based on Keras (<https://keras.io>). Keras is a Deep Learning framework for Python that provides a convenient way to define and train almost any Deep Learning model. However, it does not handle low-level operations such as tensor manipulation and differentiation. Instead, Keras relies on a specialized, well optimized tensor library to do so, serving as the *backend engine* [58]. Tensorflow was used which is developed by Google as this Backend (www.tensorflow.org).

Hundreds of different versions of the NN model have been tried and the hyper-parameters have been tuned many times in the training process. Grid search methods were used to select the right pack of hyper-parameters that goes with the best performing NN model. These tools consist in general of running rounds of training (with a smaller sample of the original data) on different NN architectures with

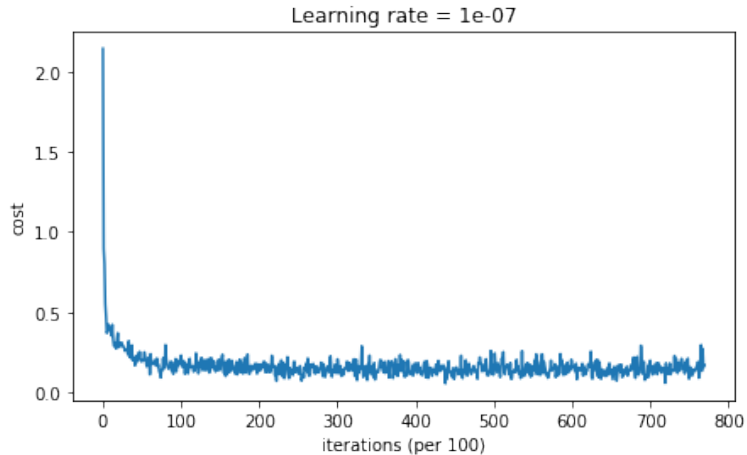


Figure 3.2: The following graph is usually called the “happy face”!

different choice of hyper-parameters from cycle to cycle. For the NN hyper-parameters search Talos library was used (<https://autonomio.github.io/docstalos/>). The outcome consisted of a list of best to worst performing architectures and hyper-parameters of the NN. This process can be done by hand, but it is highly impractical. However, if done with experience and professionalism using grid searches can save time and help boost results.

A grid search allows us to exhaustively test all possible hyper-parameter configurations that we are interested in tuning. Hyper-parameters are scanned with every combination tested on a data sample one at a time. The choice of parameters to grid scan should be educated and selective since a randomly substantial number of parameters may make the processing time grow exponentially which would prevent it to give back results within a reasonable period. A grid search will test all combinations of these hyper-parameters, trains a model for each set and then reports the best hyper-parameters (i.e., the set used in the model/architecture that demonstrated highest accuracy). The NN model hyper-parameters initial values chosen to run our grid search upon are the following:

- Learning rate values: [0.0001, 0.001, 0.01]
- First hidden layer number of nodes: [7, 64, 128, 200, 230, 530]
- Second hidden layer number of nodes: [7, 64, 128, 200, 230, 530]
- Third hidden layer number of nodes: [7, 64, 128, 200, 230, 530]

- Fourth hidden layer number of nodes: [7, 64, 128, 200, 230, 530]
- Batch size values: [64, 512, 1024]
- Activation between hidden layers: [Tanh, ReLU]
- Optimizer: [Adam, Nadam⁴]

From Talos, the “Scan” function trains the models using all the different combination of the parameters input. Then the “Reporting” class lists the hierarchy of the best to worst performing model on the metric specified from the available ones. The best metric to evaluate on is the validation set accuracy (e.g., in “table” function the metric is “val_acc”).

Also, for the comparison with different machine learning models and doing hyper-parameters⁵ tuning, SKLearn packages (e.g., GridSearchCV function from “model_selection” class) were used⁶. Five ML models other than the NN were tuned and tested (Logistic Regression, Random Forest, Gradient Boosting, Ada Boost Classifier and RBF SVM). All these models (in their default mode) can be called directly from SKLearn, either “linear_model”, “ensemble” or “svm” classes. Only the Radial Basis Function (RBF) kernel (SVM) Support Vector Machine model (i.e., “SVC” in SKLearn⁷) needed parameter tuning. The hyper-parameters initial values chosen to run grid search upon are the following:

- The trade-off parameter C values: [0.001, 0.01, 0.1, 1, 10]
- The inverse of the radius of influence parameter Gamma values: [0.001, 0.01, 0.1, 1]

Using the available limited computing power at hand, the grid search method and extensive hand picking (e.g., with different number of hidden layers). A lot of trials and error involved to reach a somehow satisfying first step performance so that the NN model with a specific set of hyper-parameters showed a relative out-performance over other trained models using different machine learning algorithms. ROC curves⁸ were used as a basis of comparison between different models. Figure 3.3 shows that

⁴Check “Nesterov Adam” optimizer at <https://keras.io/optimizers/>

⁵Only the NN model needed a special treatment with Talos packages since the tuning process is much more complicated due to the considerable number of parameters incorporated.

⁶<https://scikit-learn.org/stable/documentation.html>

⁷https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

⁸ROC curves in general measures Sensitivity (or True Positive Rate) VS 1-Specificity (or False Positive Rate) or in Physics terms Efficiency VS Mistag. An AUC (Area Under Curve) 100% score means perfectly efficient predictive model while a 20% is random.

the NN AUC score is the highest amongst the models (Boosted decision trees “Ada Boost”, SVM “Support Vector Machine”, Random forests and the others). However, that was not the only reason that led to the decision to go forward with the NN choice as final model. A lot of factors played a role, a few of which are the flexibility and adaptability of the NN with the data in hand, and the considerable possibilities to implement the model as a C++ class in the NA62 Framework which proved very useful in the final physics checks that will be presented later.

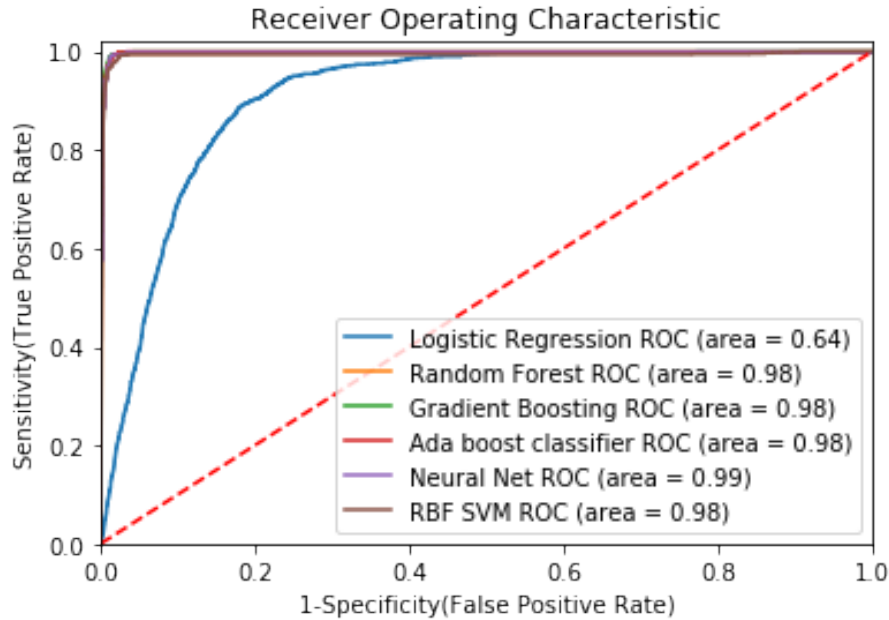


Figure 3.3: The following graph shows the ROC curves and AUC scores of the different models used in comparison tests.

The adopted final model consisted of the following selections:

- A 5 layered NN with 530 nodes on the first layer, 230 128 200 nodes on the activation layers and 1 predictive node on the output.(See Figure 3.4)
- A learning rate of 10^{-4} .
- A mini batch size of 1024.
- Optimizer “Nadam”, which is Adam with an additional parameter called the Nesterov Momentum.

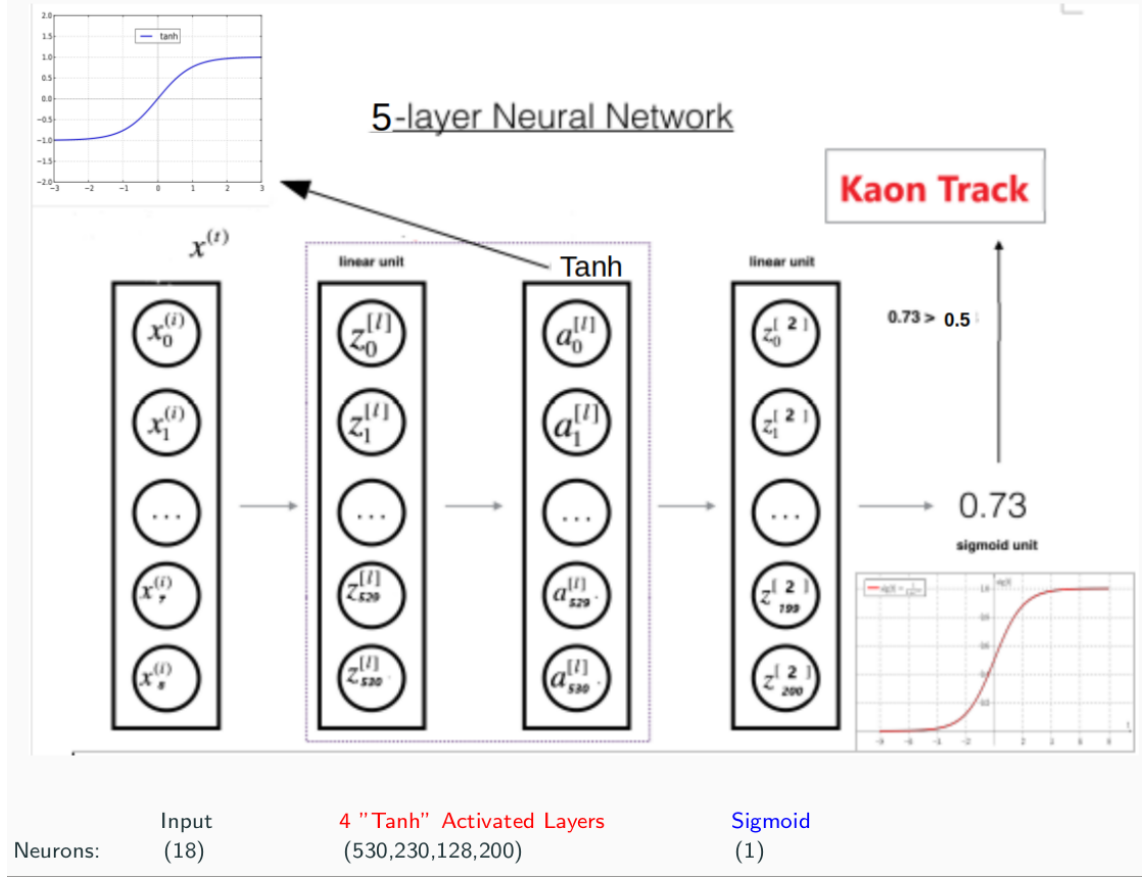


Figure 3.4: The (18,530,230,128,200,1) model diagram.

- *EarlyStopping* and *ReduceLROnPlateau* were used from Keras *callbacks*. This is to automatically reduce the Learning Rate or stop the training when you measure that the validation loss is no longer improving from one epoch to another [58].

From the above-mentioned NN, a classifier or a Discriminant algorithm was built in Python that discriminates on binary-basis between Kaon and Pileup events. Kaon tracks matched with π^+ in the final state were counted as Signal events while Pileup tracks matched with the π^+ were considered Background. It is essential to mention that these noise events are not the final $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ Upstream Background that will be addressed later in the main analysis. It is just a test model to prove that NN-based Discriminant can understand the physical topology of the $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ analysis in parameter space.

Figure 3.5 shows the probability distribution of the prediction outputs. There we can clearly see that the NN classifier successfully predicted Signal and Background to more than 98% accuracy on never-before-seen data of the test set. A green dashed line represents the probability threshold above which the discriminant will predict as Kaon Signal. Any cut on this distribution can be made and line can be moved along the horizontal axis where at each cut an efficiency and a mistag probability can be measured to finally decide which is the best point to place our threshold. On the right-hand side of it we can see clearly in red the true positives which are the rightly identified Kaons along with the false positives in blue which are the residual mistagged events where pileups are wrongly identified as kaons.

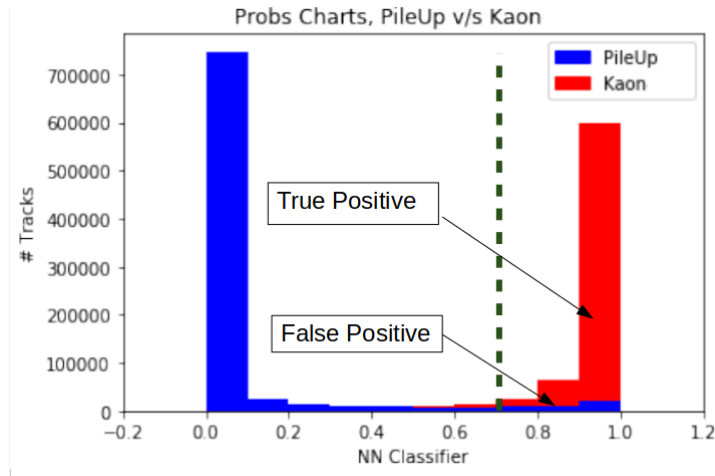


Figure 3.5: The probability distribution of the NN Discriminant output.

3.2 Results

To make the right comparison between the NN Discriminant and the old Likelihood based one we had to implement the model in NA62 Framework. A C++ class was written for the NN Discriminant, and the right statistical analysis was done in ROOT. Figure 3.6 shows the Performance⁹ plots of both the classical and new NN based classifier. The graphs shown in red and black are the outputs of threshold cuts between 0 and 1 of the NN and classical discriminant respectively on predictions of the same never-before-seen data sample. A significant improvement started showing where we got 5% more efficiency and 40% less mistag on the current working point

⁹Performance: Maximise the probability to associate the parent K^+ to π^+ (“Efficiency”) and Minimise the probability to associate an accidental or pileup track to the π^+ (“Mistag”).

of the Likelihood based classifier (which is 75% efficiency and 4% mistag).

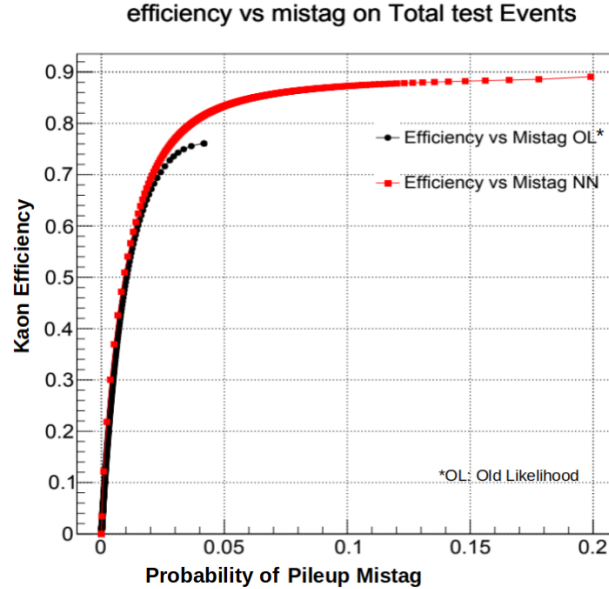


Figure 3.6: By varying the threshold on NN Discriminant (plot in red) we got 5% more Efficiency for current mistag and 40% less Mistag for current efficiency with a Likelihood-based Discriminant (in black).

By repeating the same Efficiency VS Mistag plots for every test run (2017A runs that were not used for the training), we could draw a stability margin on the predictions which are apparent in Figure 3.7. There we notice the small margin band between the worst (in blue) and the best run (in red). These two plots show the NN discriminant stability in performance through the runs which both exceeded in performance metric that of the classical Likelihood-based one even on the worst run.

The above-mentioned *result* we finally got required a lot of empirical experimentation to tune all the parameters of NN but most importantly Feature Engineering. What is meant by that is a lot of physics intuition was spent in the data preparation and model design (i.e., in picking the right variables for pattern recognition). An example of that is in Figure 3.8, we can see clearly how the time and CDA show shape difference between Kaon and Accidental beam. Other variables have subtle differences that needed to be explored as well to push the NN performance to the level we currently reached. A sizable number of trials with similar features plots guided our engineering process in addition to an extensive experience with $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ physics

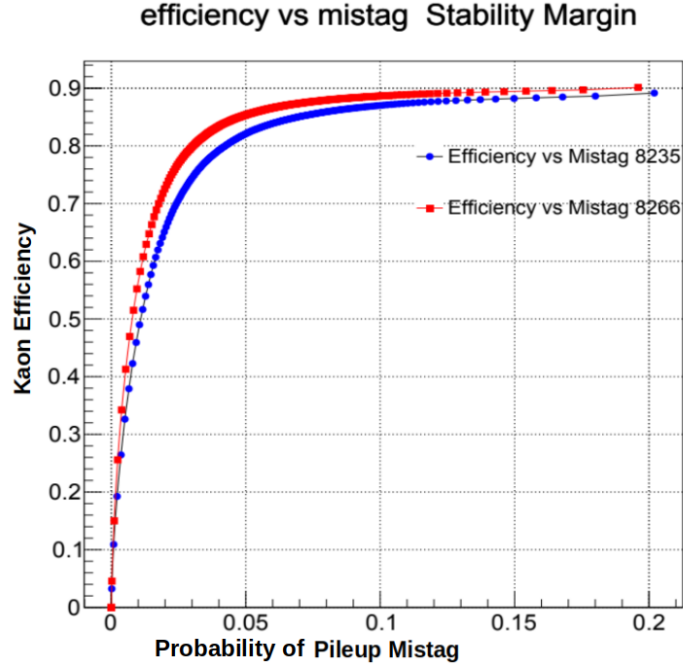


Figure 3.7: Best (red) VS Worst (blue) test Runs

parameter space and how it affects the analysis.

The CDA as an independent feature that proved its supremacy again in the NN as was known before in the Collaboration. In figure 3.9 NN performance dropped by removing the CDA even though we expected it to remain stable. Against intuition the NN could not construct the CDA from primitive raw variables of the individual tracks in initial and final states. It had to be added by hand as an extra raw variable to reach the level of performance we had.

Finally, it should be noted that the NN did learn the topology of our data in parameter space. That is obvious from a simple analysis we did on the features of outputs either being True or False Positive. To ameliorate the predictions and reduce the rate of False positive we started by plotting the instantaneous intensity (Fig. 3.10), then the time differences with GTK in both RICH and KTAG (Figs. 3.11 & 3.12), and the CDA (Fig. 3.13) in both output cases. For the first three variables mentioned we could not see any significant difference other than a bit wider variance and thicker tails in case of false Kaons. However, the CDA showed a crucial difference and a much wider distribution for the False Positives. That re-established once again the importance of CDA as a crucial parameter that will help the discriminant to improve

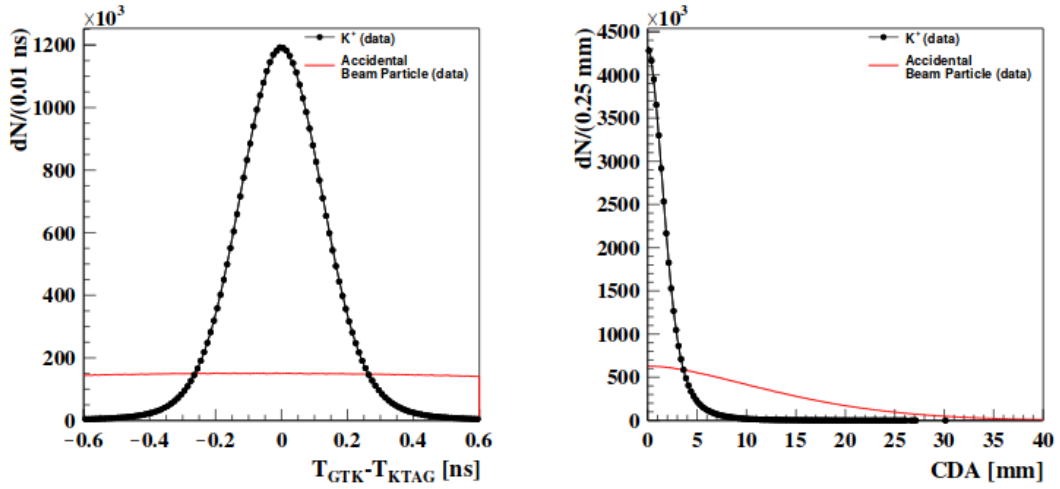


Figure 3.8: Clear pattern recognition on variables such as the time difference and CDA.

its predictions no matter what other variables are chosen or which NN architecture is adopted.

3.3 XAI Analysis

After the success of the improved test model on the $K^+ \rightarrow \pi^+\pi^+\pi^-$ environment, enough motivation was gathered to produce a better one. The main idea was to use state-of-the-art Deep Learning tools to reach the goal of getting higher sensitivity in the background estimation of $\pi\nu\nu$ analysis. The practice here falls in the context of the currently popular approach of XAI in HEP. XAI main target is to step out of the Black Box analysis using ML and especially NNs into what is also called the White Box Analysis. The idea is to make any AI model implemented as clear as possible by adding Physics optimization and educated metrics to gauge the performance of concrete particle physics evidence. A way to do that is by either invent new variables or explore/exploit additional features related to the main analysis.

For the Track-Matching task two main challenges were faced. First, the data was limited to one specific kind where for instance no images existed in the tracking system only coordinates and raw variables and measurements. This prevented us from using Deep Convolution Neural Nets used for image recognition as done in ATLAS tracking and Pileup mitigation [59] [82]. Second, the variables were few and did not give much space for feature engineering. We did succeed in defining a certain hierarchy

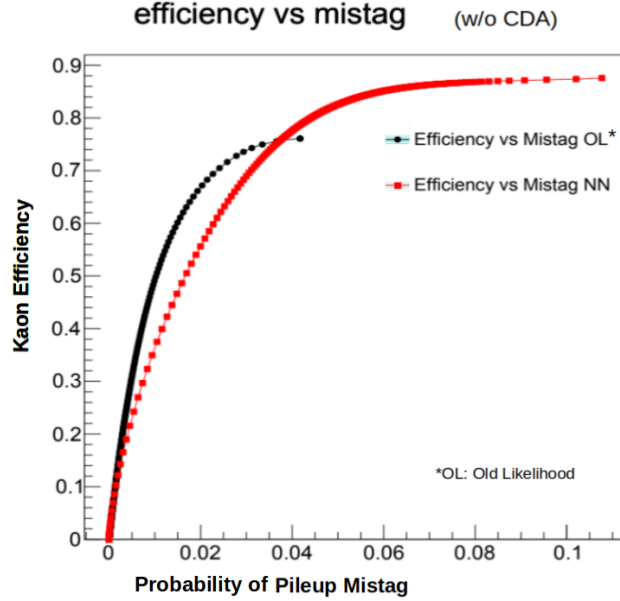


Figure 3.9: Performance plot without CDA.

on features though which will be mentioned later in detail. Having these tools in hand, the attention shifted to improving the model while the target was set to get higher efficiency on predictions in the current test environment first and the final $\pi\nu\nu$ last. So far, a discriminant was created and implemented in the main analysis framework that can discriminate and classify a Kaon event from a fake or Pileup one. To be able to improve the model, an investigation to look where it was failing was done and the parameter space explored. Since the analysis could tolerate some Kaons to be cut down as False Positives. A way to compensate was necessary because the misidentification of Pileups as Kaons was clearly alarming and dangerous. The strategy we wanted to follow is to cut as narrowly as possible on the timing and CDA of tracks in our training samples to allow the NN to learn the features of the trickiest pileup tracks that reside in the same parameter space as real kaon parent tracks. Against what was initially planned, the wider we cut on the timing and CDA in training samples the better discriminating efficiency we obtained in general. The reason for that is that pileup tracks within a certain cut window would mix up completely in features with real kaon tracks and the NN would loose track of them and its ability to learn and discriminate.

To improve the algorithm, we followed two directions: *redefinition* of the training sample, and *upgrade* of the discriminant variables.

In addition, to score the performance of the upgraded discriminant we moved

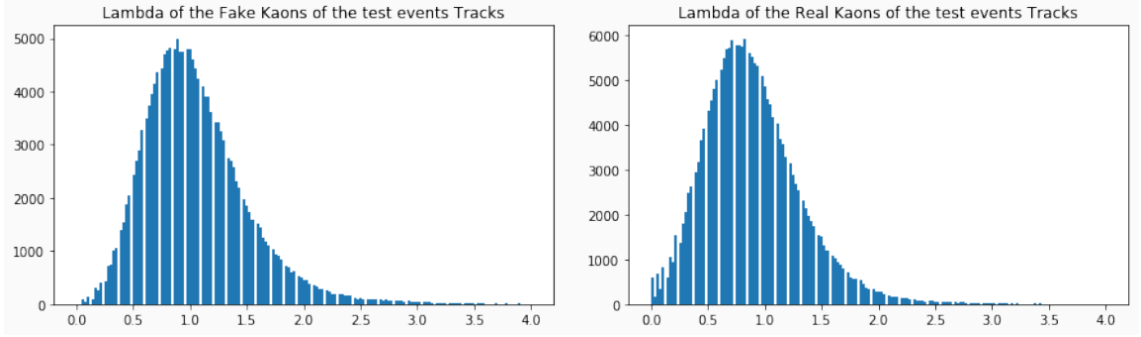


Figure 3.10: Lambda of Fake VS Real Kaons predicted from test data.

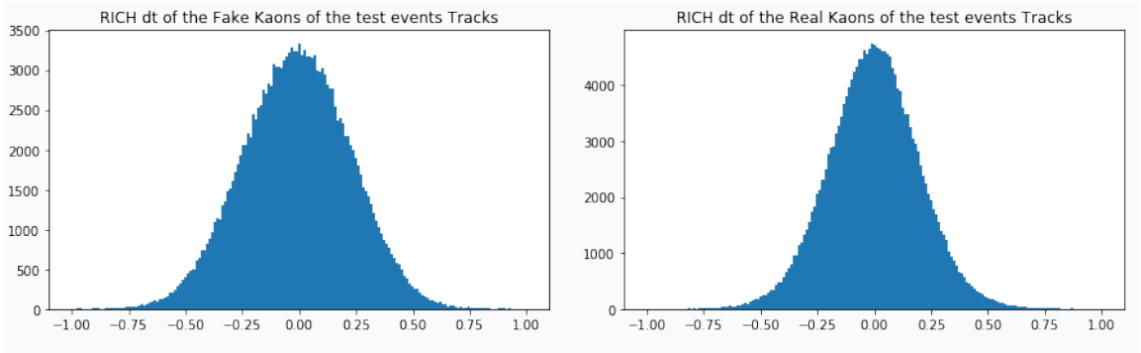


Figure 3.11: time difference between RICH and GTK of Fake VS Real Kaons predicted from test data.

from the $K^+ \rightarrow \pi^+\pi^+\pi^-$ to the $\pi\nu\nu$ analysis. As a figure of merit, we used the $K^+ \rightarrow \pi^+\pi^0$ background leftover after the full $\pi\nu\nu$ analysis versus the number of normalization events. Two track matching metrics were adopted in what follows, the Tails and the Fractional Acceptance Variation (FAV). The $K^+ \rightarrow \pi^+\pi^0$ background depends on the probability of $K-\pi$ wrong association through the fraction of the tails of the reconstructed m_{miss}^2 entering signal regions as shown in Figure 3.14. Therefore its measurement gives quantitative indication of the level of $K-\pi$ misidentification probability.

The number of normalization events are a direct measurement of the effect of $K-\pi$ association on the signal acceptance because the same $K-\pi$ matching algorithm is used to select signal and normalization events. FAV is the relative difference between the number of such normalization events selected in the standard analysis and the ones selected using the NN matching algorithm.

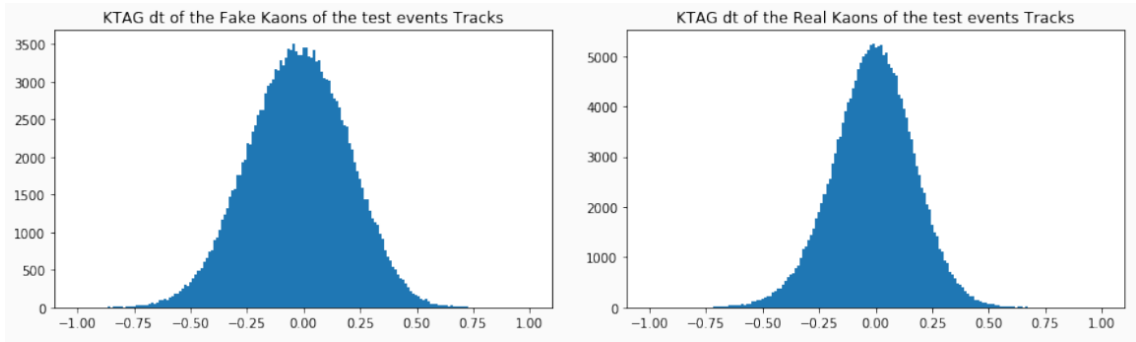


Figure 3.12: time difference between KTAG and GTK of Fake VS Real Kaons predicted from test data.

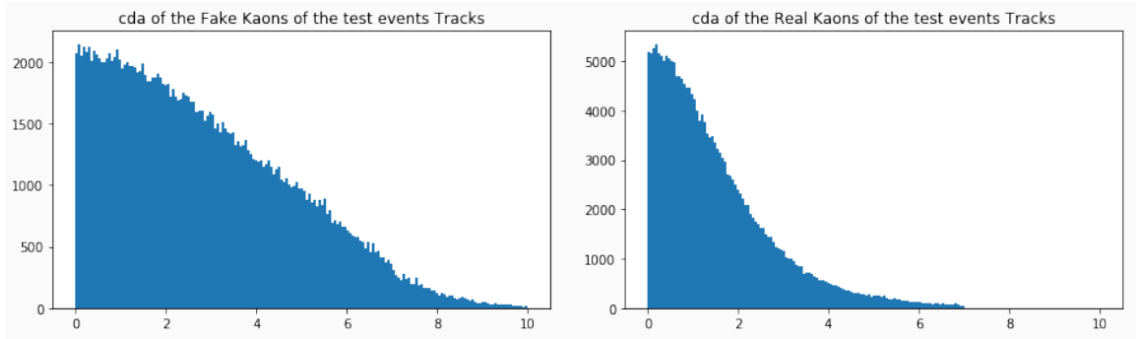


Figure 3.13: CDA of Fake vs. Real Kaons predicted from test data.

3.3.1 Redefinition of the training sample

An understanding of the nature of sampled data clarified that selected events in the time window of NA62 are not singular tracks. This is confusing to the discriminant which deals with tracks one at a time. Since in this case augmenting the training data in mere volume was not enough so work was done to enrich it in quality as well. What is meant here is that a new kind of background was prepared and added, in a hope that a new more complicated training might widen the perspective of the classifier.

The Kaon track is carefully selected on Chi2 fit criteria as a pre-filter called “Chi2Condition” from the bunch of Pileups that infiltrate instantaneously (as per our time window of 15 *nsec*). Removing this filter allowed us to store a sample of tracks that are Pileup mixed with the kaon beam tracks which was added to the training as a new kind of background different than the one we used before with pure Pileup tracks. The aim was a better understanding of the Pileup topology in general

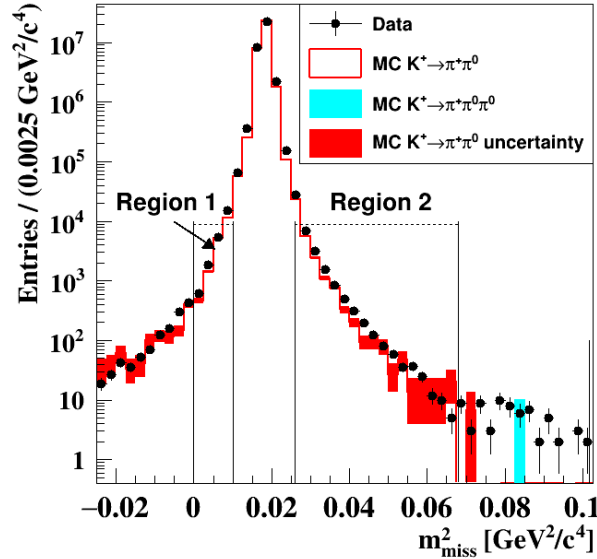


Figure 3.14: The final fraction of $K^+ \rightarrow \pi^+\pi^0$ and other similar events entering signal regions.

by dumping all unwanted kaon tracks in the background sample as well. This strategy is followed in training sample selection based on the hypothesis that all kaon beam tracks that didn't meet the Chi2Condition would be considered as background similar to Pileup.

After training on this new data sample, unfortunately, some efficiency on Pileup predictions decreased. Our model was biasing the predictions towards this new background which lowered efficiency in general. In Figure 3.15 we notice the comparison in prediction performance plots. In increasing order, in gray first we show the one belonging to the classical Likelihood based discriminant followed in blue by the pileup only background predictions with the NN trained from the redefined data which was a bit lower than the one (in red) trained with the usual data before redefinition. This result led to the conclusion that contrary to what was thought at first, this additional background was confusing the NN instead of helping it grasp more corners in parameter space. That was due to biasing the model on an artificial background that is different to the one existing in the physical analysis.

After making sure it is the only trick in our pocket for the data sampling process, we were later satisfied with our first choice of pure Pileup background and worked on

a volume-wise improved data sample production of the classical kind to be adopted for the final training of the model. Hence, a lot of tuning and adjustments still needed to take place in upgrading the model to ameliorate the prediction accuracy of the NN discriminant.

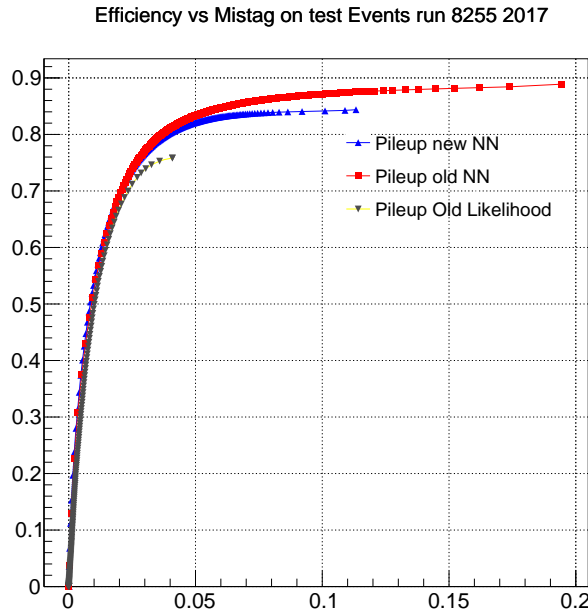


Figure 3.15: Efficiency comparisons on one test run 8255 of 2017A data. Performance plot of NN predictions (in red) before and (in blue) after data redefinition. (In grey) The performance plot for classic Discriminant.

3.3.2 Upgrade of the discriminant

It is well known that working with NNs is a black box mechanism, where the noise it does not necessarily give us hints on its working processes. Being that the case, a lot of intuitive trials were proposed from the start of this project since there is no obvious way to be conclusive and affirmative on how the NN will adapt with the data in hand and what outcome will result from it. So, to upgrade the Discriminant, we worked on three distinct aspects: the *architecture*, the *primary cuts* on the training data and *feature engineering*.

Architecture-wise when the work started with redefined data and before completing the checks on the performances, an investigation was carried out to check

if events with different number of additional tracks might be of different topological nature. However, the limitation in the choice of features motivated the invention of new ones (this latter is in feature engineering part) but the general sense was that by adjusting the design of the classification algorithm, it might become able to understand these subtle differences if they ever exist. Five separate and different discriminant classes¹⁰ are added to the model. One for events that are in time with a single additional track in GTK, another for events with two, and yet others with three, four and five, respectively. Five different samples of data that correspond to the 5 mentioned types were prepared and their related NN Discriminants trained. In addition to that, the framework was adjusted to distribute events according to their nature to be evaluated and classified by the five different NN Discriminants while the first one used before still is present to identify events with singular tracks or 0 additional tracks in the GTK. The first problem from the redefined data also reappeared here to leave its imprints on the much lesser and exclusive data samples. It was clearly noticed also that prediction performance was exceptionally low on exclusive kind of events with additional tracks due to low statistics. Proper data mining needed much more attention and time to filter enough data of these kinds to prevent under-fitting with exclusive events. Being that the case, this attempt was paused especially that at the time we were working on it, another issue was present in the redefined data (the “over-fitting” towards the new background mentioned previously in 3.3.1). Prioritizing the most essential issues, this adjustment to the model had to be paused to resume if enough time and computing power become available to excavate all the data samples needed all over again without the additional background that over-fitted the training. Then, the project moved forward with egalitarian democracy and treated all events on equal footing by using again the data with Chi2 filter in GTK tracks associated selection. At that point, the attention was given to the inputs’ features engineering with the hope to gain better than with adjusting the fundamentals of the model. However, before delving into the details of features engineering it is good to mention how the primary cuts worked on the training data against expectations as well.

Primary cuts are wide cuts applied firsthand on the data domain where it is confirmed there are no correlations in the features’ physics. These are done to avoid training the NN on irrelevant or obvious events and consuming unnecessary computing power. These cuts are imposed on the four key features which are the three-time differences between the reference detector (GTK time) and the other three, and the CDA. An average first cut of $\pm 1ns$ and $15mm$ was used on the training set. It was expected at first that a tightened cut around confusion region might produce better

¹⁰The events come most frequently as “single-track” but some exceptional events are distributed between “double-tracks” mostly and up to “five-tracks” quite rarely.

training. The intuition was that around confusion area where the NN predictions are failing, the features are remarkably similar between Kaons and pileup. Hence, if we select more events from that region in the training, the NN might become able to better generalize and give more accurate results. However, once again against expectations, the wider cuts were much better and the NN discriminant resulting from them performed better classifications.

To gauge the performance, a root routine was written that can statistically evaluate the fraction of $K^+ \rightarrow \pi^+\pi^0$ events entering signal regions after $\pi\nu\nu$ selection. This fraction is shortly called “Tails” in the following. The tails were measured as a function of the relative difference between the number of normalization events selected in the standard analysis and the same events selected using the new $K - \pi$ matching algorithm, denoted with “Fractional $\pi\nu\nu$ acceptance variation”. A scan was then performed varying the threshold on the discriminant and the performance compared to the standard analysis.

Figure 3.16, left, shows the results using a discriminant variable built with tight primary cuts corresponding to $\pm 0.6ns$ and $4mm$. As a reference, the standard analysis corresponds to a value of 0 of the fractional $\pi\nu\nu$ acceptance variation, and the tails in that analysis were measured about 0.00135 for the 2017 dataset used in this study. The figure shows that the NN discriminant trained with tight cuts provided a marginal improvement with respect to the maximum likelihood discriminant.

However, Figure 3.16 right shows the results using a discriminant variable built with wide primary cuts, corresponding to $\pm 2ns$ and $30mm$. In this case the improvement of performance starts to be significant, with about 10% lower tails than the standard ones for the same signal acceptance as in the standard analysis. This reflects in a 10% lower $K^+ \rightarrow \pi^+\pi^0$ background for the same signal acceptance, because the $K^+ \rightarrow \pi^+\pi^0$ background depends linearly on the tail fraction.

This demonstrates that the predictions of the NN trained with the wider cut data proved more accurate than the other. This behavior was traced back to the training preferences of the NN itself. It also made sense that with more pileup events, existing in the training sample (for wider cuts), the generalization process became easier and especially on the pileup events. To push the performance even further feature engineering was exploited and described next.

Features Engineering is a part of Machine Learning Data Science that deals with understanding and developing the physical features of the training data. It is a very wide topic that starts with the simplest task of classifying and understanding the discriminating importance of physical features to the much more complicated design and invention of new ones. The classification is done through understanding a certain hierarchy of the features which the path was followed through some specific

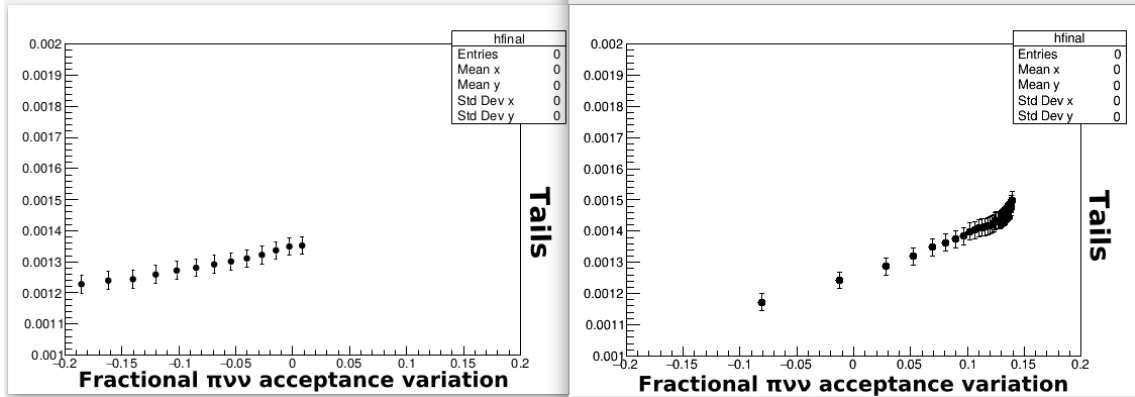


Figure 3.16: Tails versus the fractional $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ acceptance variation for different sets of cuts applied on the Single NN discriminant for the tight (left) and the wide (right) primary cuts.

packages available on GitHub¹¹. While in the features development a reliance on physical knowledge helped to select and produce features/variables that boosted results. To be able to understand the hierarchy of the current features, an Open-source library was used that works with Sklearn¹², with a specific class from ELI5 called “PermutationImportance”. This class undergoes a classic sensitivity analysis using permutation and weighting of features importance [109], [83]. The basic idea is to permute the inputs of one of the features, pass it through the model and see if the model’s predictive power would change. In case all stayed the same, then the variable under test might not be of much importance. Feature importance can be gauged in the form of weights which are computed using a metric called “Mean Decrease of the Accuracy score (MDA)”¹³ while the tested variable/feature gets permuted in the input set (i.e. becomes noise). Table 3.1 shows the hierarchy in the features where the CDA, dtKTAG and Chi2 came as top three out of 23. An insightful reasoning should be presented to try to make some physics sense out of the position of most essential variables. Starting with the most obvious and well known, an expected result for CDA and dtKTAG to take the first two positions because these are the two variables of the highest discriminating power in the Standard Analysis as shown in figure 3.8. The Chi2 of the Kaon in the initial state came third due to it being an indicator of the quality of reconstructed track which is quite significant since the Pileup presence does not only depend on accidental activity but also from fake combinatorics of tracks

¹¹<https://eli5.readthedocs.io/en/latest/autodocs/sklearn.html>

¹²The code needed to be adapted with Sklearn as well since it runs in Keras with Tensorflow backend, using the keras wrappers package <https://keras.io/scikit-learn-api/>

¹³For general equations check an instance in [97]

MDA Weights	Features
55%	CDA
18%	dtKTAG
13%	Chi2
8.4%	fTrackTimeS1
5.7%	dtRICH
4%	fTrackTimeS2
3.1%	ftKTAG
1.8%	dtCHOD
0.9%	fTrackTimeS0
0.7%	fLambda
0.6%	ftRICH
0.6%	ftGTK
0.25%	ftCHOD
0.1%	nHits
0.09%	fPositionX
0.06%	fPBMX
0.06%	fPBMX
0.03%	fPositionY

Table 3.1: List of Feature Importance weights from the “permutation importance” MDA sensitivity analysis.

associating hits of GTK. Moreover, the timing of the GTK second station came fourth most probably due to its geometric layout. GTK2 is aligned with GTK1 and GTK3 only on the horizontal plane, while demoted about 6cm below GTK1 and GTK3 in the vertical plane to measure the Kaon’s momentum. This peculiar position would allow GTK2 to be exposed to a different kind of accidentals and its timing variable to become twice as sensitive to pileup. Also, the difference in timing in the RICH is quite important because it is related to particles downstream and in the final state but less sensitive to accidentals and pileup. Moreover, time of KTAG is related to reconstructed Kaons in the initial state and CHOD’s timing would clearly be more sensitive to the charged Pions in the final state. However, the instantaneous Intensity variable fLambda was counted with the variables of lowest values and it is due to it being fully correlated with the timing and hits of GTK. On the other hand, 5 of them (Chi2Pion, fSXBM, fSYBM, fSlopeX and fSlopeY which are not mentioned in table 3.1) appeared to be useless which led to them being removed from the final model, leaving a final input of the other 18 left.

Finally, a feature called 2-D NN discriminant $\Delta(D_{best} - D_i)$ was built. 2-D NN discriminant computes the difference in Single NN discriminant outputs between the highest score (“best”) and second highest score (“i”) for tracks associated with the

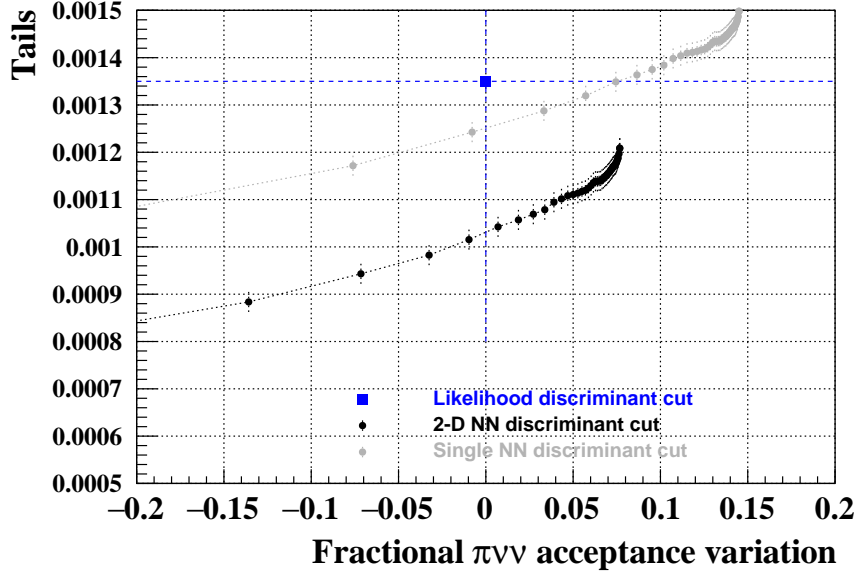


Figure 3.17: Tails versus the fractional $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ acceptance variation for different sets of cuts applied on the 2-D NN discriminant (black dots). The performance of the Single NN discriminant cut (grey dots) and of the standard likelihood cuts (blue square) are also shown for comparison. Grey dots are the same as those of figure 3.16 (right). Lines connecting dots and the blue straight lines are for visualization only.

same Pion in the final state. This variable lowered the exploitation of the additional tracks present at the same events. The fact is that when 2-D NN discriminant is sufficiently low then the two tracks of near equally high probability might be confusing the matching algorithm. The rule followed in such case is imposing a cut on events with lower than a certain value for 2-D NN discriminant, which ensures the matching is done as purely as possible on events with single Kaon parent track in initial state only. Figure 3.17 shows the clear improvement in the performance of the algorithm obtained using 2-D NN discriminant with both D_{best} and $\Delta(D_{best} - D_i)$ (black dots) with respect to Single NN discriminant using only D_{best} (grey dots). The black dots are obtained fixing a cut $\Delta(D_{best} - D_i) > 0.25$, and varying D_{best} over the entire range. The grey dots correspond to Figure 3.16. The standard likelihood-based discriminant performance is shown in the same plot for comparison. This result obtained is giving hints that pileup can pose as fake kaon parent tracks, confuse the matching and increase the tail values. The cut strategy with 2D-NN discriminant helps mitigate significantly the effect of accidentals on tails overlapping signal regions and would serve big time in future higher intensity runs when pileup effects are expected to

exponentially increase.

The working point was chosen at $FAV=0$, which is a fair point to compare track matching performance with tails v/s FAV because the two kinds of discriminant would be selecting the same number of normalisation events. For a cut applied on 2-D NN discriminant at $\Delta(D_{best} - D_i) > 0.25$ a result of 35% lower $\pi^+\pi^0$ background was obtained for the same signal efficiency, or, equivalently, a 7% larger signal efficiency for the same level of $\pi^+\pi^0$ background which is an exceptional performance of XAI methods to be clearly noted.

Chapter 4

NNODA for LKr Calorimeter

4.1 Physics Review

Photon Veto is a relevant source of signal inefficiency affecting the 2016-17-18 $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ analysis. This is due to signal events being rejected by vetoing on random activity instead of photons or, more generally, of additional particles from K^+ decays (please check 1.2). Through this chapter such a loss of signal events is referred to as “Random Veto” (RV). The fraction of signal loss due to RV also strongly depends on beam intensity, as shown in the $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ analysis [1]. Therefore, the RV seriously limits the performance of the analysis at increasing intensity. One of the most relevant contribution to RV comes from the definition of “in-time” photon used in the Liquid Krypton calorimeter (LKr).

The general classical cutting conditions are a time window of $|t_{cluster} - t_{rich}| < 50\sigma$ applied to $E_{cluster} > 15 \text{ GeV}/c$ and distance $> 100\text{mm}$ from the charged particle’s impact point. Here t_{rich} is time of the downstream charged particle, $t_{cluster}$ is time of the cluster and σ is LKr cluster time resolution which is $\lesssim 0.7 \text{ ns}$ and slightly dependent on energy. The standard analysis has shown that any attempts to shrink the time window of the cut to $\pm 30\sigma$ increased $\pi^+ \pi^0$ background events by a factor of 2. Two main reasons for this issue, one is physics related and the other is software technicality. Starting with the latter, a bug in the time sharing of the LKr cluster algorithm was found and fixed. As for the physics part, spatial merging of clusters within and outside the time window spoils the time measurement of the clusters themselves. Here rises the need for an indicator of time since only spatial reference is not enough to identify clusters. The work described here makes use of data from $\pi \nu \nu$ analysis reconstructed with the standard LKr reconstruction.

According to LKr Standard Algorithm (SA) for cluster reconstruction, cluster time

is the time of seed cell, which is the most energetic cell among those forming the cluster while the time of other cells is almost irrelevant. Improved corrections for time sharing between adjacent clusters are being implemented to distinguish in time clusters with barycenter distant down to 10 cm. The width of the LKr signal of each cell is 60 ns FWHM and the pulse is sampled with FADC working at 40 MHz. Therefore, pulses closer than 30 ns apart cannot be distinguished. So far, the SA has no present double-pulse separation. Attempts to resolve double pulses within 60 ns are under study with the goal to improve the identification of spatially merging clusters whose seeds share the same cell. The impact on RV of these improvements is currently under study, and first results look promising. Nevertheless, the hardware limitation to 30 ns will remain. The method of in-time photon identification described next intends to be independent from the LKr reconstruction and to overcome the hardware limitation. The idea is to rely on the timing of the non-overlapping cells to separate clusters of different time, even below the hardware limitation of 30 ns. This is achieved through a visual analysis of the time distribution of the energy measured across the cells event by event.

The starting point is the study of LKr calorimetric data of peculiar $\pi^+\pi^0$ decays rejected by the $\pi\nu\nu$ analysis because a SA cluster with energy greater than 15 GeV, not associated to the π^+ is found either between $(-50, -30)\sigma$ or $(+30, +50)\sigma$ off the π^+ time. In addition, these events have no signal associated with LAV, IRC and SAC, according to the photon rejection criteria of the $\pi\nu\nu$ analysis. The first step is to understand why no cluster closer than 30σ is reconstructed in the LKr despite these events being $\pi^+\pi^0$ decays. To this extent, two ways are presented to visualize calorimetric data in the LKr graphically for each of these events, one in terms of amount of energy and another in terms of the time of energy deposition measured in each cell. Figure 4.1 (top middle) shows an event on an energy scale where the color on the palette represents energy deposits in cells. The energy associated to the π^+ in this event (inside of the red circle) is consistent with a MIP. Nevertheless, energy releases consistent with hadronic showers can also be associated with the π^+ . Through this chapter we will refer to a SA cluster matching the track of the reconstructed charged particle as “track cluster”. Any attempt to use a $|t_{cluster} - t_{rich}| < 30\sigma$ time window would not be sufficient to reject the event of the figure. This can be understood by looking at figure 4.1 (bottom middle) which shows the same event on a time-based representation. The right-hand side of the figure displays a zoom-in on clusters not associated with the track, both in energy-based and time-based colors, respectively. A comparison between the two cluster images can be easily presented. There, the shape of the clusters remained the same in both cases but the colors are different. On the (top) energy-based visualization the reddish color in the middle of clusters shows the deposit of energy being the highest in the central cells. On the (bottom)

time-based visualization some distributions of colors are noticed, according to the time scale adopted all greenish cells are of the same timing as the track cluster.

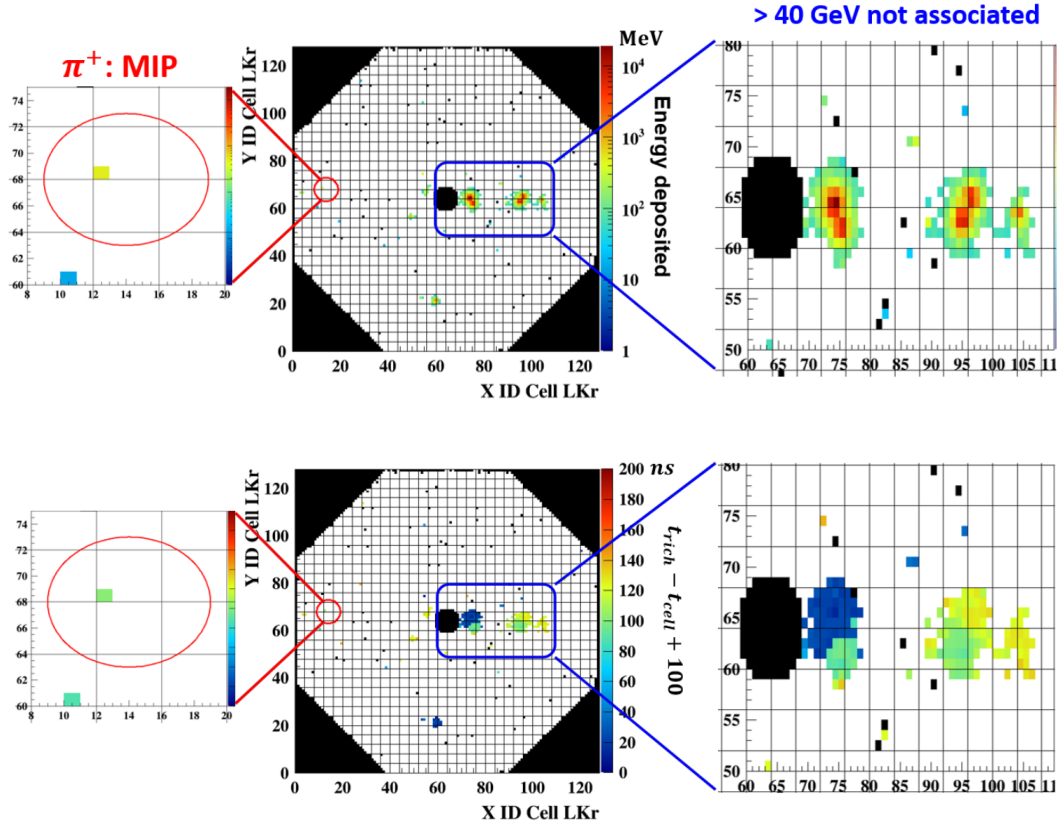


Figure 4.1: LKr's Calorimetric Visual Data: (top) energy-based visualization, (bottom) time-based visualization. Black hole showing in middle of picture is for the beam pipe of the experiment.

The algorithm described in what follows exploits the time visualization, relying on a cell time-to-color transformation to perform the clustering. Physics clusters are defined as color islands, and an *in-time* cluster is an island of the same color as that of the SA (Track cluster) cluster associated to the reconstructed π^+ track. The core of the algorithm is to use a Deep Learning Object Detection trained model at the heart of the Neural Network Object Detection Approach (NNODA) to detect in-time clusters that are not spatially associated with the MIP and ultimately help improve NA62's Photon Veto by lowering inefficiency and improving RV especially by constraining the high energy photons in-time Clusters coming from π^0 decay (i.e., $E_\gamma > 15$ GeV see Figure 4.2). Object Detection algorithms (OD) are built on top

of Convolutional Neural Networks (CNN). While CNNs extract geometrical features by running convolution kernels over the input space (i.e., images in pixel-tensors form), ODs go the extra mile of localizing the objects of interest, select and classify them individually. CNNs have been successfully applied over the past decade to classification tasks commonly encountered in particle physics [31] [15] [16] and recently in NA62's particle ID [46] and the renown work of P.T. Komiske *et al.*, 2018 [82] for Pileup mitigation from Jet images. A first visual application of an OD approach in NA62 experiment is presented to solve the problem of random activities in NA62's LKr RV. NNODA's prime objective is to work as an independent virtual AI inspired bubble chamber technique in parallel with any cluster reconstruction.

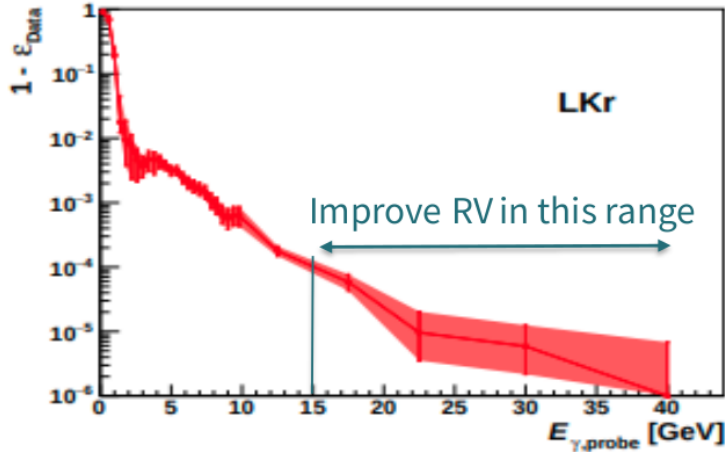


Figure 4.2: This figure taken from [6] shows the range of PV inefficiency in LKr that is prioritized for improvement.

The main goal of this innovative visual technique is to overcome the time window challenge faced with SA. A balance is sought by making the PV time window soft enough to prevent Random Activity from posing as fake background and avoid spatial merging of in-time and out-of-time clusters by efficiently differentiating/filtering their source events. On the other hand, NNODA allows the time window to be simultaneously tight enough to control RV and avoid cutting too many events of interest which allows recovery from signal acceptance loss.

4.2 Data Preparation

To train any machine learning model, a dataset should be prepared accordingly with the required annotations and model-related format. Possible methods of acquiring non-research-specific datasets include finding a preexisting annotated database online of the desired objects. However, research-specific data would be harder to get. It would be produced using a simulated model of the desired objects (e.g., CAD model) and prepare an annotated dataset in a related virtual environment. Yet another method would be to select and/or collect data manually and use an online labelling and annotating tool, the one used for most of this project is Intel's Computer Vision and Annotation Tool (CVAT). The ideal one among them is to find and use an open source dataset due to time effectiveness but these are generally available for common real-world scenarios. The calorimetric particle physics project for LKr Calorimeter requires a very specific dataset subject-related and had to be carefully prepared and annotated. Manual annotation and labelling of a collected/created dataset are time consuming tasks and more so when the desired dataset is large. A lot of attention is spent in keeping the dataset balanced in negative examples and in multi-class distributions. A non-equivalent dataset can have a massive impact on the training model from making it unable to learn features to extremely biasing its predictions. That might result in a low mAP¹ and might lead to over-training as well. Additionally, a well-balanced dataset is directly linked, empirically speaking, to lowering the number of unidentified/unselected objects of interest in images which in the case of ODs are the most dangerous errors.

CVAT tool is used for online video and image annotation for computer vision purposes. Instructions on how to install CVAT can be found on their official GitHub page². Within this tool, annotations can be done in several ways depending on the purpose. For instance, in Object Detectors (e.g., YOLO etc...), boxed annotations are needed. Boundary boxes annotation would be the method followed in both projects for object detection and computer vision.

The clusters dataset is a collection of calorimetric images of energy deposited, each of them representing a specific data event. A data event is a superposition of physical events staggered in time. These would serve as a basic dataset (**NA62_LKrCV**) to train computer vision systems to detect objects belonging to the same physics event once displayed. Additionally, as the general practice is and according to cross-validation rule, 10% of the dataset would be set aside for validation. Since the images represent clusters of different particles interacting in the calorimeter, it is only logical

¹See later section 4.4.1 for details on the mAP metric

²<https://github.com/openvinotoolkit/cvat/blob/develop/cvat/apps/documentation/installation.md>

for objects of interest to be clusters in nature. Two classes the model should learn to differentiate and detect, Single and Merged clusters. The image features that would help discriminate the two types of clusters are quite straight forward. Because the pixel color refers to a very crucial physics criteria (“relative time of the hits”), it served as the main reference in the annotation process.

Now to define the image of a physics cluster, the following selection criteria were compiled in the following:

- Fix the color of the time of the clusters associated to the Pion track time to green which corresponds to 100 ns absolute.
- *In-time* cells are all cells within ± 10 ns of the Pion time. Since within the time window the gradients of the colors are kept, and all the greenish shades can be noticed in this case.
- *Out-of-time* cells are all cells outside the ± 10 ns time window from Pion reference time. However, the time of these is pushed to the extremities of exactly ± 50 ns with respect to the Pion’s timing. Apparently, we don’t notice color gradients any longer in this other case. What we do notice is the following:
 - Either all *blue* clusters (-50 ns off the MIP) or all *yellow* ($+50$ ns off the MIP).
 - All out-of-time cells on the same side of the MIP time should otherwise have the same color
- A Physics “cluster” is defined by at least 3 adjacent same-color cells/pixels.

Using CVAT online annotation tool³, the images were annotated where a box was added around clusters of interest and the right class label was added, either Single- or Merged-Cluster (See figure 4.3). First of all, for a patch of pixels to qualify as a cluster of interest it should be formed of 3 or more connected greenish⁴ pixels in a patch. The Green color is a general indication to the existence of an in-time cluster. Now from the qualified clusters, anyone that mostly contained greenish pixels (not more than 2 attached non-greenish pixels) is labelled as Single cluster. While any cluster, made mostly of non-greenish pixels and contained 3 or more attached greenish ones, is labelled as a Merged cluster. A merged cluster represents the spatial overlap of clusters coming from particles produced by uncorrelated physics events. On the other hand, all the other clusters which are mainly made of non-greenish colors were left

³<https://github.com/openvinotoolkit/cvat/>

⁴As the time scale fluctuates the greenish pixels would not all be of the same shade of green

un-annotated to serve as negative examples⁵ (Figure 4.3).

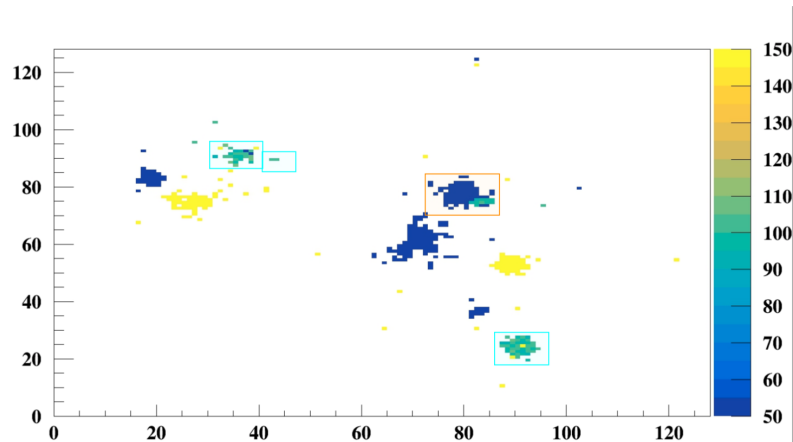


Figure 4.3: This figure shows an instance where clusters are labelled and box-style annotated using CVAT tool. Single clusters are in light-blue colored boxes while Merged ones in brown-orange and negative clusters are left without annotation.

To train the model reliably all the clusters of interest must be within the boxes in all images. The clusters dataset yielded a total of 1230 histograms for training and 155 to validate. These were taken from a sample of 2017 data 10 runs. This is sufficient for basic training purposes, nevertheless the time required to extract and annotate manually such histograms limited the ability to expand further the dataset. Since image variety and range are most essential factors for an ideal dataset, an extra effort has been spent selecting best runs that held a wide variety of highest quality events. Additional attention was given to picking an unbiased sample that includes a vast variety of clusters of various kinds. Moreover, the relative training sample quantity drawback was mitigated by applying data augmentation while training in all different models. This technical option would artificially add an extra variety to the dataset by changing the pose and texture of input images. Each state-of-the-art vision algorithm has its own related properties that will be discussed later. It should be mentioned here that the advantage CVAT presents is that, once the annotation of a dataset is completed; it can be exported in a multitude of formats. Each format would work with one or more different algorithms.

“NA62_LKrCV” dataset is sub-sampled into three parts for three different purposes

⁵Negative examples are left without annotation in the dataset to make it more robust and less prone to False Positive detections

which are training, validation and also testing. Even though these parts contain practically similar data object-wise, however they are quite different from the physics point-of-view and they serve a variety of purposes. The first two are used during training but they are completely separated, one to train the algorithm and another to test its ability to generalize while training. For these, sample data was picked from 2017A and subdivided (90% for training and 10% for validation). All images selected for these represented PNN events in $\pi^+\pi^0$ region rejected by the PNN analysis only because of the presence of clusters not associated to the π^+ reconstructed at $(-50, -30)\sigma$ or $(+30, +50)\sigma$ off the π^+ time. In addition to that, no photons are associated with them in LAV, IRC nor SAC. On the other hand, for testing a more physics specific events were selected from 2017B data and called “pathological events” to refer to their physics significance. The reason for this name comes from the careful selection made at these events to find any improvements on selection that separates NNODA from SA. A collection of 1438 events of the above-mentioned ones were picked to test $\pi^+\pi^0$ rejection of NNODA. As to test μ^+ events rejection of NNODA (or RV), $K_{\mu 2}$ sample of events were picked, these are rejected by SA because of the presence of clusters not associated to the μ^+ reconstructed at $(-50, -30)\sigma$ or $(+30, +50)\sigma$ of the μ^+ time. A collection of 3050 of these events are tested in an attempt to see how many of them can be saved by NNODA in comparison with SA.

4.3 Training

The training was performed on a hand-picked dataset, that is general enough to include most extreme cases and some extremely rare calorimetric activity. Now the main objective of training the four hybrid OD algorithms mentioned in Chapter 2 is that they learn a mapping function that would not only map the input space into a single classification prediction (i.e., the default case of CNNs) but to a tensor with multi-dimensional prediction components. The four of them share the same objective, nevertheless, the mapping mechanism is slightly different on the mathematical level which will be presented next. The input space structure would remain the same, every image is RGB pixel-encoded in a multi-dimensional input tensor (i.e, in our case $640 \times 640 \times 3$ for the resolution multiplied by the number of channels RGB). Each OD has its own mapping style.

4.3.1 YOLO model

For the output space, since the number of classes are two (i.e, Single or Merged clusters) the number of components for one predicted bounding box, in each “*central*” cell, are: 4 coordinates, 2 to flag the class labels and an additional 1 to indicate the

existence of the favorable class if two objects happened to be in the same central cell. That would leave every output vector p_i with $(B \times 7)$ components, because every central cell has the ability to be associated with B bounding boxes, B being the maximum number of boxes to be assigned around each central cell. For instance, if two different objects needs to be detected then two boxes with proper shapes would be assigned if they happen to have same central cell (i.e., $B = 2$). Finally, the output becomes a tensor P when we take into consideration the whole grid resolution. If the number of grid cells is S^2 then the tensors dimensions become $S \times S \times B \times 7$ (e.g., the most common case adopted for ‘‘Pascal’’ format [23] is $S = 7$). Then YOLO learns a map from the ground-truth annotated tensors ($P_{ij} = (p_{io}, \dots, p_{iS^2})$) to the output predicted probability tensors ($\hat{P}_{ij} = (\hat{p}_{io}, \dots, \hat{p}_{iS^2})$, where i is the index of the input image in the training set and j is the index of the grid cell) using the following loss function:

$$J(p, \hat{p}) = 5\lambda_{obj} \times L(p, \hat{p}) + \lambda_{obj} \times F(p, \hat{p}) + 0.5\lambda_{noobj} \times F(p, \hat{p}) \quad (4.1)$$

Equation 4.1 would calibrate the training process. To avoid over-fitting⁶ the data, $L(p, \hat{p})$ sum-squared error penalty is weighted 5 times thus increasing the loss and only considered if the object is found within the grid cell (i.e., $\lambda_{obj} = 1$ and $\lambda_{noobj} = 0$). On the other hand, to avoid under-fitting⁷ a penalty is added to $F(p, \hat{p})$ weighted by a 0.5 and only considered when the object is not found within the grid cell (i.e., $\lambda_{obj} = 0$ and $\lambda_{noobj} = 1$). Both localization part $L(p, \hat{p})$ and fidelity part $F(p, \hat{p})$ of the loss function use the ‘‘sum-squared error’’ in the following manner:

$$L(p, \hat{p}) = \sum_{k=0}^7 \sum_{j=0}^{S^2} [(x_{kj} - \hat{x}_{kj})^2] \quad (4.2)$$

Where x_{kj} is any location related coordinate k^{th} -component of vector p at the j^{th} cell and there are 8 of them (4 for each bounding box) and

$$F(p, \hat{p}) = \sum_{l=0}^5 \sum_{j=0}^{S^2} [(c_{lj} - \hat{c}_{lj})^2] \quad (4.3)$$

Where c_{lj} is any classification flag l^{th} -component of vector p at the j^{th} cell and there are 6 of them (3 for each bounding box).

The loss function $J(p, \hat{p})$ has to be numerically minimized by an iterative process which is some sort of gradient descent.

⁶the specialization of the model to the training set.

⁷the disconnection of the model from the training set.

4.3.2 SSD models

All SSD models as mentioned before (Chapter 2), have two kinds of prediction vectors in the output space $CONF$ and LOC . SSD generator would be producing bounding boxes around found objects on a range of different scales, and only the ones, similar to the ground-truths for a specific class label, are selected. N is the total number of boxes in this selection. So, since the number of classes are two, the number of components for one pair of prediction vectors, related to one such predicted and filtered bounding box, are: 4 components for LOC to indicate the 4 basic coordinates of bounding boxes, and 2 class flags for $CONF$ to indicate the class labels. SSD learns to map from the ground-truth location vector LOC_i to the output predicted location vector \widehat{LOC}_j , where j is the index of the matched bounding box to the i^{th} ground-truth, and the algorithm also produces predictions on the classes in $CONF$. The overall objective loss function is a weighted sum of the localization loss (L_{loc}) and the confidence loss (L_{conf}):

$$J(CONF, LOC, \widehat{LOC}) = \frac{1}{N}(L_{conf}(CONF) + \alpha L_{loc}(LOC, \widehat{LOC})) \quad (4.4)$$

Where α is a weight factor to increase penalty from localization is needed, and if $N = 0$ the loss is set to 0. While $L_{conf}(CONF)$ is simply a Soft-Max function, L_{loc} is a $smooth_{L1}$ [29] that would take the following form:

$$L_{loc}(LOC, \widehat{LOC}) = \sum_{i \neq 0}^N \sum_{m=0}^3 smooth_{L1}(LOC_{im} - \widehat{LOC}_{jm}) \quad (4.5)$$

Where LOC_{im} is any one out of 4 location-wise m^{th} -component of ground-truth vector LOC for the i^{th} matched bounding box. While \widehat{LOC}_{jm} is any one out of 4 location-wise m^{th} -component of prediction vector \widehat{LOC} for the j^{th} selected bounding box.

The loss function $J(CONF, LOC, \widehat{LOC})$ has to be numerically minimized by an iterative process which is some sort of gradient descent. Equation 4.4 shows how to avoid over-fitting the data, $L_{loc}(LOC, \widehat{LOC})$ $smooth_{L1}$ error penalty is weighted α times more than that for $L_{conf}(CONF)$ and increases the loss if the bounding boxes are matching weakly. While on the other hand to avoid under-fitting a penalty is only added from $L_{conf}(CONF)$ when the bounding boxes are unmatched [29].

4.3.3 Faster-RCNN model

There's no need to repeat the mapping procedure here, since SSD is inspired by Faster-RCNN and adopted nearly the same objective loss functions. Only two differences with Faster-RCNN:

1. The produced, selected and matched bounding boxes are the ones generated directly from RPN (see Chapter 2). These are expected to match the ground-truth boxes of input images to reward and reduce the loss function $J(CONF, LOC, \widehat{LOC})$.
2. The $L_{conf}(CONF)$ uses a log loss over two classes (object *vs.* not object) instead of a Soft-Max on all classes [27].

However, these mappings are not of our concern while using the APIs, as long as we pick the right parameters in the configuration files of the models, what would be presented later. The parameters choice is extremely empirical though, and requires experience and thorough understanding of the problem at hand.

4.3.4 Fine Tuning in Transfer Learning

Fine tuning requires both updating the CNN architecture and re-training it to recognize new object classes. It's good to note here that it is fine tuning that would be adopted for the application work considered in this thesis and adopted by all models trained using the APIs.

The final predictive layers of the network are removed first and replaced with freshly initialized ones. Then, the earlier convolutional layers in the network should be frozen⁸ to ensure that any previously learned robust features are not banished. Afterwards, two options are valid for training:

- Train only the FC layer heads.
- Unfreeze some or all of the convolutional layers and train on top with a second round.

The first option is used when the extra dataset is terribly similar in features to the one we trained before. Additionally, it is much faster to retrain only the predictive layers. On the other hand, we might need to use the second option (like in our case) when we have new object classes to be learned by CNN. Even in that case, and it is the most commonly used one, it would be much more practical than to train weights from scratch all over again. Even though it sounds trivial, the only fact is that fine tuning, however, is an advanced technique that requires care and experience. For instance, a very common pitfall is the fast tendency to over-fit a network while fine tuning. One way to overcome this is by training the network using a very small learning rate. This allows the new set of FC (Fully Connected) layers to learn patterns from the

⁸Freezing a layer means preserving the weights fixed at each node of the layer.

previously learned convolutional layers earlier in the network. This process is called allowing the FC layers to “warm up”. It is a delicate process and requires a “network surgery” following the steps mentioned above. Figure 4.4 shows in graphical details how the surgery actually works.

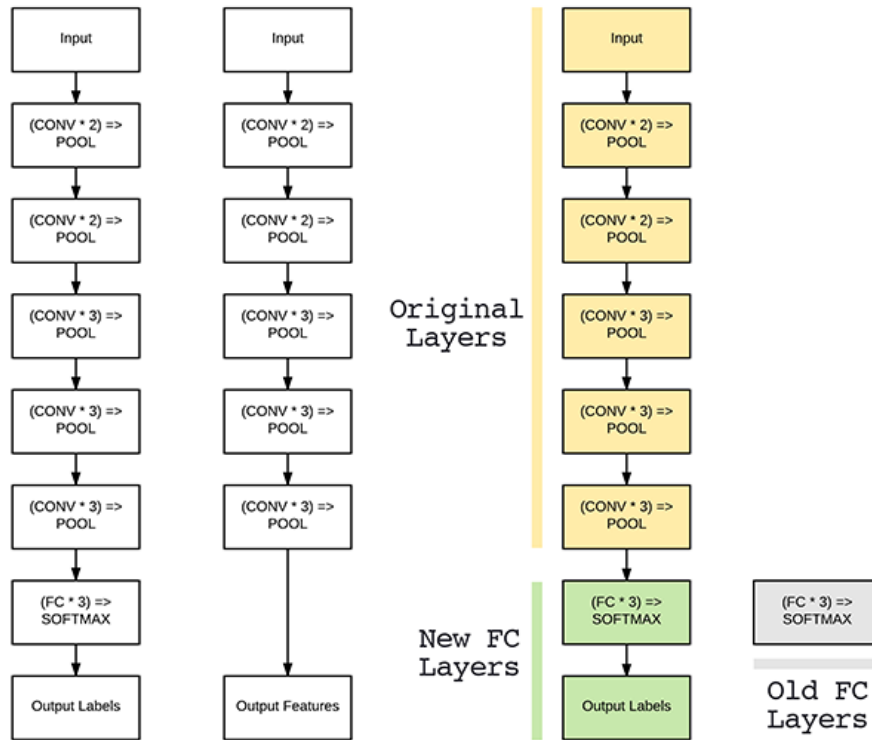


Figure 4.4: **Left:** The original network architecture that outputs probabilities for 1000 different class labels. **Middle:** Removing the FC layers from the network and the output of the final pooling layer will serve as the extracted features. **Right:** Removing the original FC layers and replacing them with a brand-new FC head. Now these can be fine-tuned to a specific dataset [102].

On the left all layers of the network used before are displayed. The final set of layers (i.e. the “head”) are the FC layers along with the Soft-Max classifier. When performing fine tuning, the head of the network is actually severed (Figure 4.4, middle). However, a new FC head is built and stitched on top of the kept old layers (Figure 4.4, right). The new FC layer head is randomly initialized (just like any other layer in a new network) and now “neurally” connected to the body. However, the convolutional layers have already learned rich, discriminating features while the

head is new and its weights are totally random with no knowledge whatsoever. If the gradient can backpropagate from these random values all the way through the network, a risk of destroying its powerful features is alarmingly present. A way to circumvent that is by letting the FC head “warm up” by freezing all layers in the body of network as depicted in Figure 4.5 (left).

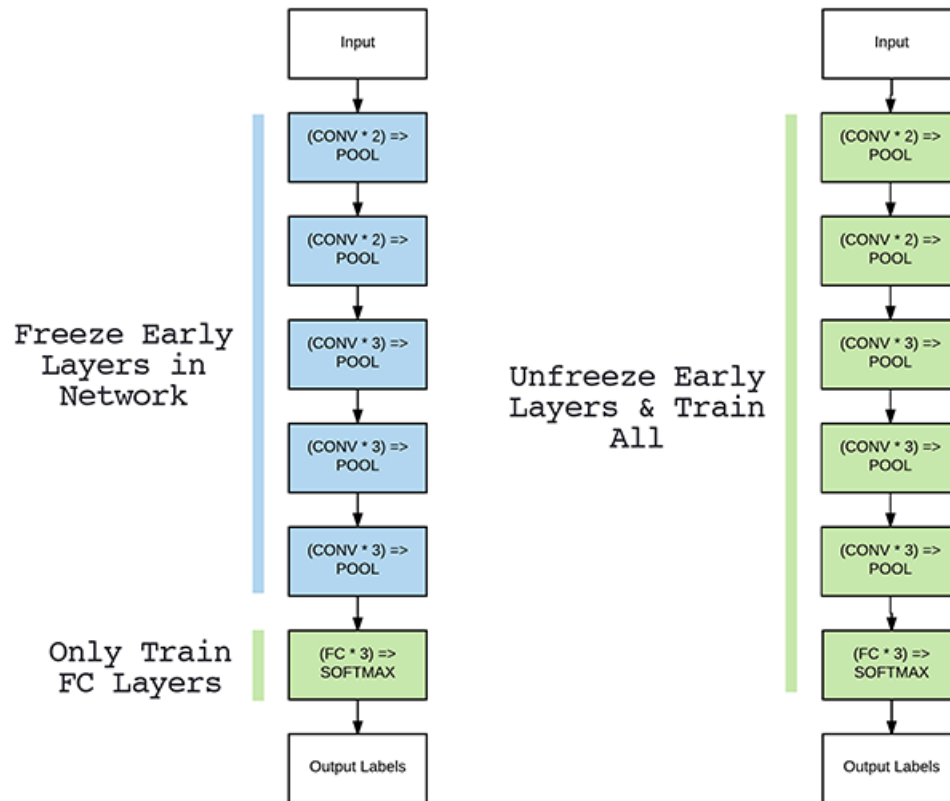


Figure 4.5: **Left:** At the start, all layers are frozen, and the gradient is only allowed to back-propagate through the FC layers to achieve a “warm up”. **Right:** Afterwards, one choice might be to unfreeze all the layers and allow each of them to be fine-tuned as well [102].

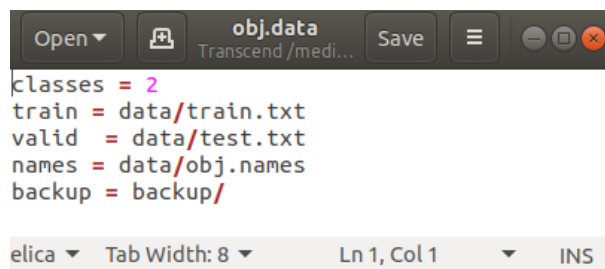
Training data is forward propagated the usual way; however, the backpropagation is stopped after the FC layers. That would allow these layers to start learning patterns from the highly discriminating convolutional layers. In some cases, there’s no need to unfreeze the other layers as the new FC head may reach the required prediction accuracy. However, as mentioned earlier for some datasets (like our case)

it is sometimes advantageous to allow the original body of layers to be retrained and adapted during the fine-tuning as well (Figure 4.5, right). So, after the FC head started to learn patterns, training is paused, the body unfrozen, and then it will resume normally. To avoid altering the convolutional filters dramatically, a very small learning rate is used. Training will once again be allowed until the targeted accuracy is obtained. Luckily enough, the API processes take care of these intricate mechanisms as long as the configurations are properly called.

4.3.5 Configurations

4.3.5.1 Charged Tracks Clusters using Darknet API

The dataset exported here from CVAT is YOLO specific. The files produced contain all the essential material needed to launch a training using YOLOv4 in Darknet setup. This includes the input images which are stored in data directory under “obj_train_data” folder. Also, the images are added with their corresponding .txt files that encode the information from annotation and labelling. Additionally, the files “obj.data”, “obj.names” and “train.txt” are produced (“test.txt” from the validation dataset as well). An extra line is added to “obj.data” to include validation data. The final content of “obj.data” file is shown in figure 4.6.



```

classes = 2
train = data/train.txt
valid = data/test.txt
names = data/obj.names
backup = backup/

```

Figure 4.6: obj.data file

Two classes are used each for one kind of cluster. The classes’ names are showing as “Single_Cluster” and “Merged_Cluster” in “obj.names”. The YOLO version used in this application is YOLOv4[18] according to the latest instructions on “AlexeyAB/darknet” GitHub page. Again, before the training started, changes were implemented to the configuration file in order for the data to be passed smoothly and the algorithm gets properly optimized. The configuration file used is “yolov4-custom.cfg” in darknet/cfg folder. This file is spared for custom object detection on a customized dataset. Explanation is provided in relation to parameters that are

adjusted for this case (Please check Appendix B for setups details).

Afterwards, the changes should be saved, and the configuration file renamed as “yolo-obj.cfg” as this is the default name used in the training command. A final step before starting the run is to make sure that the “yolov4.conv.137” initial weights are downloaded and placed in the main directory. Similarly, as training progresses, the best weights are identified and automatically saved in “backup” folder. The improved weights can always be reused if another run is needed for training.

4.3.5.2 Charged Tracks Clusters using Tensorflow Object Detection API

One of the most effective tools for custom object detection is Tensorflow Object Detection API⁹. It provides an intuitive platform (“eager execution”¹⁰) for transfer learning. Using any of the state-of-the-art pre-trained models provided the last layer is replaced for the data in hand so that the model can be fine-tuned to learn it. The version used for this study is the latest Tensorflow 2.3.

The hybrid models selected from the model zoo are computationally heavy so the Physics Department’s EPP cluster¹¹ was used for training instead of a laptop with single GPU. The installation instructions on the Tensorflow 2 Object Detection API Tutorial page¹² for a Linux machine was carefully followed in the right order. A python 3.8 Anaconda virtual environment was created and activated, Tensorflow 2.3 was installed and verified in a separate directory named “Tensorflow” and CUDA GPU support was skipped since the cluster doesn’t contain GPUs. Tensorflow Model Garden was downloaded from the main Git page¹³ and added in a folder named “models”. Protobufs were compiled and installed, these are used to configure the model and training parameters. COCO API was installed but never used since another metric proved more engaging and used to gauge the dataset’s box predictions. Then, the Object Detection API was installed and tested.

Following the tutorial (Please check Appendix B for further details) all steps leading to training were prepared in the right order:

1. Organize the workspace and training files

⁹https://github.com/tensorflow/models/tree/master/research/object_detection

¹⁰<https://www.tensorflow.org/guide/eager>

¹¹<http://py-box.lancs.ac.uk/computing-guide/>

¹²<https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/install.html#tensorflow-installation>

¹³<https://github.com/tensorflow/models>

2. Arrange the images and their annotations in the right folders
3. Configure the training pipelines
4. Train the models.
5. Monitor progress and test performances with Tensorboard
6. Export the trained model and use it as an object detector.

By default, the training process logs some basic measures of training performance. Now an additional, necessary step that goes in parallel with training is the evaluation process. To do that, standard metrics are used as mentioned before while the ones used here are “oid_V2_detection_metrics” which uses Average Precision (AP)¹⁴ for both individual classes and the total mAP. Also, as detailed in Appendix B the dataset was partitioned into two parts, where one was to be used for training and the other for evaluation. The metrics were selected along with the validation images, to get a sense of the performance achieved by our model as it is being trained. While the training runs, it will occasionally create checkpoint files which correspond to snapshots of the model at given steps. When a set of such new checkpoint files is generated, the evaluation process uses these files and evaluates how well the model performs in detecting objects in the validation part of the dataset. The evolution of this evaluation is summarized in the form of some metrics over time (or iterations).

4.4 XAI in Performance Checks

The performance checks in this section would be first limited to the training and validation processes from a pure data science point of view. That would help in the final choice of algorithm, which would logically be the most performing one according to the standard metrics, described later in a separate part with examples, and most commonly used on Hybrid models of similar criteria. All the physics analysis done on testing samples will be described later in section 4.5.

YOLO algorithm and the three Tensorflow based models, mentioned in previous section, were objects of the checks. When YOLO algorithm is referred to in this study, what was actually used is YOLOv4 which has YOLOv3 as Head or OD and CSPDarknet53 for Backbone. Additionally, for training YOLO, Darknet API was used instead of Tensorflow2 OD API which was used for the other three models.

¹⁴These basic metrics would be explained in the next section.

4.4.1 Metrics

All state-of-the-art object detector algorithms need certain basic evaluation metrics. These are used to help validate, check training performance, and get rid of unwanted low fidelity results. Each metric will have its own properties and uses. Therefore, we will focus here merely on Intersection over Union “IoU” and mean Average Precision “mAP”.

IoU: IoU is an evaluation metric used to measure the accuracy of an object detector on a particular dataset. Any algorithm that provides predicted bounding boxes as output can be evaluated using IoU. So, in order to apply IoU two kinds of bounding boxes are needed. The ground-truth ones and the predicted ones from the model. The first are generally obtained through hand labelling via rectangular boxes using specific software (e.g., CVAT). Computing IoU can be determined via the equation showing in a self-explanatory representation in Figure 4.7 below. As seen


$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Figure 4.7: Computing the Intersection over Union is simply dividing the area of overlap between the bounding boxes by the area of union.

in the equation there, the IoU is simply a ratio. In the numerator area of overlap is computed between the predicted and ground-truth bounding box. The denominator is an area of union that covers both combined together. Now, dividing the area of overlap by the area of union would give the final required metric score (i.e., IoU). In practice, an IoU score with fidelity level of 0.5 or more is considered as good enough prediction, in most cases. We need IoU fidelity score in object detectors because, unlike simple binary classifiers, they produce outputs that are very rarely exact clones of ground-truth labels (i.e., classes and bounding boxes). It is extremely unlikely to

have the dimensions, (x, y) coordinates and the height and width, of our predicted bounding boxes with the same values as the ones for the ground-truth bounding boxes. Therefore, due to varying and multitude of parameters involved, expecting a complete and total match between predicted and ground-truth is simply unrealistic. So, what IoU score does is reward predicted bounding boxes for heavily overlapping with the ground-truth. This reward is presented as a higher fidelity score. However, to ensure that the matching is as close as possible, a threshold on IoU score should help deciding on which predictions to keep and which to be disposed of (e.g., > 0.5).

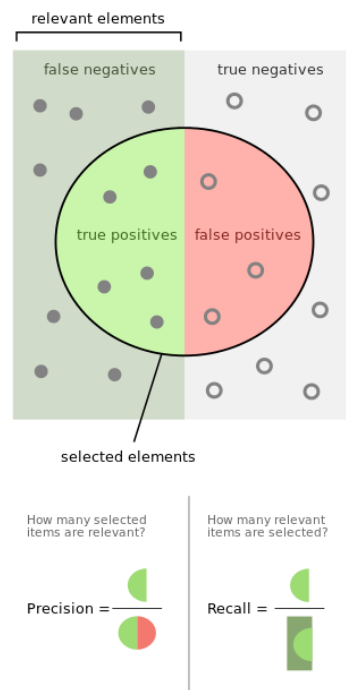


Figure 4.8: <https://commons.wikimedia.org/wiki/File:Precisionrecall.svg>

mAP: To understand mAP, precision and recall should be first introduced since all mAP calculations are based on these two sub-metrics. Basically, *Precision* for a given class is the number of positive predicted rate (i.e., The number of selected Clusters that were accurately classified):

$$Precision = \frac{TP}{TP + FP} \quad (4.6)$$

where TP is the true positive and FP is the false positive.

On the other hand, *Recall* of a given class is simply the true positive rate (TPR) or sensitivity (i.e., The number of Clusters of interest that were selected and not left out) :

$$Recall = \frac{TP}{TP + FN} \quad (4.7)$$

where FN is the false negative, making the recall represented as the ratio of TP and total of ground-truth positives.

Now the IoU metric will be used again in defining the *precision* and *recall* of the bounding boxes. A TP bounding box is considered so if it has an $IoU > 0.5$ compared with the ground-truth, while a FP is any duplicated bounding box or any with an $IoU < 0.5$.

Now FN selection process can be a bit different. In two possible ways, an object detector can miss the target and therefore produce false negatives. One case is when it produces no detection at all. Another is when the predicted bounding box has an $IoU > 0.5$ compared with the ground-truth, however, holds a wrong classification. After statistically defining all TPs, FPs and FNs, the bounding box confidence level

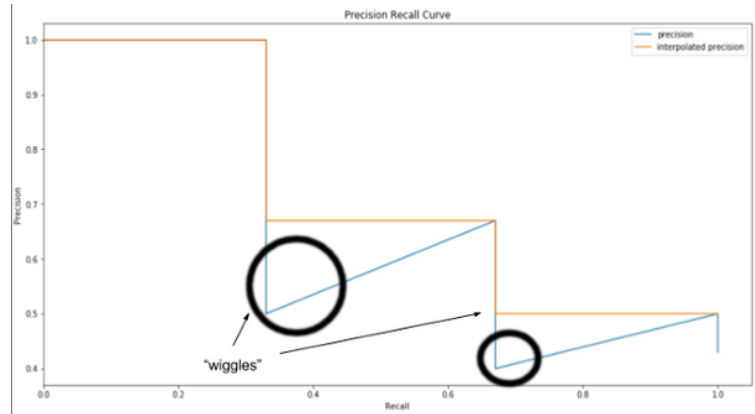


Figure 4.9: Precision/Recall curve for the example in hand. The black circles are showing the wiggles in precision that are avoided by using interpolation instead [111].

(Usually given by the soft-max layer) would be used to rank the output. But before explaining how the Precision/Recall curves are used to produce the single number metric called mAP, the interpolated precision p_{int} should be additionally introduced. The p_{int} is calculated by taking the maximum precision measured at every recall level, between r or one rank higher, using the formula:

$$p_{int}(r) = \max(p_i(r_i) : r_i \geq r) \quad (4.8)$$

where $p_i(r_i)$ is the measured precision at recall rank r_i which is the same or one rank higher than r . Interpolations are basically keeping the highest value of precision for

every other recall by projecting along the vertical axis (figure 4.9).

The Average Precision (AP) is calculated using the area under curve, when the interpolated precision is the one taken into consideration. Then, a known practice in the industry is to divide this area into 11 sectors or segments of recalls which are located at Recall = (0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1).

The formula used for calculating AP is the following:

$$AP = \frac{1}{11} \sum_{r \in (0,0.1,\dots,1)} p_{int}(r) \quad (4.9)$$

Finally, the mAP of an object detector is the average of AP calculated for all the classes that exist in the dataset used for evaluation and testing.

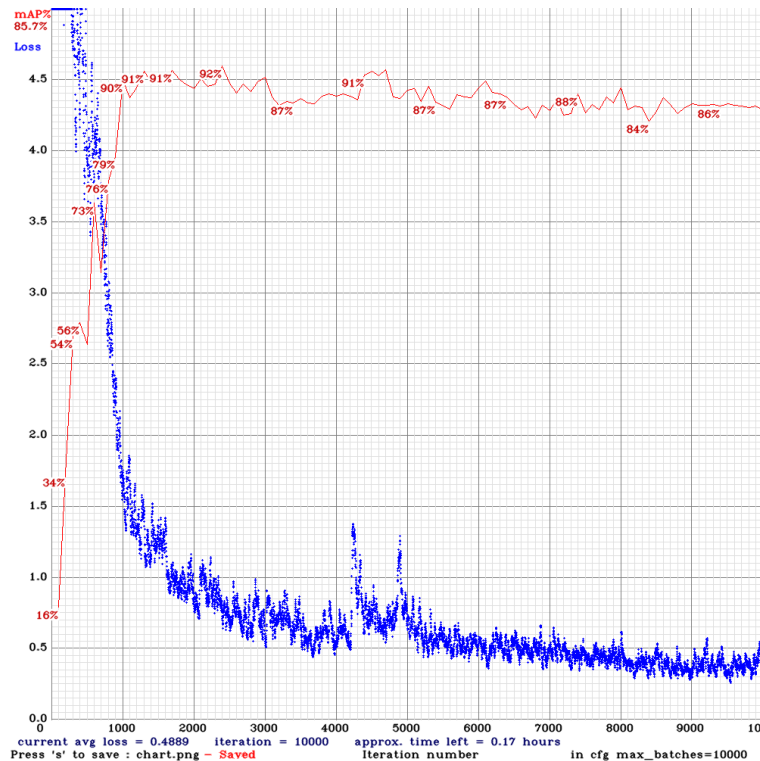


Figure 4.10: Results of initial training showing the loss (*blue*) and mAP (*red*) plots.

4.4.2 Technical Analysis & Model Competition

Figure 4.10 shows the results of initial training on the dataset with yolov4 custom starting weights, where iterations appear on the x-axis while the loss value on the

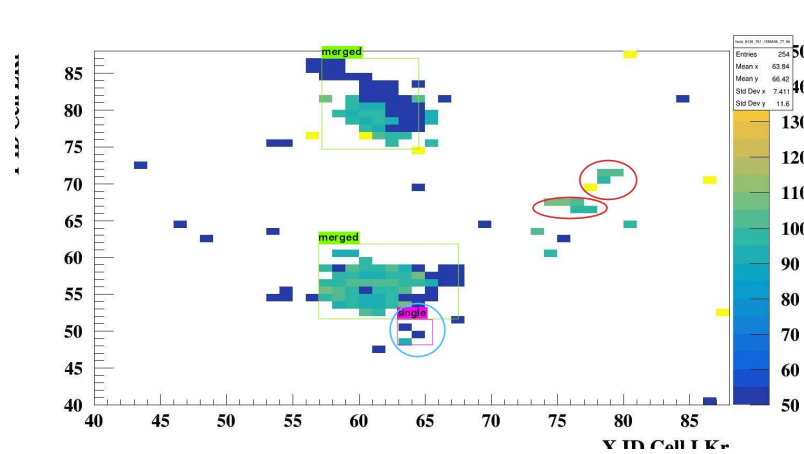


Figure 4.11: An instance of initial training predictions. True identifications in (*green*) boxes first. Then circled in (*blue*) is a FP (mid-bottom) while 2 FNs are highlighted in (*red*) (upper-right)

y-axis. After 2000 iterations the loss function (in *blue*) dropped significantly and mAP (in *red*) reached a value greater than 85%. This is a clear indication that the model was able to generalize well and predict with a high accuracy. The final result for the first run has an average mAP of 86%. Two peculiarities appeared in this run that required discussion and attention. Starting with the least alarming, two spikes appeared between 4000 and 5000 iterations as seen in figure 4.10. As well explained in [95], spikes are an unavoidable consequence of Mini-Batch Gradient Descent in Adam (Batch_size=64 in this case). Some mini batches have “unlucky” random data for optimization, producing those spikes in cost function using Adam. If stochastic gradient descent (GD) was used instead (same as using batch_size=1) there will be even more spikes in the cost function. The same does not happen in (Full) Batch GD because it uses all training data (i.e., the batch size is equal to the training set) each optimization epoch. However, the training would become too slow and not practical which is the reason Adam optimization was used in the first place. So, to say that the existence of such spikes is a natural consequence of Adam even though some spikes might appear pointier than the others like the two being discussed here. Then, it is noticed as well in figure 4.10 that after the two large spikes appeared, not necessarily related events, the loss function continued dropping while the mAP kept decreasing slowly with it. It is a first impression and intuitive to assume that mAP and the loss function value are correlated and inversely related where if one would increase the other should decrease. This cannot be further from the facts especially in box detection cases. That might apply to accuracy as a single metric but definitely not to mAP. As it was detailed in subsection (4.4.1), mAP calculation is quite complicated

and dependent on 3 parameters TP, FP and FN while averaged at 11 Recall levels. A quick reminder here is that Recall loss is directly related to Clusters of interest being left out. In this Recall-related property the danger of FN becomes concrete and clearly understood. Having more FN might lead to higher accuracy on lower Recall levels which would take away from the final mAP value. Additionally, if a common threshold of 0.5 is for instance chosen in the TP, FP and FN selection conditions, successful predictions with classification score <0.5 would still slightly decrease the loss function and the mAP value at the same time. From CV empirical experience, this issue can be somehow related to the small-scale Clusters which are excluded for being unrecognized or recognized with extremely low classification score. That is why any change in the initial configuration conditions must take these small-scale Clusters into account.

As expected, plenty of FN (False Negative) instances were found mainly on small Single clusters where the model couldn't identify them at all. Some FP existed as well where electronic noise was falsely identified as Single Cluster. An example of these drawbacks can be noticed in figure 4.11. The figure shows a true detection of two Merged clusters, which was a general trend for all other detections. The model after first run still clearly struggles to identify correctly the small Single Clusters. That is showing from the FP (circled in blue) and the two FNs (circled in red). In the case of FP, some noise pixels were identified as Single clusters. While for the FNs, two Single clusters were completely ignored as noise. Since this issue was a serious drawback for the model, a solution for this problem took urgent priority.

To overcome this small-scale obstacle, another training took place for 6000 iterations starting from the best weights of the first training. Only this time the resolution was optimized from 416×416 to 576×576 . It was the highest resolution the GPU¹⁵ in hand could handle to launch the training. An improvement was obtained on the model, specifically on small scale detections, by simply increasing the resolution of input images. This tuning should have worked very well with the other parameters adjusted in the configuration file for that same purpose (Appendix B), in this second run only. This improvement was interesting considering the few percent increase in the total mAP which became 89.9% by the end of the training (see figure 4.12) but the best weights would be finally selected which might carry mAP with a top score of 96%.

After training on higher resolution, predictions were tested on the validation set of unseen images. Both FP and FN became much less likely to appear. Another method used to reduce the likelihood of incorrect classifications was changing the threshold of detection prediction probability score (or fidelity). By default, it is set

¹⁵Nvidia Quadro RTX 4000 Max-Q

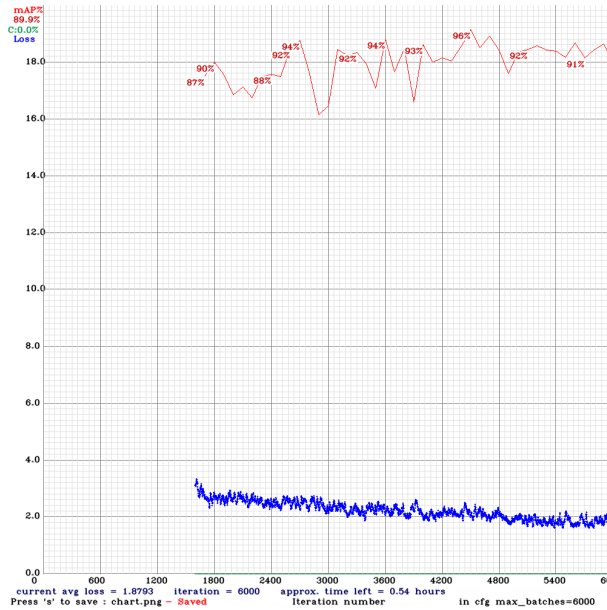


Figure 4.12: Results of second training showing the loss (*blue*) and mAP (*red*) plots

in the algorithm to 25% and an additional argument to the main testing command can be used to adjust this value as needed. This parameter is usually increased when small-scale detection is insignificant. However, in this case the opposite was needed, and the threshold was lowered to 15%. That would allow the model to have a full prediction scope even in instances that occupy very small areas. In most cases, the model started predicting both classes with 80-90% certainty. Only on the small size Single clusters identification fidelity appeared lower. Figure 4.13 shows a general trend of high-fidelity detection for both Merged and Single clusters, plus one “benign” FP (circled in *red*).

To make a proper sense out of those FP we call “benign”, their physics source was investigated. NA62’s Calorimetric experts’ interpretation is that cases happen when electronic noise overlaps with physics clusters. Electronic noise would occupy single cells with some energy deposit. These would look like rare fake MIP with energy levels of 10MeV/cell with fluctuations up to (50-60)MeV/cell and tend to be aligned along the y-axis since they almost exclusively appear in vertical cells. So, to say that misidentifying them as Merged Clusters is not as serious and can easily be recognized from their energy bulk levels.

On the other hand, very accurate detection results are noticed even for small scale

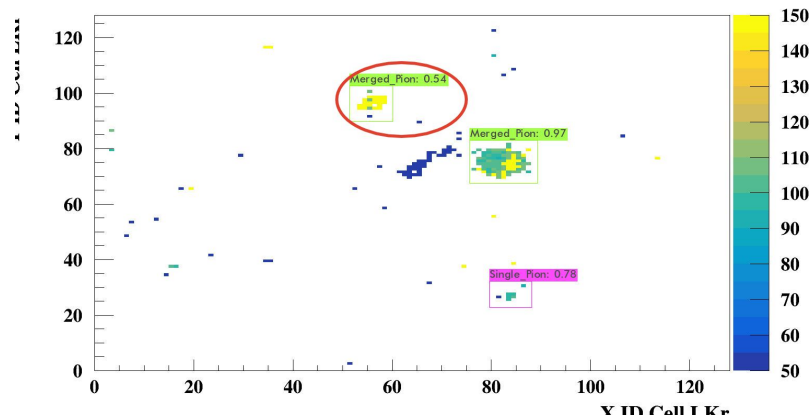


Figure 4.13: Circled in (*red*) is a benign case of FP (upper-middle)

Single clusters with a wide range of confidence levels. Even in the case of low scores, detection is both successful and accurate to an acceptable level. The command used for testing is the same one used previously only with “test” script and few other arguments.

“-ext_output” that would extract all the outputs of predictions done on images in the testing set, referred to by “test.txt”, and save them in a file named “results.txt”. While the threshold “-thresh” is for the detection confidence and was set to 0.15. However, a bigger scope performance comparison with other algorithms is a must and will be presented later on other testing samples.

To summarize, as shown in figure 4.14, the training was performed over 6000 iterations and on two stages. Where the best weights from first run served as starting point for the second run. A final total mAP of 89.9% was registered by the end of training session. In the figure we notice the evolution of the loss function’s value (*blue*) and mAP (*red*) over time/iterations.

As for Tensorflow models, using Tensorboard dashboard, a clear display of AP evolution over time can be extracted (all 20k iterations). For all three models AP at an IoU threshold of 50% for each of the two individual classes plus a total mAP plots are produced. These plots were made possible by choosing “oid_V2_detection_metrics” in the configuration files.

To use inference on the trained model with its best performing weights (i.e., Checkpoint) the model should be exported. In other words, the newly trained inference graph needs to be extracted, which will be later used to perform the object

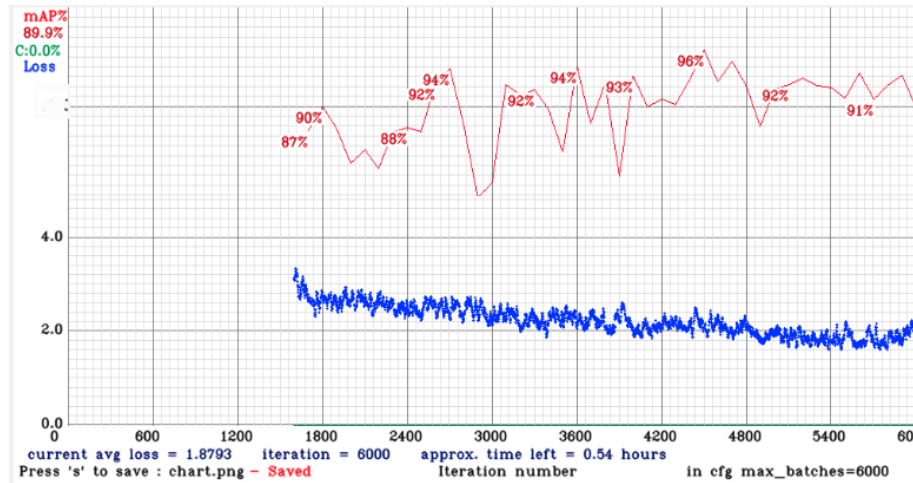


Figure 4.14: YOLO’s second run showing the loss (*blue*) and mAP (*red*) plots.

Model	AP (Single)	AP (Merged)	total mAP
YOLO	-	-	89.9%
SSD_MobileNetV2	71%	91.8%	81.44%
SSD_ResNet_FPN	74%	93.67%	83.8%
FasterRCNN_InceptionResNet_V2	60%	88.5%	74.25%

Table 4.1: Performance checks on AP for all models

detection. For that purpose, another script “`exporter_main_v2.py`” should be utilized and the saved_model would be used to run inference and perform object detection on test images.

Figures 4.15, 4.16 & 4.17 show the results for SSD_MobileNetV2, FasterRCNN_InceptionResNet_V2 & SSD_ResNet_FPN respectively. There it is clearly noticed that the peak performances happened at 15k, 16k & 5K iteration for SSD_MobileNetV2, FasterRCNN_InceptionResNet_V2 & SSD_ResNet_FPN respectively. It is noticeable from the figures some fluctuations for the AP values by the end of the training but it doesn’t really matter since Tensorflow API would pick the best weights during the training session at the highest most accurate values and save them automatically. So, these fluctuations would simply indicate that the training was better at earlier iterations using the AP metrics.

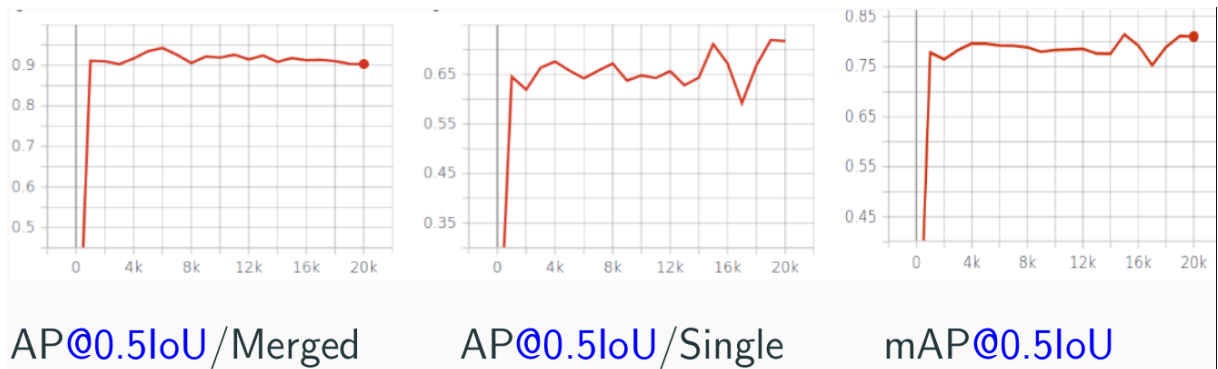


Figure 4.15: Results showing the individual AP for classes and total mAP plots (Best peak at 15k iteration) for SSD_MobileNetV2. The vertical axis is the AP while the horizontal one is for training epochs.

A slower learning of around 2k iterations is noticed in SSD_ResNet_FPN, which is most probably due to the additional OD Neck (i.e., FPN). It is only normal to have a slower training pace by just recalling that FPN recycle the features at every level in the Feature Pyramid and scan over every scale of the input image. This specific architecture showed a higher efficacy than the other two, since one of our classes (i.e., Single clusters) is formed of only a few pixels and FPN worked so well in extracting its features.

So, to conclude, both YOLO and SSD_ResNet_FPN championed the competition (As shown in table 4.1) and were the first two best performing algorithms. However, since Darknet API is quite different than Tensorflow's, a reasonable doubt was assumed that they do not share the same mAP evaluation and specifically not the same IoU threshold. Additionally, no display of AP for individual classes could be extracted for YOLO to compare fairly. The only solution, to test properly which is the best performing one, was to fix the thresholds and check the prediction results for the validation data and compare with the annotated ground-truths to be able to make the decision over which would become the final adopted algorithm.

A pattern of few FP and FN only appearing in YOLO predictions led to the conclusion that this specific dataset and the rounds of training performed¹⁶ SSD_ResNet_FPN should be the final model for NNODA.

¹⁶For clarification, it was the most convenient choice for this case study with respect to its specific set-ups and available tools at hand, and has nothing to do with models Benchmarking which is not our concern here.

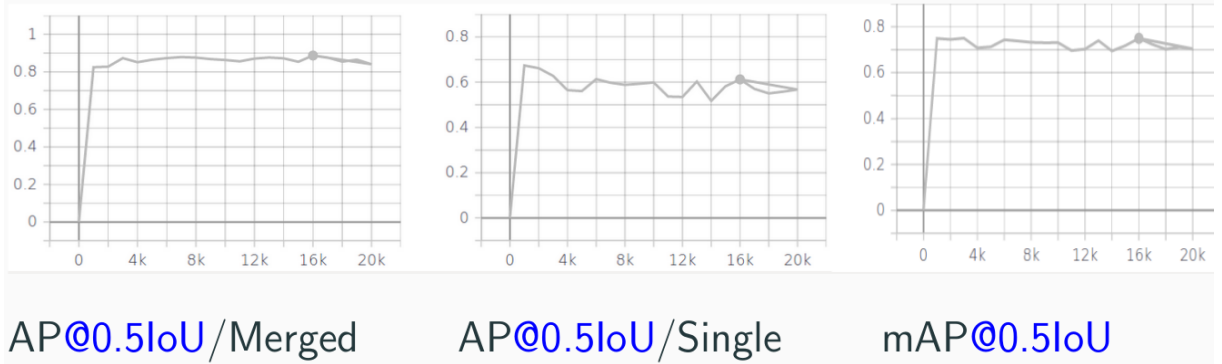


Figure 4.16: Results showing the individual AP for classes and total mAP plots (Best peak at 16k iteration) for FasterRCNN_InceptionResNet_V2. The vertical axis is the AP while the horizontal one is for training epochs.

Practically, figures 4.18, 4.19 & 4.20 show an instance of a FP with YOLO and not with SSD_ResNet_FPN which appeared in comparison with the GT (Ground Truth). While figures 4.21, 4.22 & 4.23 show an instance of FN¹⁷ with YOLO and not with SSD_ResNet_FPN which appeared in comparison with the GT (Ground Truth).

4.5 XAI in Calorimetric Implications

The physics performance of the dataset can be displayed in the following stages:

- **Selection & Classification:** In the selection mode, the goal of the vision system would be to understand the $K^+ \rightarrow \pi^+\pi^0$ background and be able to select different kinds of clusters (i.e., Single & Merged) with hits in time with the π^+ .
- **Rejection:** The system's understanding of the background would allow it to reject events at smaller time windows (i.e., $|t_{cluster} - t_{rich}| < 30\sigma$) when tested on $K^+ \rightarrow \pi^+\pi^0$ samples.
- **Goal:** Gain improvement on the signal acceptance in general and the LKr Photon Veto in specific.

For both selection and rejection tests, only the in-time vs out-of-time feature of NNODA is explored. So first, for the selection process we expect the $\pi^+\pi^0$ events

¹⁷Far more dangerous than FP also because these indicate a lack of competence in selection not only detection

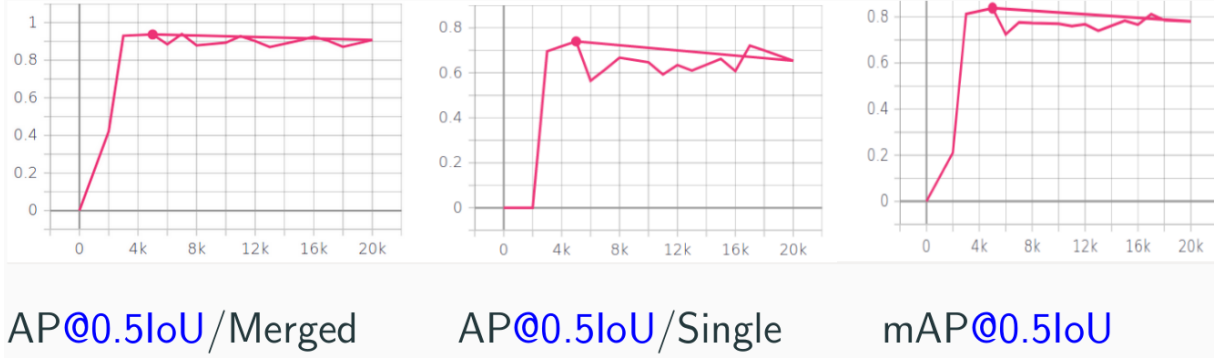


Figure 4.17: Results showing the individual AP for classes and total mAP plots (Best peak at 5k iteration) for SSD_ResNet_FPN. The vertical axis is the AP while the horizontal one is for training epochs.

to mostly have in-time clusters which would qualify them as $\pi^+\pi^0$ background events. So, selecting these accurately would contribute to reducing this kind of background in the final analysis. To compete with the SA selection, we should improve the general efficiency of selecting inside this smaller time window (i.e., $\pm 10\text{ns}$). As shown in table 4.2, SA would reject all the events used for testing but in a much larger time window of $|t_{cluster} - t_{rich}| = 50\sigma$, and reducing the time window only to $\pm 30\sigma$ would reduce the efficiency in half leading to an increase of the background with a factor of 2. Using NNODA, we could manage to keep the selection efficiency of $\pi^+\pi^0$ background events ($\varepsilon(\pi^+\pi^0)$) at 97.3% with our best performing algorithm (i.e., SSD_ResNet_FPN) while reducing the time window to a quarter its original size (from about $\pm 40\text{ns}$ to $\pm 10\text{ns}$). On the other hand, for the rejection of muon events, we expect these to mostly have out-of-time clusters which would qualify them as $K_{\mu 2}$ events or coming from ionization of muons. The reason is that muon tracks should not be associated with any calorimetric activity (i.e., clusters). Muon events imitate $\pi\nu\nu$ Single clusters from a calorimetric point of view and a sample of them proved very useful for training. As for physics-related testing, $K^+ \rightarrow \pi^+\pi^0$ events were sampled and used for evaluating the algorithm's rejection while another sample of $K_{\mu 2}$ events was spared for RV checks. The only reason SA would reject all the events we used for training is that due to the considerable time window of the cut, random activity arises creating fake associated in-time clusters. By choosing the $\pm 10\text{ns}$ time window for NNODA, we were able to reduce this random activity to a factor of 10, therefore managed to save around 88.3% of $K_{\mu 2}$ events. In table 4.2, we notice the rejection efficiency ($\varepsilon(\mu^+\nu)$) of these events is drastically reduced by a factor of 10 to around 11.7% with our most performing algorithm. The pathological testing of selection/rejection of these muon events would help in the normalization process of random activity for $\pi\nu\nu$ signal

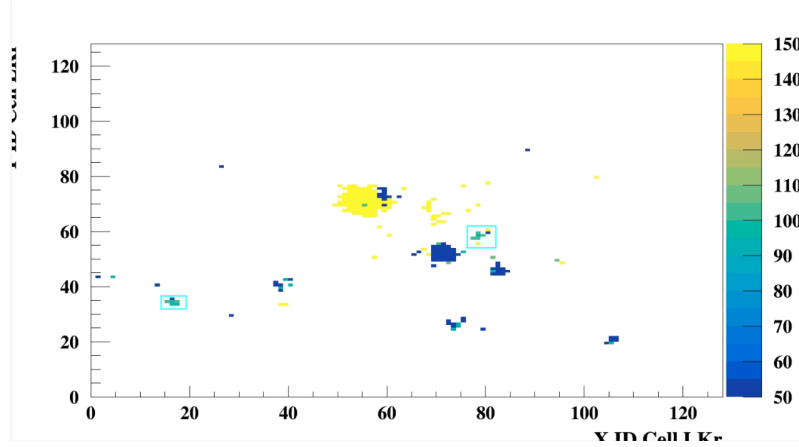


Figure 4.18: GT: Only two detections of Single clusters should exist

Model	$\varepsilon(\pi^+\pi^0)$	$\varepsilon(\mu^+\nu)$
Standard Algorithm	100%	100%
SSD_MobileNetV2	96.1%	10.9%
SSD_ResNet_FPN	97.3%	11.7%
FasterRCNN_InceptionResNet_V2	97%	20.3%

Table 4.2: Selection/Rejection Efficiencies

selection. Additionally, a new feature would be introduced to the analysis, when NNODA would learn to reject any such signal-like (i.e., single charged particle in the final state) events in case they contained in-time Clusters. A degree of freedom that could never be provided by the cut-based approach.

In the following, examples will be presented for some testing events and how SA dealt with them in comparison with NNODA while discussing their peculiarities as well. Each of these events was quite rare and pathological in its own way and lead to some interesting insights on how NNODA can be very handy in understanding and/or clarifying the random clusters' behavior otherwise improving RV in general by reducing random activity's influence on a much smaller time window.

First, as shown in figure 4.24, a $\mu^+\nu$ event rejected by the standard algorithm, but not by NNODA. There the real cell color gradient from digital filter is displayed for both energy and timing visualizations. Circled in red, the MIP is a μ^+ with all in-time

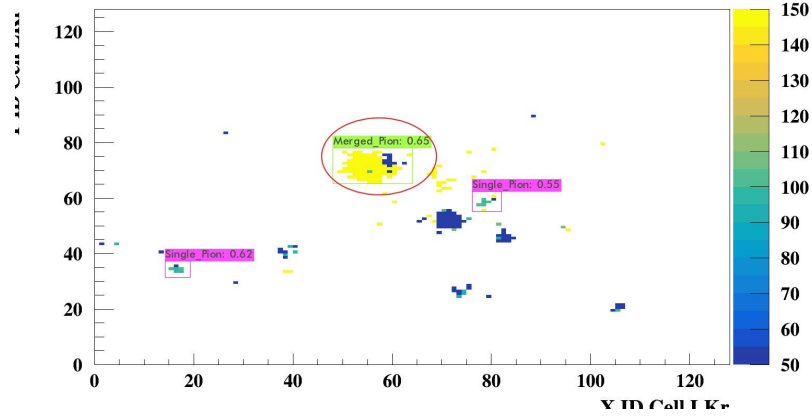


Figure 4.19: YOLO: Circled in (red) is a case of FP (upper-middle)

and out-of-time clusters around it within the same event. On the energy scale, it is noticed that clusters around MIP are showing a certain distribution of energy. Some of them, are perfect clusters in a sense that the high energy peak is showing clearly in the center cells. While others are spread at medium to low energy, like the ones circled in brown which are at an energy level less than 30 MeV. On the time scale, we notice clearly that most of the clusters are out-of-time (i.e., not greenish). Let alone the couple of in-time uni-cell clusters, which were not taken into account for not being neither large enough (i.e., less than 3 adjacent cells/pixels) nor energetic enough (i.e., $E < 30$ MeV) to fit the selection criteria of a physical cluster defined earlier. This event was rejected by the SA, on the basis of a single feature which is the large classical time window. On the other hand, it was saved by NA62's NNODA because all the significant physical clusters were out-of-time of the μ^+ .

Another example, as shown in figure 4.25, a $\mu^+\nu$ event rejected by the standard algorithm, and by NNODA. There the real cell color gradient from digital filter is displayed for both energy and timing visualizations. Circled in red, the MIP is a μ^+ with all in-time and out-of-time clusters around it within the same event. On the energy scale, it is noticed that clusters around MIP are showing a certain distribution of energy. However, here only one significant cluster has the high energy peak in the center cells. While others are spread at medium to low energy. On the time scale, we notice clearly that most of the cells/pixels scattered around are too small to be considered significant and to fit the selection criteria of a physical cluster defined earlier. This event was rejected by the SA, on the basis of a single feature which is the large classical time window. On the other hand, it was also rejected by NA62's NNODA because the only significant physical cluster associated

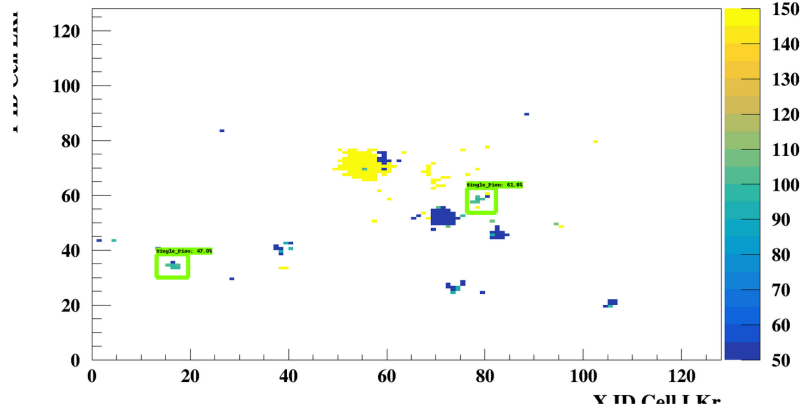


Figure 4.20: SSD_ResNet_FPN: Predicted correctly

with the μ^+ is greenish in color and considered in-time with the MIP. It is a $\mu^+\nu$ event rejected most probably due to an in-time soft photon cluster within $\pm 10\text{ns}$ window (i.e., in radiative $K^+ \rightarrow \mu^+\nu(\gamma)$). This is another kind of insight that NNODA can additionally provide.

Now, as shown in figure 4.26, a $\pi^+\pi^0$ event rejected by the standard algorithm, and by NNODA. There the real cell color gradient is displayed for both energy and timing visualizations as measured by the digital filter. The only difference here is that the in-time time window is made yellowish instead of greenish, on the time palette, for better visualization. Circled in red, the MIP is a π^+ with all in-time and out-of-time clusters around it within the same event. On the energy scale, it is noticed that clusters around MIP are showing a certain distribution of energy. However, here there are many significant clusters that have a high energy peak in their center cells which are perfect electromagnetic clusters. While others are spread at medium to low energy. On the time scale, we notice clearly that most of the significant clusters are out-of-time, while only two of them are in-time (showing in red ovals). This event was rejected by the SA, on the basis of a single feature which is the large classical time window. On the other hand, it was also rejected by NA62's NNODA because two of the significant physical clusters associated with the π^+ are considered in-time with the MIP (in this case only, greenish gradient is replaced by yellowish).

Finally, as shown in figure 4.27, a $\pi^+\pi^0$ event rejected by the standard algorithm, but not by NNODA. There the real cell color gradient is displayed for both energy and timing visualizations as measured by the digital filter. Circled in red, the MIP is a π^+ with all in-time and out-of-time clusters around it within the same event. On

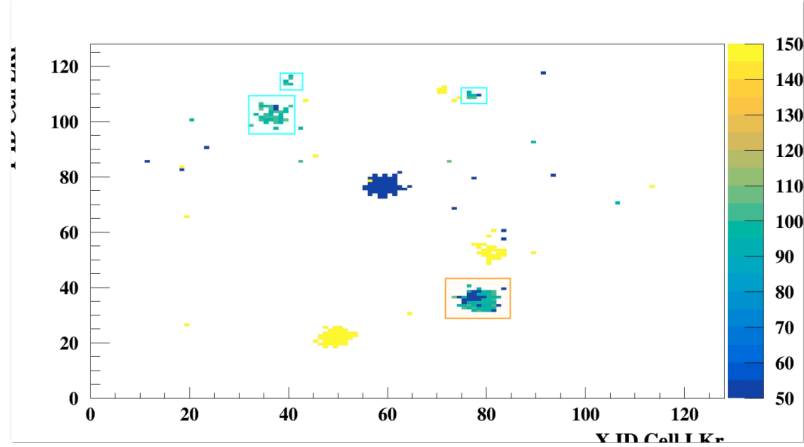


Figure 4.21: GT: All these detections should exist

the energy scale, it is noticed that clusters around MIP are showing some distribution of energy as well. A couple of them, are perfect clusters in a sense that the high energy peak is showing clearly in the center cells. While others are spread at medium to low energy. On the time scale, we notice clearly that most of the clusters are out-of-time (i.e., not greenish). A couple of out-of-time clusters showing in pink ovals, they fit the selection criteria of a physical cluster defined earlier, and they are found one at -60ns out of the MIP time and the other at -20ns out of the MIP time. This event was rejected by the SA, on the basis of single feature which is the large classical time window. On the other hand, it was not selected as a $\pi^+\pi^0$ background event by NA62's NNODA because all the significant physical clusters were out-of-time of the π^+ . However, this event shows some peculiarities that raise some questions that are still under investigation. Figure 4.28 displays a zoomed in version of the second cluster, which is found at -20ns out of the MIP time, both on energy and time scales. There we noticed that it is an ideal cluster where the bulk of energy is concentrated in the center cells which clearly are out-of-time of the π^+ . However, on the rim we could still see some random cells/pixels, pointed at with pink ovals, that are clearly in-time. This is quite strange and can be the result of many subtle scenarios. It could be either a very rare event where both γ s are lost with some γ -like pileup (random activity) or a problem in the LKr cell time reconstruction¹⁸. Studies are now under way, using the new reconstruction v2.x, to check if these effects are consistent and find the right interpretation for the existence of such rare clusters that are rejected because they fall outside the time window allowed by NNODA but still at a far marginal from the classical one for SA at nearly half the time (i.e., 20ns instead of $\pm 50\sigma$).

¹⁸No way to tell “by eye” and must rely on an algorithm

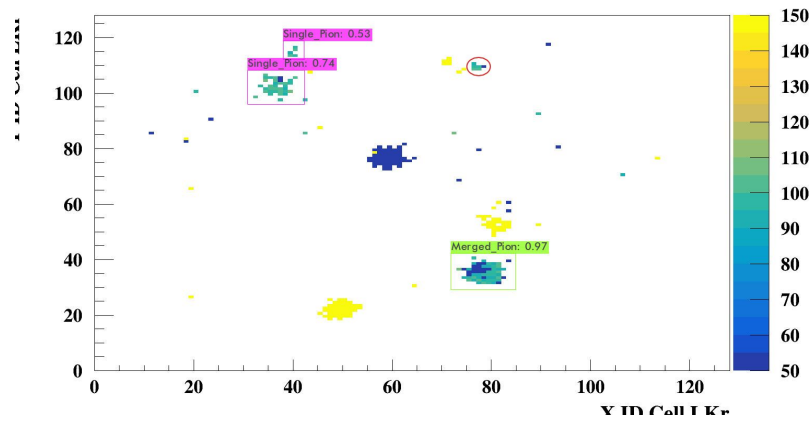


Figure 4.22: YOLO: Circled in (red) is a case of FN (upper-middle)

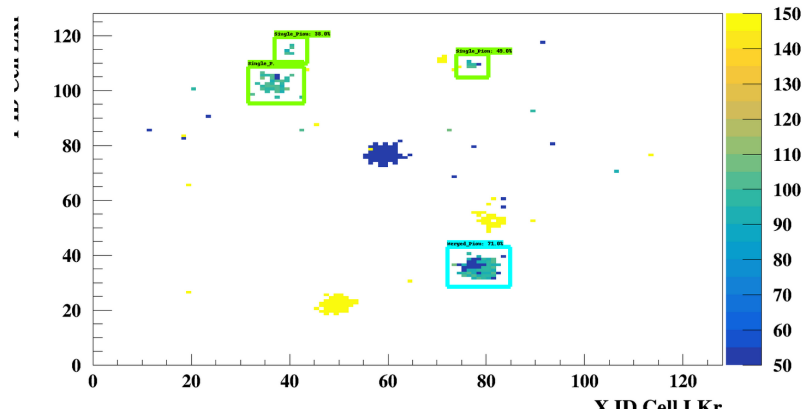


Figure 4.23: SSD_ResNet_FPN: Predicted correctly

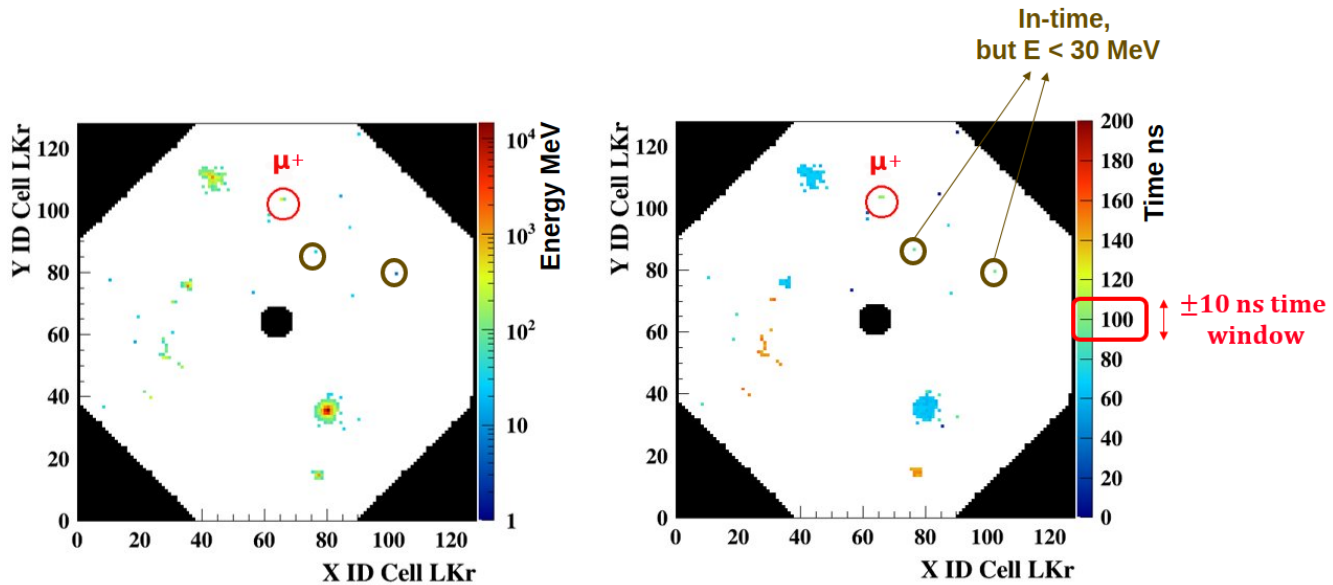


Figure 4.24: (Circled in *red*) is the MIP, here μ^+ . (Circled in *brown*) two insignificant in-time clusters with $E < 30$ MeV. (In a *red* square) is the in-time time window of ± 10 ns, setting the limit for all greenish gradients in the time palette of the time visualization.

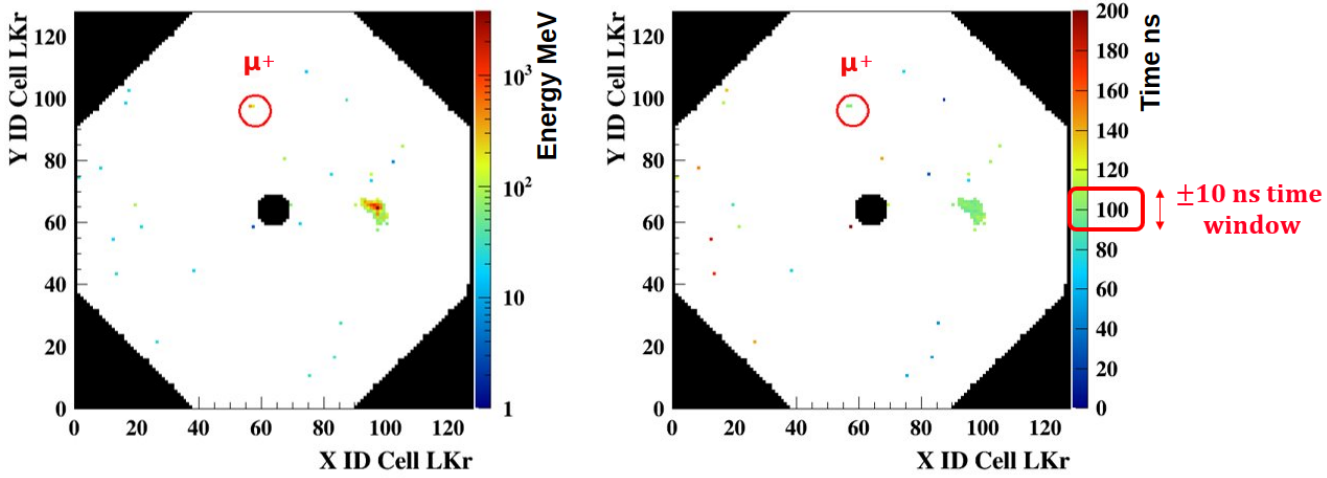


Figure 4.25: (Circled in *red*) is the MIP, here μ^+ . (In a *red* square) is the in-time time window of $\pm 10ns$, setting the limit for all greenish gradients in the time palette of the time visualization.

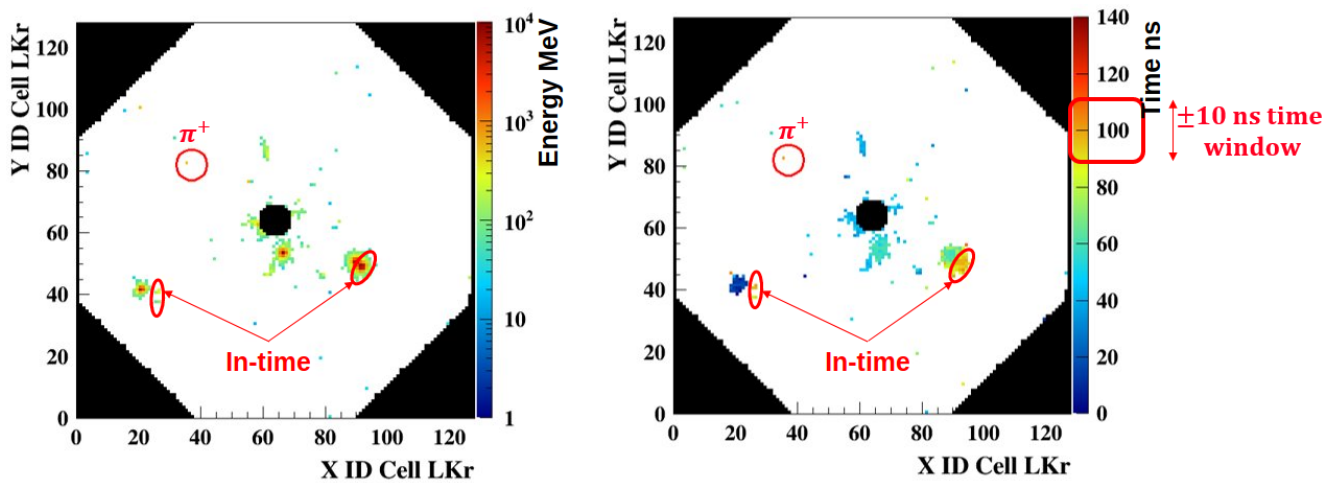


Figure 4.26: (Circled in *red*) is the MIP, here π^+ . (In *red* ovals) two significant in-time clusters. (In a *red* square) is the in-time time window of $\pm 10ns$, setting the limit for all yellowish (exceptionally instead of greenish) gradients in the time palette for a clearer time visualization.

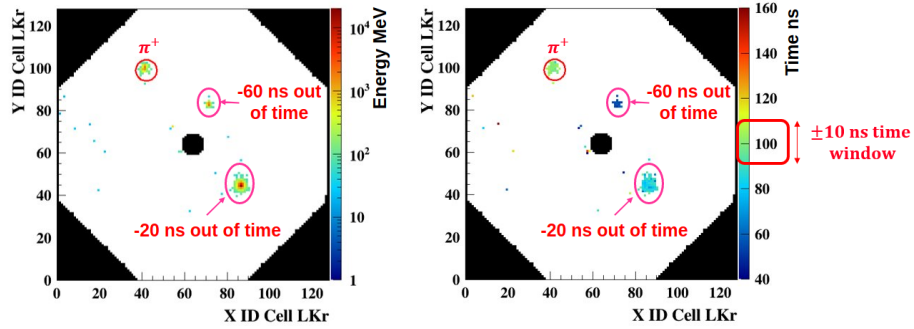


Figure 4.27: (Circled in *red*) is the MIP, here π^+ . (Circled in *pink ovals*) two insignificant out-of-time clusters at -60ns and -20ns out of the MIP time. (In a *red square*) is the in-time time window of $\pm 10\text{ns}$, setting the limit for all greenish gradients in the time palette of the time visualization.

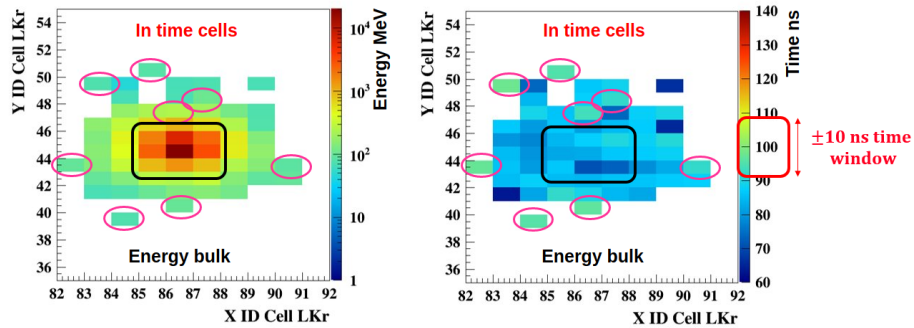


Figure 4.28: Zoomed in look at the -20ns out-of-time cluster. (In a *red square*) is the in-time time window of $\pm 10\text{ns}$, setting the limit for all greenish gradients in the time palette of the time visualization. (In a *black square*) the bulk of energy is clearly out-of-time. (In *pink ovals*) peculiar uni-cells or pixels appear on the cluster's rim and are found to be in-time with the π^+ .

Chapter 5

Summary and Conclusions

In this thesis a Neural Net (NN) code was presented from scratch and applied to the Kaon-Pion matching in the pion neutrino-neutrino analysis of particle physics data of NA62 at CERN. The NN code showed increased efficiency in Kaon decay identification with respect to the standard algorithm based on statistical analysis. Then, another project was an implementation of a new “Virtual Bubble Chamber” technique for NA62’s LKr to analyze calorimetric images of clusters in energy deposit using a NN object detector code we called NNODA (Neural Net Object Detection Approach). It is innovative unique idea to use color tags on the cluster timing to filter out and reduce random activities plaguing particle physics fixed target on-flight decay experiments like NA62. Finally, practical data science skills in Robotics are presented using Computer Vision techniques in non-trivial real-life scenarios, like training an algorithm that would help a drone to identify and locate robotic end-effectors in unusual environments. These skills have been effectively transported to NA62 particle data analysis in a multi-disciplinary fashion.

5.1 $K^+ - \pi^+$ Track Matching

The first part of this thesis proved that data analysis in HEP can be efficiently mapped into a Multivariate/ML/NN classification problem to distinguish background processes and potential signal from events. An educated selection of the physical variables, based on the physics of $\pi\nu\nu$ analysis, allowed to exploit differences between signal and background. The search was designed to maximize the statistical significance of the signal sample over background data. These tools were applied successfully in the $K^+ - \pi^+$ Track Matching of the on-flight $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ analysis of NA62 experiment.

The difficulty of the analysis is that it requires a background rejection of 11

orders of magnitude. The background enters signal regions as resolution tails which depend on irreducible multiple scattering in the material and the quality of association between kaons and Pions. The signal is ambiguously determined by searching for a single lonely π^+ event in the final state. Since a higher track matching quality is expected to solve the resolution tails problem, a NN-based matching was designed and tested.

The goal first was to choose variables accessible in $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ events and to develop the best algorithm that maximizes the probability to associate the parent K^+ to the π^+ (“Efficiency”) and minimizes the probability to associate a beam pileup track to the π^+ (“Mistag”). That way we can gain improvement on the signal acceptance in general guided by these two metrics. In fact, this is achieved when a positively charged track in STRAW with signals associated both in space and time in all the sub-detectors produces a timestamp for the π^+ with a 100 ps resolution. Ideally, an identification of a K^+ in KTAG in time with the selected π^+ in the final state is exclusively selected. The Tracks reconstruction is a complicated process where the rate in GTK is >50 times the rate in STRAWs because the physical beam contains only 6% kaons, and the track matching should only be done for the Pions coming from Kaon Decay.

A first from-scratch 3-layered NN attempt was successfully trained with 9 primitive variables and tested on a first sample of data. This dataset consisted of a whole run of 2017 data and allowed us to build a ROOT Tree with a binary¹ labelled tracks reconstructed from pure kinematics.

After the first attempt, grid search methods were utilized to select the best performing model with the best set of hyper-parameters. Then, another more advanced 5-layered NN model based on common ML libraries was selected and trained with an input of the final 18 variables 3.1.1. While the training sample consisted of multiple runs and around 16 million events. In the data, a clearer definition of Signal/Background was instated as:

- *Signal*: π^+ from $K^+ \rightarrow \pi^+ \pi^+ \pi^-$ decays with a K^+ track in GTK reconstructed using hits in a ± 1 ns time window around the average 3π time (from CHOD) and associated to the real K^+ using the kinematics of the real K^+ inferred from the 3π kinematics.
- *Background*: π^+ from $K^+ \rightarrow \pi^+ \pi^+ \pi^-$ decays with an accidental track in GTK reconstructed using hits in a ± 1 ns time window 15 ns off the 3π time. The

¹Either K^+ or accidental beam activity “Pileup”.

GTK track is then shifted to be in-time with the 3π (If more than 1 accidental track appeared, the one with the highest chi2-based Discriminant is picked).

A first NN Discriminant class was built on this trained model, to be able to compare performance between the NN and the Logarithmic Likelihood based “classical” Discriminant. By varying the prediction threshold on the NN Discriminant, a better performance over the “classical” was translated in 5% more Efficiency and 40% less Mistag.

This first implementation of a NN-based NA62 analysis proved that such models work well enough and can learn physics features and give higher performance even from raw data or low-level variables. The predictions of the NN not only outperformed the “classical” Discriminant but showed stability over the runs as well. Also, this work showed that NNs are flexible and can be adjusted architecture-wise and variables-wise to adapt with the data at hand and give better performance.

Given that, various trials were made to boost the performance of such NN Discriminant by upgrading the model from different angles. First, the data was redefined by removing the “Chi2Condition” pre-filter which restrict the final π^+ association to one best scoring GTK track. Then we adjusted the architecture of the NN Discriminant accordingly and produced 5 classes of it instead of one to adapt with reconstructed events that associate different number of multiple GTK tracks. This attempt was not remarkably successful due to a data mining problem. A robust enough data was lacking to train all 5 models of the NN Discriminant. Afterwards, work has been done on primary cuts, especially on the most influential variables (e.g., timing and CDA). A wider cut of $\pm 2\text{ns}$ on the GTK time and 30mm on CDA, proved to improve the performance of the NN Discriminant and resulted in a lower value in background tails compared with a tighter cut (See Figure 3.16).

On the other hand, feature engineering took its toll and boosted the performance of NN Discriminant from two perspectives. A feature hierarchy technique allowed the objective evaluation of input variables and their level of importance in the training. That helped in the selection process of the most important variables and the disposal of useless ones. Additionally, a new feature was added called *DeltaDiscriminant* that solved the problem of additional GTK tracks by comparing their discriminant probabilities. A null value of this feature would completely reject the event because there would be two or more tracks confusing the NN predictions since their evaluation presented same discriminant scores. Due to *DeltaDiscriminant*, we obtained as first test without optimization 8% higher efficiency on signal acceptance and 20% less tails.

Finally, it is worth mentioning that the dominance of basic features (i.e., CDA and Timing) in $K^+ - \pi^+$ matching gave little space of improvement for this NN Discriminant based on low level raw data. Any further optimization should include either the design of higher-level variables or some other kind of NN that might adapt better and curate the intricate dependencies on the very few variables.

5.2 NNODA: LKr Calorimetric Study

The Neural Network Object Detection Approach (NNODA) is developed to identify in-time clusters in LKr. A technique rarely used in HEP calorimetry is made possible in NA62 due to the fine granularity and cell time resolution of LKr Calorimeter. The visualization process is based on cell time extracted from the digital filter which is independent from cluster reconstruction and can be used with any version (i.e., v.1 or any v.2.x).

Training is done on pathological $\pi^+\pi^0$ events infested with pileup random activity. The sample picked is a selection of $\pi^+\pi^0$ events rejected by SA of $\pi\nu\nu$ analysis because the energy deposit is $> 15\text{GeV}$ and clusters are found between 30σ and 50σ from the π^+ reference timing. The ultimate target is to recognize in-time LKr clusters, even with random activity.

The data-set annotations are based on presence of at least 3-adjacent in-time cells “far” from the π^+ cluster and with the “same color” on the histogram’s time palette. For that purpose, a custom-made definition of “in-time” is adopted at $\pm 10\text{ns}$ from π^+ or MIP. Moreover, an educated visualization of “out-of-time” to help NNODA is fixed but could easily be removed/mitigated in future studies to broaden the scope of time scans. Also, additional features of Merged/Single clusters will be considered and studied.

Several hybrid Machine Learning Computer Vision algorithms are used, competed and the most convenient one picked for our final implementation, which is SSD_ResNet_FPN. The model is very well trained on the dataset and showed the upper-hand in performing accurate predictions of coordinates/locations and classifications in comparison with the ground-truths. Only the localization of in-time clusters feature is explored in this study while the classification part can also be incorporated in future work.

Testing is done on pathological $\pi^+\pi^0$ events affected by pileup and $\mu^+\nu$ events rejected by $\pi\nu\nu$ analysis (2017). A statistically independent sample from that of the training is chosen for testing. Inference is later used to produce a visual event-by-event

analysis with a virtual “**Bubble-Chamber**” technique to discriminate calorimetric activity based on an OD Deep Learning approach (NNODA). An inference-based algorithm was developed to process predictions and cluster-classifications for thousands of events in just over few minutes time, but can be much better optimized in the near future.

Preliminary results showed that NNODA with the highest performing trained model (SSD-ResNet-FPN) showed on selection:

- >96% of pathological $\pi^+\pi^0$ events rejected \sim same rejection power as 50σ ($\sim \pm 40\text{ns}$) time window compared with SA.
- $\times 10$ less rejection of $\mu^+\nu$ events otherwise rejected by SA of $\pi\nu\nu$ analysis which makes NNODA clearly much better resilient to increase intensity effects.
- $\pi^+\pi^0$ events left and not rejected are highly pathological. A physical understanding of their nature would qualify their nature and explain their selection by NNODA. Algorithms must decide to be either recovery from random activity, or (exceedingly rare) timing problems in LKr.
- $\mu^+\nu$ events rejected are due to residual random activity in $\pm 10\text{ns}$.
- About 3-4% absolute improvement in Random Veto, based only on $\pi^+\pi^0$ event not rejected.

Future studies are now prepared to concretely translate results in RV improvement *vs.* 2018 PNN analysis. A target is set to first apply NNODA to events with $E < 15$ GeV activity in LKr to explore any peculiar behavior at low energy limit. Then, an improvement of the definition of “cluster” (box size annotation) can further optimize the algorithm. Also, an optimization based on physics understanding can be done. One way is to ameliorate the definition of “in-time” (shrink/release of the time window). Another way is to modify the educated visualization of the out-of-time to profit of time/color gradient of the out-of-time cells. General optimization can also be done by: (i) using the cluster classification features to improve performance, (ii) training and testing the model(s) on the improved LKr digital filter from the most recent reconstruction versions and (iii) testing NNODA versus higher intensity dependencies.

5.3 Robotics

The Robotics part consisted of first trials to implement a multi purposed vision system to an Unmanned Air Vehicle (UAV) that would allow it to survey and detect

the dynamic motion of a fetching ground nuclear robot with twin manipulators; arms and end-effectors. It is an artificial intelligence-based vision system, and specifically uses computer vision techniques of deep learning in image classification and object detection. First, a basic code, “mobile_activity”, is presented that uses MobileNetV2 Convolutional Neural Net architecture for image classification. A fine-tuning technique from transfer learning produced an image classifier that succeeded with 90% accuracy to differentiate different activities of the manipulator from mere images. Then, multiple state-of-the art computer vision object detector algorithms were implemented through different platforms or APIs. Only a limited dataset was prepared (due to COVID19), annotated in boxes and labelled using Intel’s Computer Vision Annotation Tool (CVAT). YOLO first was implemented through Darknet API and proved to have the highest (mAP) mean Average Precision of 96%. Then, Mask R-CNN through Facebook’s Detectron2 API followed with a mAP of 83%. Finally, SSD with MobileNetV2 as backbone was tested on Nvidia’s DIGITS tool using DetectNet and Jetson-Inference API. This last one is the most practical to implement directly into the embedded system of the UAV, Nvidia Jetson Nano. However, the training was taking too much time and had to be stopped at an accuracy of 72% on mAP.

So, four vision systems were introduced to the project: mobile_activity, YOLO, SSD_MobileNetV2 (DetectNet) and Mask R-CNN. All four of them were used on a Linux system and were trained to identify an end-effector and then tested on blinded video footage that was available of an end-effector in motion. The results from the tests were analyzed and any limitations and/or advantages were highlighted. The results showed that classification and detection of an end-effector was carried on properly within a reliable accuracy at most times.

However, mobile_activity does not provide coordinate information as the other object detectors do. YOLO is quick, accurate and with a slight increase in the data-set size demonstrated promising results. It yielded a mAP of 96%. DetectNet proved to be very practical, efficient and allows one to implement models directly on embedded devices like Jetson Nano. Another advantage it shows is the ROS node that can be linked to a DetectNet model. On the other hand, it proved to be a bit slow in training through Nvidia’s DIGITS tool and had to be stopped at a lower than final mAP of 72%. While finally Mask R-CNN even though its mAP was around 83% but showed superiority in detecting tricky instances like a fading motion frame and a closed grip end-effector.

If the dataset is expanded and made more robust, the accuracy of these object detectors would be boosted further showing the full potential of these models. Additionally, and due to their lack of computational expense, it is likely that they all

can be implemented onto the main processor of the UAV without any issues arising and then successfully classify and detect with coordinates the end-effectors of the ground robots if a live feed is used.

The effect of Covid-19 was significant on the perception and impacted both the training and final testing of the vision systems developed. In addition to this, the rate at which work could be completed regarding annotation was slowed down. Implementation onto the Nvidia Jetson was also affected. The quality and quantity of the dataset created was also negatively affected due to the pandemic. Many images are required for an ideal dataset. Image variety and range are also beneficial. This includes displaying the end-effector from several different angles, lighting, distances, positions and while completing different actions. A few thousand frames are ideal as a minimum to train a vision system with confidence based on this specific application. However, for the project we had access to approximately 4500 frames and as they were from video footage, they were highly repetitive and there was not much variety shown. This was mitigated in two ways. The first was by applying data augmentation to artificially add variety to the dataset. This allowed for greater use to be made of the dataset available. To further mitigate the dataset limitation, a video was found online of the Brokk40 that provided an additional 1003 frames to the dataset once annotated. The benefit of this was a larger dataset for training and greater variation within the dataset as the video found online showed the Brokk40 in divergent backgrounds, whilst in motion and while grasping objects.

Additional work that could not be completed due to Covid-19 was the inability to implement the vision systems onto the quadrotor's Nvidia Jetson and to then do a live video test with the use of the RealSense D415 camera. To mitigate this, when working on the vision systems, care was taken to ensure compatibility between them and the Nvidia Jetson. In the case where there were issues with compatibility such as `mobile_activity` and ROS, modifications were made to the code as explained to overcome this. Therefore, implementation on to the Jetson is not expected to cause any errors. As the methods of perception could not be tested on a live video feed of an end effector, unseen end-effector footage was shown to the model instead.

Finally, a reflection regarding the project is the amount of time taken to create nodes in a Linux system and be able to link it with ROS. ROS required an understanding of its basics which consumed an extensive amount of time. Other techniques were followed as a shortcut for lack of time and all the other difficulties described previously. However, for future implementations ROS integration would be the ideal practice in the field of Robotics and especially related to sensor fusion requirements. In sensor fusion, a robot should be equipped with a communication

and decision-making unified system that coordinates the flow of data received by its sensors and takes actions accordingly.

Appendix A

Standard Notations for DL

A.1 Deep Learning Representations

In general, standard deep learning graphs (see figure A.1) are structured as:

- Nodes represent inputs, activations or outputs.
- Edges represent weights and biases.

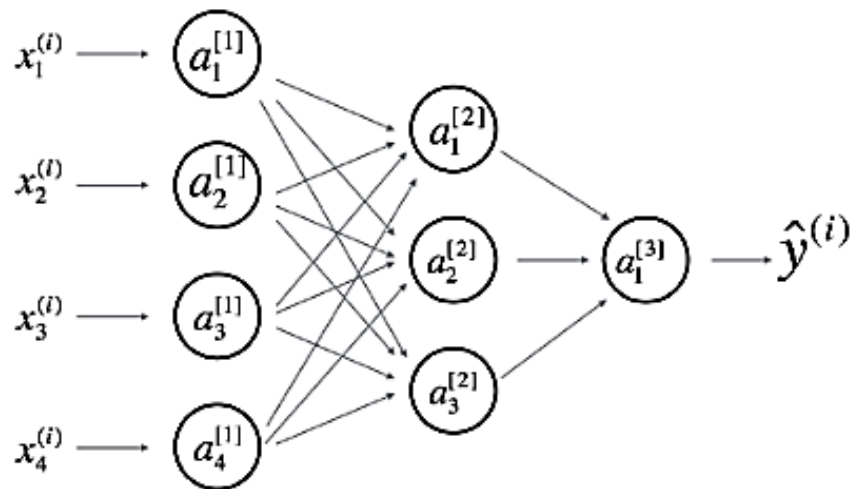


Figure A.1: A typical Neural Net graph with proper representations[95]

A.2 Fundamental CNN Notations

The General Notation is presented as the following:

- Superscript $[l]$ denotes an object of the l^{th} layer.
 - Example: $a^{[4]}$ is the 4th layer activation. $W^{[5]}$ and $b^{[5]}$ are the 5th layer parameters.
- Superscript (i) denotes an object from the i^{th} example.
 - Example: $x(i)$ is the i^{th} training example input.
- Lowerscript i denotes the i^{th} entry of a vector.
 - Example: $a_i^{[l]}$ denotes the i^{th} entry of the activations in layer l , assuming this is a fully connected (FC) layer.
- n_H , n_W and n_C denote respectively the height, width and number of channels of a given layer. If referencing a specific layer, these can also be written as $n_H^{[l]}$, $n_W^{[l]}$ and $n_C^{[l]}$.
- $n_{H_{prev}}$, $n_{W_{prev}}$ and $n_{C_{prev}}$ denote respectively the height, width and number of channels of the previous layer. If referencing a specific layer, these can also be written as $n_H^{[l-1]}$, $n_W^{[l-1]}$ and $n_C^{[l-1]}$.

Sizes are represented as:

- m : number of examples in the dataset.
- n_x : input size.
- n_y : output size (or number of classes).
- $n_h^{[l]}$: number of hidden unit of the l^{th} layer.
 - In a for loop, it is possible to replace n_x with $n_h^{[0]}$ and n_y with $n_h^{[L+1]}$ where L is the total number of layers in the network.

Objects are represented as:

- $X \in \mathbb{R}^{n_x \times m}$ is the input matrix.
- $x^{(i)} \in \mathbb{R}^{n_x}$ is the i^{th} example represented as a column vector.
- $Y \in \mathbb{R}^{n_y \times m}$ is the label matrix.

- $y^{(i)} \in \mathbb{R}^{n_y}$ is the output label for the i^{th} example.
- $W^{[l]} \in \mathbb{R}^{\text{number_of_units_in_next_layer} \times \text{number_of_units_in_previous_layer}}$ is the weight matrix and superscript $[l]$ indicates the layer.
- $b^{[l]} \in \mathbb{R}^{\text{number_of_units_in_next_layer}}$ is the bias vector in the l^{th} layer.
- $\hat{y} \in \mathbb{R}^{n_y}$ is the predicted output vector. It can also be denoted as $a^{[L]}$.

Common forward propagation equation examples:

- General Activation Formula: $a_j^{[l]} = g^{[l]}(\sum_k w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]}) = g^{[l]}(z_j^{[l]})$ where $g^{[l]}$ denotes the l^{th} layer activation function.
- $J(x, W, b, y)$ or $J(\hat{y}, y)$ denote the cost function.

Appendix B

Technical Configurations

B.1 Clusters in Darknet API Setups

The value used for batch and subdivision is 64 for both. While the starting resolution for input images was 416×416 . The maximum number of batches for the whole training is set to $2000 \times \textit{number_of_classes}$ but not less than 6000 so it was chosen accordingly. The number of channels was kept 3 as for RGB. The dataset was artificially expanded by setting to 1 the data augmentation parameter “mosaic=1”. Another change required is adjusting the values of classes and filter numbers. The class number is 2 in this case and following equation (B.1) the filter number would become 21.

$$\textit{Filters_Number} = (\textit{Number_of_Classes} + 5) \times 3 \quad (\text{B.1})$$

These adjustments should take place in the convolution layers right before YOLO and there are three of them. The anchor coordinates were not changed, though the training would be running in 2 classes. The main reason for that is the similarity in shape for existing classes. Both cluster’s kinds can fit into rectangular to squarish boxes of random dimensions. Other parameter adjustments in the configuration file are specifically done for this project only. These served the purpose of a higher resolution detection on small scale instances. Some Single_Pion clusters in the data were significantly small (i.e., occupying barely 2 to 3 pixels in input images), and needed an exceptional attention. To do those two steps were followed:

- To improve small scale detection in general:
 - In line 895 under [upsample] “stride” is changed to 4
 - In line 898 under [route] “layers” is changed to 23
 - In line 993 under [convolution] “stride” is changed to 4

- To improve small scale accuracy on detection boxes, 3 parameters are added to all three [yolo] layers:
 - ignore_thresh = .9
 - iou_normalizer = .5
 - iou_loss = giou

The following command was used to launch the run:

```
./darknet detector train data/obj.data cfg/yolo-obj.cfg yolov4.conv.137 -  
map
```

The “*map*” argument that is added at the end is optional and allows to take advantage of the validation dataset. The mAP indicator would signal that whilst it is increasing, the training should keep going. When it starts flattening up, this is the point where training should be stopped and evaluated. The rest of the command is used to direct the model to the paths of the data-set’s main file “obj.data”, the custom objects configuration file “yolo-obj.cfg” and the initial weights “yolov4.conv.137”, respectively.

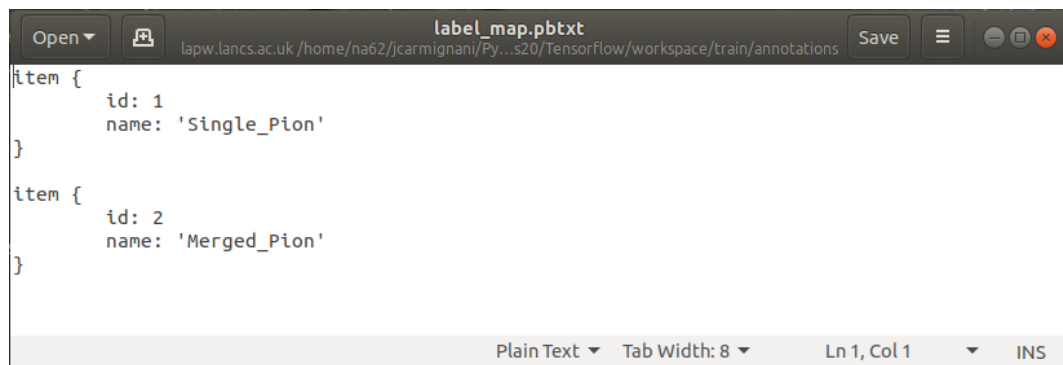
B.2 Clusters in Tensorflow API Setups

Under the main “Tensorflow” folder, a new “workspace” is created. In this last one should be stored all the training set-ups. Now under “workspace”, a “train” folder is added to keep files consecrated for training. In it should be added several folders:

- “annotations”: this is where the label map is stored.
- “exported-models”: Here should be stored the exported versions of our trained models.
- “images”: This folder contains all images in the dataset,
 - “images/train”: for which will be used for training.
 - “images/test”: for which will be used for testing.
- “tf_record”: This folder will be used to store all Tensorflow .tfrecord files which contain the list of annotations for both training and test images.

- “tf2_models”: Here all the sub-folders of training jobs would be added. Each one of these would contain the .config training pipeline configuration file, as well as all files generated during the training and evaluation of the model.
- “tf2_pre-trained-models”: This folder would contain the downloaded initial weights or checkpoint of the pre-trained models.

The annotations are encoded in .tfrecord binary files and should be stored in “tf_record” folder as train.tfrecord and val.tfrecord. The label map was then created and stored in an “annotations” folder; it has the extension .pbtxt. It is a text file that maps each of the labels used to integer values. This would be used both in the training and detection processes. Below in figure B.1 is showing the label map used for the particle physics dataset. Any name for the labels could be added if they are fixed for training and detection phases.



```

label_map.pbtxt
lapw.lancs.ac.uk/home/na62/jcarmignani/Py...s20/Tensorflow/workspace/train/annotations
Save
item {
  id: 1
  name: 'Single_Pion'
}
item {
  id: 2
  name: 'Merged_Pion'
}
Plain Text Tab Width: 8 Ln 1, Col 1 INS

```

Figure B.1: The label map for the two classes.

As mentioned earlier Transfer Learning and Fine-Tuning would be used on top of pre-trained models from TF2 object detection model zoo¹. For the goals and purposes of this case study three state-of-the-art hybrid models were picked, trained, and used:

1. “SSD MobileNet v2”: MobileNetV2 for Backbone and SSD as object detector.
2. “SSD ResNet50 V1 FPN”: ResNet50 for Backbone and a combined object detector of SSD (Head) and FPN (Neck²)
3. “Faster R-CNN Inception ResNet V2”: Inception and ResNet as double Backbone with Faster R-CNN object detector on top.

¹https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

²OD extensions are called Necks

Each of the three models checkpoints and configuration files were downloaded and added in “tf2_pre-trained-models” folder in three separate sub-folders. Each of these is named after the model of interest. For instance, the tree showing only “SSD ResNet50 V1 FPN” model sub-folders would be like:

```
train/
|---...
|--tf2_pre-trained-models/
      |--ssd_resnet50_fpn/
            |--checkpoint/
            |--saved_model/
            |--pipeline.config
|--...
```

And the others would be arranged similarly.

Few essential changes were made to the configuration files “pipeline.config” for all three models:

- Line 3: “num_classes” should be adjusted o the number of label classes used
- Line 131: under train_config , “batch_size” should be increased/decreased depending on the available memory (higher values require more memory and vice-versa)
- Line 161: “fine_tune_checkpoint” should be filled with the path to checkpoint of pre-trained model
- Line 167: “fine_tune_checkpoint_type” should be set to “detection” since the goal is to be training the full detection model
- Line 172 & 182: “label_map_path” should be filled with the path to label map file
- Line 174: “input_path” should be filled with the path to training TFRecord file
- Line 178: “metrics_set” should be set to the type of metrics needed for validation and performance checks. For the dataset in hand, the most suitable choice was “oid_V2_detection_metrics” which uses AP for both individual classes and the total mAP³

³https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/evaluation_protocols.md

- Line 186: “input_path” should be filled with the path to testing TFRecord file

In addition to the above-mentioned, a few optional changes were made as well:

- Line 147: under “optimizer {}” adam_optimizer was added with a customized scheduling dynamics for the learning rate in the following fashion:

```
optimizer {  
  
  adam_optimizer: {  
  
    learning_rate: {  
  
      manual_step_learning_rate {  
  
        initial_learning_rate: .0002  
  
        schedule {  
  
          step: 4500  
  
          learning_rate: .0001  
  
        }  
  
        schedule {  
  
          step: 7000  
  
          learning_rate: .00008  
  
        }  
  
      }  
  
    }  
  
  }  
}
```

```
step: 10000

learning_rate: .00004

}

}

}

}

use_moving_average: false

}
```

After the configuration setups are done accordingly, the main script to launch the training “model_main_tf2.py” should be copied and launched from the main training directory (in our case train/). The command used to launch training is the following:

- `python model_main_tf2.py --model_dir=models/my_model --pipeline_config_path=models/my_model/pipeline.config`

And the arguments are:

- `model_dir`: should point to the path of model’s created directory
- `pipeline_config_path`: should point to the path of the model’s configuration file

The command used to run the evaluation process is:

- `python model_main_tf2.py --model_dir models/my_model --pipeline_config_path models/my_model/pipeline.config --checkpoint_dir models/my_model`

The only extra argument here is:

- `checkpoint_dir`: which should point to the location of saved checkpoints

The results are stored in the form of tf event files (`events.out.tfevents.*`) inside `models/my_model/eval.0`. These files can then be used to monitor the computed metrics, using Tensorboard⁴.

The command used to launch Tensorboard is:

- `tensorboard --logdir=models/my_model`

The above command will start a new TensorBoard server, which (by default) listens to port 6006. Now typing `http://localhost:6006/` in the browser's address bar should be enough to visualize a detailed dashboard with all training/validation-related charts and plots.

⁴<https://www.tensorflow.org/tensorboard>

Appendix C

Robotics Applications

C.1 Introduction

In this chapter, the work presented is a collection of independent NN applications in the field of Robotics. Presented next is a set of trials on models trained to classify and identify objects related to a grasping robot. The work has been done in collaboration with Lancaster University Engineering Department and National Nuclear Lab (NNL). The desired object's nature (i.e., end-effector/manipulator's grip) not having any specific datasets online, one had to be prepared manually from scratch. Manual annotation and labelling of a collected/created dataset are time consuming tasks. A lot of attention is spent on keeping the dataset balanced in negative examples and in multi-class distributions. A non-equivalent dataset can have a significant impact on the training from making it hard to train to extremely biasing the predictions. The training might result in a low mAP and might lead to over-training as well. Additionally, a well-balanced dataset is linked, empirically speaking, to lowering the number of False Negatives (i.e., the most dangerous false predictions) like what was done in the particle physics applications.

CVAT tool is used again for online video and image annotation for computer vision purposes. Within this tool, annotations can be done in several ways depending on the purpose. For instance, in Object Detectors (e.g., YOLO etc...), boxed annotations were needed. Boundary box style annotation has been the method followed in both projects for object detection and computer vision. Only in these Robotics applications, another method was also an option which is to create a simulated dataset out of the model of the desired objects (e.g., CAD model).

C.1.1 End-Effector Data Box Annotations

The end-effector dataset is a collection of video frames, taken from portable cameras. Two feeds were taken on scene in the Engineering lab. Camera positions and rotations were carefully chosen to cover most angles over, around and under the end-effector of Brokk40 manipulator robot. To make the dataset even more robust some frames were taken from online videos of the Brokk40 manipulator¹. Using CVAT, the frames in the dataset were marked where a box was added around end-effectors and the right class label was also added. To train the model reliably end-effectors part of the images must fall within the box in all frames. The video that was available for the project yielded a total of 4556 frames which is barely sufficient for basic training purposes. The quality and quantity of the dataset created was negatively affected by the pandemic because a larger number of images is required for an ideal dataset. Image variety and range are also beneficial. This includes displaying the end-effector from several different angles, lighting conditions, distances, positions and while completing different actions. A few thousand frames per class are ideal as a minimum to train a vision system with confidence based on this specific application. However, the frames in the video footage are highly repetitive but as it was the only data available, an attempt was made to make full use of it.

The problem was mitigated in two ways. The first was applying data augmentation while training to artificially add a variety to the dataset. This allowed for greater use to be made of the data-set available. To further mitigate this limitation, an additional Brokk40's live-in-action video was found online and provided an additional 1003 frames. The benefit of this was a larger and more robust dataset because the video showed clear views of Brokk40 functioning in diverse backgrounds, whilst in motion and grasping objects.

Each frame contains an annotation and a label, as shown in figure C.1. The advantage that CVAT presents is that, once the annotation is complete, it can export a dataset in a multitude of formats compatible with different vision algorithms.

C.2 Mobile Activity project

Mobile Activity is a Keras Tensorflow code arranged and publicly available on the GitHub² page. The project consists of a standalone Python code using OpenCV and MobileNetV2 pre-trained weights on COCO dataset³. The computer vision method

¹<https://drive.google.com/file/d/1eRT7HUrOsSfhqsm-WhPfig8nnq6rF-Lf/view>

²https://github.com/Carmigna/mobile_activity

³www.cocodataset.org



Figure C.1: Boxed annotation of Brokk40 in CVAT

applied here is Fine-Tuning in Transfer Learning described in detail in earlier chapter. A model is trained to recognize main activities of a robotic end-effector (e.g., Grab, Rest, Release etc...) from mere images or video frames. It requires a few set-up steps before launching the training. One of which is the installation of Keras library with Tensorflow back-end and preferably adaptable with GPU for tensors' computations. Following the steps in “mobile_activity” Git page, instructions for data preparation, training and testing on videos and real-time live-feed cameras are stated clearly. Instructions to install Tensorflow⁴ with an Nvidia GPU are given as well with all the steps to download and install all the necessary packages.

The first practical step is to prepare a suitable dataset that contains the activities that we wish the model to learn from images and classify. For the end-effector, among the many potential activities that one could teach the model to recognize are “rest” and “grab”. Therefore, a folder must be created for data and within this, another folder in which all the images can be placed. Each activity shown in a single image should

⁴<https://github.com/Carmigna/tensorflow>

also be annotated. Once the data has been prepared the database can be created for training and testing purposes. They provide scripts that would add the paths to data, automatically create two datasets (one for training and another for validation and testing) and load the model to launch training from the main directory of the code. (Please check Appendix D for additional preparatory and functional details).

After training is done, the testing phase begins. The code includes a couple of scripts to do the testing as well. Depending on if the test is carried over via a sample video or a live streaming camera, the call for test scripts would be a bit different.

Each case requires a command (**A** or **B**) since the two scripts have different arguments as well.

If the test is on video, the following command is used:

```
A: python predict_video --model "activity.model" --label-bin "lb.pickle" --input "example_clip" --output "output_video_128avg" --size 128
```

Of which the arguments would be:

- *model*: for adding path to the saved fine-tuned model CNN (i.e., graph, weights, and classifier). The code automatically saves a copy in model directory named "activity.model"
- *label-bin*: for adding path to a .pickle file which is a serialized label binarizer. It is a binary file that contains the class labels, and automatically saved in model directory as "lb.pickle"
- *input*: for adding path to a sample video clip for testing, an instance is "activity.mp4"
- *output*: for adding path to the filename of an output video, that once played, would be showing the predicted activity with its fidelity score
- *size*: for adding the number of frames that the fidelity scores of predictions would be averaged upon.

In case the test is done on live-stream camera, the following command is used:

```
B: python predict_cam --model "activity.model" --label-bin "lb.pickle" --size 128
```

Here the arguments used are already explained in command **A**. Only this one would not save a video output. Hence, it would display the activity's name and label

on top of the live streaming feed (the fidelity score can easily be added as well). In both cases, the test is average on the predictions of 128 frames and displaying the activity's class for every other 128 frames. This parameter is chosen in a Goldilocks zone that made sense for the dataset in hand. It is not necessarily the same for other classifications. So, the point is that it can easily be tuned from the `-size` argument to get the best score for the average classification over number of frames.

The end-effector dataset is a collection of video frames, taken from a portable HD camera. Two videos were taken on scene in the Engineering lab at Lancaster University (Engineering Dept.). The camera positions and rotations were carefully chosen to cover most angles over, around and under the end-effector of Brokk40 manipulator robot existing in the lab. To make the dataset even more robust some frames were taken from online videos of the Brokk40 manipulator. In this code we only used the images from the dataset while different annotation formats are added later for the other parts related to object detection.

After the model was thoroughly trained, it was tested on an online video. This consists of an open-source advertisement available on YouTube for the Brokk40 manipulator and others as well. The goal was to test the accuracy of identification of the activities of these end-effectors. The target was not trivial since the model would not only have to be able to recognize the end-effector within the frames but also it should be able to classify its activities. The video was challenging enough because it included a range of other end-effectors in addition to the Brokk40. Moreover, it showed these end-effectors in different environments and completing several actions and tasks making it a versatile test video.

Figure C.2 below shows the correct identification of the end-effectors activity to be at rest while in good lighting and from two different angles which is a promising sign for the "mobile_activity" code.

Figure C.3 below demonstrates that an end-effector had been identified and its activity classified correctly and successfully. The model can understand now that an end-effector is within the borders of a certain collection of consecutive frames. Additionally, it is detecting the activity that is being completed by the manipulator which would provide a further understanding if applied on a drone.

A strength of the mobile_activity system is shown in figure C.4 and C.5 . These figures show the correct identification of both the end-effector and the activity it is completing. However, these images are different to the frames used while training and contain unique objects with the same general features though. That makes the

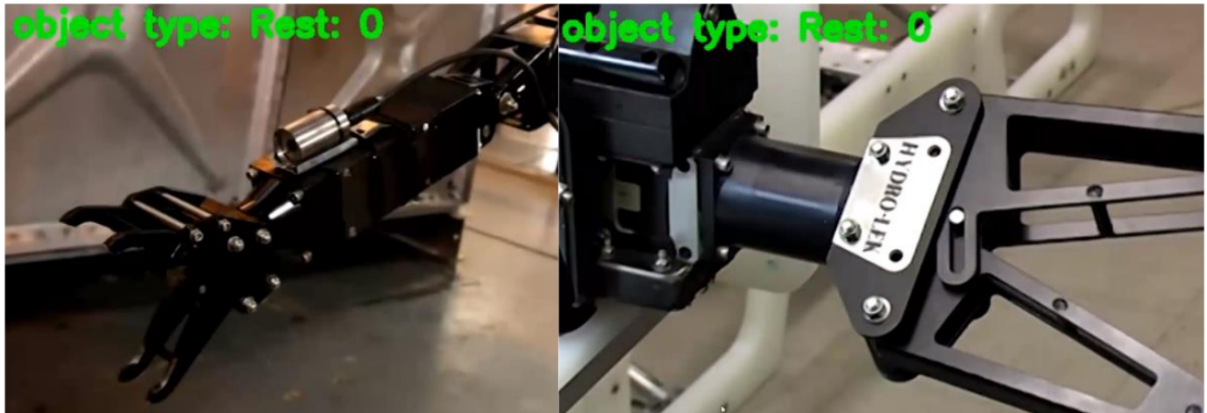


Figure C.2: Classification of the Brokk40 at rest.

results quite surprising and yet promising. Figure C.4 shows an end-effector other than the Brokk40, but the model can correctly classify its existence and activity. Here another Background class where there is no end-effector can be easily added and make the predictions even more robust. Figure C.5 is also showing a correct classification despite the low lighting in these frames.

An example of the model showing understanding between two activities is proven as well in figure C.6 below. The two frames show the Brokk40 transitioning from one activity to the other and when doing so, the classification changes to the correct one.

The above-mentioned sample results show that the fine-tuned MobileNetV2 network can be a potential method for object classification used on a drone. However, there were limitations and some cases showed confused predictions. During systematic testing, the end-effector's activity was sometimes incorrectly classified. Additionally, in some instances the model would also mistakenly classify the frame to have the "grab" activity while an end-effector was missing. As mentioned before, that might be improved by adding more negative samples as background (or "not end-effector") in the training. A richer and more robust dataset is always a working method to reduce the number of false positives. While refined training would on the other hand help reduce the false negatives (i.e., in our case "no detection" where there should be one). However, due to time constriction in preparing the dataset and to the outburst of the pandemic, reevaluating and improving the dataset could not be done within a realistic timeframe.

An original code from Robot Ignite Academy named "Domain Randomization" was

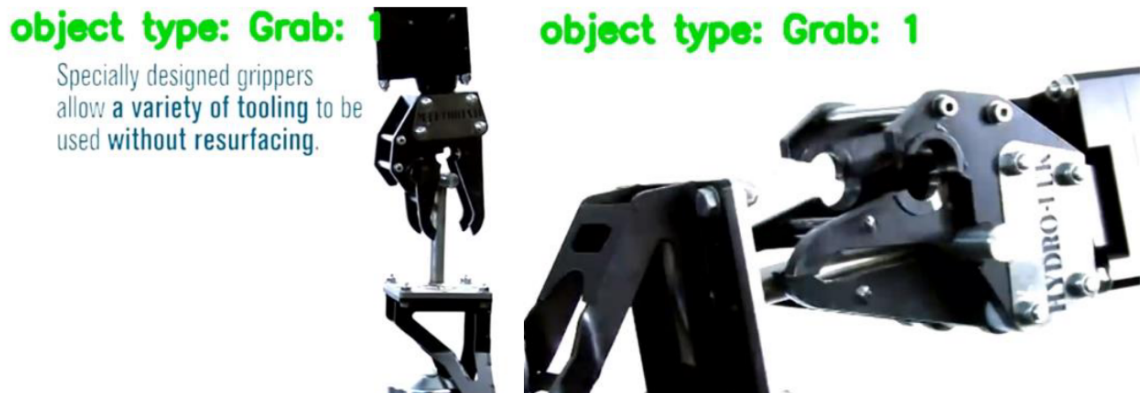


Figure 160: Classification of the Brokk40 whilst grabbing

Figure C.3: Classification of the Brokk40 whilst grabbing.

an inspiration for the starting point of this section [11]. In the “mobile_activity” code, the steps to prepare the data and call of the MobileNetV2 architecture for transfer learning were taken from “Domain Randomization” but big differences exist though. First, the classification objective of the code is different than what was initially intended originally as a simple in plane pose estimation. That led to a fine-tuning in “mobile_activity” instead of a features’ extraction in “Domain Randomization”. Then, the data’s functionalities, binarization and labelling followed a completely original pattern. In addition to all that, the GPU activation and involvement in Tensorflow calls and computations are extremely specific and run the training at a much faster rate.

Moreover, an issue was encountered whilst using the initial code, it is its compatibility with (Robot Operation System) ROS⁵. It is a tool that aids in building and using robot applications. However, ROS works mostly and much more efficiently with Python 2. While OpenCV and other vision systems on the other hand require Python 3 to work. As these are prerequisites to using Tensorflow with this specific computer vision application, any stage of the work using ROS in Python 2 was met with errors and caused difficulties.

Initially, to overcome this problem, an attempt was made to use Python 2 on the system whilst setting up a virtual environment of Python3 so that ROS would be

⁵Please refer to <http://wiki.ros.org/> for a summary of ROS functionalities, domains, and environments.



Figure C.4: Correct classification of a different end-effector and its activity.

able to run the code. Instructions followed to do that are available online at [12]. All the steps were followed properly, however the guide was not complete and lacking a lot of details to prepare an effective usable and working environment for OpenCV in ROS. Another tool to test could have been by using simulated data from a simulation environment. A practical tool is (NDDS “Nvidia Data Synthesizer”) for simulating data for training and can be easily expanded for 6D pose estimation algorithms.

Nevertheless, “mobile_activity” code was prepared to work solely with a Python3 environment without the need to operate within ROS. It still allows for the training and using the weights for testing inference within a UAV processor (e.g., Nvidia Jetson). A terminal and a ssh connection must be used to execute these tasks such as running the “mobile_activity” detection with a live-stream mode (or command B) using the best trained weights of the model.

Another objective that was sought through the project is to make the vision system not only recognize the end-effector but locate its coordinates as well. Since the first attempt lacked any object detection coordinates or bounding boxes, the vision system had to be upgraded by using Object Detection Algorithms. Although knowledge of the classification and activity are useful to a drone, without information regarding the coordinates of the end-effector, the vision system would be limited in its practicality. Since the best approach of a vision system would be a coherent inseparable code that works end-to-end and achieving the objectives, the project took path of best available



Figure C.5: Correct classification of the Brokk40 and the activity in poor lighting conditions.

Deep Learning state-of-the-art object detection algorithms. YOLO, RCNNs and SSD were used in this project on different platforms (API) and the results were compared.

C.3 End-effector detection using Darknet API

The dataset exported here from CVAT is YOLO specific, a dataset within which is added first a folder that contains all the essential material to fine-tune YOLO algorithm and train it on custom objects data. It includes all the input frames (if from video) or images in a JPG or PNG format. (Please check Appendix D for setups and technical details). The command that should be used to start the training is showing below with its default arguments:

```
./darknet detector train data/obj.data cfg/yolo-obj.cfg yolov4.conv.137 -  
map
```

The “*map*” argument that is added at the end is optional and allows to take advantage of the validation dataset. This results in a mAP plot being drawn as shown in figure C.7. The mAP indicator would signal that while it is increasing, the training should keep going. When it starts flattening up, this is the point where training should be stopped and evaluated. The rest of the command is used to direct the model to the paths of the data-set’s main file “obj.data”, the custom objects configuration file



Figure C.6: Classification showing transition in activity - (left) “Rest” and (right) “Grab”

“yolo-obj.cfg” and the initial weights “yolov4.conv.137”, respectively.

Figure C.7 shows the results of training on the initial dataset with yolov4 custom starting weights. It shows that with approximately 1500 iterations, the loss function has dropped a considerable amount and mAP has reached a value of 98%. This value was higher than anticipated and implies that training went a bit off balance which is an indication of over-training. However, the model was able to generalize well and predict with acceptable but not ideal accuracy. Figure C.8 demonstrates the strength point of the trained model as the object detection was tested on an unseen video. It contained an end-effector that differed slightly from the Brokk40. This test showed that the model was able to recognize this new end-effector thereby demonstrating some effectiveness of the trained model. Moreover, this end-effector was identified with a high-fidelity score. This is notable as it is grasping an object, an instance that was not included in the dataset, and despite this the model is still identifying and classifying the object correctly.

Although training on the initial dataset yielded a 98% mAP value, there were some clear indications of over-training when testing. In some cases, items other than the end-effector were wrongly identified as False Positives (FP). Both cases are shown in figure C.9. It should be noted though that the fidelity percentage or certainty score when identifying a FP was low relative to when detecting a real end-effector.

It is likely that an FP is due to the limited dataset that was available. The video footage of the end-effector that was annotated and labelled contained mostly close shots of the object and did not display the arm in motion. Due to this, the model was

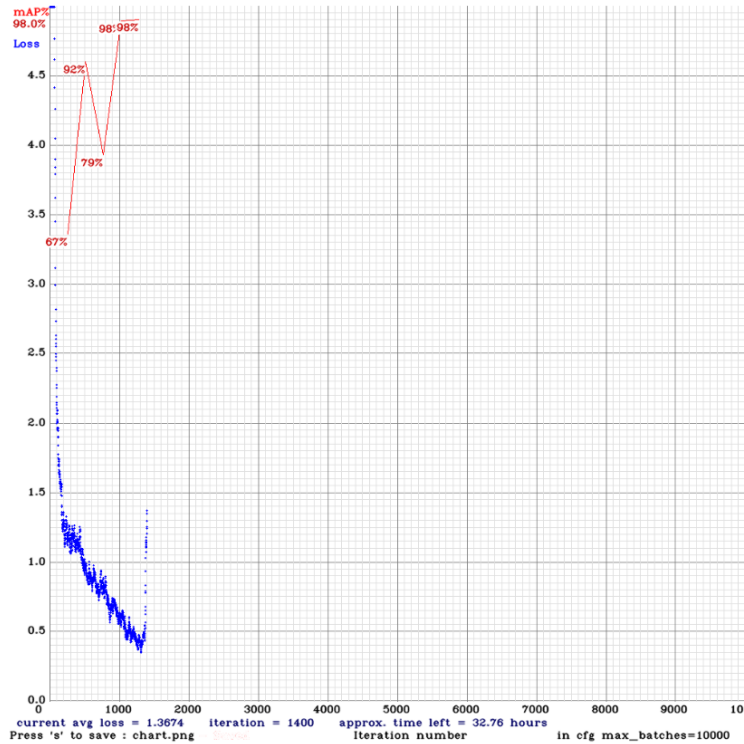


Figure C.7: Results of initial training showing the loss (*blue*) and mAP (*red*) plots

detected less accurately when the end-effector is displayed from a distance or while in motion. Moreover, the FP in figure C.9 (right) displays relative similarities in the components of certain frames within the dataset. This is apparent for instance in figure C.10 below. It is showing the end-effector to have similar colors to the FP in figure C.9 and they both contain bolts that could be the cause of confusion. Therefore, making the dataset more robust was necessary to improve the model’s accuracy.

To overcome this problem the expended dataset was used. This one included an online video of the Brokk40 on a Hydrolek arm. It contained clear shots of the end-effector from a distance and displaying motion of the arm. Therefore, the new footage was annotated and labelled as described earlier and yielded an additional 1003 images. These were mixed with the original ones, then exported as two final datasets, one for training and another for validation. The object detection model was retrained on the expended dataset whilst utilizing the best weights from the initial training with a goal to further boost accuracy of the second training. Retraining picked up from where the first session stopped iteration-wise. The final model was then trained on a combined dataset containing approximately 6500 images and its validation yielded a final mAP

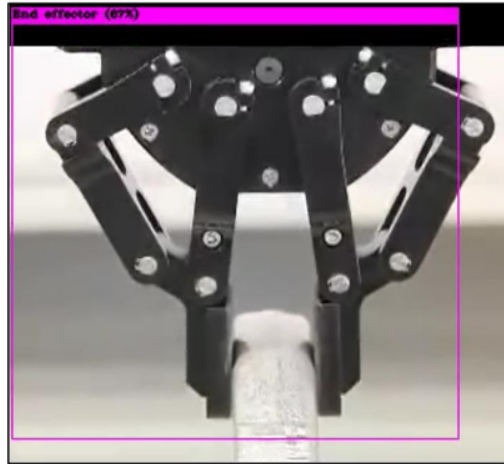


Figure C.8: Correct classification of a strange kind of end-effector that is grasping an object

of 96.5%. Although this is slightly lower than the initial mAP, the training had a much wider range of images to scope and so was less likely to have FP. Fortunately, it was possible to reduce the effect of over-training by adding this variety to the dataset.

Another method used to reduce the likelihood of incorrect classifications was altering the detection threshold within YOLO. By default, it is set in the algorithm to 25% however a command can be used to change this value accordingly by the user. Effectively, increasing the detection threshold on fidelity score would help reduce the error on detection by getting rid of the uncertain ones. In other sense, an object now will only be classified as an end-effector if the model is more confident that an end-effector is present. As the test showed, in most cases an end-effector was recognized with 80_90% certainty. Therefore, increasing the detection threshold in this case will not adversely affect the identification of an end-effector, instead it will only reduce the chances of false detection thereby improving the model's reliability. Therefore, the threshold on detection was set to 77% by adding the “-thresh 0.77” argument to the test command as shown below. Here are few commands used for testing:

To test on images: `./darknet detector test data/obj.data cfg/yolo-obj.cfg backup/yolo-obj_best.weights images.jpg -thresh x`

To test on videos: `./darknet detector demo data/obj.data cfg/yolo-obj.cfg backup/yolo-obj_best.weights video.mp4 -thresh x`

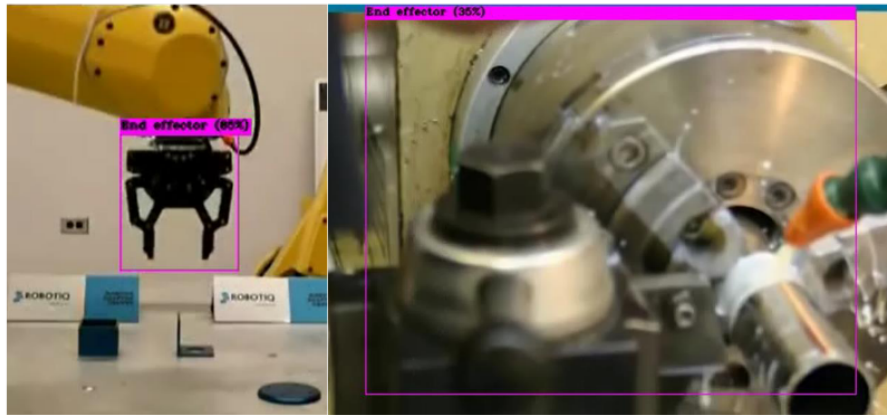


Figure C.9: Correct (*left*) against incorrect (*right*) identifications

To test on cam: `./darknet detector demo data/obj.data cfg/yolo-obj.cfg backup/yolo-obj_best.weights -c 0 -thresh x`

The arguments added to the above commands are explained below:

- “cfg/yolo-obj.cfg” -Path to the configuration file adjusted for our custom object
- “backup/yolo-obj_best.weights” -Path to the best weights from the training within “backup” folder
- “-thresh x” -The value of detection threshold will be set to “x”
- “images.jpg” -Path to test images
- “video.mp4” -Path to test video
- “-c 0” to test on cam

To save the output on a separate image or video file filename.extension we add:

- “-out_filename filename.ext”

Additional especially useful feedback, that can be received as an output of the model, is coordinate display of detection boxes from every frame. This could provide localization data for an object in a 2D plane at a certain distance from the camera. Plus, a percentage of accuracy can be displayed clearly in the terminal. Figure C.11 shows an instance of coordinates feedback for objects detected within a terminal. To



Figure C.10: Boxed annotation of Brokk40

receive such digital feedback, in a separate terminal, the below commands are entered:

On video: `./darknet detector demo data/obj.data cfg/yolo-obj.cfg backup/yolo-obj_best.weights -ext_output test_video.mp4 -thresh x`

On Cam: `./darknet detector demo data/obj.data cfg/yolo-obj.cfg backup/yolo-obj_best.weights -ext_output -c 0 -thresh x`

The coordinate feedback received separately in the terminal is beneficial as it shows that this information can be extracted from the vision system with ease. This data can then be sent to the receiving communication system. This final feature is a major step that can be exploited further in future research independently.

C.4 DetectNet using Jetson-Inference API

Nvidia DIGITS tool is a user-friendly Graphical User Interface (GUI) to prepare datasets and train a custom object detector from a pre-trained model (e.g., SSD_MobileNetV2). The easiest way to use DIGITS is by installing Nvidia's Docker

```

Objects:
End effector: 100%      (left_x: 398  top_y: 176  width: 136  height: 268)
End effector: 75%      (left_x: 377  top_y: 13   width: 134  height: 183)
FPS:34.2              AVG_FPS:28.3
Objects:
End effector: 99%      (left_x: 404  top_y: 181  width: 135  height: 256)
End effector: 64%      (left_x: 379  top_y: 14   width: 131  height: 184)
FPS:34.2              AVG_FPS:28.3
Objects:
End effector: 99%      (left_x: 405  top_y: 181  width: 130  height: 255)
End effector: 65%      (left_x: 378  top_y: 13   width: 133  height: 184)
FPS:33.9              AVG_FPS:28.3
Objects:
End effector: 90%      (left_x: 414  top_y: 171  width: 142  height: 273)
FPS:34.0              AVG_FPS:28.3
Objects:
End effector: 99%      (left_x: 426  top_y: 177  width: 131  height: 261)
FPS:34.0              AVG_FPS:28.3
Objects:
End effector: 99%      (left_x: 437  top_y: 186  width: 127  height: 244)
End effector: 43%      (left_x: 386  top_y: 10   width: 135  height: 194)
FPS:33.9              AVG_FPS:28.3
Objects:
End effector: 100%     (left_x: 436  top_y: 183  width: 128  height: 252)
End effector: 78%     (left_x: 387  top_y: 6    width: 136  height: 201)
FPS:34.1              AVG_FPS:28.3
Objects:
End effector: 99%      (left_x: 432  top_y: 176  width: 122  height: 265)
End effector: 28%      (left_x: 390  top_y: 13   width: 132  height: 186)
FPS:34.2              AVG_FPS:28.3
Objects:
End effector: 97%      (left_x: 423  top_y: 174  width: 136  height: 265)
FPS:33.8              AVG_FPS:28.3
Objects:

```

Figure C.11: Coordinate feedback of the object detected shown within a terminal

image (“nvidia_docker2”) then pull and run “digits” after registering to NGC online⁶. Running DIGITS has some requirements: Ubuntu operating system, Nvidia graphic card drivers, CUDA drivers and a CUDA capable Nvidia graphic card. The DIGITS instance running in this project was a docker image linked to a laptop with Nvidia GTX 1050 Ti graphic card.

The power of Nvidia DIGITS is that it allows the use of other practical Nvidia tools like “Jetson-Inference” which will be introduced later. The importance of jetson-inference is that it allows the user to use and customize an object detector like YOLO (only in functionalities and objectives) called “Detectnet”. It is an object detection network implemented using Nvidia’s branch of the popular Caffe deep

⁶All the detailed instructions can be found here: <https://github.com/dusty-nv/jetson-inference/blob/master/docs/digits-setup.md>

learning framework also called “nvcaffe”. Jetson-Inference API through DetectNet could broadcast a node in ROS as well to make robotic use of the trained object detection model. This distinctive feature makes the use of this tool a very essential part of this project.

Preparing datasets is the first thing to start with. As was mentioned earlier the end-effector dataset was prepared with CVAT however it can be exported in many different common formats. One of which is MS COCO format which is one of the most common formats available. It is related to COCO dataset which is mentioned before. On the other hand, Nvidia’s DetectNet would be the object detection model adopted and used in DIGITS. The DetectNet configuration follows a specific format, that is terribly like MS COCO, it is called KITTI. Since CVAT does not export data in this specific format, a script available on the jetson-inference API was found to do that pretty easily⁷. As was mentioned many times earlier, labeling images for object detection is a process where files are created that contain descriptions about regions of interest on images. Besides class names and bounding boxes, KITTY format can accept some other parameters like truncated, occluded, etc. These extra ones can always be omitted or set to zeros⁸.

Next, as figure C.12 is showing, under networks tab “Custom Network” was selected and “Caffe” tab chosen underneath. Under “Pretrained model(s)” the path to pretrained weights was added. It is especially important to use these, otherwise training will take ages to be completed. GoogLeNet backbone weights were used and are available online⁹.

Then, modification should be made to the “prototxt” file. Prototxt is a Caffe file describing the structure of a neural net. DetectNet file is available online¹⁰ so it was downloaded, copied, and pasted in the “Custom Network” Caffe model text window (figure C.12). This is meant for one object custom detection, but it can easily be adjusted to 2 or more classes. Additionally, Nvidia provides another one for 2 custom objects and it would be a good start for multi-class object detection¹¹. Since in this project only a single class is used, the default file worked well without many modifications. The only adjustment made was replacing the correct input image size. A resolution of 640x640 pixels was used, but the default prototxt uses 1248x352. That

⁷<https://github.com/dusty-nv/jetson-inference/tools/coco2kitti.py>

⁸More on KITTY: <https://github.com/NVIDIA/DIGITS/tree/master/digits/extensions/data/objectDetection>

⁹Can be downloaded from here: https://github.com/BVLC/caffe/blob/rc3/models/bvlc_googlenet/readme.md

¹⁰https://raw.githubusercontent.com/NVIDIA/caffe/caffe-0.15/examples/kitti/detectnet_network.prototxt

¹¹https://github.com/NVIDIA/caffe/blob/caffe-0.15/examples/kitti/detectnet_network-2classes.prototxt

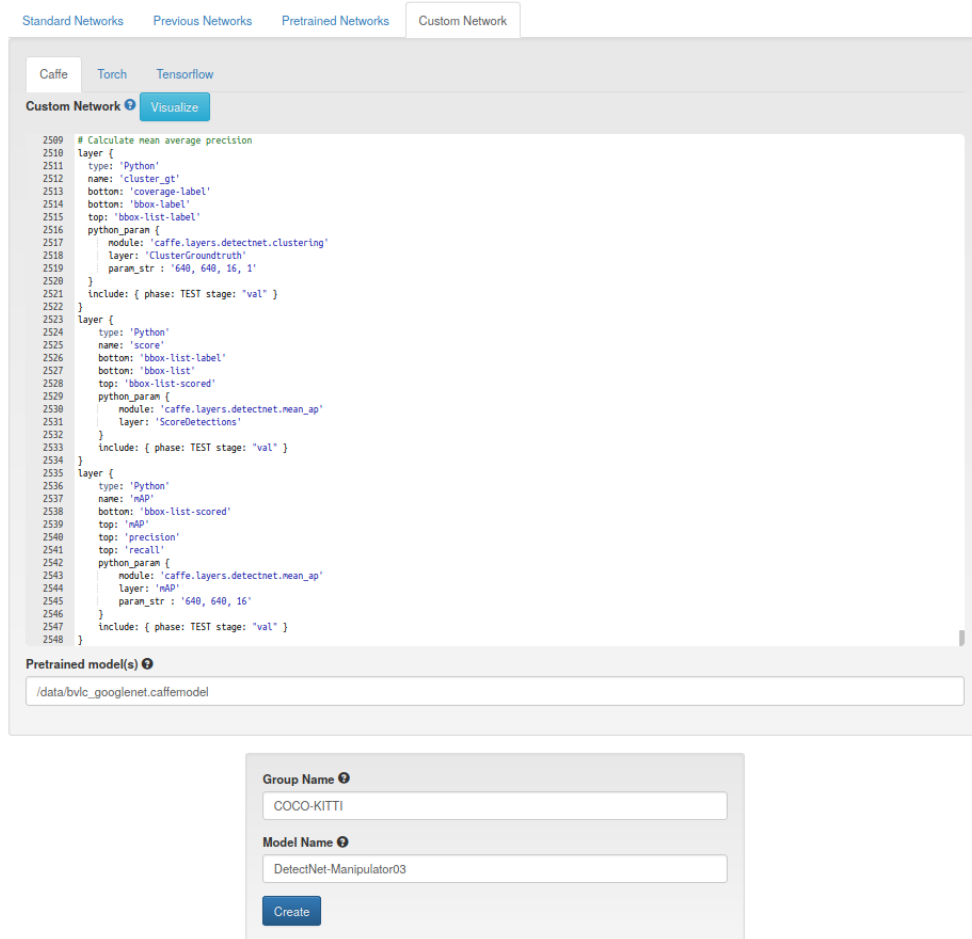


Figure C.12: Network display board

was fixed by finding and replacing occurrences of those numbers in the entire file. A Group and Model names were chosen accordingly and “Create” button to start the training. With all the parameters set up correctly, the model started training first 30 epochs to test then for 200. The first round took around 3 hours to finish with the capacities of GPU in hand. The results were promising but not ideal (figure C.13).

The test run performed at the highest peak 83% precision, 67% recall and 65% mAP. This is a clear indication that the model started identifying the object and localizing properly the bounding boxes around it. The padding used for this run was quite tight and barely covered the corners of images used in the dataset which as mentioned before were of varied sizes and inconsistent resolutions. Another run was

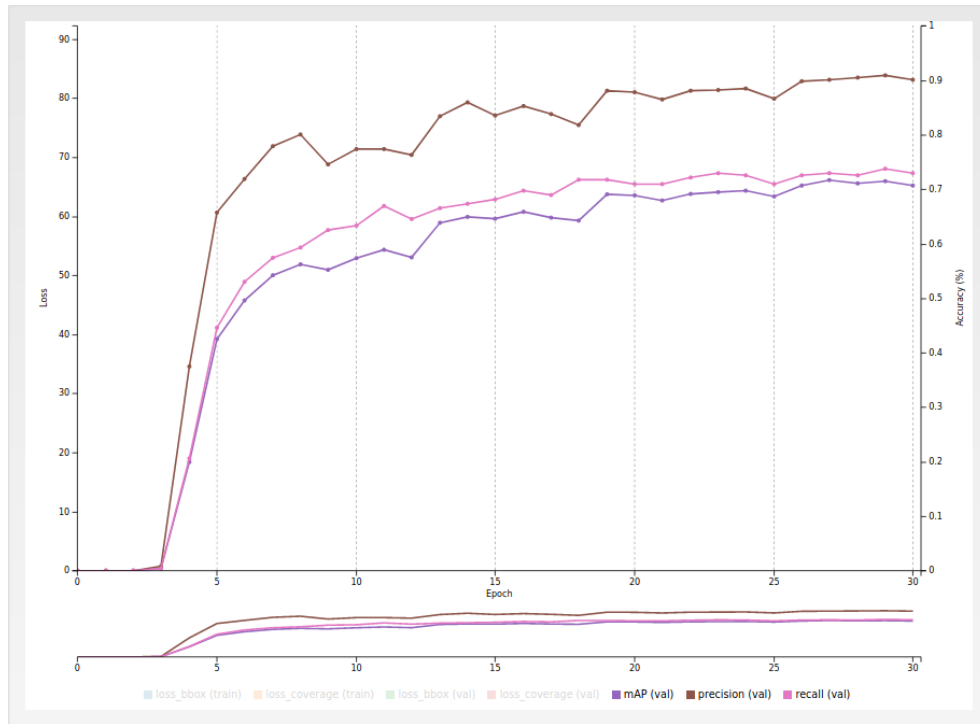


Figure C.13: First training round for 30 epochs and 2000x1500 padding

prepared by increasing the padding from 2000x1500 to 2500x2500. By that the goal was to keep the features of images centered so nothing would be lost on the edges while convoluting. This time 200 epochs were set, and better results were obtained after 24 hours of continuous training. The accuracy metrics gave at the peak 90% precision, 74% recall and 72% mAP (figure C.14). While the loss function was still dropping, slowly though, the training had to be stopped since it was taking such an extended period. However, a trend is clearly noticeable in the plot that accuracy was still increasing. That indicates a plateau was still not happening, and an extra training time could have further improved the overall performance and therefore the final mAP.

After training the model, inference should be used to practically experience how the model is detecting the object of interest. Nvidia’s official GitHub page dedicated for DetectNet inference in “jetson-inference” API¹² has a clear guide that was followed carefully to test inference of the custom model and implement it to the Jetson Nano¹³.

¹²<https://github.com/dusty-nv/jetson-inference/tree/d2bb14ba4b60bbd8fb26bc952857daa20624fa97#locating-object-coordinates-using-detectnet>

¹³Nvidia’s main robotic embedded system: <https://www.nvidia.com/en-gb/autonomous->

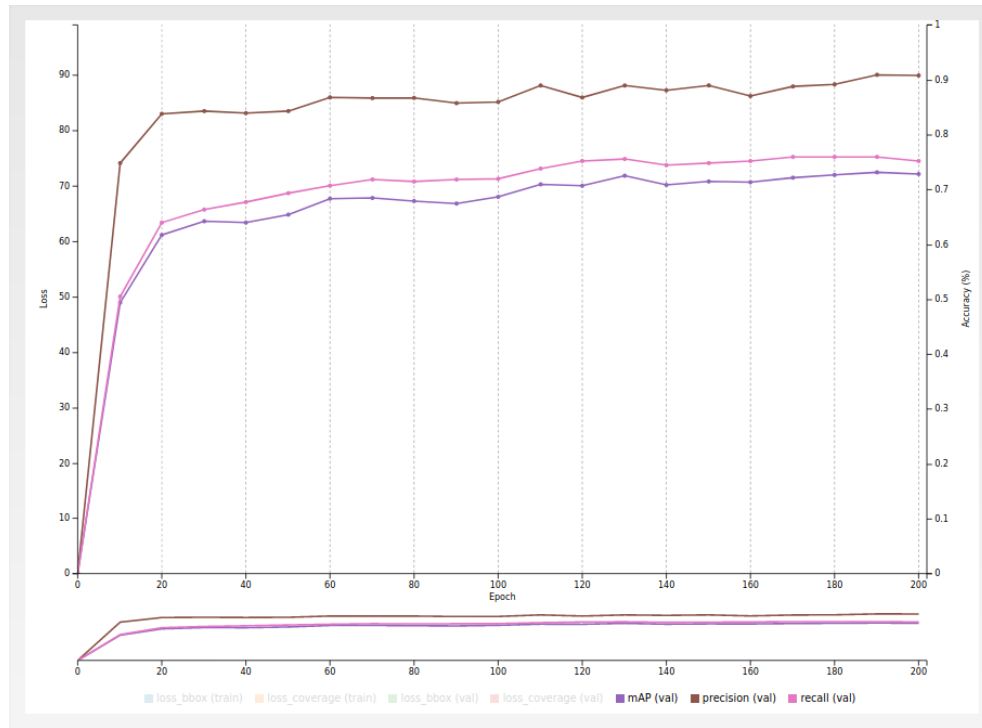


Figure C.14: Second training round for 200 epochs and 2500x2500 padding

From the same DIGITS instance running, single or multiple images inference tests could be performed. Few of the detection results are showing in figure C.15 below.

However, as seen in figure C.16, errors like what was obtained in YOLO appeared again due to data-set's limitations. The figure below is showing two instances, a FN on the *right* and a FP on the *left*. In the case of FN an end-effector was apparent while the model could not detect it. As for the FP a joint of the manipulator's arm was falsely detected as an end-effector. It is noticed also some double counting or double-box detections even on the successful ones (figure C.15 upper right and lower left). That is most probably due to the model being confused on two occurrences, while the end-effector grip is closed and on the fading frame of a manipulator in action. These two problems can be overcome in future work by making the dataset contain more images of the moving arm and grip closing on objects.

Jetson-Inference web page contains also instructions on how to implement the model directly into the embedded system and run it on images, videos, and live camera machines/embedded-systems/jetson-nano/

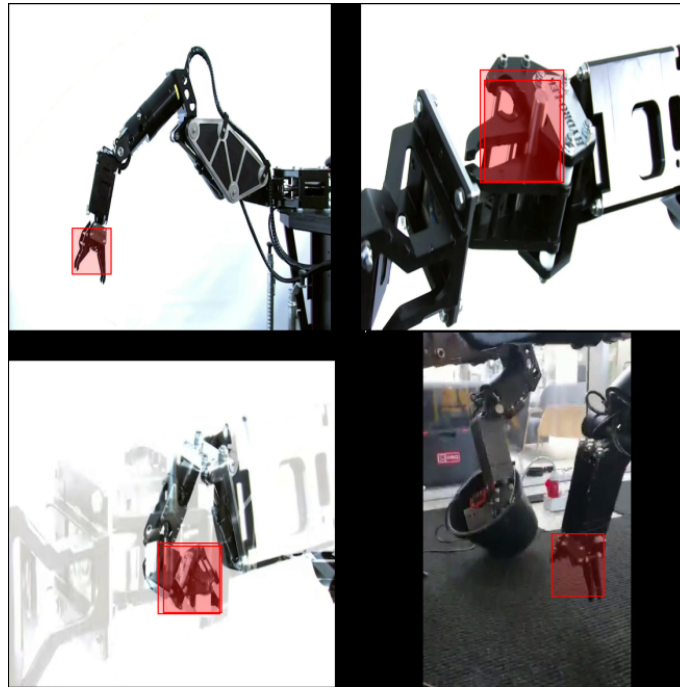


Figure C.15: Some successful detections from inference of the DetectNet model

as well. A series of YouTube video-tutorials are prepared for these purposes as well by Dusty NV ¹⁴. A super advantage for jetson-inference is that a link with ROS¹⁵ can be done. A node can be created for listening to the object detection model and display its outputs. Unfortunately, no realistic ground environment tests could be done directly on the drone (i.e., it is embedded system and camera) due to the pandemic and lockdown. Access to the engineering lab was not allowed and the final application of the project could not be achieved. However, it was proven that DetectNet through jetson-inference API is a powerful tool to be used for future research and can give valuable results through direct applications on the robotic embedded device (e.g., Nvidia Jetson Nano) and operating system (e.g., ROS). It could provide coordinate information and create bounding boxes on an object with Inference. This proved extremely useful for the application of this project on an end-effector. In addition to this, the confidence of classification is also output. “Detectnet” allows for extracting the bounding box coordinates within the terminal. Therefore, this information can be sent to a receiving communication system. Unfortunately, at the time this work was done the ROS link could only be called on images and not on live camera feed.

¹⁴<https://github.com/dusty-nv/jetson-inference>

¹⁵https://github.com/dusty-nv/ros_deep_learning



Figure C.16: Some failed detections from inference of the DetectNet model, a FP (*left*) and a FN (*right*)

An extra reason that prevented me from working directly on ROS and no time or resources were available to start developing one from scratch. That could be another proposal for future research work on the same topic.

C.5 Facebook's Detectron2 API

In this section Detectron2 API will be introduced and presented in a simple (single custom class “end-effector”) demo to show its power and speed in object detection. Detectron2 API was developed by facebookresearch¹⁶. By combining Detectron¹⁷ and maskrcnn-benchmark¹⁸ it became the easiest and most practical platform to include and use Faster R-CNN (introduced and explained thoroughly in earlier chapters). A tutorial demo in Colab, is publicly available and can be executed immediately. Instructions to install detectron2 API with all its dependencies can be found here¹⁹. Following the steps in Colab tutorial, a Mask R-CNN model was trained for a single custom object detection (in this case the “end-effector” of a robotic manipulator). Then the different testing methods will be presented with performance checks. Detectron2 API makes it possible to test inference on single images, a whole testing dataset, videos, and live web cam²⁰.

¹⁶<https://github.com/facebookresearch>

¹⁷<https://github.com/facebookresearch/Detectron/>

¹⁸<https://github.com/facebookresearch/maskrcnn-benchmark/>

¹⁹<https://www.linkedin.com/pulse/fast-way-go-faster-r-cnn-facebooks-detectron2-api-joe-carmignani/>

²⁰https://github.com/facebookresearch/detectron2/blob/master/GETTING_STARTED.md

All that is needed for this demo was an exportation of the dataset from CVAT in MS COCO format. The MS_COCO_endeffector dataset was exported and divided into two subsets one for training and other for validation in the structure showing in figure C.17.

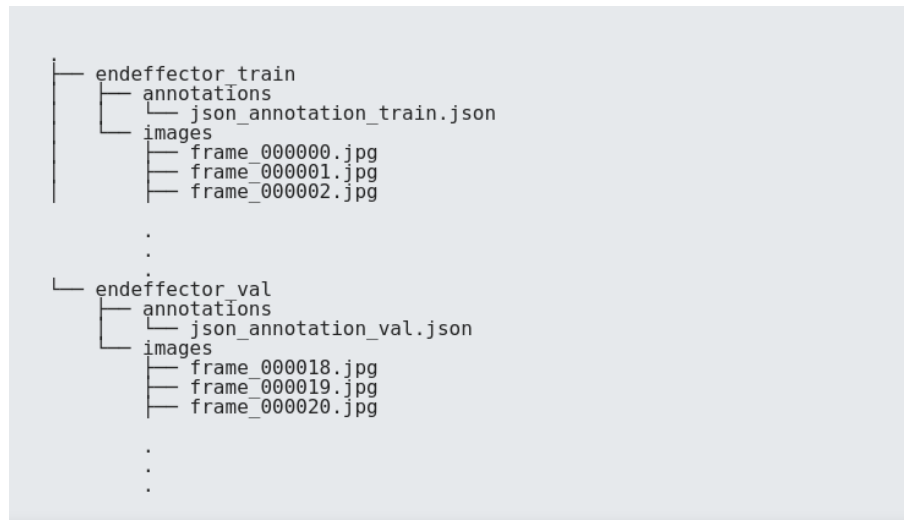


Figure C.17: Structure of the MS_COCO_endeffector dataset

Now that the datasets are prepared in a suitable format, a simple script is rearranged as to launch training directly as instructed on GitHub page²¹. After adding the datasets to main directory, `mask_rcnn_R_50_FPN_3x_train.py` script is the one to run and start training. In this code following the Colab tutorial code a coco-pretrained R50-FPN_Mask_RCNN model was trained on the end-effector dataset. It took 80 minutes to train 15000 iterations on a single Nvidia GeForce 1080 FX GPU. Another way to do the training is by using the API commands directly as will be presented in what follows.

Training using Detectron2 API: To train a model with “`train_net.py`” script, first a setup is needed for the corresponding datasets following `datasets/README.md`²². Now from Facebookresearch detectron2 GitHub page the main directory was cloned, and the following commands are used:

```
cd tools/
```

²¹<https://github.com/Carmigna/detectron2.git>

²²<https://github.com/facebookresearch/detectron2/blob/master/datasets/README.md>

```
./train_net.py \
-config-file ../configs/COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_1x.yaml \
-num-gpus 1 SOLVER.IMS_PER_BATCH 2 SOLVER.BASE_LR 0.0025
```

The arguments here refer to the number of GPUs used “num-gpus”, then the batch number and the initial learning rate. For more options, `./train_net.py -h` would list all the other arguments available for the online API.

Training using my Github directory: In all that follows the results will be presented exclusively for the training done from the following GitHub directory²³. Nevertheless, the main API will be used later for general testing.

To check and monitor the training progress on Tensorboard, from an Ipython terminal these commands are used:

```
%load_ext tensorboard
%tensorboard --logdir output
```

Now the graphical display for Tensorboard can be opened on a browser using `http://localhost:6006`. There is a lengthy list of attributes, and their graphs are displayed actively with progressing iterations as seen in figure C.18.

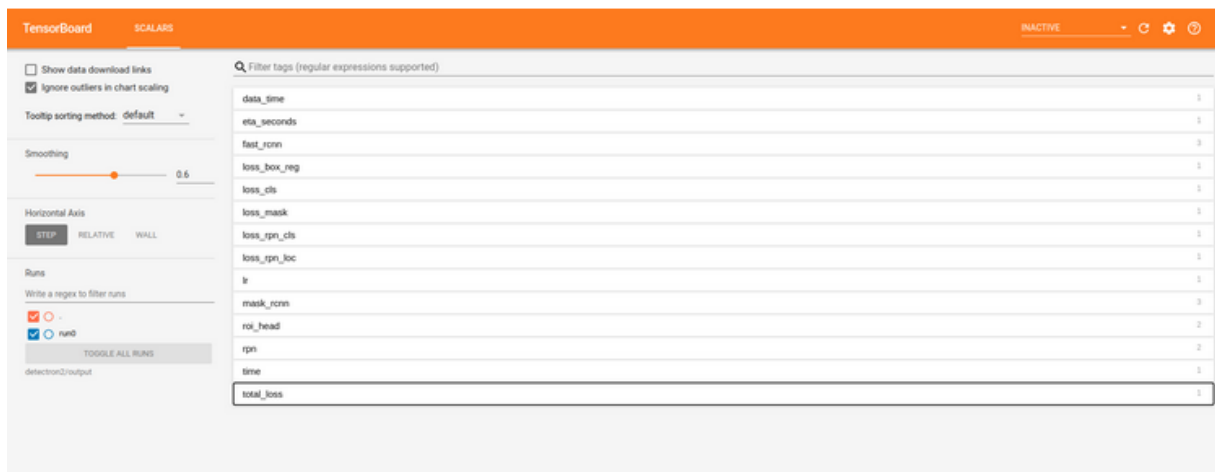


Figure C.18: Tensorboards showing components of training

²³<https://github.com/Carmigna/detectron2.git>

Most importantly in the loss function plot (figure C.19) the steep drop in the total loss is noticed clearly. This indicated that the model was learning the features of input images. The first slope that goes down to the blue dot refers to a test run for about 1500 iterations. After adjusting the learning rate another real run started for 15k iterations. The difference in the slope of the loss function is obvious between the two runs. That indicated that a minor change in adjusting the main parameters, one of which is the learning rate (from $1e^{-5}$ to $1e^{-3}$), can have a drastic effect on the training in general.

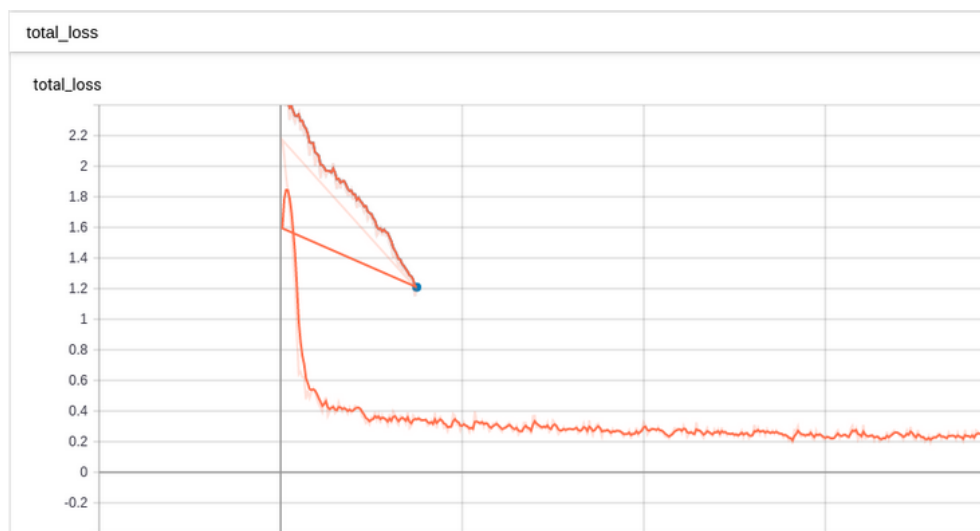


Figure C.19: The loss function plot for the test and real run

Another informative attribute is the “mask_rcnn” one, under which we find three charts that give us the progress rate of the Accuracy, False_negative FN and False_positive FP respectively (see figure C.20).

The measures presented in figure C.20 are the most important metrics in any Machine Learning model because they give us a direct reference measure on how accurate our model’s predictions are on blinded data. Here in the charts, it is apparent that the predictions were true almost 98% of the time. As for the missing 2% they were mostly false positives which are much less dangerous than the false negatives FNs. FPs are so because they are much less confusing and can be included within statistical error. As for FNs, they have a much more dangerous effect in any classification problem, and a substantial percentage of them gives a hint of a biased sample of training data and/or Over-training. However, other more specific metrics should be

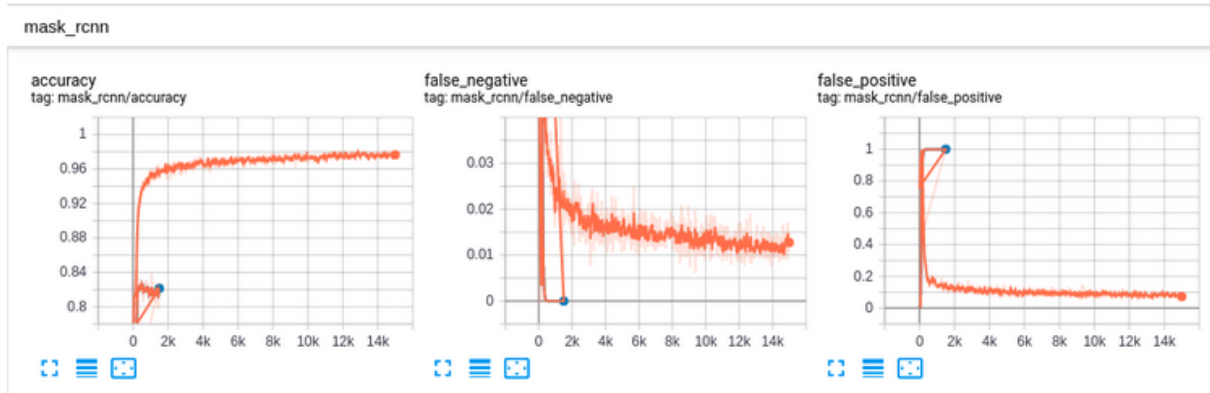


Figure C.20: From left to right the corresponding plots for Accuracy, FN and FP

studied when it is related to box detection since these include localization as well. The most important one of them is mAP (explained in detail in previous chapter). For performance checks and testing on the whole validation dataset, from my GitHub detectron2 directory another script “test_val_rcnnR50.py” is used. It would display all the detected images in the order that they appear in the validation dataset. After testing the trained model on all the images, performance checks would appear in the terminal and are showing in figure C.21.

Having a mAP (or mean Average Precision) of 83% on large areas in such a small dataset is acceptable. However, the result should be taken as usual from the live feed camera as the selected sample might be biased or off total balance. It is noticed as well that the average precision for small to medium areas is null, but that was expected since our dataset only contained the class in large form. The model did not have pictures showing the end-effector in small to medium areas, to train upon. This could be a future project with mask-rcnn detector.

In figure C.22, three detections are worth noticing and discussing. The top part of image shows two successful detections where the end-effector is closing grip (*top right*) and another accurately predicted instance where it was grabbing and moving in a faded frame. It is worth mentioning here that these are the instances where SSD-MobileNetV2 based DetectNet algorithm failed. So, this feature shows a promising superiority in the case of this project for Mask R-CNN in these kinds of detections. No need to mention that such high-level accuracy for detections is specially needed in dangerous environments surveying. As for the FP showing in the *bottom* is an instance of those failing ones. Even though the end-effector was properly detected but the arm and joint were mistakenly identified as end-effectors as

```

File Edit View Search Terminal Help
[07/04 05:00:31 d2.evaluation.evaluator]: Total inference pure compute time: 0:01:08 (0.260298 s / img per device, on 1 devices)
[07/04 05:00:31 d2.evaluation.coco_evaluation]: Preparing results for COCO format ...
[07/04 05:00:31 d2.evaluation.coco_evaluation]: Saving results to ./output/coco_instances_results.json
[07/04 05:00:31 d2.evaluation.coco_evaluation]: Evaluating predictions ...
Loading and preparing results...
DONE (t=0.00s)
creating index...
Index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=0.11s)
Accumulating evaluation results...
DONE (t=0.03s)
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.772
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.906
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.895
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.014
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.829
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.796
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.801
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.801
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.040
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.861
[07/04 05:00:32 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
| AP | AP50 | AP75 | APs | APm | APl |
|:-----|:-----|:-----|:-----|:-----|:-----|
| 77.178 | 90.634 | 89.521 | 0.000 | 1.386 | 82.866 |
Loading and preparing results...
DONE (t=0.01s)
creating index...
Index created!
Running per image evaluation...
Evaluate annotation type *segm*
DONE (t=0.15s)
Accumulating evaluation results...
DONE (t=0.02s)
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.773
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.906
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.895
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.021
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.830
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.797
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.803
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.803
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.060
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.863
[07/04 05:00:32 d2.evaluation.coco_evaluation]: Evaluation results for segm:
| AP | AP50 | AP75 | APs | APm | APl |
|:-----|:-----|:-----|:-----|:-----|:-----|
| 77.250 | 90.634 | 89.542 | 0.000 | 2.079 | 83.022 |

```

Figure C.21: Performance checks list for the Mask R-CNN model

well. Here is another example to make it more relevant to stress on the importance of having a much more robust dataset and including much more negative example to immune the model against such FPs and increase its noise discriminating power.

Inference using Detectron2 API: Another testing method would be carried on directly from the API. Like what was done in the training, from facebookresearch detectron2 repository a simple command with many arguments is used. So then, to test on single images the following command is used:

```

cd demo/
python demo.py --config-file ../configs/COCO-InstanceSegmentation/

```

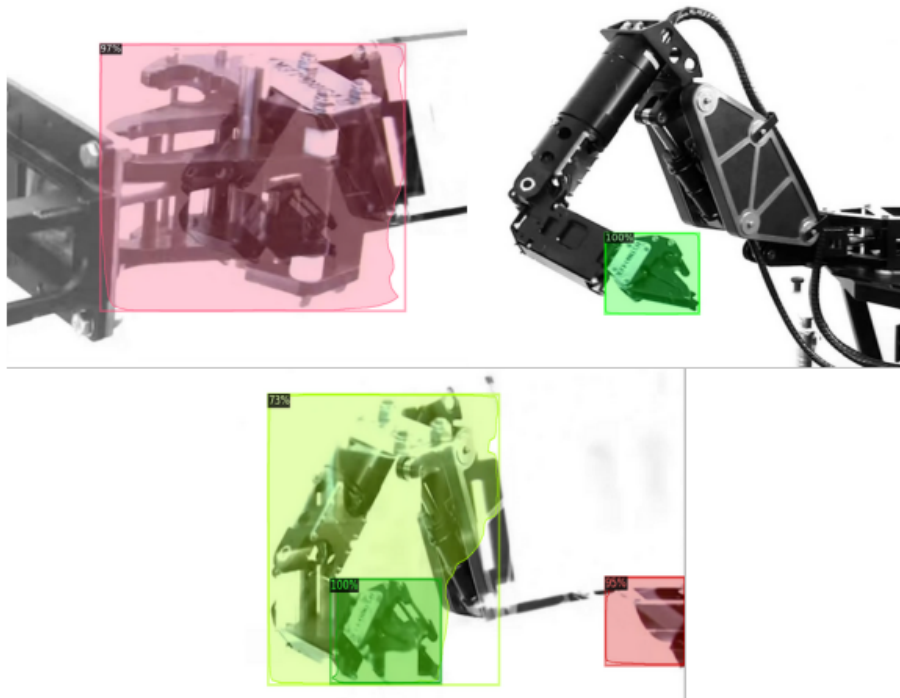


Figure C.22: Two successful detections (*top*) along with false positives (*bottom*)

```
mask_rcnn_R_50_FPN_3x.yaml \
  -input input1.jpg input2.jpg \
  -opts MODEL.WEIGHTS model_final.pth
```

It is essential to use here the trained model's final weights to be added to the `-opts weights` argument. Tensorboard automatically saves the customized model weights in `output/` directory within a file named "model_final.pth".

Finally, to implement this model on the Jetson Nano, as was done with the other cases, the best way was to use the API directly. Here are few other useful arguments²⁴:

- To run on webcam, `-input files` is replaced with `-webcam`
- To run on a video, `-input files` is replaced with `-video-input video.mp4`
- To save outputs in a directory (for images) or a file (for webcam or video), `-output` is used additionally

²⁴For details of the command line arguments, see `demo.py -h`

Appendix D

Robotics Setups

D.1 Mobile Activity

From the terminal, we can see clear proof if our installation with GPU adaptation has been successful, right after we launch the main script. Figure D.1 shows that the name of an existing Nvidia GPU would appear (*underlined in red*) along with its compute capability.

```
name: GeForce GTX 1050 Ti with Max-Q Design major: 0 minor: 1 memoryClockRate(GHz): 1.2905
pciBusID: 0000:01:00:0
2020-05-29 14:11:45.118388: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudart.so.10.2
2020-05-29 14:11:45.147735: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcublas.so.10
2020-05-29 14:11:45.165597: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcufft.so.10
2020-05-29 14:11:45.170816: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcusand.so.10
2020-05-29 14:11:45.215012: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcusolver.so.10
2020-05-29 14:11:45.225386: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcusparse.so.10
2020-05-29 14:11:45.342011: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudnn.so.7
2020-05-29 14:11:45.342377: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:983] successful NUMA node read from SysFS had negative value (-1)
ode zero
2020-05-29 14:11:45.343507: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:983] successful NUMA node read from SysFS had negative value (-1)
ode zero
2020-05-29 14:11:45.344387: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1767] Adding visible GPU devices: 0.
2020-05-29 14:11:45.345407: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudart.so.10.2
2020-05-29 14:11:45.436984: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1180] Device interconnect StreamExecutor with strength 1 edge matrix:
2020-05-29 14:11:45.437006: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1186] 0
2020-05-29 14:11:45.437011: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1199] 0: N
2020-05-29 14:11:45.437562: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:983] successful NUMA node read from SysFS had negative value (-1)
ode zero
2020-05-29 14:11:45.437790: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:983] successful NUMA node read from SysFS had negative value (-1)
ode zero
2020-05-29 14:11:45.437984: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:983] successful NUMA node read from SysFS had negative value (-1)
ode zero
2020-05-29 14:11:45.438162: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1325] Created TensorFlow device (/job:localhost/replica:0/task:0/dev1
GeForce GTX 1050 Ti with Max-Q Design, pci bus id: 0000:01:00:0, compute capability: 6.1)
2020-05-29 14:11:45.439933: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x6f14e80 initialized for platform CUDA (this does not gua
2020-05-29 14:11:45.439946: I tensorflow/compiler/xla/service/service.cc:176] StreamExecutor device (0): GeForce GTX 1050 Ti with Max-Q Design, Co
```

Figure D.1: The GPU would appear clearly in the terminal if it is running. *Underlined in red* are the GPU's name, unit number and computing capability.

After showing the GPU specifications, the model would be loaded. It is a CNN based on MobileNetV2 architecture. However, since this code would be fine-tuning the model on a custom dataset with different classes, the script would only load a headless MobileNetV2. In other words, as explained in transfer learning section in Chapter 4, the old, trained output layer (i.e., FC + SOFTMAX) would be removed and replaced with a new one with randomly initialized weights. This code makes use

of the pre-trained layers but trains the whole network on the new custom classes as well, not only the output layer. Figure shows that the model is loaded signaling that it is training from scratch. It is not re-training the whole network from ground zero, that only applies to the FC and Softmax layers. The figure D.2 shows ground-truth class labels, of the training set's images, in one-hot vector format explained in Chapter 2.

```

W0529 14:11:45.441009 139623036491584 module_wrapper.py:139] From /usr/local/lib/python3.6/dist-packag
compat.v1.is_variable_initialized instead.

W0529 14:11:45.509400 139623036491584 module_wrapper.py:139] From /usr/local/lib/python3.6/dist-packag
mpat.v1.variables_initializer instead.

W0529 14:11:48.383069 139623036491584 module_wrapper.py:139] From /usr/local/lib/python3.6/dist-packag
pat.v1.nn.fused_batch_norm instead.

W0529 14:11:48.451618 139623036491584 module_wrapper.py:139] From /usr/local/lib/python3.6/dist-packag
.compat.v1.placeholder_with_default instead.

No load_weight_starting_file...We will start from scratch!
[[ 0 1 0]
 [ 0 1 0]
 [ 0 1 0]
 [ 1 0 0]
 [ 1 0 0]
 [ 1 0 0]]
  
```

The terminal output shows several warnings from TensorFlow. The key message is "No load_weight_starting_file...We will start **from scratch!**", where "from scratch!" is circled in red. Below this, a 6x3 matrix of ground-truth labels is shown in one-hot format: $\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$. The first three rows have a '1' in the second column, and the last three rows have a '1' in the first column. A red circle highlights the first row of this matrix, and a red arrow points from the text "One-hot format" to it.

Figure D.2: Loading the model and ground-truth labels of the dataset.

After that, the MobileNetV2 graph would be displayed, a part of it is showing in figure D.3 then the training would start (figure D.4). The type of layer, shape of the image and number of parameters trained in every layer of the model are also stated in the graph. Showing with a *red circle* in figure D.4 is the customized head added through in MobileActivity code to the MobileNetV2 architecture. A dense (FC) layer, that flattens the input features of the Bottleneck or 1x1 CONV layer, is introduced by hand. While the SoftMax activation function would output the final categorical probability scores class-wise. After all that, the training would begin with “Epoch 1”, while at every epoch level the loss function’s value and accuracy, for the training and validation, would be displayed. During the training process, average loss would start decreasing after a short while, if not there would be something wrong that is worth stopping the training and checking. If all went well, a sharp drop in loss value should be noticed because the code uses Adam optimization. It is recommended that training should be stopped when this loss value no longer decreases for several iterations.

Layer (type)	Output Shape	Param #	Connected to
Input_1 (InputLayer)	(None, 96, 96, 3)	0	
Conv1_pad (ZeroPadding2D)	(None, 97, 97, 3)	0	Input_1[0][0]
Conv1 (Conv2D)	(None, 48, 48, 32)	864	Conv1_pad[0][0]
bn_Conv1 (BatchNormalization)	(None, 48, 48, 32)	128	Conv1[0][0]
Conv1_relu (ReLU)	(None, 48, 48, 32)	0	bn_Conv1[0][0]
expanded_conv_depthwise (DepthwiseConv2D)	(None, 48, 48, 32)	288	Conv1_relu[0][0]
expanded_conv_depthwise_BN (BatchNormalization)	(None, 48, 48, 32)	128	expanded_conv_depthwise[0][0]
expanded_conv_depthwise_relu (ReLU)	(None, 48, 48, 32)	0	expanded_conv_depthwise_BN[0][0]
expanded_conv_project (Conv2D)	(None, 48, 48, 16)	512	expanded_conv_depthwise_relu[0][0]
expanded_conv_project_BN (BatchNormalization)	(None, 48, 48, 16)	64	expanded_conv_project[0][0]
block_1_expand (Conv2D)	(None, 48, 48, 96)	1536	expanded_conv_project_BN[0][0]
block_1_expand_BN (BatchNormalization)	(None, 48, 48, 96)	384	block_1_expand[0][0]
block_1_expand_relu (ReLU)	(None, 48, 48, 96)	0	block_1_expand_BN[0][0]
block_1_pad (ZeroPadding2D)	(None, 49, 49, 96)	0	block_1_expand_relu[0][0]
block_1_depthwise (DepthwiseConv2D)	(None, 24, 24, 96)	864	block_1_pad[0][0]
block_1_depthwise_BN (BatchNormalization)	(None, 24, 24, 96)	384	block_1_depthwise[0][0]
block_1_depthwise_relu (ReLU)	(None, 24, 24, 96)	0	block_1_depthwise_BN[0][0]
block_1_project (Conv2D)	(None, 24, 24, 24)	2304	block_1_depthwise_relu[0][0]
block_1_project_BN (BatchNormalization)	(None, 24, 24, 24)	96	block_1_project[0][0]
block_2_expand (Conv2D)	(None, 24, 24, 144)	3456	block_1_project_BN[0][0]
block_2_expand_BN (BatchNormalization)	(None, 24, 24, 144)	576	block_2_expand[0][0]
block_2_expand_relu (ReLU)	(None, 24, 24, 144)	0	block_2_expand_BN[0][0]
block_2_depthwise (DepthwiseConv2D)	(None, 24, 24, 144)	1296	block_2_expand_relu[0][0]
block_2_depthwise_BN (BatchNormalization)	(None, 24, 24, 144)	576	block_2_depthwise[0][0]
block_2_depthwise_relu (ReLU)	(None, 24, 24, 144)	0	block_2_depthwise_BN[0][0]
block_2_project (Conv2D)	(None, 24, 24, 24)	3456	block_2_depthwise_relu[0][0]

Figure D.3: A part of MobileNetV2 graph.

D.2 End-effector in Darknet API Setups

As done with Clusters, the information extracted from the annotation and labelling would be added in .txt files, one for each image. In every .txt file a new line is added for every object annotated with a box within each image. The lines indicate inputs of the following order:

$$\langle object_class \rangle \langle x \rangle \langle y \rangle \langle width \rangle \langle height \rangle \quad (D.1)$$

Where *object_class* refers to the label given for each class, starting with 0. the *x* and *y* refer to the x and y coordinates of the upper left corner of the bounding box indicating its location with respect to the image origin, which is its upper left corner as well. Finally, *width* and *height* refer to the width and height of the bounding box as a % in comparison with the width and height of the image. Figure D.5 shows an instance of the .txt files, where only 1 class, 1 object and one box is found within the frame.

```

block_io_expand_relu (ReLU) (None, 3, 3, 960) 0 block_io_expand_relu[0][0]
block_io_depthwise (DepthwiseConv (None, 3, 3, 960) 8640 block_io_expand_relu[0][0]
block_io_depthwise_bn (BatchNorm (None, 3, 3, 960) 3840 block_io_depthwise[0][0]
block_io_depthwise_relu (ReLU) (None, 3, 3, 960) 0 block_io_depthwise_bn[0][0]
block_io_project (Conv2D) (None, 3, 3, 320) 307200 block_io_depthwise_relu[0][0]
block_io_project_bn (BatchNorm (None, 3, 3, 320) 1280 block_io_project[0][0]
conv_1 (Conv2D) (None, 3, 3, 1280) 409600 block_io_project_bn[0][0]
conv_1_bn (BatchNormalization) (None, 3, 3, 1280) 5120 conv_1[0][0]
out_relu (ReLU) (None, 5, 5, 1280) 0 conv_1_bn[0][0]
flatten_1 (Flatten) (None, 11520) 0 out_relu[0][0]
fc (Dense) (None, 3) 34563 Flatten_1[0][0]
----- Customised Head -----
total_params: 2,792,587
trainable_params: 2,258,435
non-trainable params: 534,152
2022-05-29 14:11:57.593509: I tensorflow/core/python/ops/math_grad.py:1424: where (from tensorflow/python/ops/array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
use tf.where in 2.0, which has the same broadcast rule as np.where
2022-05-29 14:11:59.026786: I tensorflow/core/python/module_wrapper.py:139: From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:986: The name tf.assign_add is deprecated. Please use tf.compat.v1.assign_add instead.
2022-05-29 14:12:00.209979: I tensorflow/core/python/module_wrapper.py:139: From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:973: The name tf.assign is deprecated. Please use tf.compat.v1.assign instead.
2022-05-29 14:12:03.839847: I tensorflow/core/python/module_wrapper.py:139: From /usr/local/lib/python3.6/dist-packages/keras/callbacks.py:850: The name tf.summary.merge_all is deprecated. Please use tf.compat.v1.summary.merge_all instead.
2022-05-29 14:12:03.839800: I tensorflow/core/python/module_wrapper.py:139: From /usr/local/lib/python3.6/dist-packages/keras/callbacks.py:853: The name tf.summary.FileWriter is deprecated. Please use tf.compat.v1.summary.FileWriter instead.
Epoch 1/10 Training Starting Point
2022-05-29 14:12:10.835442: I tensorflow/stream_executor/platform/default/dso_loader.cc:44: Successfully opened dynamic library libcublas.so.10
2022-05-29 14:12:10.880983: I tensorflow/stream_executor/platform/default/dso_loader.cc:44: Successfully opened dynamic library libcudnn.so.7
04/64 [=====] - 22s 340ms/step - loss: 0.8513 - acc: 0.9832 - val_loss: 0.3149 - val_acc: 0.9383
Epoch 00001: val_loss improved from inf to 0.31491, saving model to /home/carnigna/mobile_activity/training/model_weight_checkpoints_gen/model_activity_model-96-1.0-10-64-TIME-1590757914.1704998-0.31491096.h5
2022-05-29 14:12:147.322837: I tensorflow/core/python/module_wrapper.py:139: From /usr/local/lib/python3.6/dist-packages/keras/callbacks.py:195: The name tf.summary is deprecated. Please use tf.compat.v1.summary instead.
Epoch 2/10
04/64 [=====] - 12s 180ms/step - loss: 0.8345 - acc: 0.9968 - val_loss: 1.0650 - val_acc: 0.9141

```

Figure D.4: Showing above are the customised head and the training’s starting point.

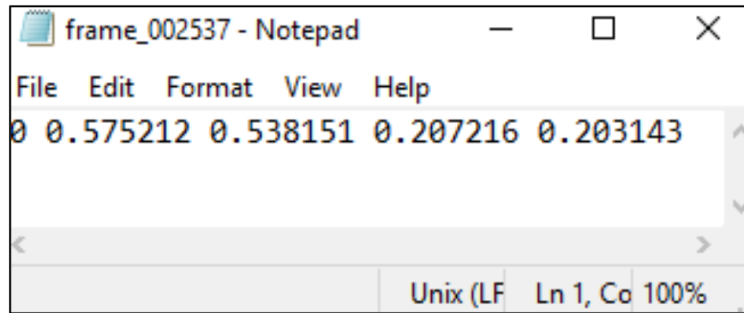
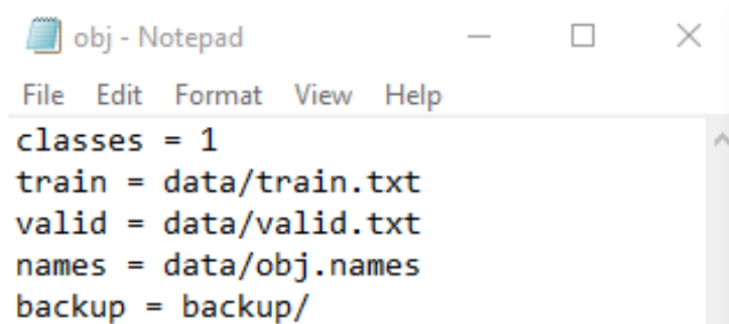


Figure D.5: Text file displaying annotation and labelling information

Additionally, when exporting the data from CVAT, the files “obj.data”, “obj.names” and “train.txt” (in addition to “valid.txt” for validation) are created. The “obj.data” file was slightly altered to include validation data¹, and this is shown in figure D.6.

The first row of the file contains the number of classes within the dataset. As there is only one class in this case, the end-effector, it is set to one. The following three lines define the path to such files as “train.txt”, “valid.txt” and “obj.names”. These paths are added as “train”, “valid” and “names” variables, respectively. The train.txt and valid.txt files contain the paths to all the input data frames be it for testing or validation. The reference to all defined paths should be the main “Darknet”

¹In fact, two datasets are created using CVAT, one for training and the other for validation following the rule of *cross-validation* (see NNs section).



```
obj - Notepad
File Edit Format View Help
classes = 1
train = data/train.txt
valid = data/valid.txt
names = data/obj.names
backup = backup/
```

Figure D.6: obj.data file

directory². Figure D.7 below shows a section of these paths.

As for “obj.names” it is a straightforward text file that contains the names of the classes (i.e., end-effector in this case) (Figure D.8). The final line creates a path to a backup folder where the best weights from the training as well as the ones from every other 1000 iterations are all saved to be used later for testing.

The YOLO version used in this application is YOLOv4. Relative to other detectors, it is extremely fast and depending on the requirement of the task, speed can be sacrificed for accuracy without the need for retraining thereby making the system flexible. YOLOv4 can be installed either on a Windows or Linux device, and instructions on how to install can be found at YOLOv4 main GitHub page³.

Before the model can be trained on the dataset, changes must be implemented to the configuration files and the dataset must be prepared to ensure that training goes smoothly and gets properly optimized. The data-set folder must contain all the images that need to be trained on and their corresponding .txt files. Additionally, as was mentioned before, images for validation are also required in a separate dataset, so that the training can be verified whilst taking place. The validation images are frames that contain the end-effector, but the model has not been seen before while training. If the model identifies the validation image to contain an end-effector in most instances, then the training has been validated successfully.

To compare the datasets, almost 90% of the overall images were added for training

²<https://github.com/pjreddie/darknet>

³<https://github.com/AlexeyAB/darknet>

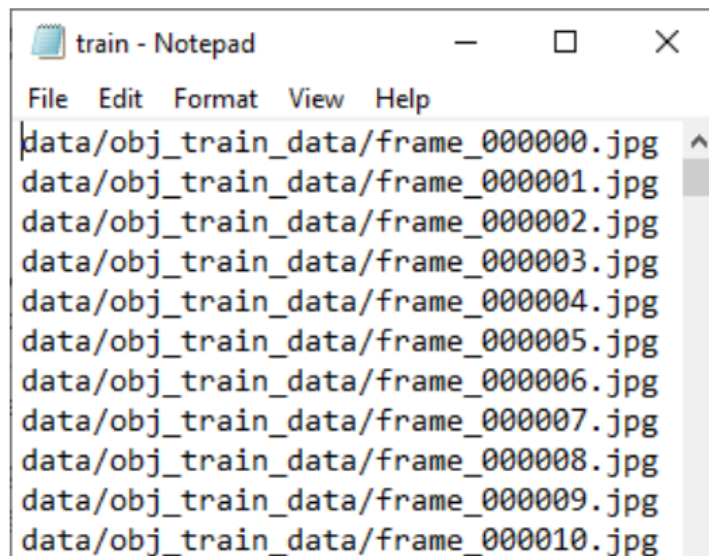


Figure D.7: A section of the train.txt file.

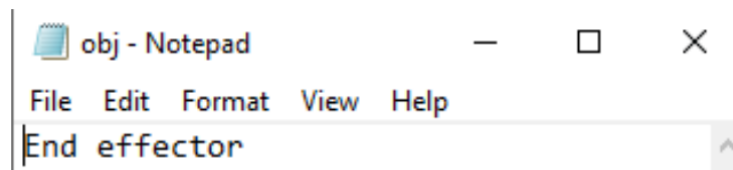


Figure D.8: The obj.names file

while the remaining 10% were left for validation. In case the two datasets were not created separately from the beginning with CVAT, the training and validation files require splitting into separate folders. Therefore, a python script can split them into the necessary folders to save time. One such script can be found online article⁴. The configuration file used is “yolov4-custom.cfg”. This is the configuration file that is used for custom objects which is the case in this project. It contains several important parameters such as the batch/subdivision number, resolution, number of channels and data augmentation which require adjusting depending on the case. A batch is the number of images that are loaded per iteration. A subdivision is the amount a batch is split into. Each one from the batch is processed and trained upon. The

⁴<https://medium.com/@anirudh.s.chakravarthy/training-yolov3-on-your-custom-dataset-19a1abbdaf09>

two parameters are dependent on the computing power available for training. Due to this, the value used for batch and subdivision for training in this case was 64 and 64. Resolution should be adjusted according to processing power available as well. If the resolution on input images is set too high, errors may start to appear in the training. Therefore, a Goldilocks value was used at 256x256. In practice, one starts with the lowest resolution which is 96x96. If the system can handle it without a problem, it would be increased fast by doubling or slow by adding 32s on dimensions. The number of channels describes the colors of the dataset. A channel value of 1 would result in training to be completed in Greyscale. This is not ideal and so the value was instead set to 3 thereby keeping the RGB scale which is for training with colors.

Data augmentation is important when training with a limited dataset. Although approximately 4000 images to train on is sufficient, it is not an ideal number in case the frames are repetitive. This is likely to be the case in video footage. In most instances, a greater dataset will increase the reliability of the system. Therefore, to improve this, the dataset can be artificially expanded by applying changes to the images such as flip, rotation, shearing and change in hue. The names of these parameters in the configuration file are mosaic, mix-up, and blur. The effect of this is that the model can learn from a greater, more robust dataset, and so this should improve validation results at the next stage. The augmentations applied to the dataset are mosaic=1, mixup=1 and blur=1 and so full advantage can be taken of the limited data-set available.

Another change required to the configuration file is the filters number. This value is evaluated using the following equation:

$$Filters_Number = (Number_of_Classes + 5) \times 3 \quad (D.2)$$

As there is only one class within the dataset the filters number used is 18. It should be noted that the classes number is only changed in the YOLO layers and the filters number, according to the above-mentioned formula, is only adjusted in the one convolutional layer that appear before each YOLO layer in the main configuration file. These adjustments should take place three times as there are three YOLO layers. The anchor coordinates can be changed as well according to the shape of objects as was mentioned in previous sections. However, since the training is running on one class only and no crowding existing in our frames, it is very unlikely that any confusion would be caused or solved by the shape of the anchor boxes.

Once the changes described above are made to the “yolov4-custom.cfg” file, it is renamed and saved as “yolo-obj.cfg” as this is the default name of the configuration file called within the training command. A final requirement is the initial weights

to be used. For this case, the ones that must be downloaded are for custom objects recognition. These weights are available online as the “yolov4.conv.137” file. As training occurs, the best weights are identified by YOLO and are automatically saved. The improved weights can be used for retraining if needed.

D.3 End-effector in DIGITS' DetectNet Setups

In DIGITS, images must be resized to satisfy the DetectNet input dimensions. The configuration can be altered to accept custom image sizes, and by default it is set to 1392x512. Since these dimensions are ideal only for the KITTY original dataset, another tweak should be done to images dimension if another custom dataset is to be used instead. The good thing about DIGITS is that it allows images to be padded to take one final set of dimensions that are compatible to both images of assorted sizes in the custom dataset, and DetectNet input dimensions. For the set of images used in this project a perfect dimensional Goldilocks used was a total padding of 2500x2500. The reason for that huge padding was the extreme difference in sizes of the images used in our limited dataset, even the extended one. Another power point of DIGITS is that it requires users to introduce a class called “dontcare”. Within this class objects contained in bounding boxes are allowed to be ignored during the training. This would prevent more FP from appearing as detection during the training.

The last stage in data preparation is importing all the images and labels we created, in KITTY format, from their proper folders into Nvidia DIGITS. Once an instance of DIGITS is opened, the Data-sets tab is selected and, on the right side, Object Detection from New Dataset is clicked from the list in scroll down.

Figure D.9 is showing all the parameters that should be added and entered correctly. These are the following:

- Training image folder: path of the folder where images for training exist (e.g., /path/to/train/images)
- Training label folder: path of the folder where labels for training exist (e.g., /path/to/train/labels)
- Validation image folder: path of the folder where images for validation exist (e.g., /path/to/val/images)
- Validation label folder: path of the folder where labels for validation exist (e.g., /path/to/val/labels)

The screenshot shows the 'New Object Detection Dataset' configuration interface. It includes the following fields and options:

- Object Detection Dataset Options**
 - Images can be stored in any of the supported file formats ("png", "jpg", "jpeg", "targa", "ppm").
 - Training image folder**: /data/Manipulator0/train/images/endeffector
 - Training label folder**: /data/Manipulator0/train/labels/endeffector
 - Validation image folder**: /data/Manipulator0/val/images/endeffector
 - Validation label folder**: /data/Manipulator0/val/labels/endeffector
 - Pad image (Width x Height)**: 2500 x 2500
 - Resize image (Width x Height)**: 640 x 640
 - Channel conversion**: RGB
 - Minimum box size (in pixels) for validation set**: 0
 - Custom classes**: dontcare, endeffector
- Feature Encoding**: PNG (lossless)
- Label Encoding**: None
- Encoder batch size**: 32
- Number of encoder threads**: 4
- DB backend**: LMDB
- Enforce same shape**: Yes
- Group Name**: 0000-KITTI
- Dataset Name**: Manipulator0
- Create** button

Figure D.9: DIGITS New Object Detection Data-set board

- Custom classes: class names should be added here separated with a comma. It is important that the “dontcare” class be added first to the list if predefined weights are to be used in the training as is the case in this project. If “dontcare” is not included first in the list, the first class will be ignored, and it will not be trained at all.
- Data-set name: a name for the dataset is added here.

After completing the form, create button at the bottom is clicked and the dataset will be produced. If all went well “Job Status Done” would appear in green. If an error occurred during the process, a “Job Status Error” would be displayed in red instead (see figure D.10). A verbose debugging display would point to the source of

error. Failures mostly happen due to missing or mis-spelled label names. Mis-typing a folder path can cause an error as well or even any unbalance in the input image dimensions. Now once the dataset is ready, DetectNet object detection model can start training in DIGITS.



Figure D.10: Job Status window showing “Done” (in *green*) or “Error” (in *red*)

First, an Object Detection Task should be created. To do that on the DIGITS home page, Models tab is selected then New Model→ Images→ Object Detection chosen. A form will be opened to enter some parameters that will be discussed in what follows (See figure D.11).

To get high accuracy, enough time should be allowed for training, 200-600 “Training epochs” are required. Depending on the data-set size and number of classes that parameter might vary. In the “Select Dataset” window the name of the dataset should be selected. The dataset prepared for this project is “Manipulator08”. “Batch size” and “Accumulation”, or how many pictures can be processed at once, should be changed according to the graphics card computing capacity. In this project’s case, the numbers 2 and 5 worked properly. These parameters should be acquired by trial and error by escalating with small increments until the processing power is exhausted. Another parameter that should be tuned is the image size. It is related to processing power available as well. Adjusting the image size in the dataset might require repeating the production of it. So, image size, batch and accumulation should be tuned in a balanced way, to get the best out of all. Image size would affect learning, the larger it gets the faster a model grasps its features, and more memory is consumed. One had to repeat this process many times to find the right dimensions suitable. That is the reason for this project 8 datasets had to be reproduced. The Goldilocks zone was for image size of 640x640 in the data-set parameters. “Blob format” was chosen to be NVcaffe which is the default Caffe platform used by Nvidia in DIGITS, however

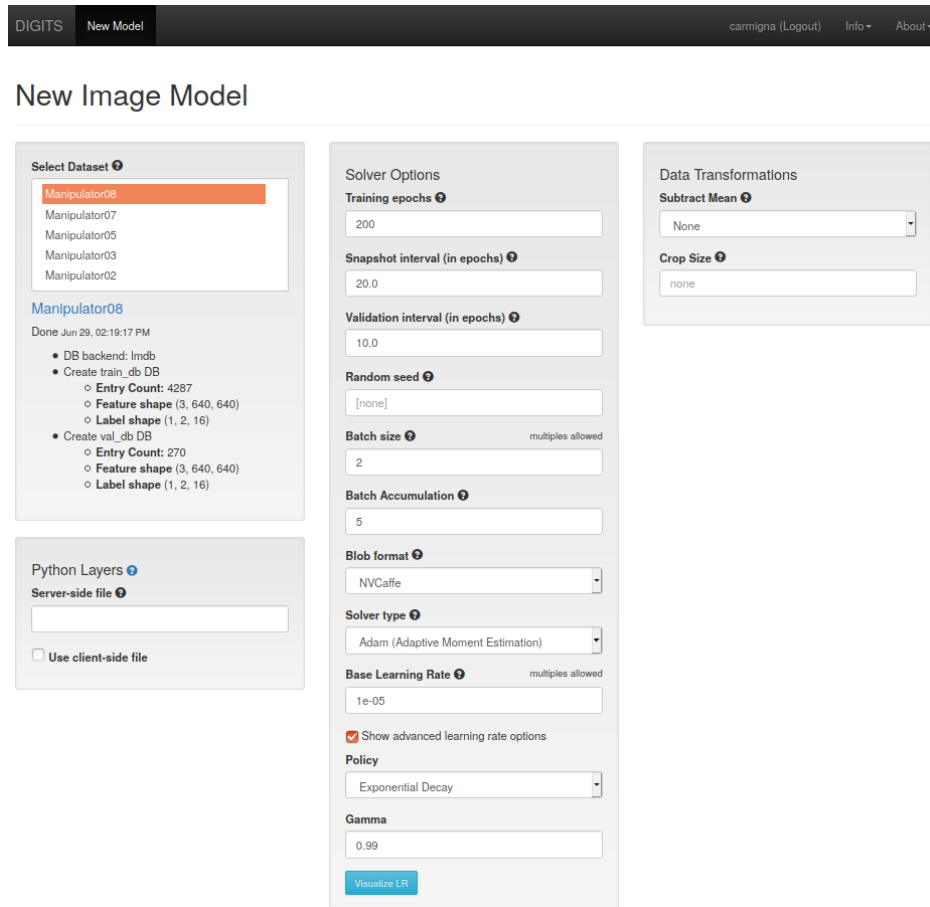


Figure D.11: New Image Model parameters board

others can be used as well in the compatible option. As for the solver type it was chosen to be ADAM. It is an algorithm that optimizes the training, and makes the learning or updating of the weights process run much faster and smoother⁵. For “Base Learning Rate” $1e^{-05}$ has proven to be a perfect starting point since a lower learning rate means slower learning with more accuracy. This feature is needed at the start of training. Under “Policy” exponential decay was selected which is a function that varies the learning rate in an exponential manner with time. Multiple starting learning rates can be selected at once, separated by a comma, however that would consume much more memory as well. If that option is chosen, training “loss” plots with distinct colors would be displayed for each selected “Base Learning Rate”. Finally, in “Data

⁵Please refer to chapter 2 for more details or check <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>

Transformations” subtract mean is set to none.

References

- [1] E. Cortina Gil et al (NA62 Collaboration). “An investigation of the very rare $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ decay”. In: *JHEP11(2020)042* (2020).
- [2] E. Cortina Gil et al (NA62 Collaboration). “Measurement of the very rare K^+ to π^+ ν $\bar{\nu}$ decay”. In: *JHEP06(2021)093* (2021).
- [3] E. Cortina Gil et al (NA62 Collaboration). “Search for a feebly interacting particle X in the K^+ to π^+ X decay”. In: *JHEP03(2021)058* (2021).
- [4] E. Cortina Gil et al (NA62 Collaboration). “Search for heavy neutral lepton production in the K^+ decays to positrons”. In: *Phys. Lett. B 807 (2020) 135599* (2020).
- [5] E. Cortina Gil et al (NA62 Collaboration). “Search for K^+ decays to a muon and invisible particles”. In: *Phys. Lett. B 816 (2021) 136259* (2021).
- [6] E. Cortina Gil et al (NA62 Collaboration). “Search for π^0 decays to invisible particles”. In: *JHEP02(2021)201* (2021).
- [7] E. Cortina Gil et al (NA62 Collaboration). “Search for production of an invisible dark photon in π^0 decays”. In: *JHEP05(2019)182* (2019).
- [8] E. Cortina Gil et al (NA62 Collaboration). “Searches for lepton number violating K^+ decays”. In: *Phys. Lett. B 797 (2019) 134794* (2019).
- [9] D. Buttazzo A.J. Buras and R. Kneijens. In: *JHEP11(2015)166* (2015).
- [10] U. Haisch A.J. Buras M. Gorbahn and U. Nierste. In: *Phys. Rev. Lett B 95(2005)261805* (2005).
- [11] Robot Ignite Academy. *Deep Learning with Domain Randomization [online]*. <https://www.robotigniteacademy.com/en/path/machine-learning-for-robots/>, 2020.
- [12] Robot Ignite Academy. *Live Clas n81: How to use Python3 with ROS [online]*. Robot Ignite Academy course, 2020.
- [13] Diederik P. Kingma et al. “ADAM: a method for stochastic optimization”. In: *arXiv 1412.6980v9* (2017).
- [14] E. Cortina Gil et al. In: *J. Instrum.* 12 (2017), p. 05025.

-
- [15] J. Cogan et al. “Jet-images: computer vision inspired techniques for jet tagging”. In: *J. High Energy Phys.* 2015 (2015), p. 118.
- [16] P. Baldi et al. “Jet substructure classification in high-energy physics with deep neural networks”. In: *Phys. Rev. D* 93 (2016), p. 094034.
- [17] V. Fanti et al. In: *Nucl. Instrum. Methods, A574* (2007), p. 433.
- [18] Alexey Bochkovskiy et al. “YOLOv4: Optimal Speed and Accuracy of Object Detection”. In: *arXiv:2004.10934* (2020).
- [19] Andrew Howard et al. “Network In MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: *arXiv:1704.04861v1* (2017).
- [20] C. Szegedy et al. “Scalable, High-Quality Object Detection”. In: *arXiv:1412.1441* (2014).
- [21] Christian Szegedy et al. “Going deeper with convolutions”. In: *arXiv:1409.4842v1* (2014).
- [22] J. Redmon et al. “YOLO9000: Better, Faster, Stronger”. In: *arXiv:1612.08242* (2016).
- [23] J. Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: *arXiv:1506.02640* (2015).
- [24] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *arXiv:1512.03385v1* (2015).
- [25] M. Sandler et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. In: *arXiv:1801.04381v4* (2019).
- [26] Min Lin et al. “Network In Network”. In: *arXiv:1312.4400v3* (2014).
- [27] S. Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *arXiv:1506.01497v3* (2016).
- [28] Tsung-Yi Lin et al. “Feature Pyramid Networks for Object Detection”. In: *arXiv:1612.03144v2* (2017).
- [29] W. Liu et al. “SSD: Single Shot MultiBox Detector”. In: *arXiv:1512.02325* (2015).
- [30] Shun-ichi Amari and Si Wu. “Improving support vector machine classifiers by modifying kernel functions”. In: *Neural Networks* 12.6 (1999), pp. 783–789.
- [31] P. Baldi, P. Sadowski, and D. Whiteson. “Searching for exotic particles in high-energy physics with deep learning”. In: *Nat. Commun.* 5 (2014), p. 4308.
- [32] Robert E Banfield et al. “A comparison of decision tree ensemble creation techniques”. In: *IEEE transactions on pattern analysis and machine intelligence* 29.1 (2006), pp. 173–180.

- [33] “Belle II Technical Design Report”. In: (2010). arXiv: 1011.0352 [physics.ins-det].
- [34] Monika Blanke, Andrzej J. Buras, and Stefan Recksiegel. “Quark flavour observables in the Littlest Higgs model with T-parity after LHC Run 1”. In: *The European Physical Journal C* 76.4 (Apr. 2016). ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-016-4019-7. URL: <http://dx.doi.org/10.1140/epjc/s10052-016-4019-7>.
- [35] Monika Blanke et al. “RareKandBDecays in a warped extra dimension with custodial protection”. In: *Journal of High Energy Physics* 2009.03 (Mar. 2009), pp. 108–108. DOI: 10.1088/1126-6708/2009/03/108. URL: <https://doi.org/10.1088/1126-6708/2009/03/108>.
- [36] Tomáš Blažek and Peter Maták. “Left–left squark mixing, and minimal supersymmetry with large $\tan \beta$ ”. In: *International Journal of Modern Physics A* 29.27 (2014), p. 1450162.
- [37] Christoph Bobeth and Andrzej J. Buras. “Leptoquarks meet ε'/ε and rare Kaon processes”. In: *Journal of High Energy Physics* 2018.2 (Feb. 2018). ISSN: 1029-8479. DOI: 10.1007/jhep02(2018)101. URL: [http://dx.doi.org/10.1007/JHEP02\(2018\)101](http://dx.doi.org/10.1007/JHEP02(2018)101).
- [38] A. Bonner. “The Complete Beginner’s Guide to Deep Learning: Artificial Neural Networks”. In: [*towardsdatascience Online article*] (2019).
- [39] Marzia Bordone et al. “Probing lepton-flavour universality with $K \rightarrow \pi \nu \bar{\nu}$ decays”. In: *The European Physical Journal C* 77.9 (Sept. 2017). ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-017-5202-1. URL: <http://dx.doi.org/10.1140/epjc/s10052-017-5202-1>.
- [40] A. P. Bradley. “The use of the area under the ROC curve in the evaluation of machine learning algorithms”. In: *Pattern Recognition* 30 7 (1997), pp. 1145–1159.
- [41] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.
- [42] Francesco Brizioli. “Measurement of $Br(K^+ \rightarrow \pi^+ \nu \bar{\nu})$ with the NA62 experiment at CERN”. Presented 26 Apr 2021. 2020. URL: <http://cds.cern.ch/record/2765463>.
- [43] Joachim Brod and Martin Gorbahn. “Electroweak corrections to the charm quark contribution to $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ ”. In: *Phys. Rev. D* 78 (3 Aug. 2008), p. 034006. DOI: 10.1103/PhysRevD.78.034006. URL: <https://link.aps.org/doi/10.1103/PhysRevD.78.034006>.

- [44] Joachim Brod, Martin Gorbahn, and Emmanuel Stamou. “Two-loop electroweak corrections for the $K \rightarrow \pi\nu\bar{\nu}$ decays”. In: *Phys. Rev. D* 83 (3 Feb. 2011), p. 034030. DOI: 10.1103/PhysRevD.83.034030. URL: <https://link.aps.org/doi/10.1103/PhysRevD.83.034030>.
- [45] Jason Brownlee. “Softmax Activation Function with Python”. In: *[Machine Learning Mastery Online article]* (2020).
- [46] D. Bryman et al. “Particle identification in NA62 using a Light Gradient Boosting Machine and Convolutional Neural Networks”. In: *CERN Note NA62-21-02* (2021).
- [47] A. E. Bryson and Y.-C. Ho. *Applied Optimal Control*. Blaisdell, New York, 1969.
- [48] Gerhard Buchalla and Andrzej J. Buras. “ $\sin 2\beta$ from $K^+ \rightarrow \pi^+\nu\bar{\nu}$ ”. In: *Physics Letters B* 333.1 (1994), pp. 221–227. ISSN: 0370-2693. DOI: [https://doi.org/10.1016/0370-2693\(94\)91034-0](https://doi.org/10.1016/0370-2693(94)91034-0). URL: <https://www.sciencedirect.com/science/article/pii/0370269394910340>.
- [49] Gerhard Buchalla and Andrzej J. Buras. “The rare decays $K^+ \rightarrow \pi^+\nu\bar{\nu}$, $B \rightarrow X\nu\nu$ and $B \rightarrow l+l-$: an update”. In: *Nuclear Physics B* 548.1 (1999), pp. 309–327. ISSN: 0550-3213. DOI: [https://doi.org/10.1016/S0550-3213\(99\)00149-2](https://doi.org/10.1016/S0550-3213(99)00149-2). URL: <https://www.sciencedirect.com/science/article/pii/S0550321399001492>.
- [50] Andrzej J Buras. “Weak Hamiltonian, CP violation and rare decays”. In: *arXiv preprint hep-ph/9806471* (1998).
- [51] Andrzej J. Buras, Dario Buttazzo, and Robert Knegjens. “ $K \rightarrow \pi\nu\bar{\nu}$ and ε'/ε in simplified new physics models”. In: *Journal of High Energy Physics* 2015.11 (Nov. 2015). ISSN: 1029-8479. DOI: 10.1007/jhep11(2015)166. URL: [http://dx.doi.org/10.1007/JHEP11\(2015\)166](http://dx.doi.org/10.1007/JHEP11(2015)166).
- [52] Andrzej J. Buras, Selma Uhlig, and Felix Schwab. “Waiting for precise measurements of $K^+ \rightarrow \pi^+\nu\bar{\nu}$ and $K_L \rightarrow \pi^0\nu\bar{\nu}$ ”. In: *Rev. Mod. Phys.* 80 (3 Aug. 2008), pp. 965–1007. DOI: 10.1103/RevModPhys.80.965. URL: <https://link.aps.org/doi/10.1103/RevModPhys.80.965>.
- [53] Andrzej J. Buras et al. “ $K^+ \rightarrow \pi^+\nu\bar{\nu}$ and $K_L \rightarrow \pi^0\nu\bar{\nu}$ in the Standard Model: status and perspectives”. In: *Journal of High Energy Physics* 2015.11 (Nov. 2015). ISSN: 1029-8479. DOI: 10.1007/jhep11(2015)033. URL: [http://dx.doi.org/10.1007/JHEP11\(2015\)033](http://dx.doi.org/10.1007/JHEP11(2015)033).

- [54] Dario Buttazzo et al. “B-physics anomalies: a guide to combined explanations”. In: *Journal of High Energy Physics* 2017.11 (Nov. 2017). ISSN: 1029-8479. DOI: 10.1007/jhep11(2017)044. URL: [http://dx.doi.org/10.1007/JHEP11\(2017\)044](http://dx.doi.org/10.1007/JHEP11(2017)044).
- [55] C. Bobeth and A.J. Buras. In: *JHEP02(2018)101* (2018).
- [56] Nicola Cabibbo. “Unitary Symmetry and Leptonic Decays”. In: *Phys. Rev. Lett.* 10 (12 June 1963), pp. 531–533. DOI: 10.1103/PhysRevLett.10.531. URL: <https://link.aps.org/doi/10.1103/PhysRevLett.10.531>.
- [57] J. Carmignani and G. Ruggiero. “Neural Network Object Detection Approach (NNODA) for Photon Veto in Liquid Krypton (LKr) Calorimeter of NA62 Experiment”. In: *In preparation* (2021).
- [58] F. Chollet. *Deep Learning with Python*. Manning Publications Co., 2018.
- [59] “Convolutional Neural Networks with Event Images for Pileup Mitigation with the ATLAS Detector”. In: (2019).
- [60] Padraig Cunningham and Sarah Jane Delany. “k-Nearest neighbour classifiers: (with Python examples)”. In: *arXiv preprint arXiv:2004.04523* (2020).
- [61] A.V. Artamonov *et al.* In: *Phys. Rev. D* 79(2009)092004 (2009).
- [62] P.A. Zyla *et al.* “Particle Data Group”. In: *Prog. Theor. Exp. Phys.* (2020) 083C01 (2020).
- [63] J. H. Christenson et al. “Evidence for the 2π Decay of the K_2^0 Meson”. In: *Phys. Rev. Lett.* 13 (1964), pp. 138–140. DOI: 10.1103/PhysRevLett.13.138.
- [64] F. Mescia and C. Smith. In: *Phys. Rev. D* 76(2007)034017 (2007).
- [65] S. Fajfer, N. Košnik, and L. Vale Silva. *Footprints of leptoquarks: from $R_{K^{(*)}}$ to $K \rightarrow \pi\nu\bar{\nu}$* . 2018. arXiv: 1802.00786 [hep-ph].
- [66] FreeCodeCamp. “Demystifying Gradient Descent and Backpropagation via Logistic Regression based Image”. In: <https://www.freecodecamp.org/news/demystifying-gradient-descent-and-backpropagation-via-logistic-regression-based-image-classification-9b5526c2ed46/> (2020).
- [67] G. Buchalla and A.J. Buras. In: *Nucl. Phys. B* 548(1999)309 (1999).
- [68] F. Mescia G. Isidori and C. Smith. In: *Nucl. Phys. B* 718(2005)319 (2005).
- [69] P. Paradisi G. Isidori F. Mescia, C. Smith, and S. Trine. In: *JHEP08(2006)064* (2006).
- [70] S. L. Glashow, J. Iliopoulos, and L. Maiani. “Weak Interactions with Lepton-Hadron Symmetry”. In: *Phys. Rev. D* 2 (7 Oct. 1970), pp. 1285–1292. DOI: 10.1103/PhysRevD.2.1285. URL: <https://link.aps.org/doi/10.1103/PhysRevD.2.1285>.

- [71] Xavier Glorot. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of Machine Learning Research* (2011).
- [72] Ian Goodfellow. *Deep Learning*. The MIT Press, 2016.
- [73] Yuval Grossman and Yosef Nir. “beyond the standard model”. In: *Physics Letters B* 398.1-2 (Apr. 1997), pp. 163–168. ISSN: 0370-2693. DOI: 10.1016/S0370-2693(97)00210-4. URL: [http://dx.doi.org/10.1016/S0370-2693\(97\)00210-4](http://dx.doi.org/10.1016/S0370-2693(97)00210-4).
- [74] Xiao-Gang He et al. “Breaking the Grossman-Nir bound in kaon decays”. In: *Journal of High Energy Physics* 2020.4 (Apr. 2020). ISSN: 1029-8479. DOI: 10.1007/jhep04(2020)057. URL: [http://dx.doi.org/10.1007/JHEP04\(2020\)057](http://dx.doi.org/10.1007/JHEP04(2020)057).
- [75] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*. Vol. 398. John Wiley & Sons, 2013.
- [76] Gino Isidori, Federico Mescia, and Christopher Smith. “Light-quark loops in $K^+ \rightarrow \pi^+ \nu \bar{\nu}$ ”. In: *Nuclear Physics B* 718.1 (2005), pp. 319–338. ISSN: 0550-3213. DOI: <https://doi.org/10.1016/j.nuclphysb.2005.04.008>. URL: <https://www.sciencedirect.com/science/article/pii/S0550321305002932>.
- [77] Gino Isidori et al. “Exploring the flavour structure of the MSSM with rareKdecays”. In: *Journal of High Energy Physics* 2006.08 (Aug. 2006), pp. 064–064. ISSN: 1029-8479. DOI: 10.1088/1126-6708/2006/08/064. URL: <http://dx.doi.org/10.1088/1126-6708/2006/08/064>.
- [78] A.J. Buras J. Aebischer and J. Kumar. In: *JHEP12(2020)097* (2020).
- [79] M. Gorbahn J. Brod and E. Stamou. In: *Phys. Rev. D* 83(2011)034030 (2011).
- [80] C. Jarlskog. “Commutator of the Quark Mass Matrices in the Standard Electroweak Model and a Measure of Maximal CP Nonconservation”. In: *Phys. Rev. Lett.* 55 (10 Sept. 1985), pp. 1039–1042. DOI: 10.1103/PhysRevLett.55.1039. URL: <https://link.aps.org/doi/10.1103/PhysRevLett.55.1039>.
- [81] Makoto Kobayashi and Toshihide Maskawa. “CP-Violation in the Renormalizable Theory of Weak Interaction”. In: *Progress of Theoretical Physics* 49.2 (Feb. 1973), pp. 652–657. ISSN: 0033-068X. DOI: 10.1143/PTP.49.652. eprint: <https://academic.oup.com/ptp/article-pdf/49/2/652/5257692/49-2-652.pdf>. URL: <https://doi.org/10.1143/PTP.49.652>.
- [82] Patrick Komiske et al. “Learning to Remove Pileup at the LHC with Jet Images”. In: *Journal of Physics: Conference Series* 1085 (Sept. 2018), p. 042010. DOI: 10.1088/1742-6596/1085/4/042010.

- [83] Josua Krause, Adam Perer, and Kenney Ng. *Interacting with predictions: Visual inspection of black-box machine learning models*. New York, NY, USA: Association for Computing Machinery, 2016.
- [84] Paul Langacker. “The physics of heavy Z' gauge bosons”. In: *Rev. Mod. Phys.* 81 (3 Aug. 2009), pp. 1199–1228. DOI: 10.1103/RevModPhys.81.1199. URL: <https://link.aps.org/doi/10.1103/RevModPhys.81.1199>.
- [85] *LHCb : Technical Proposal*. Geneva: CERN, 1998. URL: <https://cds.cern.ch/record/622031>.
- [86] Louis Lyons. *Statistics for nuclear and particle physicists*. Cambridge University Press, 1986.
- [87] A.J. Buras M. Blanke and S. Recksiegel. In: *Eur. Phys. J. C 76(2016)182* (2016).
- [88] B. Duling M. Blanke A.J. Buras, K. Gemmler, and S. Gori. In: *JHEP03(2009)108* (2009).
- [89] G. Isidori M. Bordone D. Buttazzo and J. Monnard. In: *Eur. Phys. J. C 77(2017)618* (2017).
- [90] M. Tanimoto and K. Yamamoto. In: *Prog. Theor. Exp. Phys. 2016(2016)123B02* (2016).
- [91] Federico Mescia and Christopher Smith. “Improved estimates of rare K decay matrix elements from $K_{\ell 3}$ decays”. In: *Phys. Rev. D* 76 (3 Aug. 2007), p. 034017. DOI: 10.1103/PhysRevD.76.034017. URL: <https://link.aps.org/doi/10.1103/PhysRevD.76.034017>.
- [92] Yaser S. Abu Mostafa. *Learning From Data*. AML, 2012.
- [93] K. P. Murphy. *Machine learning a probabilistic perspective*. MIT Press, 2012.
- [94] Alexey Natekin and Alois Knoll. “Gradient boosting machines, a tutorial”. In: *Frontiers in Neurorobotics* 7 (2013), p. 21. ISSN: 1662-5218. DOI: 10.3389/fnbot.2013.00021. URL: <https://www.frontiersin.org/article/10.3389/fnbot.2013.00021>.
- [95] Andrew NG. *Deep learning specialization*. deeplearning.ai by Coursera, 2018.
- [96] Andrew NG. *Machine learning yearning*. Stanford e-book, 2018.
- [97] Vivian Ng and Leo Breiman. “Bivariate variable selection for classification problem”. In: *Technical report, Department of Statistics, University of California-Berkeley* (Jan. 2005).

- [98] “Review of Particle Physics”. In: *Progress of Theoretical and Experimental Physics* 2020.8 (Aug. 2020). 083C01. ISSN: 2050-3911. DOI: 10.1093/ptep/ptaa104. eprint: <https://academic.oup.com/ptep/article-pdf/2020/8/083C01/34673722/ptaa104.pdf>. URL: <https://doi.org/10.1093/ptep/ptaa104>.
- [99] Irina Rish et al. “An empirical study of the naive Bayes classifier”. In: *IJCAI 2001 workshop on empirical methods in artificial intelligence*. Vol. 3. 22. 2001, pp. 41–46.
- [100] G.D. Rochester and C.C. Butler. “Evidence for the Existence of New Unstable Elementary Particles”. In: *Nature* 160 (1947), pp. 855–857. DOI: 10.1038/160855a0(cit.onp.1).
- [101] Byron P Roe et al. “Boosted decision trees as an alternative to artificial neural networks for particle identification”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 543.2-3 (2005), pp. 577–584.
- [102] Adrian Rosebrock. “Fine-tuning with Keras and Deep Learning”. In: [*pyimagesearch Online article*] (2019).
- [103] Adrian Rosebrock. “Object detection with deep learning and OpenCV”. In: [*pyimagesearch Online article*] (2017).
- [104] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing*. MIT Press, Cambridge, Massachusetts, 1986.
- [105] S. Russell. *Artificial intelligence a modern approach*. Pearson Education, 2002.
- [106] N. Kosnik S. Fajfer and L. Vale Silva. In: *Eur. Phys. J. C* 78(2018)275 (2018).
- [107] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural networks* 61 (2015), pp. 85–117.
- [108] S. Sharma. “Activation Functions in Neural Networks”. In: [*towardsdatascience Online article*] (2020).
- [109] Erik Strumbelj and Igor Kononenko. “An efficient explanation of individual classifications using game theory”. In: *Journal of Machine Learning Research* (2010).
- [110] T. Blazek and P. Matak. In: *Int. J. Mod. Phys. A* 29(2014)1450162 (2014).
- [111] Ren Jie Tan. “Breaking Down Mean Average Precision (mAP)”. In: [*towardsdatascience Online article*] (2019).

-
- [112] Morimitsu Tanimoto and Kei Yamamoto. “Probing SUSY with 10 TeV stop mass in rare decays and CP violation of kaon”. In: *Progress of Theoretical and Experimental Physics* 2016.12 (Dec. 2016). 123B02. ISSN: 2050-3911. DOI: 10.1093/ptep/ptw160. eprint: <https://academic.oup.com/ptep/article-pdf/2016/12/123B02/10436266/ptw160.pdf>. URL: <https://doi.org/10.1093/ptep/ptw160>.
- [113] Simon Tong and Daphne Koller. “Support vector machine active learning with applications to text classification”. In: *Journal of machine learning research* 2.Nov (2001), pp. 45–66.
- [114] Raju Vaishya et al. “Artificial Intelligence (AI) applications for COVID-19 pandemic”. In: *Diabetes & Metabolic Syndrome: Clinical Research & Reviews* 14.4 (2020), pp. 337–339.
- [115] Lincoln Wolfenstein. “Parametrization of the Kobayashi-Maskawa Matrix”. In: *Phys. Rev. Lett.* 51 (21 Nov. 1983), pp. 1945–1947. DOI: 10.1103/PhysRevLett.51.1945. URL: <https://link.aps.org/doi/10.1103/PhysRevLett.51.1945>.