CellSight: Machine Vision Defect Detection During Automated Cell Assembly

CellSight: Machine Vision Defect Detection During Automated Cell Assembly Thesis
Author: Joe Henshall
Supervisor: Dr David Cheneler / Dr Xiaonan Hou
Date: 26/09/23

Table of Contents

1.0 – Introduction

During the last decade there has been a paradigm shift in our society to move away from non-renewable energy sources. The Paris Agreement, a legally binding international treaty, was signed by 196 parties in 2015 to establish a climate goal for the near future [1]. The established climate goal places a higher focus on renewable energy sources and storage methods. Battery science has presented itself as one of the most promising technologies to achieve the sustainable future outlined in the Paris Agreement. It is estimated that the demand for lithium-ion battery energy storage could reach 9300 Gigawatt hours by 2030. A 17-fold increase from the demand of 0.5 Gigawatt hours back in 2010 [2]. This surge in production volume has led to "an 85% decline in prices" [3], making many energy storage applications viable for the first time in history.

The rise of renewable energy sources such as wind and solar power has been one of the driving factors in the accelerated deployment of energy storage systems. Moreover, the exceptional growth of the electric vehicle market has become a serious alternative as manufacturers seek to meet stringent emissions regulations and consumer demands for cleaner transport.

Because of this exponential rise, industry leaders and research groups around the world are frantically pursuing innovation in the field of battery science to produce the next optimal chemistry [4]. Much of this research is completed by small research teams producing batteries by hand, which is both demanding on the operator, time inefficient and unreliable. In addition, most of these cells are built inside pressurised glove boxes where visibility and tactile feedback is further reduced. Repeatability is also an important factor in research as handmade cells have a number of uncontrollable variables surrounding the build process that muddy the waters around testing.

Cellerate are a company that look to optimise the process of battery research and development. They manufacture robots that fully autonomise the process of coin cell production. Their core system takes coin cell components in a loading tray format, automatically assembles and crimps them together to produce working cell [5]. Their system looks to not only speed up the build process, which has been quoted by earlier adopters by a 30% increase in speed over cell manufacture by hand, but to also improve repeatability as each cell is constructed with the same maximal accuracy. The operator is also free to do other tasks while the system is making a cell, saving further resources.

During automated assembly it is possible that sub optimal components are used, resulting in sub optimal cells. These cells go on to absorb test channel resources that could otherwise be used on cells with controlled variables and quality. This project looks to offer maximum visual data on the cell build process back to the operator to optimise their workflow. Defects and features detected by the systems described in this project during the build process can be inextricably linked to a cell and quickly referenced when analysing the cells performance. This gives research teams more analysis tools when reviewing cell data to discard bad cells that were improperly made and to change their processes to reduce the rate of bad cell production moving forward. In addition, the cell component quality data can be used to optimize component sourcing and preparation. This project looks to actively drive forward the rate of battery development which is not only a benefit to the adopters but also, ultimately, the environment as the switch to greener energy can be accelerated.

1.1 - Aims

The aim of this project is to produce a computer vision system that can not only dynamically locate cell components in an image, for alignment purposes, but can also inspect the quality of cell components by detecting defects. These results shall be inextricably linked to a cell and give operators a vital resource when correlating test data to cell build reports.

1.2 - Objectives

1. Isolate a cell component within an image.
2. Produce an algorithm to perform auto exposure to accommodate different lighting conditions.
3. Produce an algorithm to find the outer contour of the cell component and perform morphological analysis to determine:
    a. Edge quality
    b. Punching overlap
    c. Curvature
    d. Component alignment
4. Develop a machine learning model that can determine the presence of:
    a. Electrode creases
    b. Surface scratches
    c. Cracks
    d. Flaked electrode coating
5. Integrate the developed algorithms into the existing Cellerate system.

2.0 - Literature Review

Computer vision encompasses many technologies such as cameras, edge computing, cloud-based computing, software and artificial intelligence that can be combined and used to extract meaningful information from digital images [6]. Humans perceive 80% of all impressions through the sense of sight [7] and with 88% of people ranking sight as their most valuable sense [8], it is clear that capturing the real world through vision data and harnessing the power of computer vision is a huge leap with almost endless applications to enrich our existing systems. This has never been more applicable as the compression, reduction in cost and increase in power of computer chips in recent years has led to increased expansion of the Internet of Things (IoT). As more systems become smart and connected there is more demand for interaction and control of real world systems and it is clear that vision data is the biggest opportunity.

It is still not fully understood how the brain interprets light entering the eye into understanding about the surrounding environment. However, It is believed that groups of neurons in the back of the brain called V1 and V2 neurons react to edges in the image, combining edges of similar orientations and supressing those orthogonal to that orientation at the same location, a process called cross orientation [9]. These cross oriented edge combinations are assembled in various ways to allow us to detect various shapes and give meaning to light information [9].

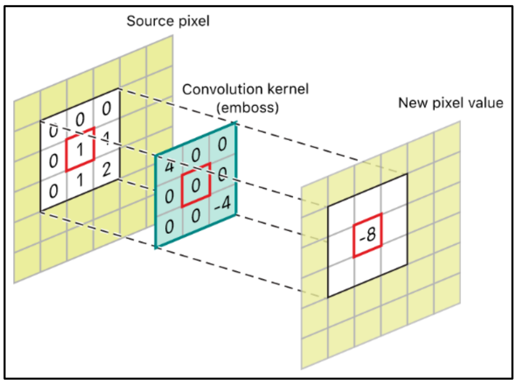


Fig.1 – Convolutional kernel [51]

This understanding was applied to the computer vision problem by means of the convolutional kernel. A kernel is a matrix, that can be multi-dimensional and any size, which is slid across an image and multiplied with the input such that the output is changed [10]. These form the most fundamental mathematical process in computer vision and describe how useful information can be extracted from pixel data. The weights within the kernel can be altered to produce a wide range of effects. These can range from edge detection, blurring, sharpening and filling in gaps.

Traditional computer vision and image processing techniques are somewhat outdated with the rise of deep learning and convolutional neural networks, due to their adaptability. Traditional computer vision techniques, such as edge detectors and contouring algorithms, are very rigid in their implementation. They are very susceptible to random variance in their environments and applications. For example, lighting changes in real time systems can have drastic effects on the success of an edge detector as the bounds for the edge detection are defined. Of course, dynamic traditional computer vision methods can be implemented, but there is always the vulnerability that unseen edge cases can destabilize the tuning of the system. Although these methods are somewhat outdated, it is important to understand the theory behind these approaches to gain more of an understanding of how a modern convolutional neural network can achieve a better result.

Many computer vision problems rely on the ability to discern between the composite objects within an image. Humans perform this processing innately and intuitively know where an object ends, and another begins. Programming computers to perform similar tasks automatically is a challenge as a computational process must be defined to determine features and ultimately separate one object from another within an image.

One powerful technique that is referenced through a large body of image processing and computer vision literature is edge detection [11] [12]. As the name suggests, edge detection allows the edges to be determined within an image. This information can be used to locate objects within an image for further processing. There have been many iterations of edge detecting algorithms throughout the development of the literature; Sobel, Prewitt, Laplacian, Robert [12] are some examples. However, the most widely adopted edge detector was developed by John Canny in 1986, described in A Computation Approach to Edge Detection [13].

It is through the isolation of the most important edges in an image that the location of objects can be identified. The most popular computer vision programming library, OpenCV [14], provides functions to find contours against these edges which provides coordinate representations of the edges that can be used in analysis.

Due to the proven usefulness of the technique, the edge detector is still used extensively for computer vision problems. Even modern techniques, such as deep learning using convolutional neural networks (CNN), which have somewhat replaced traditional computer vision and image processing methods, use edge detectors extensively in their convolutional layers [15]. Moreover, even though their use in the most modern computer vision approaches is mostly with deep learning, traditional image processing techniques, such as edge detection, still have applications.

A problem when using the canny edge detector with OpenCV is that the intensity thresholds that determine an edge are static throughout the image. What is found in practice is that found edges can be discontinuous and contain gaps. This leads to frustration when trying to optimally set the Canny thresholds to preserve as much of the target edge as possible. Moreover, applying this to many images with varying image parameters, such as exposure and lighting conditions, is exponentially harder. The work presented in [16] provides a method to actively recover edges that the eye would consider continuous, but the edge detection method has considered broken. The work uses a snakelet method [16] in which each piece of the broken edge is considered as such. Two energy equations, internal and external, are used to quantitively predict how to join the snakelets together and recover the disjoined edge. The internal energy equation concerns the snakelets existing points and the external energy equation considers the points of the other snakelets. What has been shown in [16], is that this method successfully and accurately reconnects edges that are intuitively seen as continuous from the human perspective.

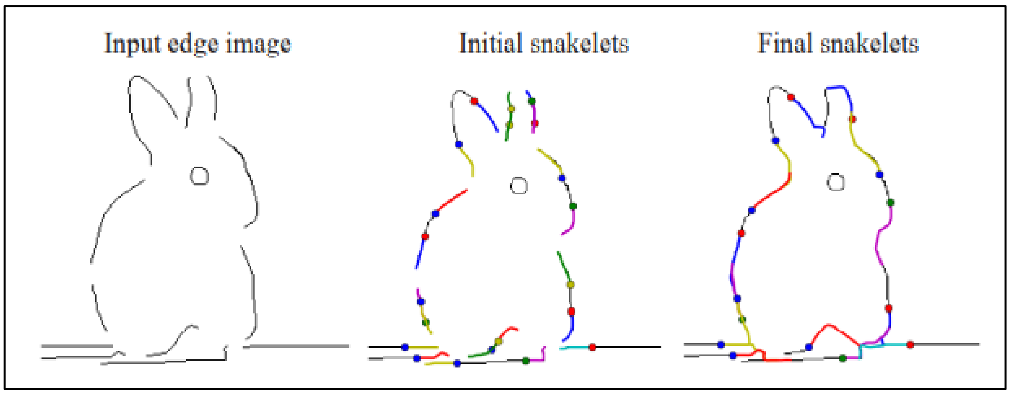


Fig.2 - Active canny recovery with snakelets [16]

Another key method in traditional image processing is the Hough transform [17]. It is a statistical approach for line and edge detection that can be used after a canny threshold to determine shapes in an image such as lines and even circles. It does this by finding statistical hot points between many imaginary lines centred at points on a given intensity boundary [17]. These hot points give statistical confidence that vertices or circle centres are found there by computing each point’s allegiance with other points in the image. This a very important concept as it allows threshold images to be described mathematically using combinations of lines and shapes. This allows contours in images to be described by mathematical structures of points, allowing computer algorithms to analyse their shapes and positions. This information is crucial for real world applications such as object detection and control systems.

Another popular technique used in traditional computer vision applications is background subtraction. Often with computer vision tasks there is focus on tracking or recognising an object or objects in an image and performing some action based on this information. This could be controlling another device or saving information about the objects for analysis. Of course, to do this the computer needs to make the distinction between the parts of the image that are useful and contain the object and the parts that are not useful. The process of background subtraction gives the computer a reference to what constitutes the background in an image. If the background is treated as a ground truth then subtracting this known background from an image will leave behind the foreground and areas of interest. A popular algorithm to perform this is known as the Mode of Gaussian (MoG) algorithm. This algorithm uses multiple Gaussian distributions to model each pixel in an image [18]. The weights of the mixture represent the amount of time the colour of a pixel has remained the same, indicating the background. As the pixel remains the same for a longer duration the probability that it belongs to the background in an image increases [18] [19]. This effect creates a dynamic model of the background that adapts with the scene as lighting conditions change and other things come in and constitute the background of an image. Fig.3 shows this concept. It should be noted that The MoG algorithm relies on a video feed to determine the background as the time component is critical to provide the weightings. However, a similar approach can be used with still images if an appropriate representative background is provided and the state of the scene in the image can be controlled. This is most applicable to repeatable scenarios with fixed backgrounds such as production lines or repetitive robotic tasks.

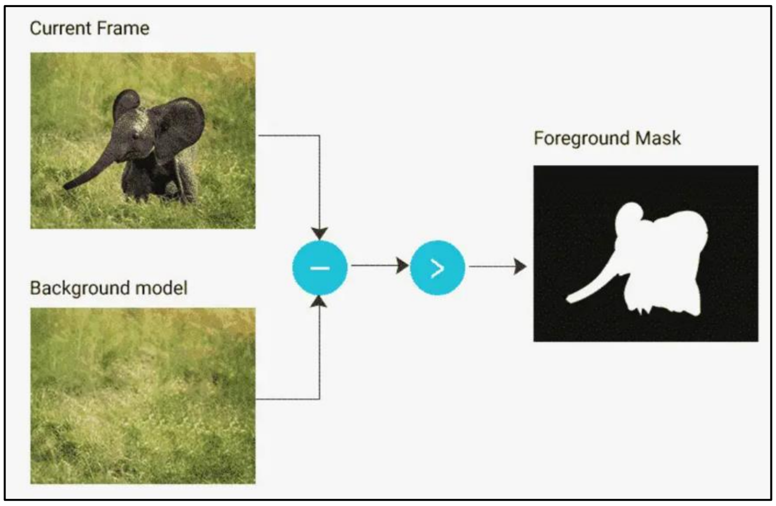


Fig.3 - Mode of Gaussian background removal [57]

The most novel and ground-breaking work in the field of computer vision over the last decade has been through the use of deep learning and convolutional neural networks. However, traditional computer vision approaches by manual feature extraction and defined computational approaches still have useful, but usually specific, applications. The work in [20] describes a computer vision system that detects edge defects in sheet steel rolls. Using a Sobel edge detector [12], the edge of the roll can be identified. A Hough transform [17] searching for horizontal lines is then be applied to determine the edge gradient, which can be plotted. Any deviation of the signal from a flat response is known to be a defective area and can be identified. The work [20] showed reliable results over numerous widths of material. It is important to consider traditional computer vision techniques when appropriate as they are much simpler to implement and are computationally less expensive than equivalent deep learning approaches.

The next stage in the development of computer vision was to introduce artificial neural networks (ANN) into the fold to make predictions on images [15]. ANNs mimic neural pathways in the brain which fire in a specific way leading to a decision being made. ANNs have some set of input nodes which contain parameterised values. These are then fully connected to a hidden layer of neurons, which are then fully connected to another layer of neurons and so on. The number of hidden layers is variable but can become computationally expensive as the number increases. The neurons fire based on a nonlinear activation function, most commonly the ReLU function [15]. Finally, the last hidden layer is connected to an output layer consisting of the defined set of classes on which to make the classification [15].

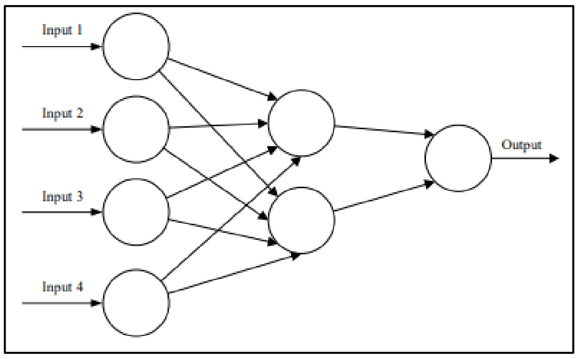


Fig.4 - Simple ANN [15]

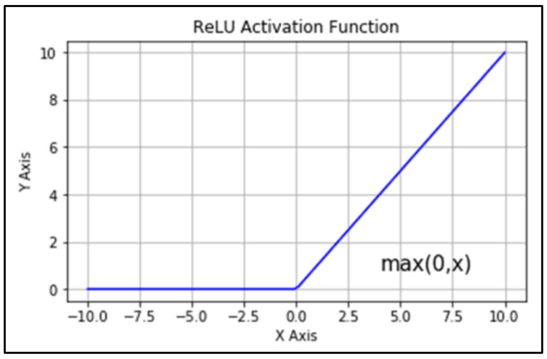


Fig.5 - ReLU Function [58]

Each node and connection between nodes have an adjustable weight assigned to it. Using a training dataset, an optimisation process is conducted to best adjust the weights to have the network fire correctly and classify an image to what is expected. This process should train the network to give the correct output classification response to varying input stimuli [15].

The challenge when using ANNs in computer vision problems is providing input values, this is known as feature extraction. What is it about an image that you can train the network to pick up on and classify based upon? Feeding each pixel into the input layer of a network would give you all the information you would ever need about an image, but the network would become impractically large and contains far too many parameters to realistically optimise. The literature in [21] describes defect detection system using an ANN. A series on invariant moments are calculated from an image which are fed into the input layer of an ANN. These invariant moments are based on regional shape recognition that is maintained when the image is subject to rotation, ratio, scale and pan [21]. They ultimately result in a series of scalar values about an image that the ANN can be trained to recognise and classify the image based upon. The defects are subdivided into 6 subcategories: pits, holes, scratches, burrs, indentations and smearing. The system detected the defects to an accuracy of: 88.9%, 94.4%, 94.4%, 96.1%, 96.3% and 90.6% respectively [21]. This work speaks to the advocacy of using ANNs when the circumstances allow for reliable feature extraction.

The problem encountered when trying to apply the methodology of ANNs to wider computer vision problems is that of feature extraction. Diverse datasets make feature extraction very difficult as the target object within the image will vary greatly in size, position, aspect ratio, colour and so on. And as mentioned previously, feeding each pixel into the input layer of a neural network is not feasible. Deep learning using convolutional neural networks was the breakthrough technology that solves this problem.

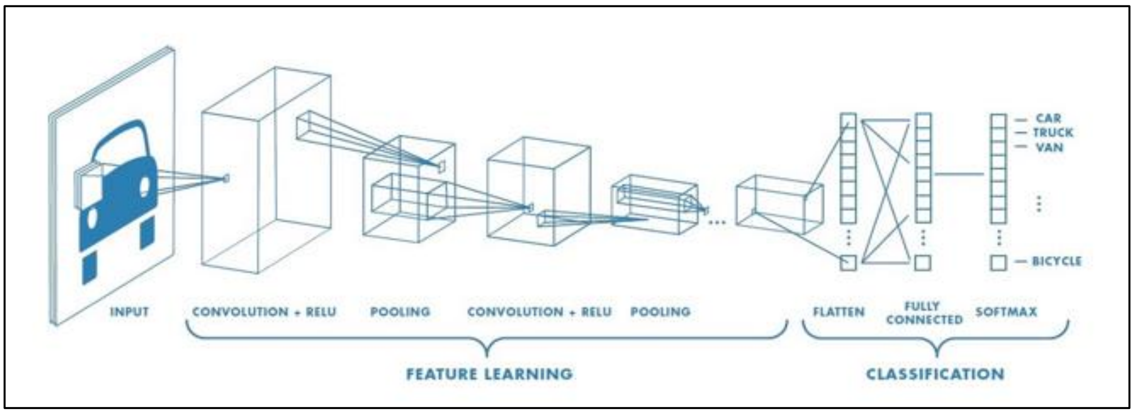


Fig.6 - Convolutional neural network architecture [59]

Convolutional neural networks (CNN) allow the network to learn, through the optimisation process, which features of an image are the best to classify it [15]. It is through the learning of the feature extraction stage that defines 'deep' learning. These features are extracted through many layers of kernel convolutions [22]. The first layer may only contain features as simple as horizontal edges, corners and so on. But when combinations of combinations of convolutions are combined, filters for picking out features such as noses, eyes and ears can be optimised [15] [22], see Fig.7. The convolutional layers reduce the spatial dimension of the image but increase the feature dimension with each successive stage. The pooling stage is also used to reduce the spatial dimension of the image through sampling. The result is a flattened image with a tiny spatial dimension but with large feature depth. This can then be connected to a fully connected ANN network for final classification. These output classifications could be binary (Dog or not a Dog), probabilities that the image contains one of many classes or pixel-wise, where each pixel in the original image is given a class: semantic segmentation [23].

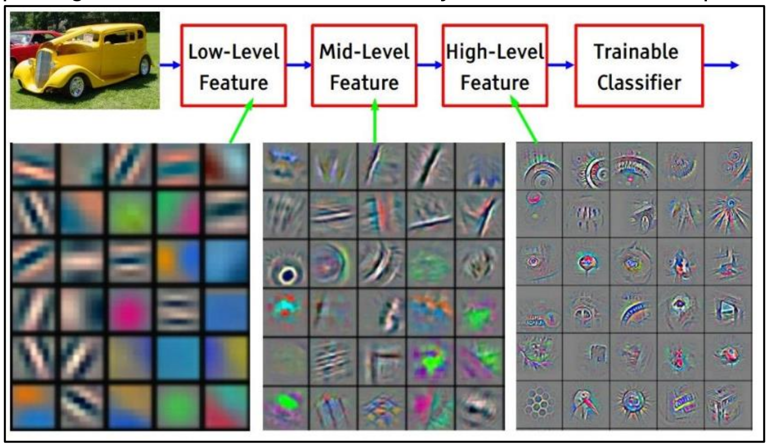


Fig.7 - Deep learning feature maps [60]

A project that used CNNs in a practical sense in defect detection is described in [24]. The work details a system to detect and classify the coating on material from 5 distinct coating classes: cracked, running, orange peel, adhesion failure and defect free. The work compares several different CNN architectures to evaluate which yields the best results: VGG19, Resnet50, Xception, MobileNetV2 and Densenet121. The training dataset contained 1500 images distributed between the classes listed. It was found that Resnet50 and DenseNet gave the best results of 97% and 96% accuracy respectively [24]. The method shows promise with great performance metrics. In addition, the number of images in the dataset is typically low. The high success rate with this amount of

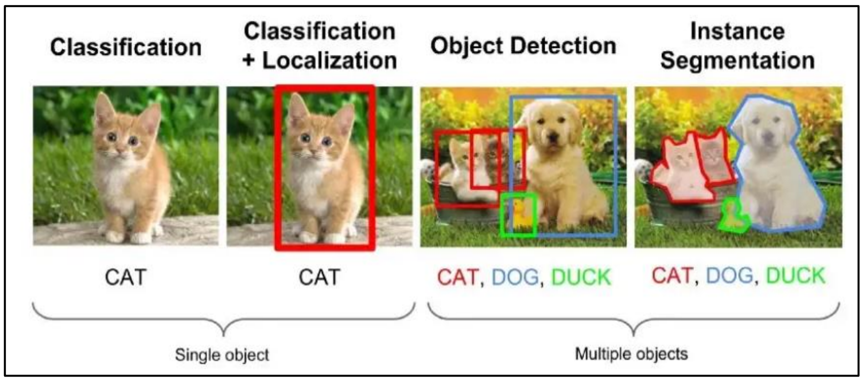


Fig.8 - Types of image classification [61]

training data can be attributed to the low variance in the context of the images. The more specific the problem and dataset can be, the less variety of context are needed in the training data

An image does not have to be classified as a whole, different classes can exist within the same image and be distinguished separately. This is known as object detection. Commonly, bounding boxes around objects in an image are generated along with the class the object belongs to. The most common object detection algorithm in the literature is the YOLO (you only look once) algorithm [25]. The YOLO algorithm splits the image into an NxM grid. It makes a bounding box prediction at each grid reference for an object that is centred there [23]. Any convolutional neural network backbone can be adapted for the YOLO algorithm, the difference being the classification head at the end of the network is replaced with an object detection and classification head [23]. The object detection and

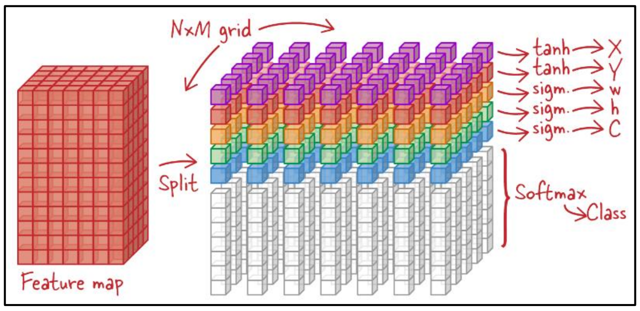


Fig.9 - Classification head of the YOLO algorithm [23]

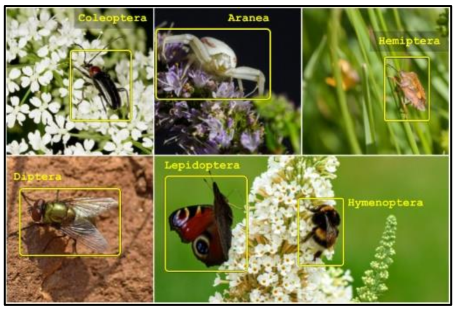


Fig.10 - Output examples from YOLO algorithm [23]

classification head predicts the *x* and *y* coordinates of the centre, the width and height of the bounding box, a confidence factor and the class of the object.

The third way of classifying an image is by semantic segmentation. This goes a step further than object detection by making pixel-wise classifications, resulting in much higher accuracy of identification within the image. The most popular and notable network architecture for achieving semantic segmentation is U-Net [26]. U-Net was initially devised for biomedical applications and won the 2015 ISBI cell tracking challenge [23]. The network design excels in biomedical applications as it is receptive to small training datasets, as large datasets are very rare for biomedical applications [26]. However, the literature [26] relies on strong use of data augmentation**,** such as elastic deformation, rotation and cropping on the available training images to enrich the data set [26].

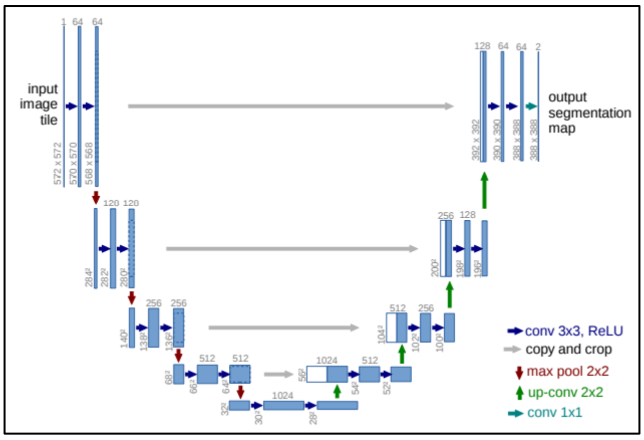


Fig.11 - U-Net architecture [26]

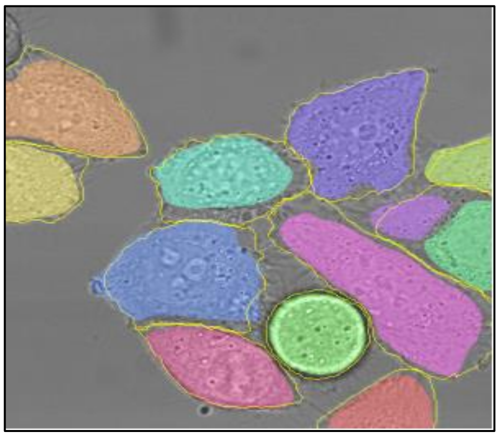


Fig.12 - Semantic segmentation of cells with coloured masks [26]

Unlike typical CNN architecture, the network does not contain any fully connected layers. The network has a contracting path (left) and expanding path (right). The contracting path utilises repeated application of two 3x3 convolutions (unpadded), each followed by a ReLU and a 2x2 max pooling operation with stride 2 for down sampling [26]. At each down sampling step, the number of feature channels are doubled. Every step in the expansive path consists of an up sampling of the feature map followed by a 2x2 convolution (up convolution) that halves the number feature channels [26]. A concatenation with the correspondingly cropped feature map from the contracting path is then performed followed by two 3x3 convolutions and a ReLU activation function [26]. At the final layer a 1x1 convolution is used to map each 64 component feature vector to the desired number of classes [26].

U-Net was able to achieve 92% accuracy when classifying PhC-U373 cells and 77% accuracy when classifying DIC-HeLa cells compared to 83% and 46% respectively of the second price network in the 2015 ISBI cell tracking challenge [26]. Because of these outstanding results U-Net has become very well adopted in the literature surrounding image classification by semantic segmentation. Moreover, in the field of defect detection in cell assembly, semantic segmentation allows for localised information to be fed back about the defect. This information could be critical for traceability of defective cells or to optimise the workflow to avoid repeating failure.

The works detailed in [27] looks to further improve on U-Net by making it more lightweight and efficient. ELU-Net was developed for the brain tumour segmentation benchmark (BraTS) 2018 Challenge and the ISBI liver tumour segmentation benchmark (LiTS) 2017 Challenge, in which it was one of the top performing architectures and the top performer respectively [27]. The architecture builds on U-Net by implementing deep skip connections, in which layers in the contracting path are concatenated directly to the corresponding layers in the expanding path [27]. These deep skips enable full capture of “fine-grained details and coarse-grained semantics in the encoder” [27] as the features extracting in the encoder stage are the key to image segmentation. Therefore, deep skips help maintain these features.

The network architecture, when built using a ResNet backbone, contains only ~4% of the parameters of U-Net 3+, making it much more lightweight and easier to train [27]. The method also scored highest Dice coefficient, 97%, of any of the state-of-the-art semantic segmentation networks that were tested when applied to the ISBI LiTS dataset [27]. This variation on U-Net is specifically optimized for resource constrained applications, such as the one in this work, and should be considered.

Another implementation of U-Net implemented for lightweight applications is the system, described in [28]. The network is again, designed for semantic segmentation of cells in biomedical images. However, it has been optimised by applying methods that reduce useless information in the images as a pre-processing step. Removing redundant information in the images allows for the network to be reduced in complexity as a result. This is done by applying the watershed algorithm [29] which removes the non-edge pixels, leaving only the boundaries between cells [28]. This reduction in information in the input images allowed for a reduction in the number of convolutional layers and features channels when compared to standard U-Net [28]. This optimisation resulted in a 94%, 83% and 68% reduction in memory consumption, training time and testing time respectively compared to U-Net [28].

An example of the U-Net architecture for use in a defect detection application is described in [30]. It adapts the U-Net architecture by implementing SE-Res blocks in the skip connections. The SE-Res block comprises of a convolutional layer, a global max pooling layer, two fully connected layers, proportional method block, a skip path and an add block. The SE-Res block performs adaptive weighting on each channel to suppress harmful or invalid channels [30].

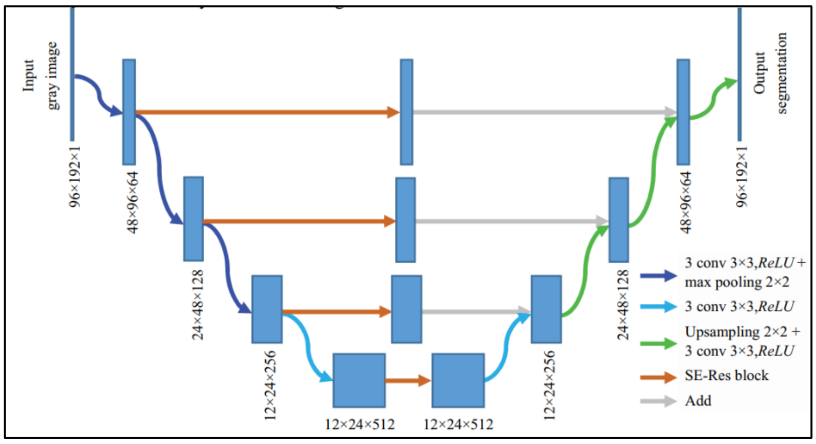


Fig.13 - SE- U-Net Architecture [30]

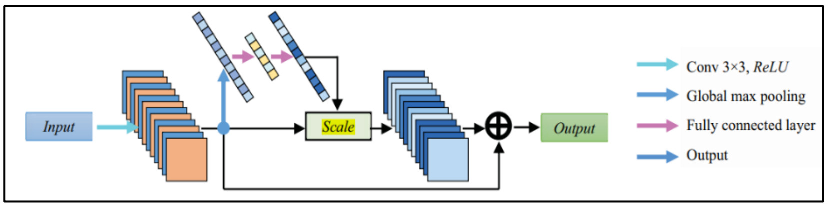


Fig.14 - SE-Res Block [30]

The system was trained using a small dataset of 1344 samples, subdivided into 6 defect categories: Blowhole (115), Crack (57), Fray (32), Break (85), Uneven (103) and Free (952) [30]. As the average defected area in the original dataset images does not exceed 10%, extensive data augmentation was used to enrich the images by copying closed graphics onto empty spaces. A test accuracy of 97.3% was achieved using SE-U-Net, a 2.5% increase on standard U-Net [30]. The test data was also augmented with increasing suppression ratios to simulate uneven lighting conditions. It was shown that SE-U-Net increasingly outperformed U-Net as the suppression ratio was increased, resulting in 7.5% better accuracy compared to U-Net for the largest suppression ratio tested [30].

Another application of U-Net and semantic segmentation for defect detection is shown in [31]. The paper describes an adapted U-Net architecture: BSU-Net used for defect detection in sheet steel and magnetic tile manufacture. BSU-Net was adapted with the objective of maintaining small details and defects through network traversal. To achieve this, BSU-Net varies the size of the kernel convolutions on the last layer of the encoding stage and first layer of the decoding stage from a 5x5 kernel to a 15x15 kernel. Using larger kernel sizes gives better semantic segmentation accuracy and the mixing of different convolutional kernel sizes can enhance the performance of the network [31]. In addition, BSU-Net includes a FEN network to process the input image and concatenate the results with the image pre and post the enhanced U-Net. The FEN network allows for better maintenance of smaller details by magnifying then reducing the feature dimension of the image, avoiding the loss of tiny defects by continuous sampling [31].

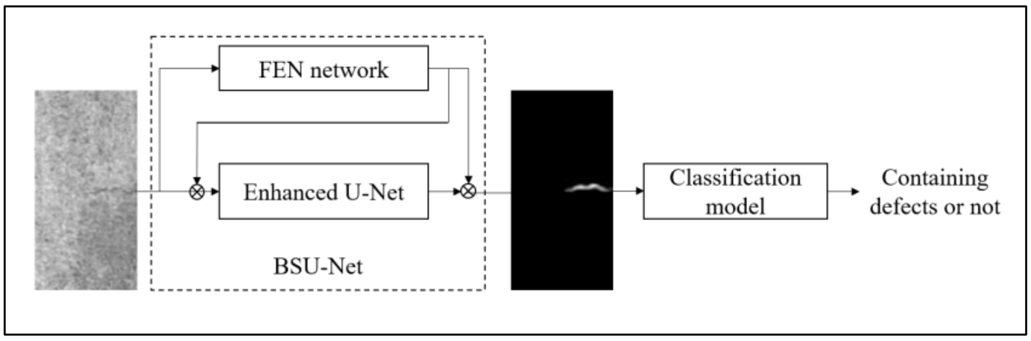


Fig.15 - BSU-Net Architecture [31]

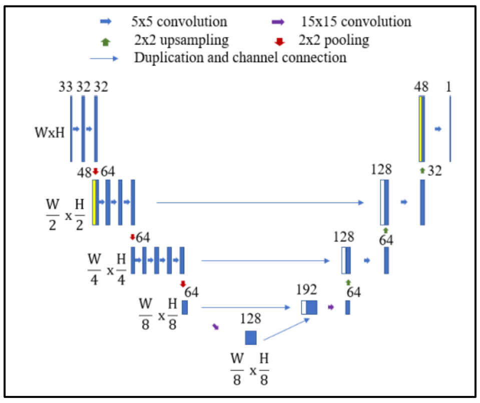


Fig.16 - Enhanced U-Net Architecture [31]

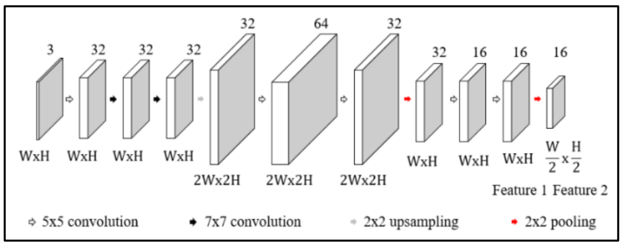


Fig.17- FEN Architecture [31]

The network was trained using steel and magnetic tile datasets containing 10,671 and 1,009 images respectively, both using a 7:3 split between training and testing images. The network was measured to achieve an accuracy of 90.2% using the steel network and 75.0% using the magnetic tile dataset, an improvement of 8.83% and 6.15% when compared to the standard U-Net architecture. However, the most staggering improvement was the intersection of union (IOU) improvement using the steel dataset, an increase of 45.7%. This metric describes the degree of agreement between the ground truth pixels and the predicted pixels.

Another implementation of U-Net for defect detection is described in [32]. This work, again, looks to develop a network capable of detecting defects in sheet steel. The network design uses a U-Net architecture but makes use of a residual block with a skip connection.

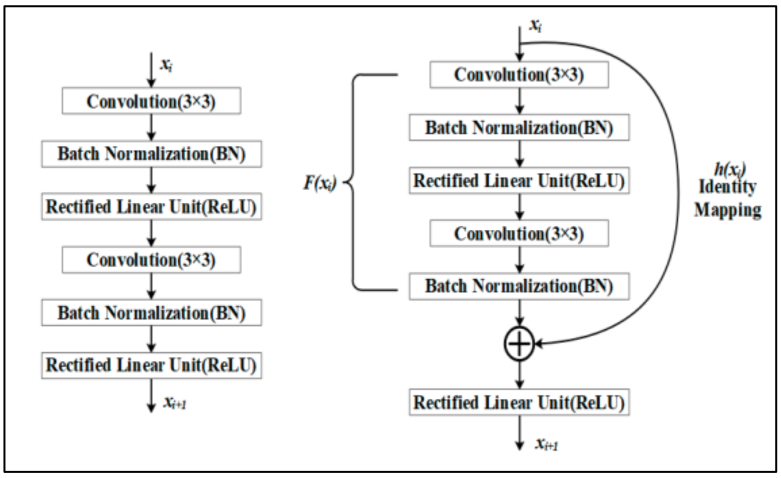


Fig.18 - Common neural network unit and residual neural network unit [34]

It is observed that accuracy can saturate when the network has learned the intricacies of the data. A first reaction to this problem may be to add more layers to the network, but this can sometimes lead to even worse performance. This is known as degradation [33]. The residual block looks to overcome this limitation and contains two parts: identity mapping and the residual. Identity mapping integrates the input with the output of the residual to maintain subsequent feature information [34] and to solve the problem of gradient disappearance and non-convergence. Moreover, any poor ability to extract features in the residual part of the block is compensated [34]. Using residual blocks have also been shown to improve the training of the network and using U-Net structure allows for the use of skip connections to facilitate information propagation without degradation [32]. The network was trained using a dataset of 12568 images for training and 1801 images for testing, distributed between 4 classes of defects [32]. The network scored a dice coefficient of 0.731, which is a metric of pixelwise agreement between the ground truth and the network predictions [32]

As mentioned previously, the success of these machine learning models is partly down to the amount of data available to train the network on. Therefore, more bespoke applications of the technology can be limited due to the limited amount of data available to teach the network. The success of the U-Net architecture, as described earlier in the paper, was also partly down to its receptibility and positive results with relatively smaller, bespoke datasets of medical images. As briefly mentioned previously, a technique that many of the papers [26] [30] already discussed use to enhance their results is data augmentation. Data augmentation is the process of enriching a dataset by taking the original images and applying various transformations such as rotations, flips, translations, crops and zooms to artificially create many more images to train the network on, Fig. 19 shows this concept. Data augmentation helps to prevent the model from overfitting the data [35] which occurs when the network 'memorises' the training data and does not generalise the information. The means it cannot respond to unseen data and will give poor results as it's only optimised to respond to the training data. If done correctly this process can increase the size of the training dataset dramatically, providing more context to the network to learn how to classify. It has been shown in [36] that by using data augmentation the validation accuracy can be increased by as much as 5%, which is a great improvement in terms of validation accuracy and through only reuse of the current data. Of course, over augmentation of the data can be detrimental to the performance of the model. If the dataset is over augmented the original context of what is trying to be achieved could be lost. Care should be taken deciding on not only the amount of augmentation to perform but also the types of transformations to perform. For example applying stretches and warping to pictures of cats will produce images that don't resemble cats at all and this can muddy the waters with the features the network should be learning.

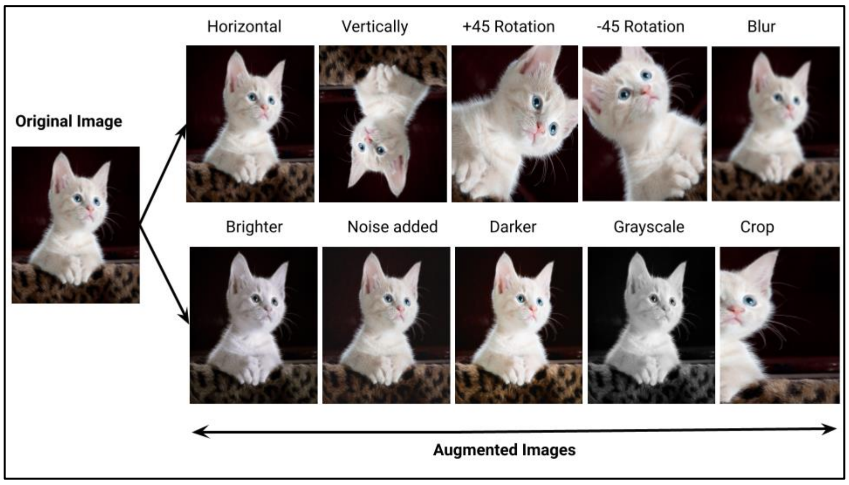


Fig.19 - Data augmentation [52]

Many conclusions can be drawn from the review of the literature surrounding this topic. Through review of the major milestones of the technology including traditional image processing techniques, manual feature extraction paired ANNs, convolutional neural networks and classification to very modern semantic segmentation models it is important to recognise the pros and cons of using each technology.

Although easy to implement and low computational demand, traditional image processing techniques for computer vision are rigid and difficult to optimise for dynamic systems. Consistency in the use case is required to properly define the processes and thresholds for algorithms like edge detectors.

It is clear after reviewing the modern state of computer vision projects that using convolutional neural networks is essential to get the best results. This is partly due to fantastic open source libraries such as TensorFlow [37] and Keras [38] becoming available, allowing anyone to be able to create and train their own networks. Also, this is partly due the availability and power of modern GPU chips ever increasing which allows for training the networks in a reasonable time period.

It is clear that some of the requirements for the project are better suited to applying traditional machine learning techniques where strictly the shape of the cell component is required. Electrode curve, edge quality and punching overlap are examples of these. A machine learning approach would not be suited to analyse these criteria. On the other hand the remaining requirements absolutely lend themselves to a machine learning approach. Furthermore, the type of classification that would be most applicable to this problem would be a semantic segmentation approach. This gives more extensive localisation of the defects which could be useful information for an end user to refine their processes. The most successful semantic segmentation model that exists in the literature is U-Net. There are many variations on this architecture and the one that seems to suit best is very dependent on the dataset. With this in mind, extensive testing and investigation will need to be done, using the dataset in this project, to find the optimal architecture for this problem.

3.0 - System Architecture

3.1 - Hardware Configuration

The project looks to integrate a software package into the existing Cellerate system. The system, shown in Fig.20 comprises of multiple modular elements that can be passed individually through an antechamber to the controlled environment of an glovebox.

The autoloader module of the system feeds trays, which are loaded with the various components of the cell to the assembler. The XY Stage and Z Axis placer work together to build up the cell layer by layer. It is during this process that this project will capture an image of each component, using the top and bottom cameras, sequentially and perform analysis on their condition. Once the cell has been constructed it is passed to the sealer which applies a large pressure to seal the cell ready for test. The cell is finally returned to a position in the autoloader and the next cell is then loaded.

This equipment provides value due to its versatility, reconfigurability, repeatability and throughput. Moreover, it allows the user to set multiple cell configurations and build programs to run automatically to explore a large parameter space, even overnight if required. This frees up the time of research scientists to complete more urgent tasks that require more ingenuity and creative thinking. Moreover, the repeatability of the cell build process eliminates randomness in the build process and restricts the variables to only those being explored in the tests.

The addition of the system developed during this project will allow scientists to have a full understanding of the quality of the cell, including any defects that are present and the alignment of the components. This allows for isolation of dud cells and freeing up of vital test channel resources for those that were built to a consistent standard. In addition, this information allows for optimisation of their component preparation methods to eliminate possible reoccurring defects as these will become clear through the cell inspection.

Following the definition of requirements, it was determined that the existing system hardware would suffice to achieve the project objectives. The system is currently controlled by a Raspberry Pi 4 microprocessor. Two camera modules are used for active alignment of cell components as they are placed during the build process. The design choice to fully utilise the existing components is both an economical and design decision as the physical space within the system is lacking due to the arrangements of existing components.

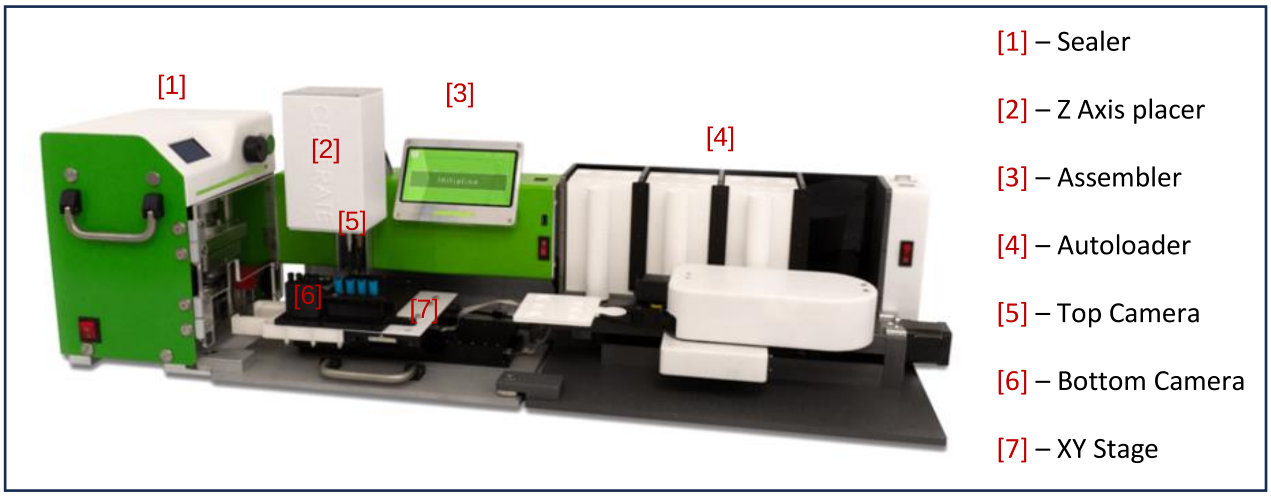


Fig.20 - Cellerate Automatic Cell Assembly System [5]

3.1.1 - Coin Cell

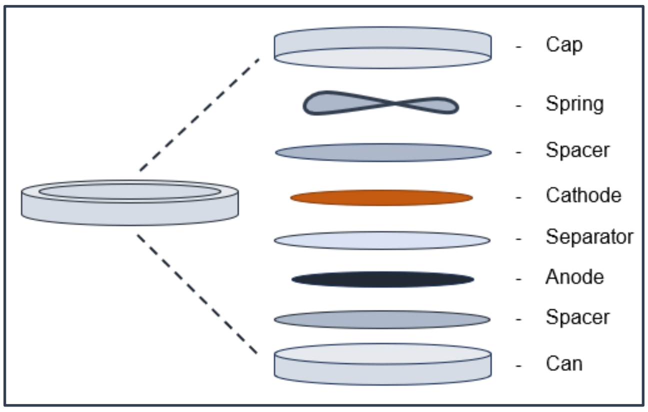


Fig.21 - Typical coin cell build configuration

Coin cells are used in the battery research space as a small scale analogue to the large battery packs used in most applications. They are used to assess various chemistries and cell designs before scale up. Fig.21 shows the cross section and of a typical coin cell. The Cellerate system assembles coin cells by sequentially placing each component onto a build plate, using active alignment to centre each component. When the stack is complete the entire arrangement is fed into the crimper unit which applies a large pressure to the stack, locking the components in place. The typical coin cell, shown in Fig.21, consists of a: case, spacer, positive electrode, separator, negative electrode, spacer, spring and cap in order bottom to top. Electrolyte is also dispensed at various stages in the build process by an electronically controlled pipette.

3.2 - Software Design

The main bulk of the work for this project comes through the design and implementation of various software modules. The software will be implemented in the Python programming language and any additional imported distributed modules will be highlighted. Moreover, each module described will be implemented as a Python class which are instantiated in various places within the codebase. Fig. 22 shows the hierarchal design of how the software modules will integrate with the existing Cellerate system software. Additional flowcharts for the processes described can be found in Appendix A.

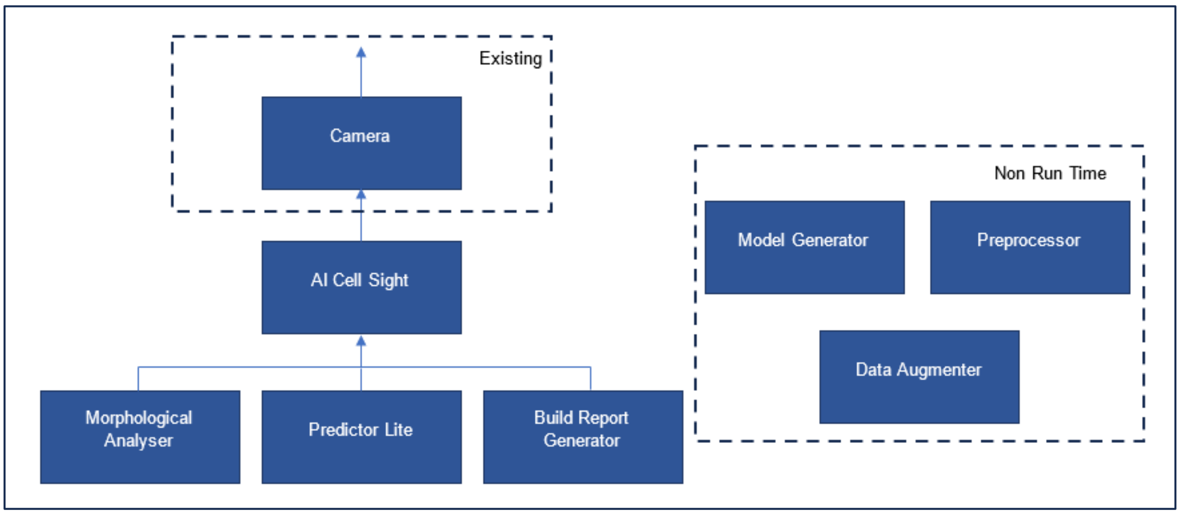


Fig.22 - Hierarchical Class Diagram for the Project Software

3.2.1 - AI Cell Sight Module

The AI Cell sight module is the parent module to the system. It is instantiated by the existing camera module within the current Cellerate system software. The module has been designed to contain all the high level functions that describe the overarching functionality of the system, such as finding component alignment, performing morphological analysis, performing surface inspection and generating the final build report. The module instantiates the morphological analyser, predictor lite and build report generator modules and controls their running.

3.2.2 - Morphological Analyser Module

The Morphological Analyser module controls the processes surrounding the morphological or shape based analysis of the component. The methodology involves using background subtraction methods and contouring to isolate the outer contour of the component and subsequent mathematical analysis on the shape of the component can be done to determine: the component alignment, the component curvature, the component edge quality and the presence of punching overlap.

3.2.3 - Predictor Lite Module

The Predictor Lite module controls the machine learning surface inspection analysis. It is named ‘Lite’ because of the implementation of Tensorflow Lite as opposed to the Tensorflow. TensorFlow Lite is optimised to run on resource constrained and mobile applications [39] such as the Raspberry Pi 4 platform. The module controls inferencing the trained machine learning model with a target image to yield predictions to the presence of: creases, scratches, flaked coatings and cracks.

3.2.4 - Build Report Generator Module

The Build Report Generator module has been designed to manage the creation of the build report. The report consists of a feedback slide which highlights the found defects and morphological analyses. It is designed to be a quick reference to the user containing all the critical information of the build process.

3.2.5 - Model Generator Module

The Model Generator module is one of three standalone modules that are not implemented on the target Raspberry Pi platform and is used specifically for model development and run on a more powerful, GPU enabled PC. This module contains the Keras python code to describe the neural network model architectures that will be evaluated in this work. The module is used for the compiling and training of the machine learning models used throughout the project. It controls the Keras implementation of the model architectures, the preprocessing of the training data and the training procedure and validation. The module uses both the Tensorflow [37] and Keras [38] libraries to achieve this. In addition a script provided by [40] was used as a reference.

3.2.6 – Preprocessor Module

The Preprocessor Module is another standalone module that has been designed to control the processing of the raw image/mask pairs into the suitable format to be used in the model training process. This involves assembling the separate defect masks into a master mask for each image with the correct pixels encodings for each defect type. Moreover, the image/mask pairs have to be divided into suitably sized tiles of 256x256 pixels for training. The useless tiles that contain no useful defect information are also discarded.

3.2.7 – Data Augmenter Module

The Data Augmenter Module is another standalone module that has been designed to control the augmentation of the training data for the model optimisation. It applies various operations such as shifts, flips, crops and resizing to enrich the original dataset and increase its size by a large factor. This artificial expanding of the dataset can provide more context to the training process and increase the accuracy of the models when used correctly. The script provided by [41] was used a reference for this module.

4.0 – Morphological Analysis

The first body of work surrounds morphological analysis of the cell components. This analysis can be performed using traditional image processing techniques and does not require a machine learning component. This analysis concerns the shape of the cell component, from this the amount of curvature, punching overlap and edge quality can be ascertained.

4.1 – Isolating the Component

To perform analysis on the morphological aspects of the cell components they first must be isolated within the image. The neural pathways in the human brain have evolved to perform this processing without intent and can recognise where one object end and another begins. A computer stores an image as a three dimensional array of values, and cannot intuitively know how to segment an image into its constituent components. Using image processing libraries, such as OpenCV [14], the cell components can be isolated within an image and analysis can be performed.

4.1.1 – Approach 1: Dynamic Canny Edge Thresholding and Filtering

The first approach taken to isolate the electrode within the image was to find the optimal combination of pre-processing methods available within OpenCV to fit this specific application.

4.1.1.1 Methods

The problem was defined as being able to isolate the outer edge of the cell component. The literature surrounding using edge detectors, such as a Sobel or Canny edge detector [13] [12], suggests to firstly convert the image to grayscale to produce a 2D image array and then to apply a blur filter of some kind.

To determine the optimal combination of pre-processing to isolate the cell component edge an experiment was designed and conducted. The stages of pre-processing were broken down into the flow chart shown in Fig.23.

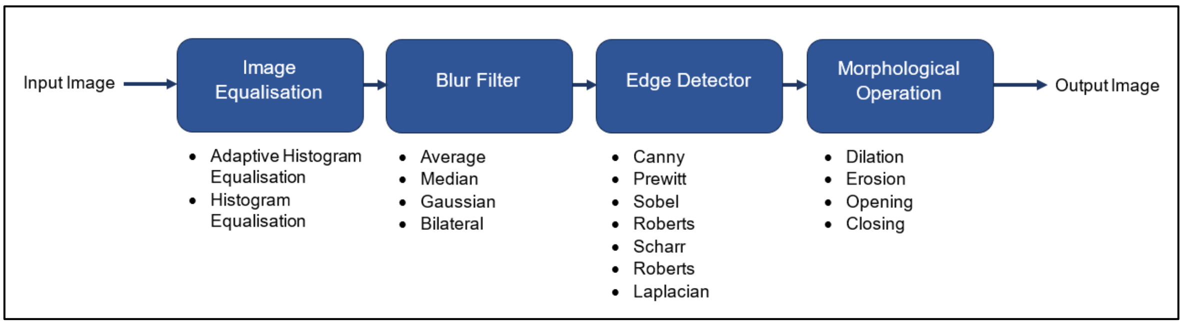


Fig.23 – Preprocessing Flowchart

To account for the requirement of dynamic lighting compensation, an image equalisation stage was also included which distributes the range of pixel values more evenly across the available range.

Each of the methods listed were tried in combination with each other to determine the optimum set of pre-processing stages to isolate the outer edge of the cell component.

4.1.1.2 Results

It was determined that the optimal combination of pre-processing methods to best isolate the outer edge of the cell component was as described in Fig.24.

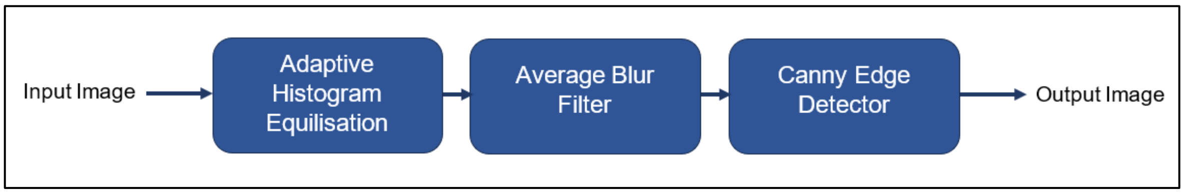


Fig.24 – Optimal Preprocessing Flowchart

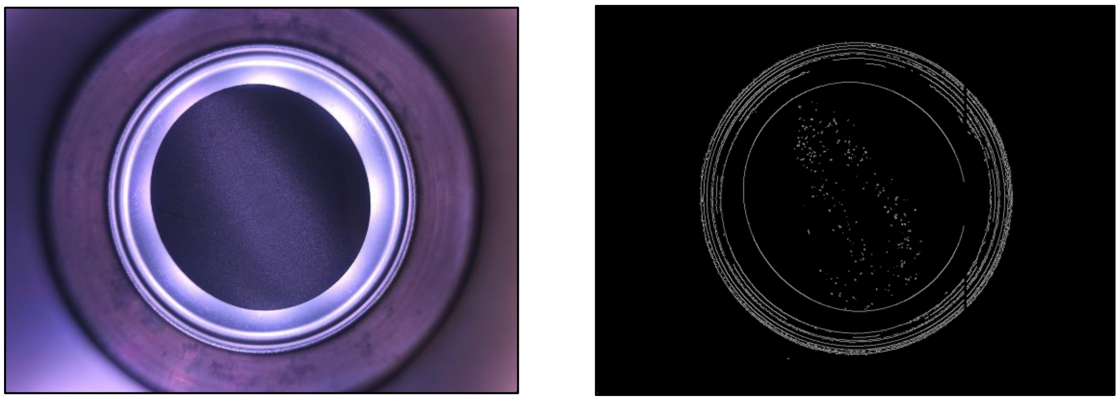


Fig.25- Input image of cell component from top camera (b) Output image after pre-processing steps in Fig.24 have been applied

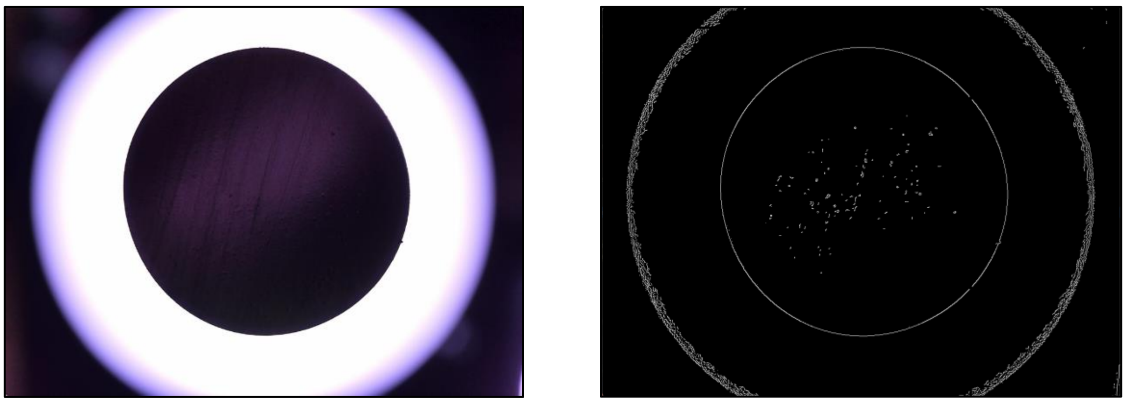


Fig.26 – (a) Input image of cell component from bottom camera (b) Output image after pre-processing steps in Fig 24 have been applied

Fig.25 and Fig.26 show the results of the pre-processing steps when applied to the corresponding source images from the top camera and bottom camera respectively.

4.1.1.3 Discussion

As can be seen from Fig.25 and Fig.26 the algorithm performed well on the bottom camera image, successfully isolating the outer edge of the cell component for analysis further down the pipeline. This is due to the large focal difference between the cell component in the foreground and the background of the image. The results from the top camera suffer from the fact the component of interest and the rest of the build plate exist on the same plane, meaning that when the pre-processing is applied the edge detector picks up noise and edges that are of no interest to this process. Extensive filtering and cropping are required to isolate the useful outer edges of the component.

After applying the methods described further in this paper it was clear that this method was not sufficient as it became impossible to differentiate between noise and the actual component edge in excessively noisy images. Moreover, the static thresholds for the canny edge detectors proved to be ineffective to dynamic changes in the environment from image to image. As lighting conditions and material types for the components changed, so too did the optimum thresholds for the edge detector to isolate the correct edges.

4.1.2 – Approach 2: Background Subtraction and Thresholding

Due to the sensitivity of lighting conditions and complexity to isolate the electrode when dealing with excessive noise from the top camera, a new method was considered for acquiring the component's outer edge.

4.1.2.1 Methods

An attribute of how images are captured on the Cellerate system is that images are always captured from the same repeatable position with high precision. This allows the method of background subtraction to be used: to set a static ground truth image of the build area and subtract it from the target image with the desired component present [42]. This subtraction should leave the parts of the image that are not common with the ground truth image: the cell component. Fig.27 shows this concept.

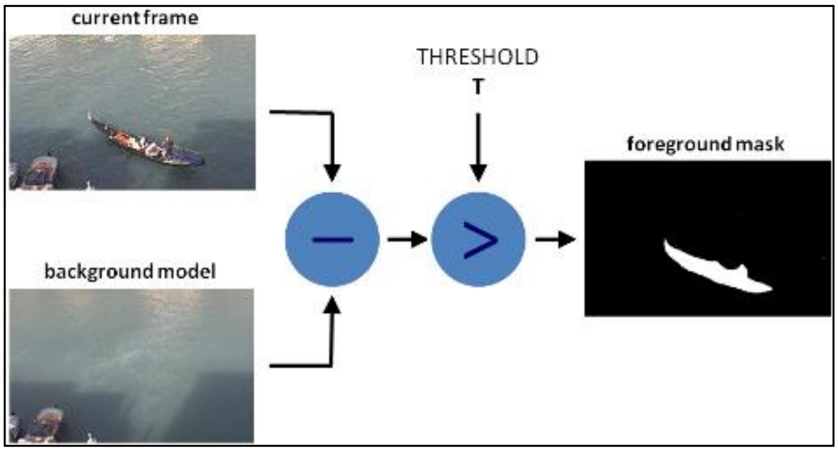


Fig.27 - Background subtraction method [28]

This implemented methodology was based on what is described in [42] with an additional step of a HSV (Hue, Saturation, Value) filter, which was included to remove noise after the subtraction. Small changes in the lighting conditions within the room can cause subtle differences in the background space between the ground truth and the target image. The HSV filter, optimally calibrated for each different material type, removes this noise to leave the target component. Fig.28 shows the flowchart for the final background subtraction process.

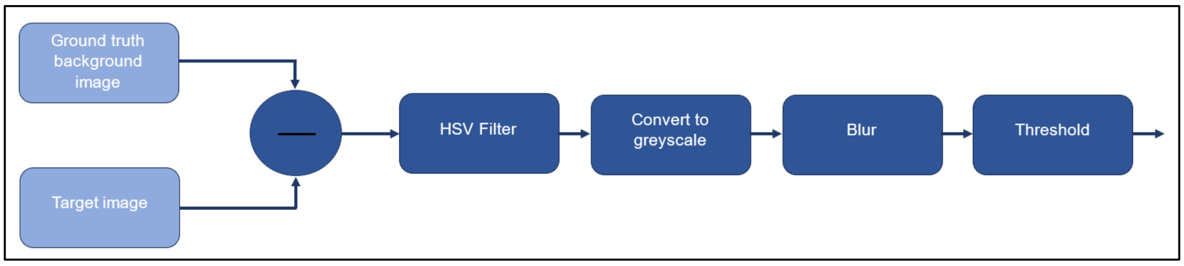


Fig.28 - Updated method of background

4.1.2.3 Results

Fig.29 and Fig.30 show the input images to the background subtraction algorithm and the output respectively. As can be seen, the method of background subtraction leaves a much better representation of the cell component as compared to results of Section 4.1.1. The noise seen in Fig.21 has been eliminated and what remains is the only part that is required: the component.

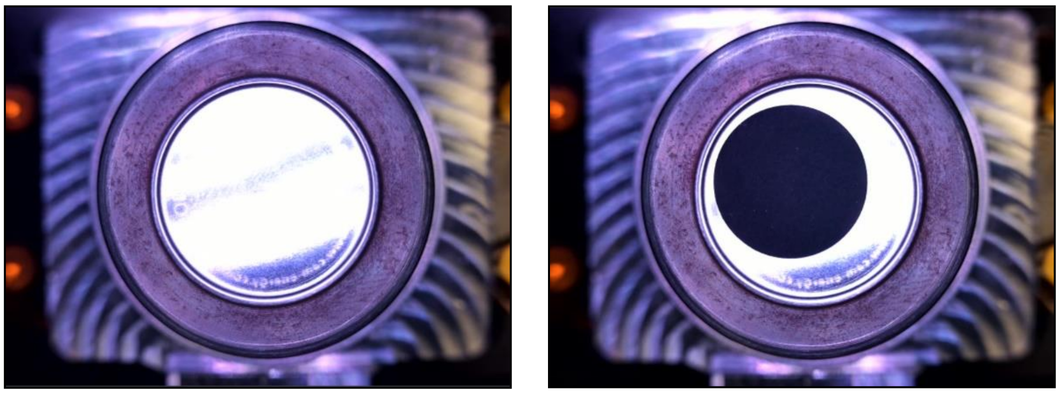


Fig.29 - (a) Ground truth background image taken from the top camera (b) Target image taken from the top camera.

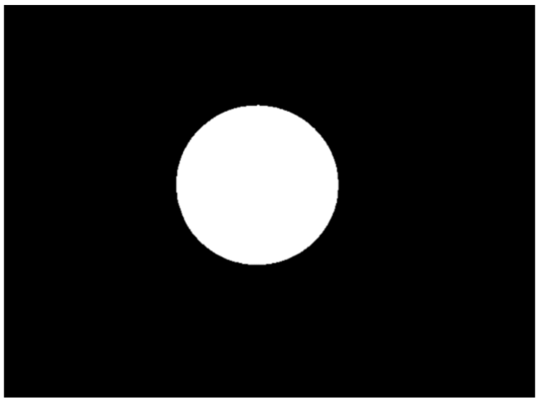


Fig.30 – Output image after thresholding

4.1.2.3 Discussion

The method of background subtraction has shown to address the limitations of the first approach. That is, the top camera does not have the benefit from a focus difference between the target component and the background. Because of this, a lot of noise can pass through the edge detection process that is not useful for morphological analysis. Background subtraction not only addresses this by removing everything that is similar between the images, but also compensates for changing lighting conditions that the first approach can also suffer from. If the lighting conditions are consistent between the background image and the target image they will be accounted for by the subtraction. Moreover, this method reduces the execution time of the algorithm as the extensive filtering that was required in the first approach to remove the noise is no longer needed. In addition, this method is more suitable for multiple analyses on multiple components as the cell is built. This is because the ground truth image is dynamic and can be updated with the current state of the build pad before the next component is added, isolating each component as the cell is built.

A negative aspect of this method is that an extra image must be captured for each component: the background image. Moreover, to optimally isolate different components of different materials, a calibration process that determines the optimal settings for the HSV filter should be completed for each material. However, the trade-off for these concessions is a much more accurate isolation of the target component in the image, shown in Fig.30, that can then be fed downstream for analysis on its outer edge for morphological analysis.

4.2 - Edge Quality Analysis

Section 4.1 describes how the component is isolated within an image for analysis. Contours can be found on the image in Fig.30 to produce a data structure that details the points that make up the outer edge of the cell component for analysis. Fig.31 shows the contours found on image in Fig.30.

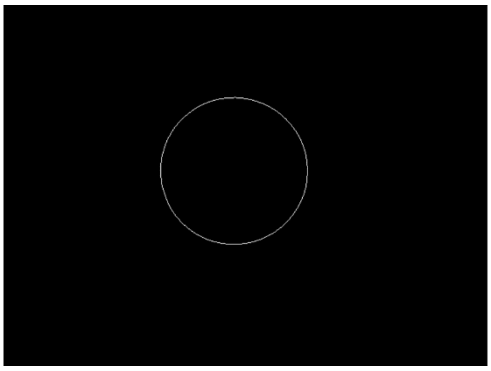


Fig.31 - Image Contours

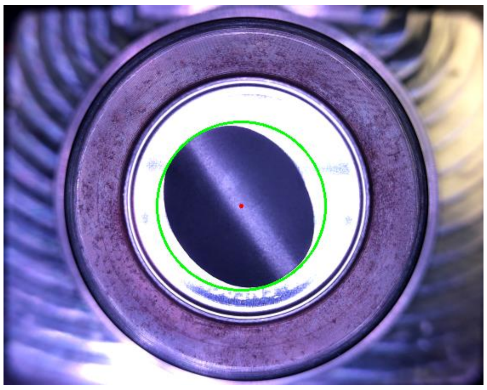


Fig.32 - Curved electrode compared to ideal circle

The first morphological requirement of the system is to assess the quality of the edge of the component. An ideal edge on a perfectly circular and flat component would be resemble a perfect circle. However, in practice components are often not perfectly circular when viewed from a plan view above or below due to component curvature, an example of this is shown in Fig.32. Therefore, comparing the edge of a component to an idealised circle that encompasses it would not give an accurate result even though curved components may still have pristine edges.

To solve this issue, a method of comparing the component edge to idealised version of itself is presented. The component edge is filtered in such a way to remove the rough edge (noise) and present a smoothened version of itself. The actual edge can then be compared to the smoothened version to determine how noisy the actual edge is, and therefore it's quality. This method accommodates components that are not completely flat and circular like the curved electrode in Fig.32. The following sections describe the methodologies to achieve this smoothened contour.

4.2.1 – Approach 1: Smoothing by Erosion/Dilation

The first approach that was taken was to use large morphological operations [43] in erosion and dilation, otherwise known as opening, to smoothen out the component contour so an assessment of the edge quality could be done.

4.2.1.1 – Methods

The process of using operations such as dilation and erosion change the shape of a binary image by growing or shrinking the object pixel area respectively [43]. They can be useful for closing in contours that have gaps or by removing noise in the form of flecks on a binary image [43].

In this application, a large kernel of (150x150) was used to erode the component contour shape down followed by a dilation of the same size to expand it. Finding the contours on the resulting shape was shown to give a smoothening effect, as shown in Fig.33 and Fig.35. The imperfections

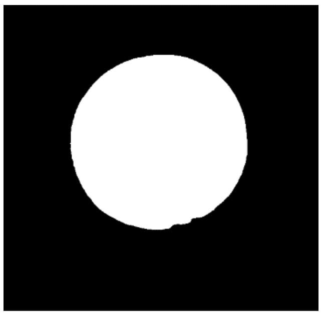


Fig.33 - Original contours found on the component

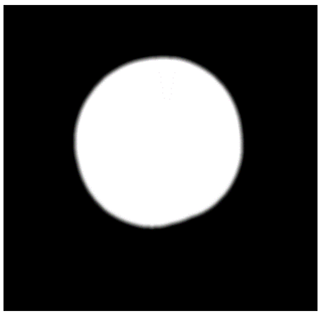


Fig.34 - Result of erosion/dilation process

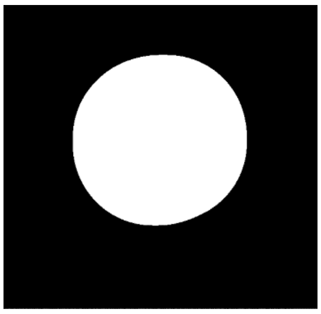


Fig.35 - Found contours on the resulting shape from the erosion/dilation process

in the edge present in the original contours are shown to have been removed and an idealised version of the original shape is found.

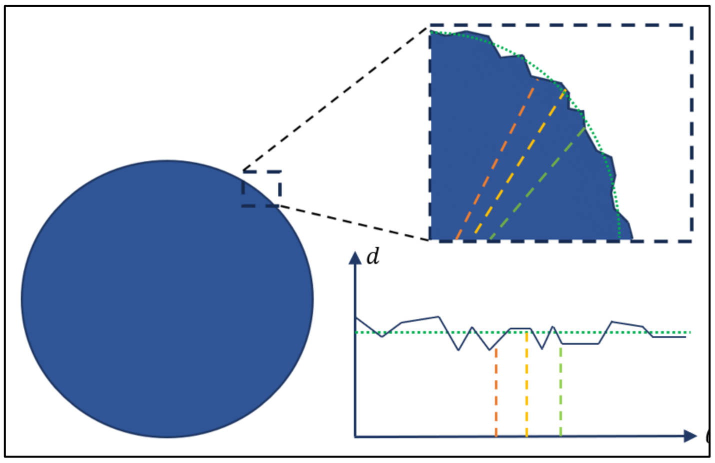


Fig.36 – Edge quality analysis method

The contour points on both the original contour and the smoothened version can then be compared to determine the edge quality. To do this the method shown in Fig.36 is implemented. Each point on the actual contour is sampled and its distance from the centre is determined. The corresponding radial distance to the smoothened contour point, shown in green on Fig.36, is also found and its distance sampled. The distances are then compared to determine the percentage error between the idealised, smoothened contour point and the actual contour point. Note that the green line representing the radial distance of the smoothened contour points will never be constant as the component will likely have some curvature, resulting in a non-ideal circle when viewed from above or below.

4.2.1.2 – Results

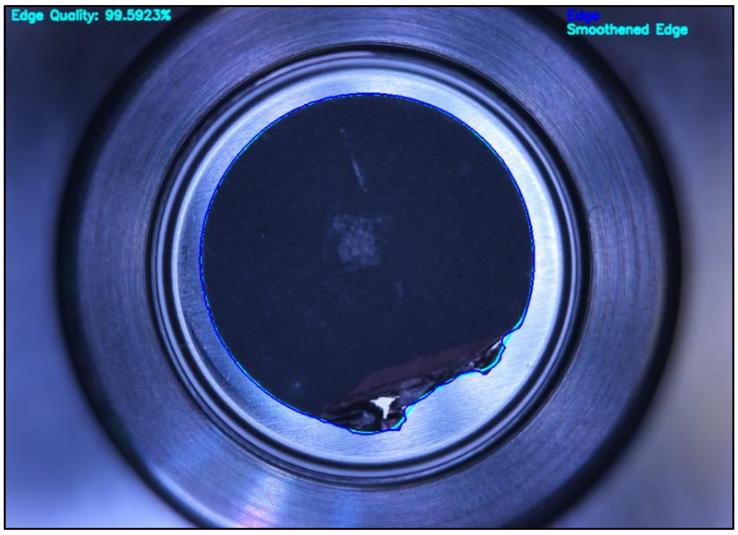


Fig.37 - Edge quality detection output

As can be seen from Fig.37 , the algorithm was able to produce estimates for the quality of the edge.

4.2.1.3 Discussion

When profiling the execution time demand of the algorithm running on the Raspberry Pi, it was revealed that the edge detection stage of the algorithm was causing significant slowdown. The erosion/dilation processes were found to be highly resource demanding as each pixel in the image has a large kernel operation associated with it. Therefore, as the image or kernel increases in size so does the amount of processing required to perform morphological operations. Moreover, the method of quantifying edge quality, shown in Fig.32, is also highly inefficient. This is due to the method of matching a contour point on the actual contour to a corresponding point on the smoothened contour. There is no association between them as smoothened contour points are found by a secondary process on the shape after the erosion/dilation operation. This means there is no correspondence between the n^{th} actual contour point and the n^{th} smoothened contour point. Because of this, a best matching point must be found. Each point on the contour must be checked and compared to identify the optimal point for comparison. This is done by checking distance from the smoothened points to the line that passes through the centre of the component to the actual contour point in question. This is, of course, highly inefficient as the number of comparisons required is:

$$No.of\ comparisons = N_{smooth} \cdot N_{actual}$$

Where N_{smooth} is the number of number smoothened contour points and N_{actual} is the number of contour points on the actual contour.

4.2.2 – Approach 2: Smoothing by 1D Blur Kernel

Due to the poor performance of the method discussed in Section 4.2.1 a new approach was considered.

4.2.2.1 - Methods

The method in [44] describes applying a 1D blur kernel to the contour points. The blur kernel samples each contour point using a window of width n, replacing the contour point with the average value of the points within the window. The method can be visualised with a kernel size of 3 in Fig.38. The red box represents the kernel window.

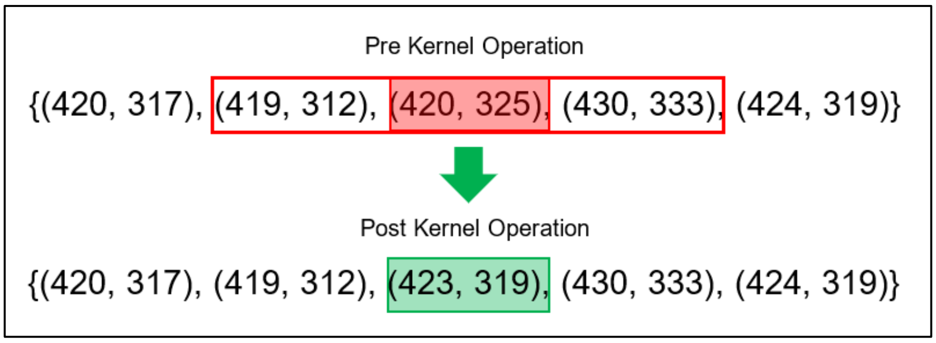


Fig.38 - Blur kernel operation

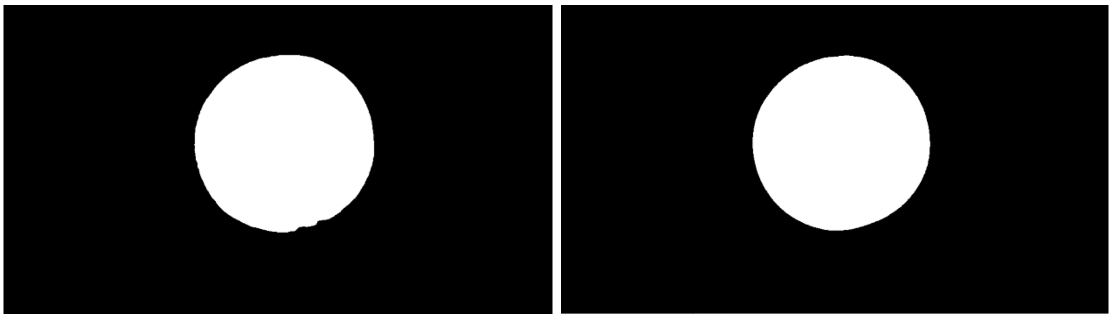


Fig.39 - (a) Actual contour of component (b) Resulting contours of 1D blur kernel smoothing

The method shown in Fig.38 was used with a much larger kernel size of 80 and applied to the entire contour points data structure. Fig.39 shows the results of the blur kernel smoothing technique. The imperfections have once again been removed and a smoothened contour remains that can be used as a reference of comparison to determine the edge quality of the component. The method achieves the same result as the approach described in Section 4.2.1, but with much less processing.

4.2.2.2 Results

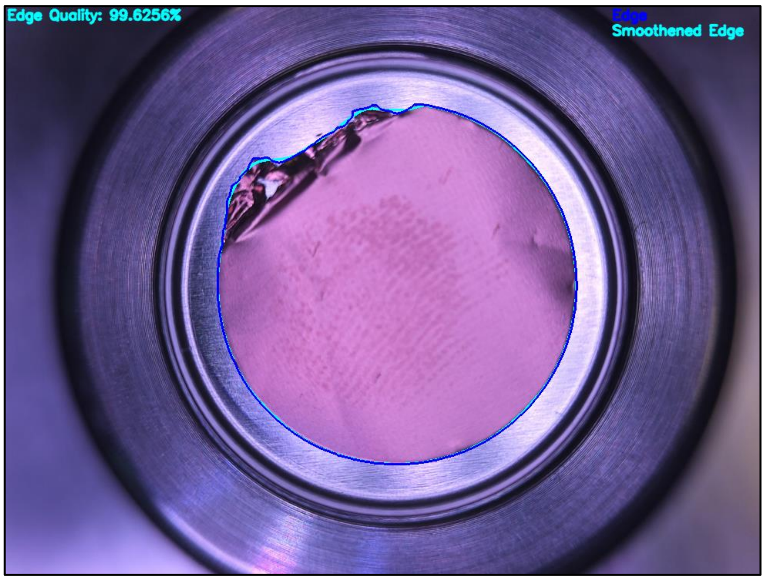


Fig.40 - Edge quality detection output from top camera

As can be seen from Fig.40, the algorithm was able to produce estimates for the quality of the edge from images as per the previous method. Fig 41 and 42 show the effect of the optimisations when

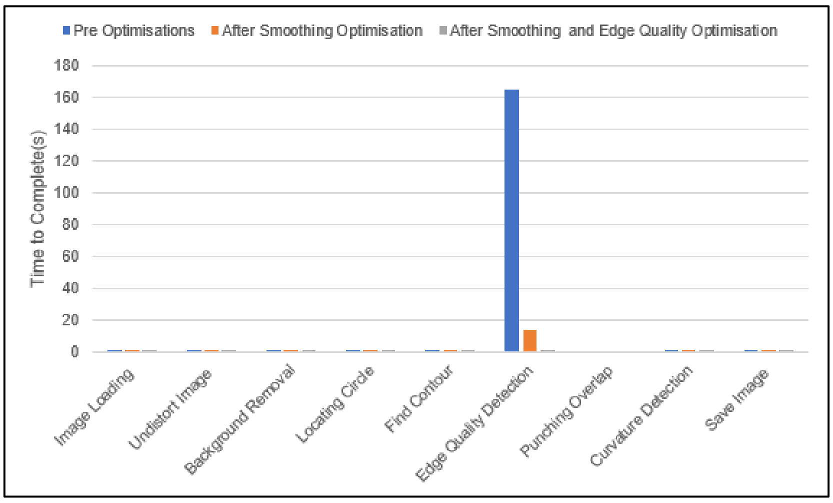


Fig.41 – Algorithm Execution Time by Stage

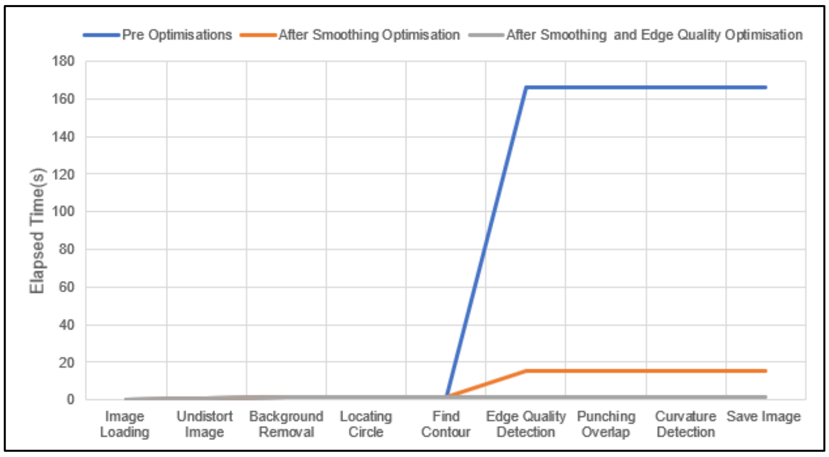


Fig.42 – Cumulative Algorithm Execution time

the execution time of the process was measured with the second methodology. The execution time was seen to be significantly reduced when compared to the first methodology. The is mostly in part to the removal of the extensive kernel operations and comparison algorithm, described in Section 4.2.1.1.

4.2.2.3 Discussion

The results yielded from this method are consistent with what was achieved using the method outlined in Section 4.2.1. Moreover, this method maintains a 1:1 relationship between the actual contour points and the smoothened contour points. As the blur kernel acts directly on the existing contour points, the n^{th} point in the actual contour points is directly associated with the n^{th} point in the smoothened contour points. By virtue of this, there is no requirement to search for the corresponding matching point as described in Section 4.2.1.1. Fig.41 and Fig.42 show the reduction in the execution time of the edge quality detection stage, running on the Raspberry Pi, as compared to the previous method. The method of the 1D blur kernel for contour smoothing reduced the execution time demand for this stage by 99.94% from 164.5s to 0.1s, which highlights the inefficiency of the large erosion/dilation and point matching processes.

4.2.3 - Edge Quality Analysis End-to-End Validation

4.2.3.1 Methodology

An experiment was designed to attempt to validate the effectiveness of the edge quality detection algorithm. This was challenging as the ‘quality’ of an electrode component’s edge is subjective and what looks bad to the eye might not be reflected when empirically processing the data.

50 electrodes were imaged and reviewed by a human. Each electrode was given a score between 0-10 to describe the perceived quality of the electrode edge, with 0 being the worst and 10 being impeccable. These images were then passed through the edge quality detection algorithm and the scores given by the system were captured and compared. Among the electrodes were examples of a wide array of edge qualities, coating types and varying amounts of curvature.

4.2.3.2 Results

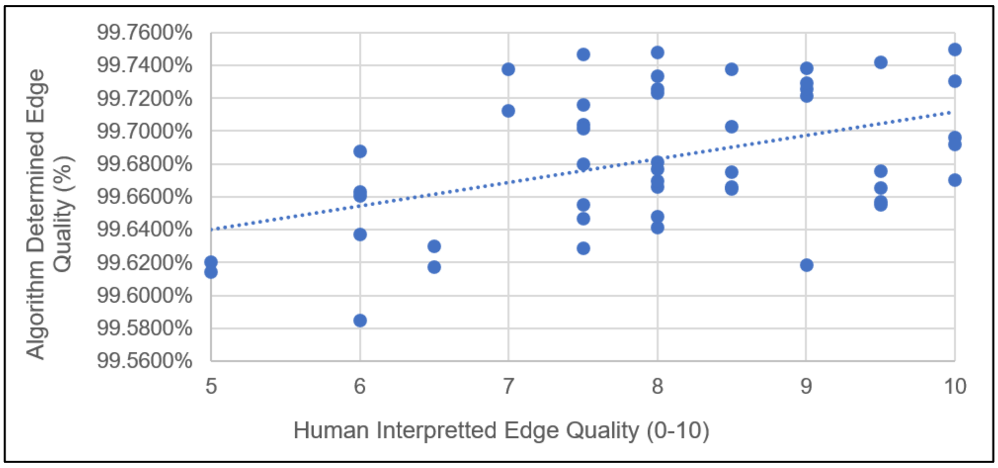


Fig 43 - Algorithm determined edge quality against human perceived edge quality of an electrode components

Fig.43 shows the association between the determined edge quality by the algorithm and the human perceived edge quality of the electrode.

4.2.3.3 Discussion

As can be seen from Fig.43, the association between the algorithm output and the human perceived edge quality is very weak. There is a vague trend that passes throughout but not something definitive enough to validate the method. A weakness of the methodology is that if pointwise calculations are going to be performed on the outer contour of the component, then it is required that the found contour be very accurate to give meaningful results. Any noise that passes the filtering in the background subtraction stage of the method will negatively affect the accuracy of the measurement.

Another observed weakness is that the contour smoothening mechanism has more transformative impact to areas of the component perimeter with greater curvature. As the smoothening method applies a 1D average kernel to the contour point, they are dragged closer together and ultimately shrink the contour slightly inwards. This effect is more pronounced on curved electrodes in areas where the perimeter curvature is more pronounced. This cause a larger discrepancy between the contour and the smoothened ideal contour, reducing the perceived edge quality. This can be seen when reviewing the images in the experiment of curved electrodes with perceived excellent quality edges that are given uncharacteristically low edge quality scores by the algorithm.

Another conclusion that can be drawn from the experiment is that the method itself could be inherently flawed. There will always be a bias when assessing these electrode components by eye and certain characteristics that look bad do not result in low scores when processed analytically through the algorithm. Another approach could be used in the future work of the project to reattempt the validation. This other approach could be to use a more powerful segmentation tool such as Meta's segment anything model (SAM) to segment the image and extract the contour of the component. Using this contour within the algorithm to produce an edge quality assessment and comparing to these results would give validation that the methodology is working as designed.

It is clear that this method has limitations, mainly around the sensitivity required to gain an accurate estimate of the edge quality of a component. Presence of noise in the image after the filtering is complete will negatively affect the accuracy of the method. Moreover, the HSV filter can be sensitive and the settings to perfectly achieve a component isolation in one image may vary slightly in the next image of a similar component. An effective autocalibration method to achieve the best HSV settings for each individual component would remove this uncertainty and improve the effectiveness of the method. In addition, the degree of curvature in the component will also negatively affect the accuracy of the measurement. A possibility to explore in future work of this project is the inclusion of edge quality to the machine learning model. If instance of bad edge quality can be captured in the labelling process of images then this knowledge can be captured by the model and used to predict other instances where the edge quality has been compromised.

4.3 – Electrode Curvature Analysis

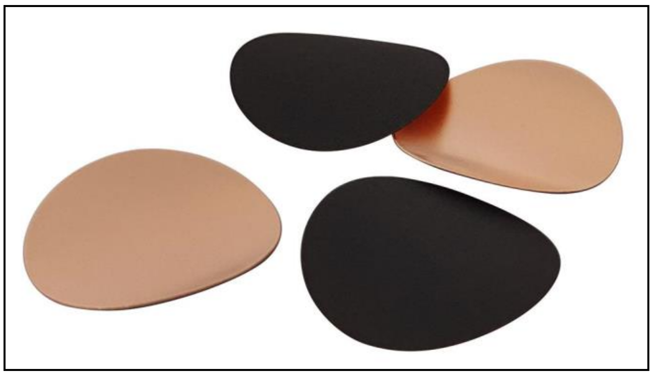


Fig.44 - Examples of curved electrodes [53]

Electrodes often don't conform to perfectly flat circles when punched from a sheet of material. They often distort vertically created a hyperbolic parabola, as shown in Fig.44. Electrode curvature is an important parameter to identify and track during the build procedure of a cell as it could have a negative impact on the alignment of components. Furthermore, the ability to correlate the performance of a cell to its curvature among other components is a powerful tool to optimise the cell build workflow.

4.3.1 - Methods

The expected circular nature of coin cell components can be exploited to determine the amount of electrode curvature present in a cell component. It can be assumed that, when laid flat, the component resembles a circle. Using this, the found outer contour of the cell component can be compared to an ideal circle encompassing the component. The discrepancy between the values represents the amount of vertical deflection in the component, the curvature.

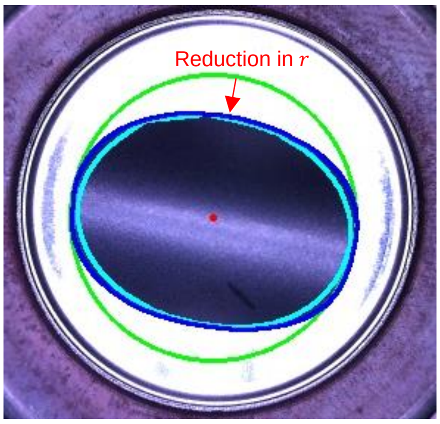


Fig.45 - Observed reduction in r in one axis as curvature increases

The method used to find the component edge, described in Section 4.2, was compared to the minimum bounding circle around the component and the discrepancy of the area between the two shapes was used to quantify the amount of curvature present.

An experiment was designed and carried out to validate the performance of the methodology. A set of 20 electrodes were produced of two differing sizes: 14mm and 16mm diameters. The electrodes were a mixture of coated and non-coated, meaning the material exposed to the camera when imaged could be the electrode coating and the copper material underneath. Curvatures of random magnitudes were introduced into the electrodes by rolling them against rounded surfaces of differing diameters. The electrodes were then measured using digital callipers and their heights recorded. The electrodes were subsequently passed through the Cellerate system where they were imaged and the resulting determined electrode curvatures were determined. Both positive and negative deflections of equal amount were used, shown in Fig.46.

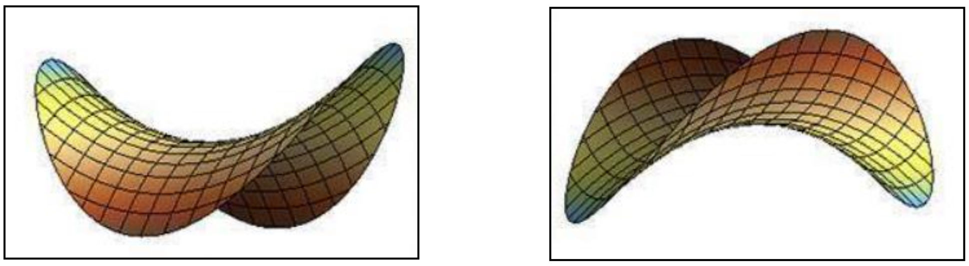


Fig.46 - a) Positive deflection of electrode b) Negative deflection of electrode [36]

4.3.2 – Results

What was expected from the testing is that a relationship would exist between the measured height of the electrode and the algorithm’s determined curvature. The algorithm’s methodology looks at the 2D plan view of the component and treats this as a flat plane, determining the area of the shape and comparing to an idealised circle gives a reflection of how much vertical deflection, and therefore curvature, is present. As the component gains more curvature, there is a reduction in the apparent radius in one axis, see Fig.45. The effect that this transformation will have on the area of the 2D plan view of the component is expected resemble an inverse square relationship, as the area of a circle is proportional to r^2. Although, the radius does not uniformly reduce at all points on the circle, resulting in an oval and an imperfect relationship. Therefore, the expected relationship between the measured height of the electrode and algorithm's determined curvature is something resembling a second order polynomial.

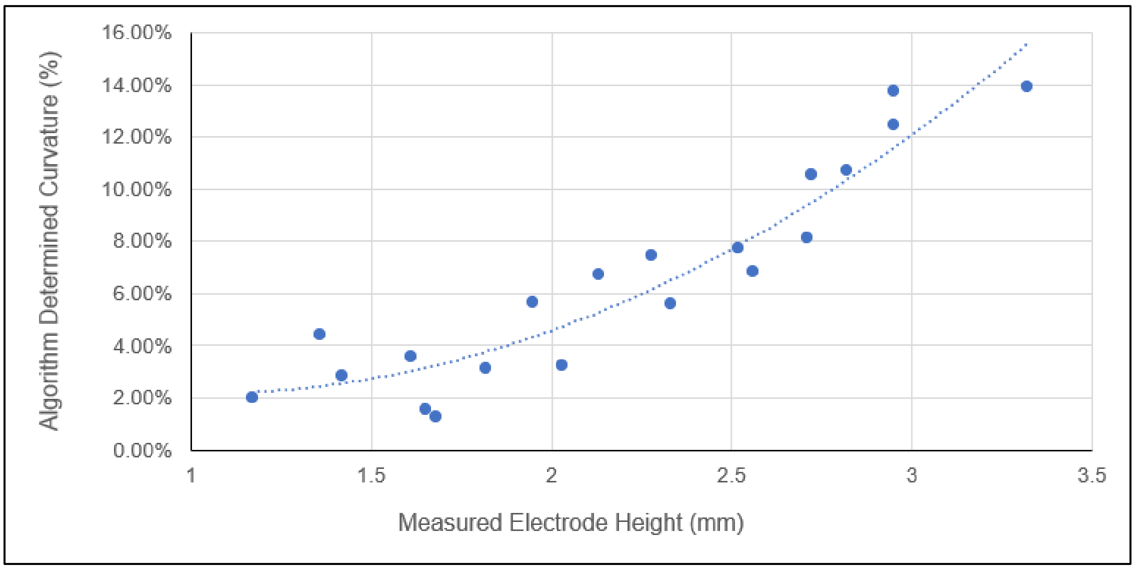


Fig.47 - Measured electrode height versus algorithm determined curvature

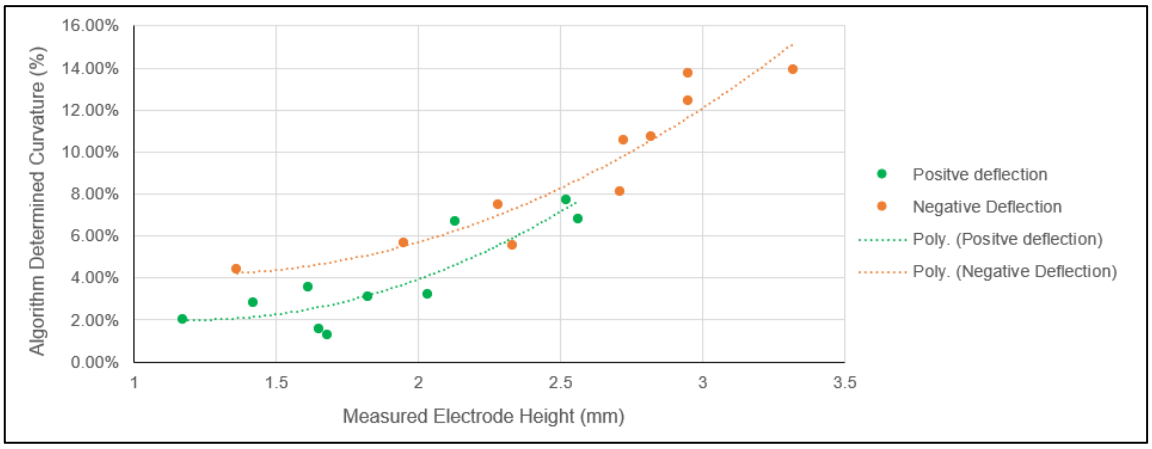


Fig.48 - Measured electrode height versus algorithm determined curvature with separate positive and negative deflections plotted

Fig.47 shows the results gathered from the experiment plotted in their entirety. Fig.48 shows the results split between the positive and negative deflections.

4.3.3 – Discussion

As can be seen from Fig.47 the expected relationship was observed. This is to be expected as the rate at which the area of the 2D shape of the plan view of the component decreases as the radius in the affected direction is reduced increases non-proportionally.

What could be noticed initially is the ambiguity of the algorithms output. The algorithm curvature percentage is simply the percentage of discrepancy of the 2D plan view of the component compared to the idealised bounding circle. What should be remembered is that this value can be quantised and scaled as appropriate to reflect something more reasonable to the user. An example of this could be scaling the value between 0-10 where 0 represent no curvature and 10 represents maximal curvature.

Moreover, the results positively validate the effectiveness of the methodology proposed. The relationship between the height and the detected curvature is all that is needed to prove out the method. Confidence can be gained that the algorithm can distinguish between electrodes of differing curvatures and this information can be quantised and presented back to the user for their cell performance data correlation.

An interesting feature of the results can be seen when splitting out the positive and negative deflections. It seems that the results gathered from the positive deflections are shifted downwards. Before the experiment was conducted it was expected that both the positive and negative deflections would yield the same results, due to the 2D plan view of the component being expectedly identical in both cases. What can be seen is that that is perhaps not the case. Additional investigation into this phenomenon could be done to identify the cause of this discrepancy.

The data points also, however, contain some deviation from the trendline. This is most likely due to two main considerations: human error when measuring the height of the electrodes and misshapen electrodes that are not perfect circles. The human error comes in both using the eye to determine when both sides of the calliper are just touching the top and bottom of the electrode and placing the electrode optimally with the apex of the deflection at the centre point of the calliper. Secondly, small variation in the shape of the electrodes, perhaps due to imperfect edges can cause the bounding circle to be slightly mis sized which results in an inaccurate calculation.

4.4 – Punching Overlap

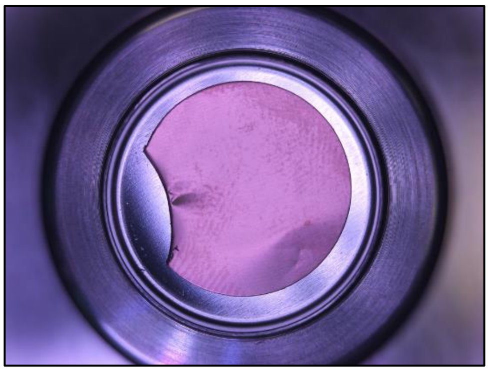


Fig.49 – Punching Overlap

Often when the electrodes are punched out of a sheet of material the punch can overlap between successive punches when trying to leave minimal excess material. These punching overlaps can be subtle and missed by operators, especially when making many electrodes. The presence of these leave non-symmetrical components with uneven impedance profiles. Therefore, it is important that these mistakes are captured, and traceability is maintained to connect subpar cells to test data.

4.4.1 - Methods

The method of detecting punching overlaps involves comparing the found outer contour of the cell component to the convex hull found on the same contour. A convex hull of an object is a minimum bounding polygon that can enclose it [45]. Therefore, it can be used to highlight missing material of a component as the convex hull will approximate a straight line across regions of the cell component that are concave. Fig.50 demonstrates this concept. Comparing the areas of the convex hull and the actual contour of the component for discrepancy reveals the presence of a punching overlap. The algorithm assesses the delta between the area of the cell contour and the area of the accompanying convex hull. A punching overlap detection is triggered according to the equation below:

Condition	**Outcome**
$A_c - A_{ch} > Threshold\ Factor\ \cdot A_c$	True
$A_c - A_{ch} \leq Threshold\ Factor\ \cdot A_c$	False

Where:

Term	**Description**
A_c	Area of the contour
A_{ch}	Area of the convex hull
$Threshold\ Factor$	Calibrated threshold factor

The threshold factor was calibrated based on the smallest punching overlap possible that could be introduced to a cell component. However, it is possible that this factor may need to be reduced upon further update to the requirement on the system.

An experiment was designed and carried out to validate the method of detecting punching overlap. 40 electrodes were made, purposely introducing punching overlaps to them. These punch overlaps were subdivided into 4 subgroups: small, medium, large and no punch overlap. Within these subgroups, curvature was introduced to 5 of the electrodes, to prove that the algorithm can detect punch overlap with the presence of curvature. These electrodes were then passed through the Cellerate system and the algorithm's determination of whether there is punch overlap present was recorded.

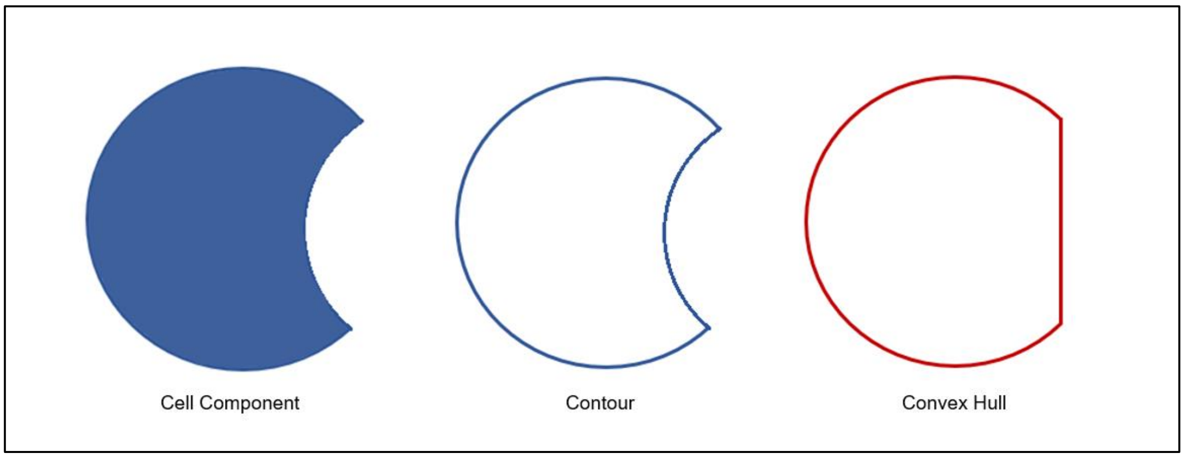


Fig.50 - Convex hull

4.4.2 – Results

Electrode Group	Curvature	Punching Overlap Detected (%)
Small Punch Overlap	Curvature	100%
	No Curvature	100%
Medium Punch Overlap	Curvature	100%
	No Curvature	100%
Large Punch Overlap	Curvature	100%
	No Curvature	100%
No Punch Overlap	Curvature	0%
	No Curvature	0%

Table 1 - Punching overlap validation results

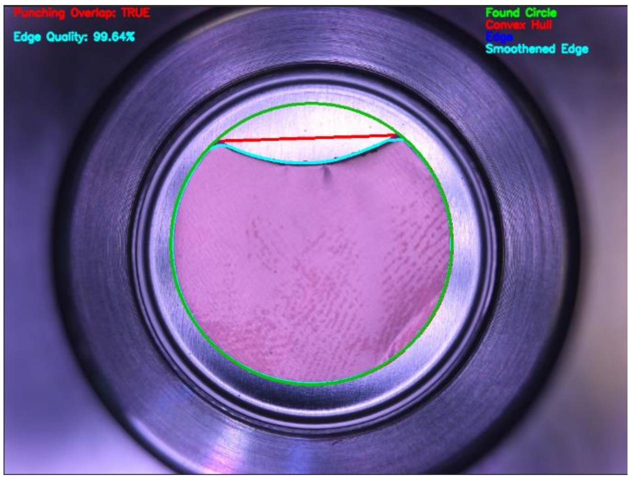


Fig.79 - Execution time of each of surface inspection

4.4.3 - Discussion

As can be seen from Table 1, the algorithm was able to detect the presence of punching overlap flawlessly on all electrodes that contained an punching overlap. Fig.79 shows an example of the output from the experiment. The algorithm was able to accurate locate the component contour and apply a convex hull process to it. The difference between the internal area of the convex hull and contour was calculated and a decision can be made whether there is material missing and there is a punching overlap present. The most impressive thing about these results is the ability of the algorithm to detect punching overlap on electrodes that contain curvature. This is possible as the cell contour is adaptable to component curvature and the convex hull is derived from the component contour. A downside of this is that the presence of punching overlap will affect the accuracy of the electrode curvature estimations, as more material is missing from the component. It should be made clear to the user that if punching overlap is detected that the accuracy of the electrode curvature will suffer. Moreover, this method is not able to describe the amount of material that is missing due to the punching overlap. This again, due to the fact that any electrode curvature will affect the validity to any comparisons made to the idealised flat circle. This method therefore requires a calibration to be performed to the smallest allowable percentage area to be the critical point where a punching overlap is flagged, described as the threshold factor.

5.0 – Machine Learning Surface Inspection

The remaining project objectives: to detect scratches, creases, flaked coating and cracks on the electrode surface were determined to be most suited to a machine learning approach. This section details the methodology to evaluate and determine the most appropriate neural network architecture for this application.

5.1 - Architecture and classification type

As determined by the review of the current literature on machine learning for defect detection applications, it is clear that the U-Net is the most popular architecture currently available. This is partly due to the classification type that is offers, semantic segmentation. Semantic segmentation is a pixel wise classification that gives each pixel within an image a specific class. The result from the classification process is a segmentation map which gives localised information about the classes present within an image. This has a particular advantage over object detection methods, in this application, as there will often be many defects contained within a small area and the many overlapping boxes that an object detection algorithm would yield could make the information uninterpretable and confusing to an end user. A segmentation map solves this issue by removing the box and applying a colour to the pixel to highlight the areas of interest. The difference in the approaches can be seen in Fig.52. In addition, the U-Net architecture was cited to be very responsive to small datasets, due to its initial application of medical image segmentation with few training images [26].

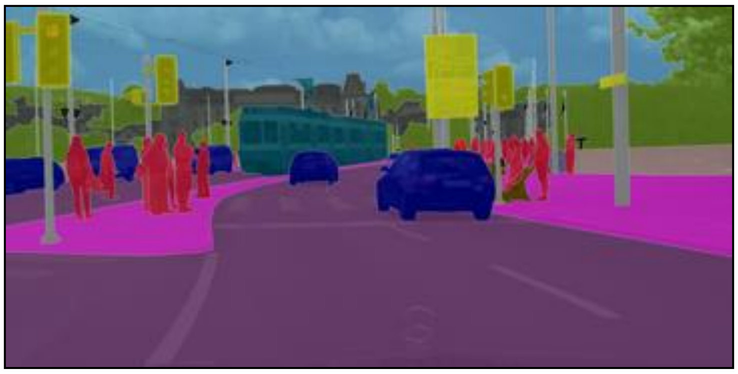


Fig.51 – Segmentation Map [62]

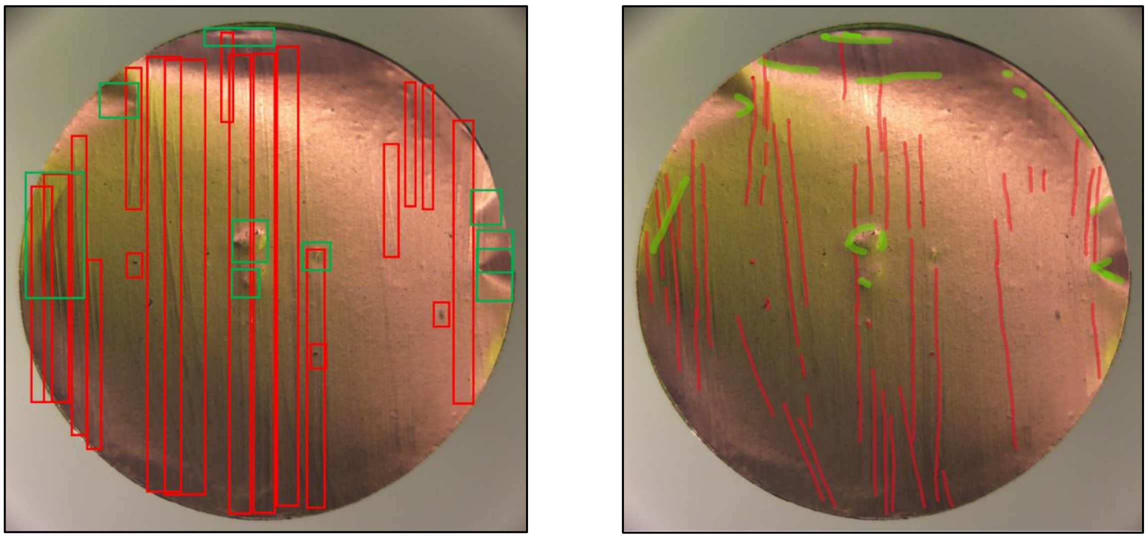


Fig. 52– (a) Defect detection by object detection (b) by semantic segmentation

5.2 - Prerequisites

5.2.1 - Data collection

To gather reliable data on the performance of the CNN architectures, a dataset was needed to consistently test each architecture against. The dataset had to be consistent with each test to isolate the variables to solely the architecture.

To be able to validate neural network architectures against each other a dataset was created. Many images were taken of electrode components in a variety of lighting conditions. The images were taken using cameras used on the Cellerate system to maintain as much consistency between the dataset and the final use case.

Firstly, a set of electrode components were sourced. The components had not been stored in an ideal manner, meaning many imperfections and defects were already present on them. A first round of images was collected from them. Subsequently, additional defects were artificially introduced through representative methods that would be mimic processes found in a lab scenario. These methods included mishandling the electrode components with tweezers and rattling within a storage box. After the additional defects were introduced the electrodes were subject to another round of imaging. Fig.53 show examples of these images.

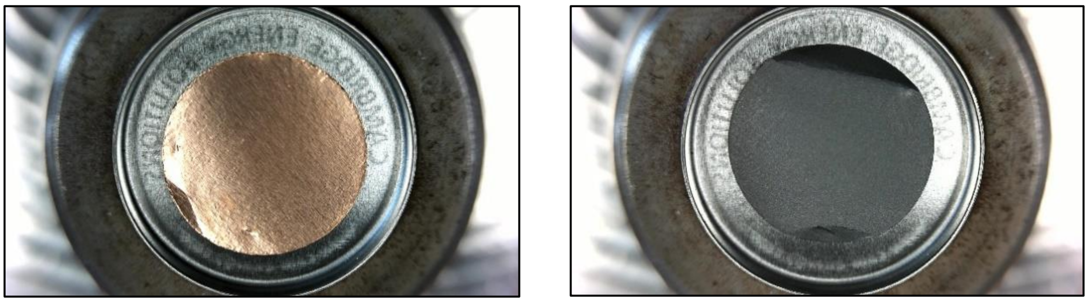


Fig. 53 – Examples of images gathered for the dataset

5.2.2 - Labelling

The process of training a neural network for semantic segmentation, as shown [26], requires labelled images. This is how the knowledge is encoded into the weights of the neural network.

To do this a human must use as labelling tool to draw on the regions which represent the various defects within the images and then create masks that can be fed into the model training procedure. The masks provide a reference to the network to which areas in a target image represent which region, including defects. The training process uses these masks to optimise it’s learning of how to classify these defects and to validate that the learning process is successful when applied to new data. Fig. 54 shows an example image and associated mask from the final dataset. The difference between the initial dataset and the final dataset is described further in this section.

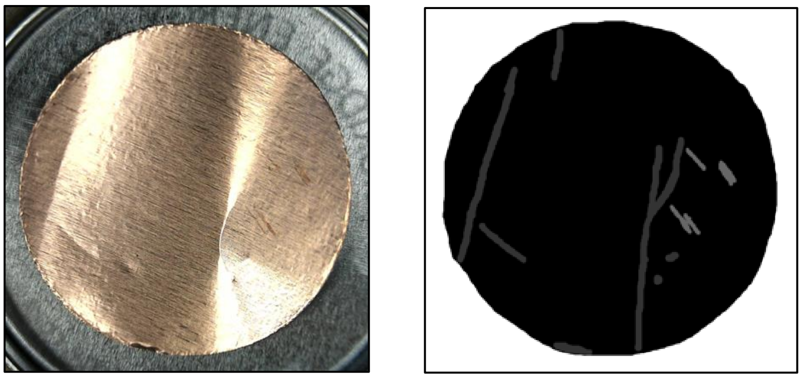


Fig. 54 – Example image and corresponding mask

The different shades of grey indicate different defect types. In this example the white represents the background of the image, the black represents the defect free areas of the cell component, the lighter grey represents scratches and the darker grey represents creases.

5.2.3 - Initial Dataset

As described previously, the original dataset contained 1308 image/mask pairs. The masks contained labels of the following types: scratches, creases, flaked coating, cracks. As every pixel within the image must belong to a class for semantic segmentation the remaining pixels that are not classified into the classes described belong to a background class. This class encompasses the non-component area and the parts of the cell component that do not contain defects.

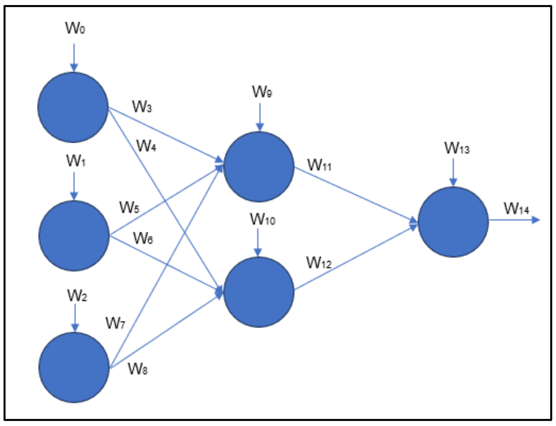


Fig.55 - Network weights

5.3 - Training

The training process of a convolutional neural network involves using the training data, where image/mask pairs are used to optimise the weights of the network so that the output activation of the network matches the ground truth. Each node and each connection between nodes contains a weight which is adjusted by the optimiser to align the output with the ground truth. In addition, within each node is a nonlinear activation function which determines the magnitude of the node output. This process of optimisation is achieved through use of the stochastic gradient descent algorithm. In addition, there are many other hyperparameters to the optimisation which can be tuned to achieve the best possible result.

5.3.1 - Stochastic Gradient Descent

The stochastic gradient descent method [46] is commonly used in machine learning applications to train models. It was adapted from the gradient descent algorithm by implementation of the stochastic approximation [47] in 1951. It improved on the gradient descent algorithm by optimising the process for large datasets. In traditional gradient descent, model parameters are updated based on the average gradient computed over the entire dataset. This can be computationally and time expensive for large datasets. Stochastic gradient descent improved on this by subdividing the dataset into mini batches and updating model parameters based on the average gradient of the random subset.

For each mini-batch the gradient of the loss function with respect to the model parameters is calculated. The model parameters are adjusted by taking a small step, with size determined by the learning rate, in the opposite direction to the loss function gradient to head towards a minima. This is done across all mini batches, this process is repeated a number of times, which is referred to as an epoch.

The desired result is that the loss function should have settled within a minima which corresponds to optimal model performance.

5.3.2 – Training Epochs

A tuneable hyperparameter in the training process is the number of epochs. As mentioned previously, an epoch is a round of training. Where the optimizer has passed through every mini-batch and made appropriate adjustments to the model parameters to minimise the loss function. Typically, more epochs gives the model more chance to better learn and generalize the training data. However, too many epochs of training can lead to overtraining. This is generally noticeable when the training accuracy continues to decrease but the validation accuracy becomes unstable. This is because the model is ‘memorising’ the training data and has learnt its intricacies to further reduce the training loss function. This means that is has not generalised the learning from the training data and cannot be applied to new data or transfer the performance to the validation dataset.

To mitigate against this, an early stopping callback was used. This early stopping callback monitors the validation accuracy, which shows how much the model has generalised to new data, for an overall decreasing gradient. If the validation accuracy has failed to decrease at all over a defined number of epochs the training is stopped and the weights which achieved the best validation accuracy are restored. A maximum of 100 epochs were used in each case. However, this is rarely reached as the model will converge much sooner than that, approximately around the 40th epoch in most cases.

5.3.3 - Loss Function

The loss function used during testing was categorical focal loss. This loss function was selected through some initial testing and because it is the most adopted loss function for multiclass classification. It is so popular because it uses weighting to lessen the influence of easy examples in the optimisation process and focusses on the hard examples to better understand the nuances of how to classify underrepresented classes. In the context of this problem, the background class is very dominant as the individual defect instances are very small. Another loss function could lead to the model being very able to distinguish the background as this would instantly lead to a very high accuracy due to its prevalence. Focal loss weights down overrepresented classes and weights up underrepresented classes such as the small scratch defects in this instance.

5.4 - Architecture Investigation

5.4.1 - Introduction

In this section, the methods taken to evaluate the performance of the various U-net architectures tested during the project will be outlined. As reviewed in Section 2, there are many variations and adaptations to the U-Net architecture. This section describes the various architectures tested under the pretences already discussed.

5.4.2 - Architectures

The initial architecture that was tested was the first iteration of U-Net, described in the initial paper [26] and shown in Fig.56. Unlike typical CNN architecture, the network does not contain any fully connected layers. The network has a contracting path (left) and expanding path (right). The contracting path utilises repeated application of two 3x3 convolutions (unpadded), each followed by a ReLU and a 2x2 max pooling operation with stride 2 for down sampling [26]. At each down sampling step, the number of feature channels are doubled. Every step in the expansive path consists of an up sampling of the feature map followed by a 2x2 convolution (up convolution) that halves the number feature channels [26]. A concatenation with the correspondingly cropped feature map from the contracting path is then performed followed by two 3x3 convolutions and a ReLU activation function [26]. At the final layer a 1x1 convolution is used to map each 64- component feature vector to the desired number of classes [26].

Due to memory constraints on the GPU used to train the network , the number of convolutional layers was reduced to that of Fig.56b.

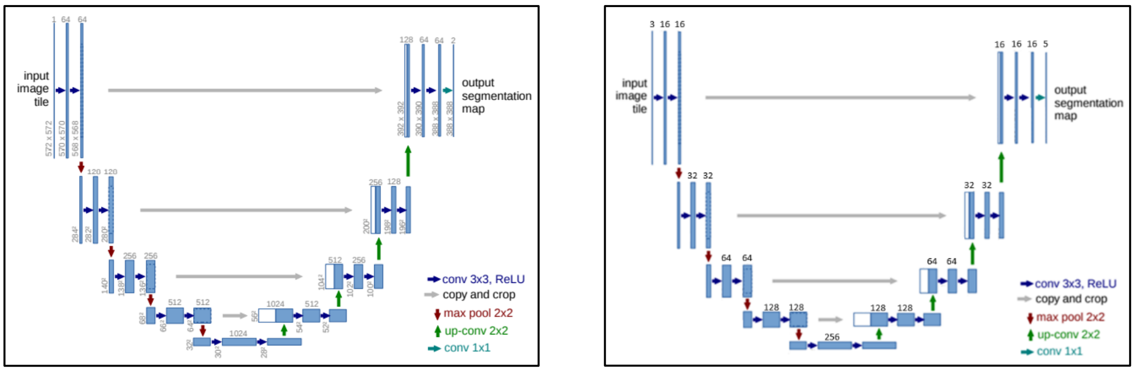


Fig. 56 – (a) Original U-net Architecture [26] and (b) Simplified version

5.4.2.1 - Deeper U-Net

The deeper u-net architecture that was used was identical to the u-net described previously, but the number of convolutional layers at each stage was multiplied by 2, see Fig.57. The effect is that the number of parameters increases significantly, in theory increasing the capacity to encode the information imparted on it by the training process.

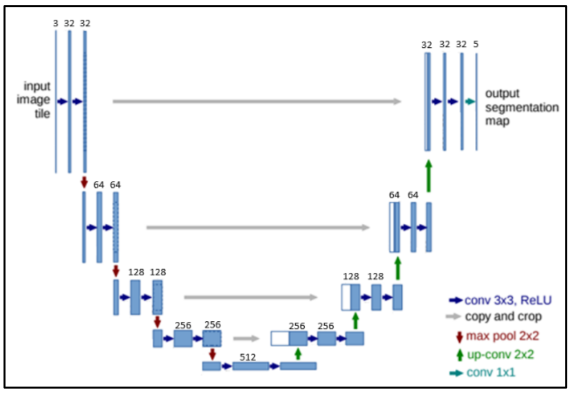


Fig.57 - Deeper U-Net Architecture

5.4.2.2 - ELU-Net

This architecture builds on U-Net by implementing deep skip connections, in which layers in the contracting path are concatenated directly to the corresponding layers in the expanding path [27]. These deep skips enable full capture of “fine-grained details and coarse-grained semantics in the encoder” [27] as the features extracting in the encoder stage are the key to image segmentation. Therefore, deep skips help maintain these features. The network architecture, when built using a ResNet backbone, contains only ~4% of the parameters of U-Net 3+, making it much more lightweight and easier to train [27]. The method also scored highest Dice coefficient, 97%, of any of the state-of-the-art semantic segmentation networks that were tested when applied to the ISBI LiTS

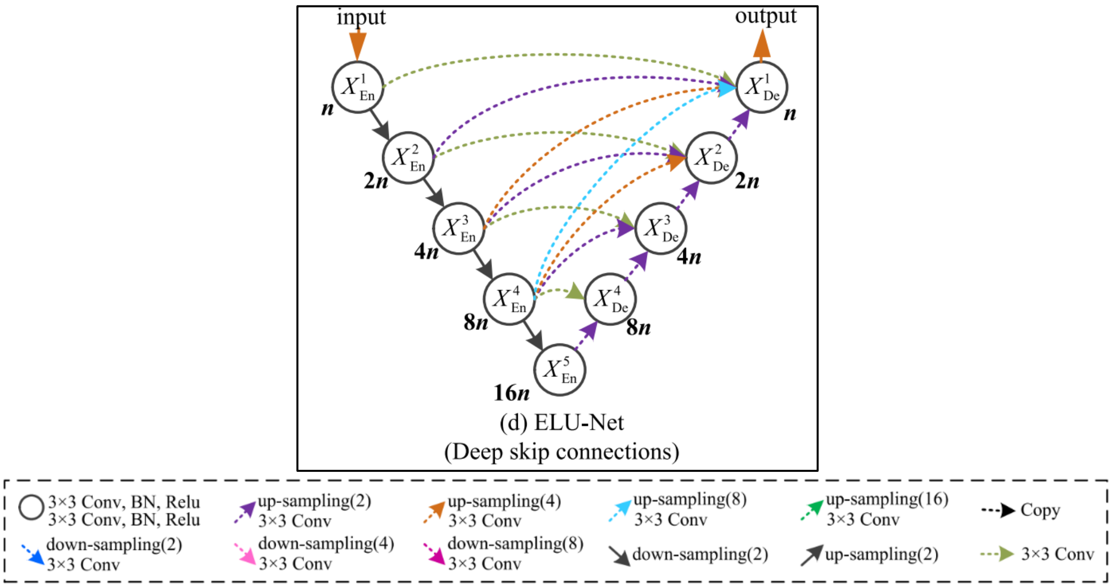


Fig.58 – ELU-Net architecture [27]

dataset [27]. This variation on U-Net is specifically optimized for resource constrained applications, such as the one in this work, and should be considered.

5.4.2.3 - Deeper ELU-Net

The Deeper ELU-Net architecture applies the rationale discussed in the deeper U-Net section. It takes the basic ELU-Net architecture but increasing the number of convolutional layers to each stage by a factor of 2 in both the expanding and contracting path.

5.4.2.4 - Residual U-Net

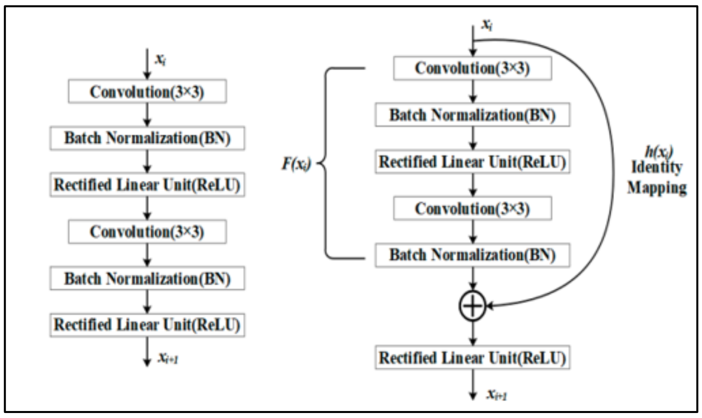


Fig.59 – Residual Convolutional Block [34]

Residual U-Net uses the U-Net architecture described above but adapts the convolutional blocks into residual convolutions, see Fig.59.

It is observed that accuracy can saturate when the network has learned the intricacies of the data. A first reaction to this problem may be to add more layers to the network, but this can sometimes lead to even worse performance, known as degradation [33]. The residual block looks to overcome this limitation and contains two parts: identity mapping and the residual. Identity mapping integrates the input with the output of the residual to maintain subsequent feature information [34] and to solve the problem of gradient disappearance and non-convergence. Moreover, any poor ability to extract features in the residual part of the block is compensated [34]. Using residual blocks has also been shown to improve the training of the network and using U-Net structure allows for the use of skip connections to facilitate information propagation without degradation [32].

5.4.2.5 - Residual ELU-Net

The residual ELU-Net carries the principle outlined in the Residual U-Net architecture but applies it to the ELU-Net architecture described above.

5.4.2.6 - Lighter U-Net

The lighter u-net architecture takes the standard u-net architecture and replaces the convolutional blocks with depth wise separable convolutional blocks, see Fig.60.

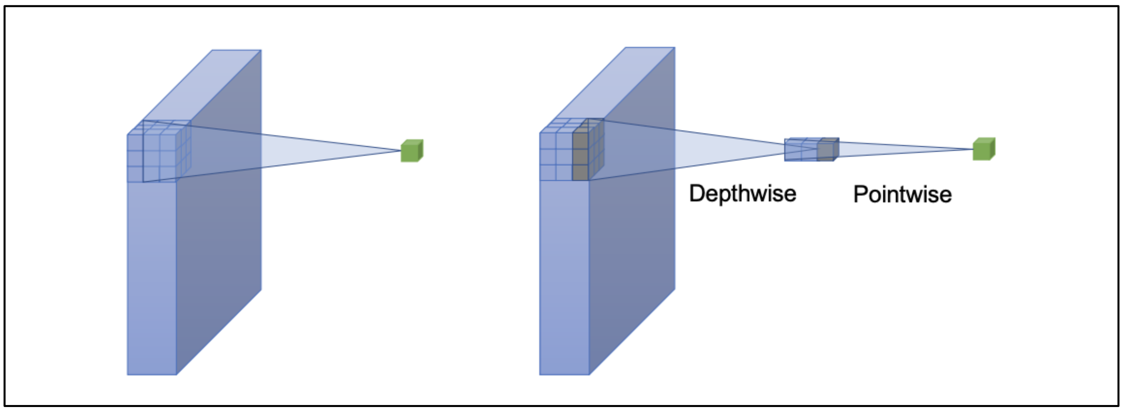


Fig.60 – Depth wise separable convolution [54]

The breaking of the standard convolution into a separate depth wise and pointwise convolutions significantly reduces the parameter count as the 3D kernel used in the standard convolution is broken into 2 separate 2D kernels. By reducing the parameter count, the size of the model and inference time can be reduced. This is applicable to use cases with limited compute, such as the Raspberry Pi used in this project. This architecture gives an option to reduce the complexity of the model easily without sacrificing too much performance if the model's inference time is not feasible running on the Raspberry Pi.

5.4.2.7 - Inception U-Net

The inception U-Net architecture used in the testing uses the basic architecture of the standard U-Net, however the convolutional blocks are replaced by inception blocks, shown in Fig.61. An inception block is a culmination of various convolutions with different kernel sizes. The idea of using multiple different filter sizes is to provide more flexibility in the feature sizes it can optimise for.

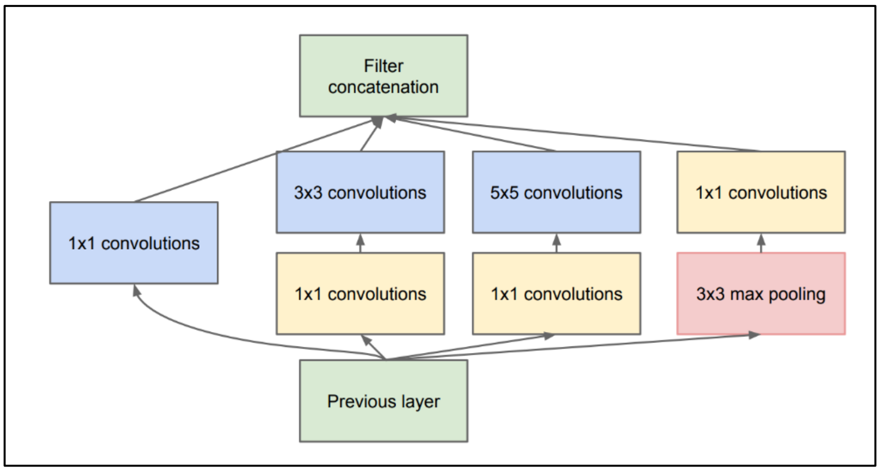


Fig.61 – Inception Convolutional Block [55]

5.4.3 - Methods

This experiment looks to quantitively evaluate the performance of the various U-Net derived network architectures described and find the U-Net architecture variation that responds best for this dataset.

The networks were constructed through implementation of the Data Preprocessor and Model Generator module classes, described in Section 3. These modules were designed and implemented in Python to be the classes that oversee the model architecture creation and training.

The Data Preprocessor module took the dataset images and the corresponding labelled masks and formatted them into many tiles of 256x256x3 pixel pairs. This was done to provide the expected suitable input tensor size for the network. Using the Tensorflow [37] package with Keras [38] Python API library the Model Generator module was responsible for describing the model architectures in Python that would allow the training procedure to take place. The Tensorflow library was written to take advantage of GPU capabilities to perform parallel operations which optimise the training process with multi-dimensional tensors. Through this library, the model files created by the Model Generator module are able to be applied to the parallel architecture of a GPU for optimised parallel training. A code reference from [40] was used to write the keras interface methods responsible for constructing for U-Net model object.

The models were trained on the dataset using a 70:30 split between training and testing as this is commonly used split among the machine learning community. Each model was trained and optimised using the training subset of the training data. The testing subset is purely used for validation and represents how the model performs against unseen data as the data in this subset is never seen by the model during the optimisation process.

The metric for performance used was accuracy. The accuracy metric is described as the number of pixels in the model prediction that align with ground truth masks. The higher the accuracy the more consistency there is between the prediction and the known ground truth, and thus higher performance of the model the results of the final validation accuracy of the model.

Each model architecture was individually implemented using the Keras library and trained against the training dataset, the final validation accuracies of the models were collected and shown below.

5.4.4 – Results

Architecture	**Validation Accuracy**	**Parameter Count**
U-Net	96.63%	1,947,078
Deeper U-Net	96.86%	7,772,038
ELU-Net	96.47%	2,720,966

Deeper ELU-Net	96.59%	10,864,006
Residual U-Net	95.57%	2,037,846
Residual ELU-Net	96.57%	2,818,646
Lighter U-Net	93.23%	275,312
Inception U-Net	96.21%	1,160,868

Table 2 – Results using the original dataset

5.4.5 – Discussion

The results, shown in Table 2 show that the best performing architecture was deeper U-Net followed closely by Deeper ELU-Net. However, the gained improvements over the standard U-Net are very marginal when considering the vastly increased parameter count. An increased parameter count means that not only is the model larger and occupies more space in memory but also that the inference times when passing new data through it are also increased. This may be detrimental to this application where quicker inference times are optimal to maintain the smooth operation of the Cellerate system. Moreover, all the model architectures except the lighter U-Net performed quite similarly. It should be noted that the Lighter U-Net still performed admirably when considering the significant reduction in the parameter count. The architecture that presents the best balance between performance and parameter count be said to be the standard U-Net.

5.4.6 - Amending the dataset

During the testing process it was clear that some defects, that were seemingly more obvious than some detected defects, were getting missed. Fig.62 shows an example of this. It was hypothesised that encoding the background and the defect free area of the cell component with the same value may be detrimental to the performance of the model. The ambiguity between the defect free areas and background class may cause the model to misclassify actual defects during the SoftMax activation function at the end of the network. If the areas of the cell component without defects and

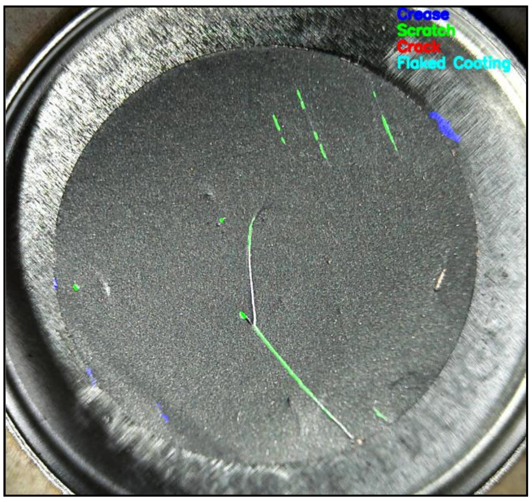


Fig.62 – Model classification results of standard U-Net architecture with missing defect detection

the background can be split and have their own separate encoding, then maybe the network can learn what it means to be a defect free component, and thus boost the effectiveness of classifying the actual defects.

To implement this theory the dataset had to be amended with another class. This meant drawing in the background on every image using the labelling tool to create an amended dataset with 6 classes: background, defect free cell, scratches, creases, flaked coating and cracks. The most successful network architectures tested using the old 5 class dataset was retested with the added class to see if the performance could be boosted

5.4.7 - Results

Architecture	**Validation Accuracy**	**Parameter Count**
U-Net	95.66%	1,947,078
Deeper U-Net	95.48%	7,772,038
Lighter U-Net	92.90%	275,312

Table 3 – Results using the amended dataset

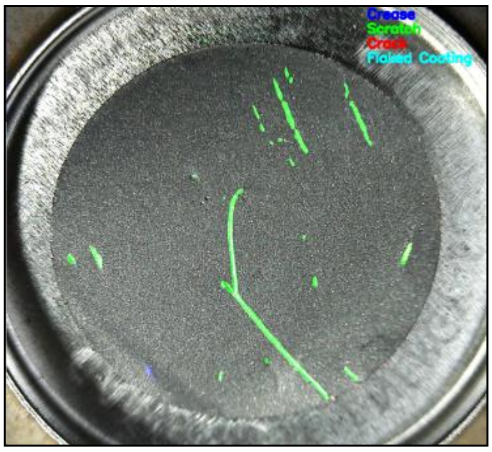


Fig.63 - model classification results using standard U-Net with amended dataset

5.4.8 – Discussion

It is clear that amending the dataset with the additional background class improved the performance of the model. When comparing Fig.63 to Fig.62 it can clearly be seen that more of the defects have been detected and the significant defects missed by the U-Net trained on the initial dataset have been highlighted. It should be noted that the validation accuracy of the model has decreased. This is likely due to more ambiguity between the background class and the defect free areas of the cell component. As this is such a large area encompassing the image any misclassification between these areas will cause the validation accuracy to decrease.

Fig.64 shows an example of this effect. Fig.64 shows an example image and the raw segmentation map produced by the model. Each colour represents a separate class, the most important in this distinction is the red background class and the dark blue defect free class. The addition of the background class to separate out the background and the defect free class causes some ambiguity within the model when trying to distinguish between these two. In the previous dataset they would be one blanket class, dominating the image. It is easy to see how a metric like accuracy that does not account for class imbalances would report a very high result if the model is able to determine the background class correctly. With the amended dataset there will most likely be more misclassification between these two classes, which can be seen in Fig.64 on the left hand side of the component. This will naturally cause the overall validation accuracy to be reported as lower when the actual performance of the model (how well the defects are detected) could improve.

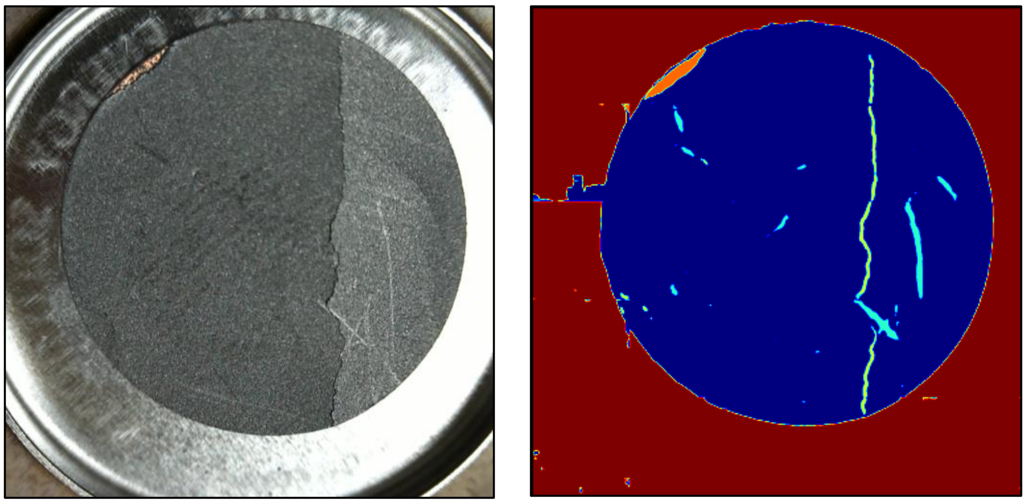


Fig 64 - Example image with accompanying raw segmentation map

5.5 - Data augmentation

A popular technique used throughout the literature surrounding convolutional neural networks is data augmentation. Data augmentation is the process of artificially enriching your existing dataset to create more training data. This can be done by applying transformations to your images such as rotations, axis flips, zooms and translations. By applying some data augmentation techniques the size of the dataset can be expanded dramatically. However, there is always a point where extensive augmentation becomes a detriment to the final model performance. The types of augmentations should also always be considered.

Data augmentation was used in this testing to evaluate whether it would boost the best performing models performance further still.

5.5.1 - Batch training

The data augmentation algorithm was tuned to produce three output images per one input image from the 10,700 image dataset, yielding 32,100 images to use during training. Due to memory constraints on the host PC used to train the model, this amount of data could not be held concurrently in RAM. The approach used to overcome this shortcoming was to use batch training.

Simply, the dataset was split into 3 chunks of around 10000 images each and separate training phases were completed on each chunk. The weights from the previous training cycle were able to be reloaded and applied to the current batch for batches 2 and 3.

Testing was done using the standard U-Net architecture to compare whether using the data augmentation method described had any effect on the accuracy of the model. The improvement from applying the augmentation to the training data can be seen below:

Architecture	**Validation Accuracy No Data Augmentation**	**Validation Accuracy With Data Augmentation**
U-Net	95.66%	96.26%

Table 4 – The effect of data augmentation using batch training

At first, it appears that the data augmentation improved the model performance by around 0.5% which is still a notable gain when reaching into the 95+% accuracy range. However, this reading is represents the validation accuracy of the model when applied to the final batch in the training. For it to have truly improved the model's performance. This improvement should be seen when applied to the other batches.

The problem with training in batches is that although knowledge from the previous batches is carried forward in the weights of the model, the final optimisation is only ever performed on the final batch. This effect leads to fast convergence on the final batch of training but if the data is not equally represented in all batches there is a chance that knowledge gained in training using the initial batches may be lost.

Architecture	**Batch A Validation Accuracy**	**Batch B Validation Accuracy**	**Batch C Validation Accuracy**
U-Net batch trained with data augmentation	92.99%	82.79%	96.26%

Table 5 – The validation accuracy of the training batches using the final optimised model using batch training

Table 5 shows this effect. The model trained with data augmentation, referenced in Table 4, is validated against all the batches that were used to train it. As is shown, the model does not respond as well to the earlier batches as It does to the final batch, where the ultimate optimisation of the weights is performed. This indicates that knowledge gained during the training of batches A and B was lost as it was not important to the optimisation of the final batch. It is also important to note that the inner workings of the weights of the neural network are too complicated to decipher and understand by humans, therefore it is impossible to tell what information was not lost. It may appear that each batch is equally representative of the dataset as a whole and contains an equal spread of the classes but this is clearly not the case to the optimiser, Therefore, training in this method comes with inherent risk and may lead to poor performance on new data.

5.5.2 - Data generator

To overcome the flaw in the batch training process, described above, a new method to process such a large amount of training data was implemented. A data generator is a class in Keras that allows data to be dynamically grabbed from memory during the training process, one mini-batch at a time. This method removes the memory constraint of batch training as the training data can remain on disk while it not being used. It also means that the model can optimise for the whole dataset instead of the final chunk as with batch training.

The testing was repeated with the new method of feeding the training data and the results are shown below:

Architecture	**Validation Accuracy No Data Augmentation**	**Validation Accuracy With Data Augmentation**
U-Net	95.66%	94.29%

Table 6 – The effect of data augmentation using data generator

In this instance the training with data augmentation actually gave a worse validation accuracy than with no data augmentation. This could be a sign that the types of augmentations applied to the data set are detrimental to preserving an accurate representation of the dataset. It could also be due to the variability and non-deterministic nature of model training. If the same model architecture is trained using the same data successively, the final validation accuracy will never be the same. This is in part due to the non-linear nature if the activation functions used. It should be noted that the data generator method still outperformed the batch training method overall. When considering the average validation accuracy over the three batches, the batch training did not perform well and shows that this method of transfer learning is not optimal.

Architecture	**Batch A Validation Accuracy**	**Batch B Validation Accuracy**	**Batch C Validation Accuracy**
U-Net data generator trained with data augmentation	94.79%	92.29%	96.31%

Table 7 – The validation accuracy of the training batches using the final optimised model using a data generator

Table 7 repeats the testing shown in Table 5 with a model trained using a data generator. As all the data can be used in the optimisation process concurrently, the final optimisation of the model is performed on all available data, meaning that no knowledge is lost due to the batching process. As expected, the model performed more consistently over the 3 batches as the knowledge of the whole dataset was retained as best as possible during the optimisation.

It is generally accepted that data augmentation can improve the performance of models and has been cited in many papers to do so. It is however, also accepted that each machine learning application is very much unique and that it is naïve to assume that techniques that improve performance in one instance can be ported and the results carry over to another instance. This is due to the black box model approach taken by deep learning model programmers that a trial and error methodology should be taken to find the correct parameters for the specific application.

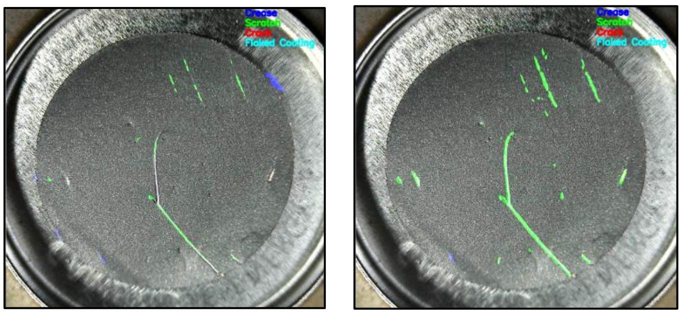
In this case, the use of data augmentation did not improve the performance of the model and was not used in further testing.

5.6 - Extended Results

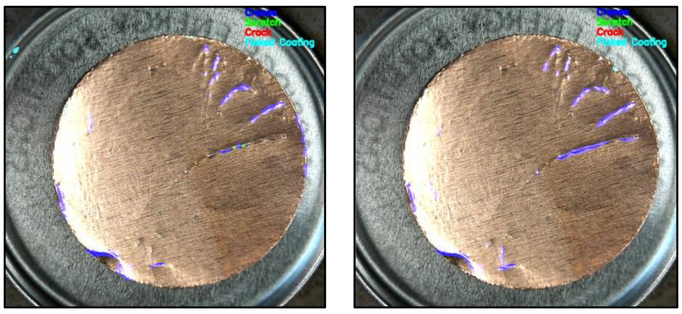
As discussed, the standard U-Net architecture with reduced convolutional layers was the stand out performer in the initial testing and remained so when tested once again with the amended dataset. However, it is curious that the validation accuracy of the model decreased from 96.63% to 95.66%. At first glance it is obvious that a direct comparison between the two results is irrelevant as the two datasets are largely different due to the addition of the extra background class. It did however, raise the question whether the adding the additional class improved the performance of the model to pick out defects.

A set of example images from the dataset were passed through both models and the output was reviewed. Creases, scratches, cracks and flaked coating are shown with blue, green, red and cyan respectively.

(a)



(b)



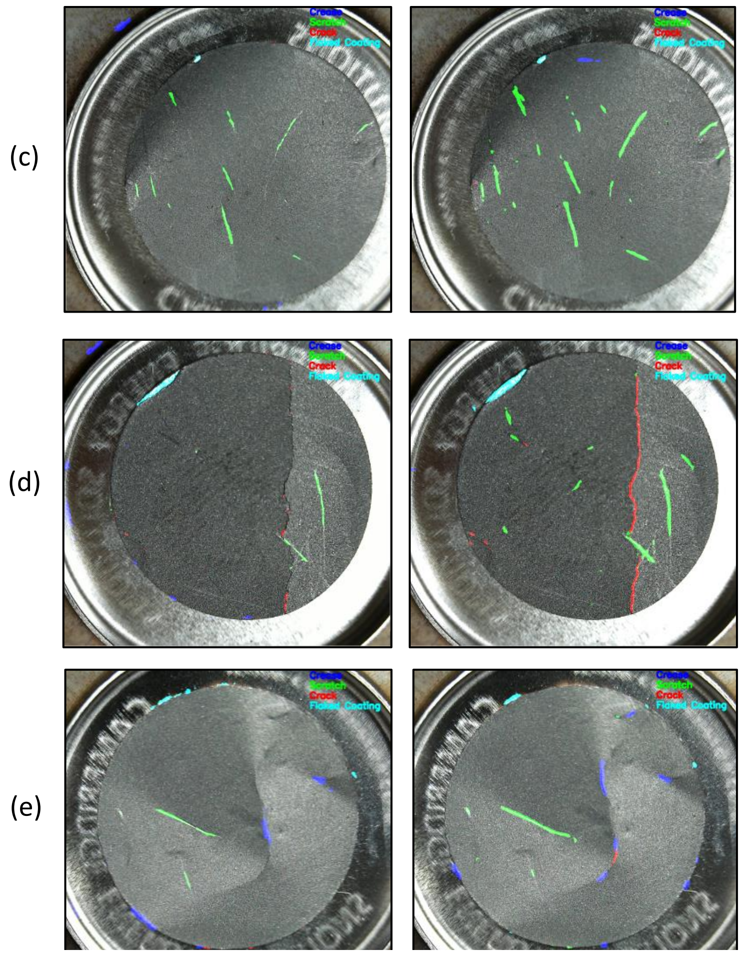


Fig.65 – Examples of images passed through U-Net architecture trained with original dataset (left) and amended dataset (right)

Fig.65 shows that training the model using the amended dataset achieved better performance at picking out defects. If this is the case, then why was the validation accuracy lower?

As discussed previously, adding an additional class to separate out the background and defect free component pixels will add additional uncertainty to the distinction between them.

Of course this raises questions whether accuracy is the best metric to track performance of these models. The models should be further verified with a metric that weights the classes according to their prevalence in the dataset. This inspired another round of testing and evaluation of the model architectures with metrics that weight classes based on their prevalence in the dataset.

The two additional metrics that were used in this round of testing are: Intersection over Union and F1 Score.

5.6.1 Intersection Over Union

Intersection over union is a commonly used metric in computer vision, particularly for object detection, semantic segmentation and instance segmentation tasks. It describes the ratio of the intersection of two regions to the area of the union of the two regions [48]. The formula to describe IOU is as follows:

$$IOU = \frac{Area\ of\ Intersection}{Area\ of\ Union}$$

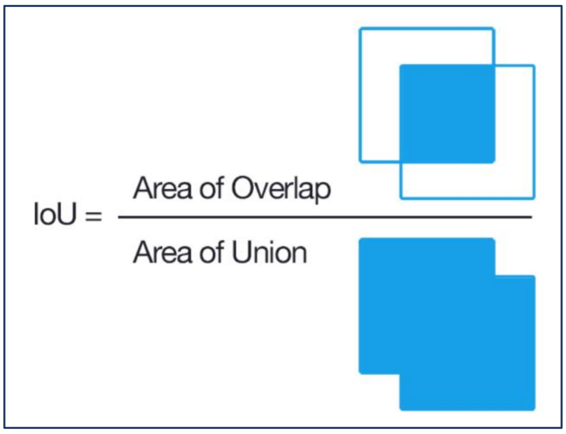


Fig.66 – Intersection over Union [56]

A visual aid of the premise of IOU is shown in Fig.66. IOU helps to determine how well machine predictions of bounding boxes or masks align with the ground truth annotation. A high IOU describes a good match between the prediction and the known truth.

In addition, the nature of the metric provides dynamic weighting of classes based on their prevalence in the dataset [48]. As it is scaled by the area of the union, each class can be weighted accordingly. Therefore, the metric is not dominated by the model’s ability to decern the most prevalent class in the dataset.

5.6.2 F1 Score

The F1 score is a commonly used metric to quantify machine learning model performance. It provides a single value that reflects a model’s performance considering both false positives and false negatives [49].

It is numerically described below as:

$$F1 = 2(Precision \cdot Recall)/(Precision + Recall)$$

Where:

$$Precision = TP/(TP + FP)$$

$$Recall = TP(TP + FN)$$

$$TP = True\ Positive$$

$$FP = False\ Positive$$

$$FN = False\ Negative$$

F1 score is particularly useful when dealing with imbalanced datasets, where one class is more prevalent that the others, such as this case. It balances the trade-off between precision and recall and presents a mean of the two [49].

Architecture	**Validation Intersection Over Union**	**Validation F1 Score**	**Validation Accuracy**
U-Net	31.77%	41.67%	95.66%

Deeper U-Net	29.62%	41.28%	95.48%
ELU-Net	28.26%	38.23%	95.37%
Deeper ELU-Net	30.33%	39.26%	95.22%
Residual U-Net	32.82%	40.56%	94.95%
Residual ELU-Net	30.54%	40.24%	95.01%
Lighter U-Net	22.89%	31.19%	92.90%
Inception U-Net	32.42%	38.17%	94.62%

Table 8 – Architecture performance validation using IOU, F1 score and Accuracy

Architecture	**IOU Ranking**	**F1 Score Ranking**	**Combined without accuracy**	**Combined Ranking**
U-Net	3	1	4	1
Deeper U-Net	6	2	8	2
ELU-Net	7	6	15	5
Deeper ELU-Net	5	5	10	4
Residual U-Net	1	3	4	1
Residual ELU-Net	4	4	8	2
Lighter U-Net	8	8	16	6
Inception U-Net	2	7	9	3

Table 9 – Combined rankings of the scores from Table 7

The model architectures from the initial testing were applied once more with the additional performance metrics of IoU and F1 score to monitor during training, shown in Table 8. A combined ranking was then taken, shown in Table 9, to indicate the model architecture that performed the best across the three metrics.

The standard U-Net was once again the best performing architecture for this application, ranking joint first with Residual U-Net architecture. However, the higher validation accuracy score is enough to secure it as the best performing architecture.

5.7 – Discussion

The additional round of testing was performed with the additional metrics to settle the assumption that accuracy was not the best metric to track performance where a large class imbalance was present. This question was raised when the validation accuracy decreased when the additional background class was introduced yet the model appeared to perform much better when applied to the eye test. Fig.64 confirmed the hypothesis that the addition of the background class to separate out the background and the defect free cell component from the original data gave the opportunity for misclassification and ambiguity between these two separate classes. Whilst this did not affect the model's ability to detect defects, on the contrary it actually greatly improved, it did cause the validation accuracy to report lower than when trained using the original dataset.

As shown in Table 9, the standard U-Net architecture performed the best among the architectures tested. The final ranking of the architectures was chosen on the combined ranking of the IoU and the F1 score as these two metrics are receptive to imbalanced datasets. It could be said that accuracy was not the correct metric to track as for the application as there is more commonality between the rankings of IoU and F1 score for a given architecture. And as these metrics account for dataset class imbalance they hold more weight to indicate which model architecture performs the best.

What should be noted is the how relatively low the final IoU and F1 scores for the architectures came out to be. It is interesting to see how low these values are when compared to the validation accuracy. This further shows that the accuracy metric is unable to account for class imbalance and the model's ability to identify the dominating class artificially boosts the accuracy and gives a misleading metric.

Moreover, it is interesting to note how low the IoU and F1 scores are when viewing the model output. Reviewing the images in Fig.65 it is clear to see that the most, if not all, of the defects in the images are detected in some capacity. The hidden classes of the background and defect free cell areas are not drawn on the image. It is possible that there is some confusion in the model's ability to discern these two classes, which could drastically bring down the IoU and F1 score. Although this distinction is not important to this application.

It must also be considered that it might not be necessary to achieve a very high IOU of >80% for this application. While it is optimal to have very accurate mask/ground truth union of the output, the end user may only care about the indication that a defect is present. A fully accurate representation of the defect drawn on the image is not necessary, as long as some part of the defect is detected and highlighted the user is made aware of its presence and can act accordingly. The best in class performance of the U-Net architecture in the F1 score category confirms that it is the best at discerning false negatives and false positives and gives the most confidence that when a user sees a defect highlighted on the output image that it is in fact there.

6.0 - System Integration

This section looks to review the project as a black box as seen from the user's perspective to evaluate the culmination of the work. The way the gathered data is fed back to the user, in the form of a cell build report will also be evaluated. Moreover, benchmarking will be done to evaluate the performance of the software and how this will affect the current runtime of the system.

6.1 – Black Box Testing and Reporting

This experiment looks to evaluate the project as a culmination of all the work detailed so far in this report. This testing will be carried out as black box testing carrying forward the confidence in the validity of the individual methods of the system, through the testing detailed in Sections 4 and 5.

6.1.1 – Methods

These tests are designed to represent a genuine use case of the system, by passing in cell components and producing a build report summarizing the defects present.

5 cells were built using the Cellerate system, comprising of 10 active cell components, 5 anodes and 5 cathodes. The components were purposely prepared with little care to artificially introduce defects of all types outlined in the objectives of this project. The anodes and cathode used were 16mm and 14mm in diameter respectively. These cell components were prepared using typical materials: copper with a graphite coating. 5 trays were prepared with these components and the relevant additional components required to produce a coin cell, including a coin cell base, separator, spacer spring and coin cell cap. The Cellerate system was programmed to build these cells and the AI Cell Sight system produced a build report as an output, summarizing the required build information outlined in the objectives of this project.

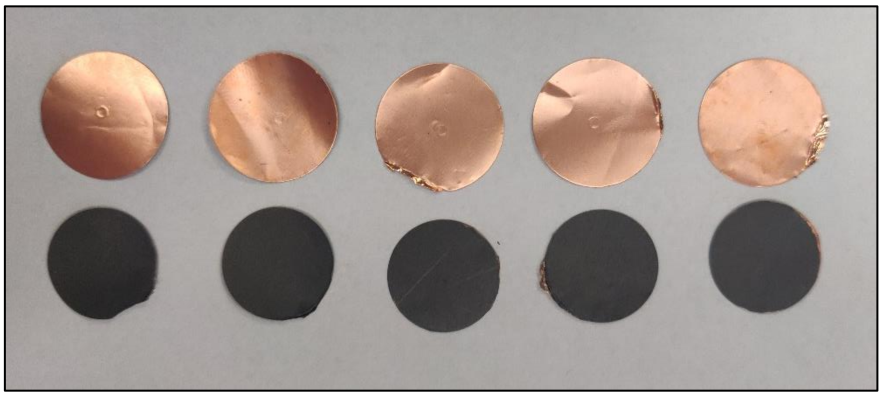
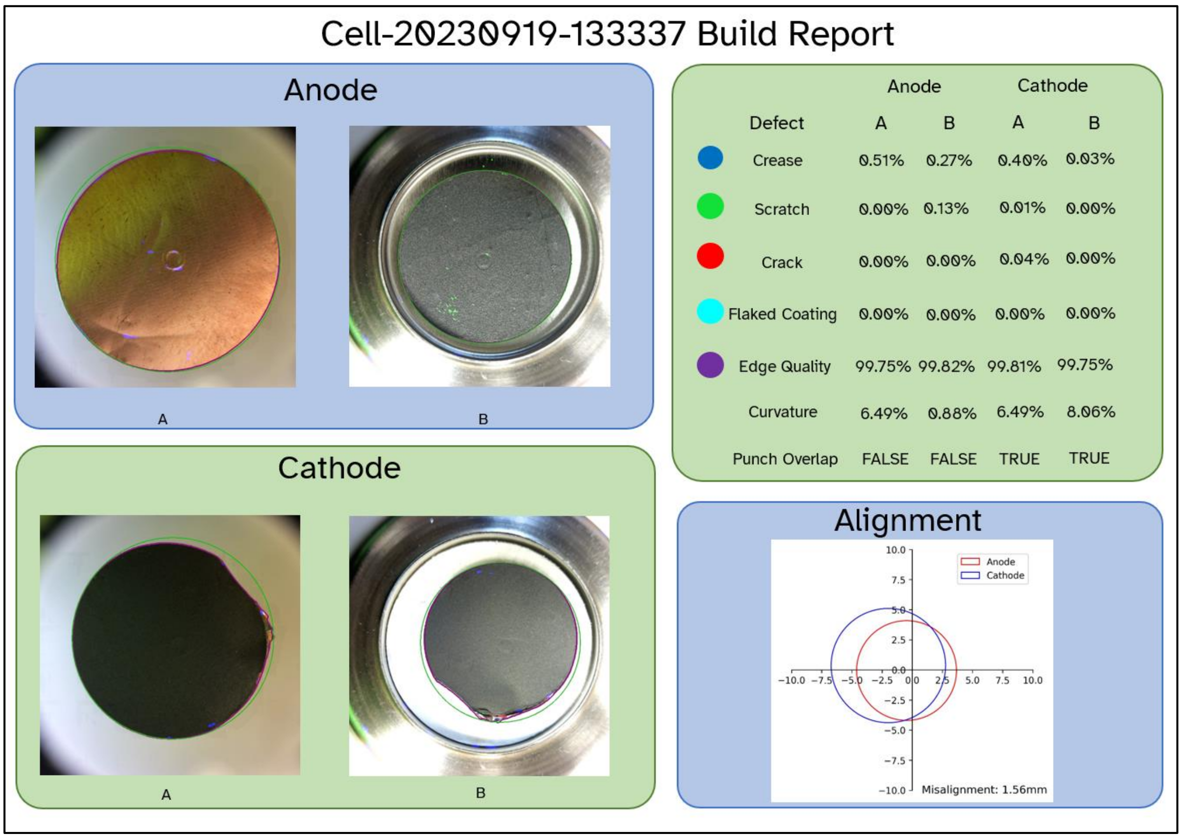


Fig.67 - Cell components used in experiment

Fig.67 show the cell components before the build procedure.

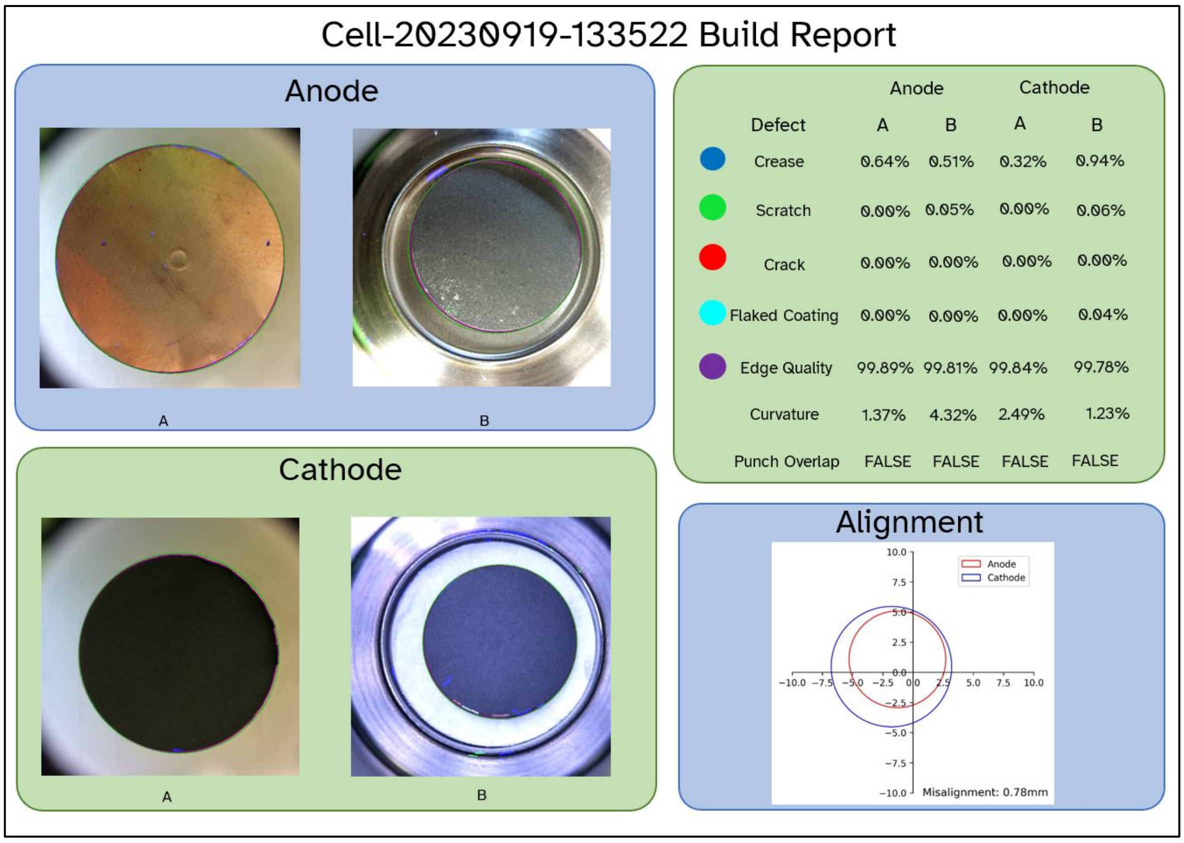
6.1.2 – Results

The figures below show the build reports produced by the system for each of the 5 cells that were built.


Cell-20230919-133337 Build Report

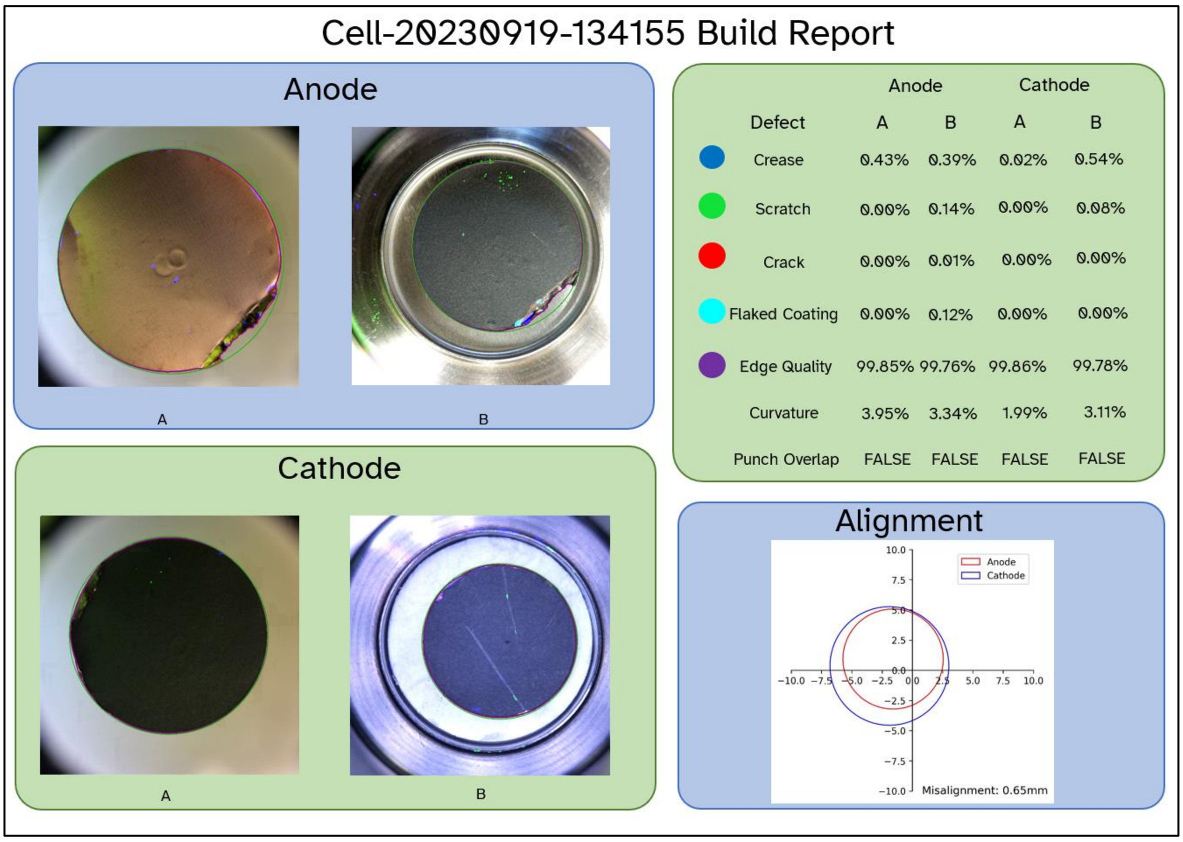
		Anode		Cathode	
	Defect	A	B	A	B
🔵	Crease	0.51%	0.27%	0.40%	0.03%
🟢	Scratch	0.00%	0.13%	0.01%	0.00%
🔴	Crack	0.00%	0.00%	0.04%	0.00%
●	Flaked Coating	0.00%	0.00%	0.00%	0.00%
🟣	Edge Quality	99.75%	99.82%	99.81%	99.75%
	Curvature	6.49%	0.88%	6.49%	8.06%
	Punch Overlap	FALSE	FALSE	TRUE	TRUE

Fig.68 - Cell 1 Build Report


Cell-20230919-133522 Build Report

		Anode		Cathode	
	Defect	A	B	A	B
🔵	Crease	0.64%	0.51%	0.32%	0.94%
🟢	Scratch	0.00%	0.05%	0.00%	0.06%
🔴	Crack	0.00%	0.00%	0.00%	0.00%
●	Flaked Coating	0.00%	0.00%	0.00%	0.04%
🟣	Edge Quality	99.89%	99.81%	99.84%	99.78%
	Curvature	1.37%	4.32%	2.49%	1.23%
	Punch Overlap	FALSE	FALSE	FALSE	FALSE

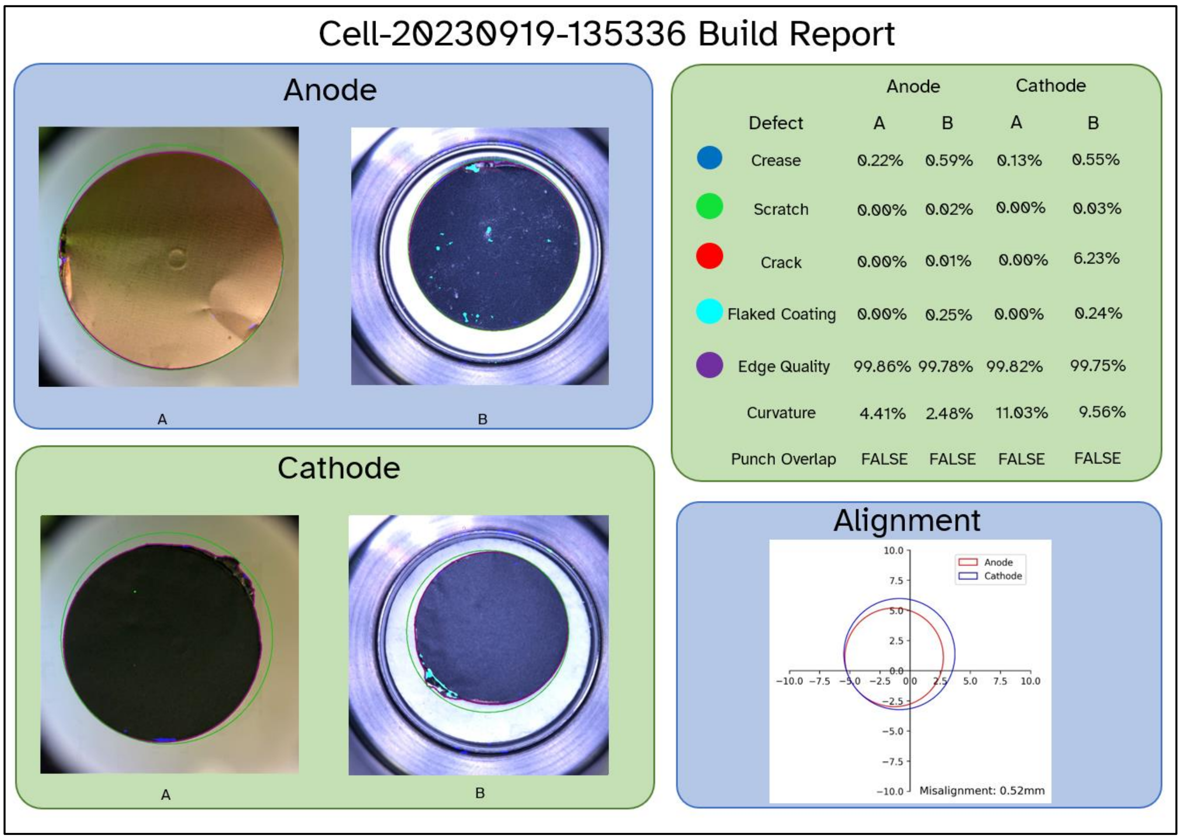
Fig.69 - Cell 2 Build Report



Cell-20230919-134155 Build Report

		Anode		Cathode	
	Defect	A	B	A	B
●	Crease	0.43%	0.39%	0.02%	0.54%
●	Scratch	0.00%	0.14%	0.00%	0.08%
●	Crack	0.00%	0.01%	0.00%	0.00%
●	Flaked Coating	0.00%	0.12%	0.00%	0.00%
●	Edge Quality	99.85%	99.76%	99.86%	99.78%
	Curvature	3.95%	3.34%	1.99%	3.11%
	Punch Overlap	FALSE	FALSE	FALSE	FALSE

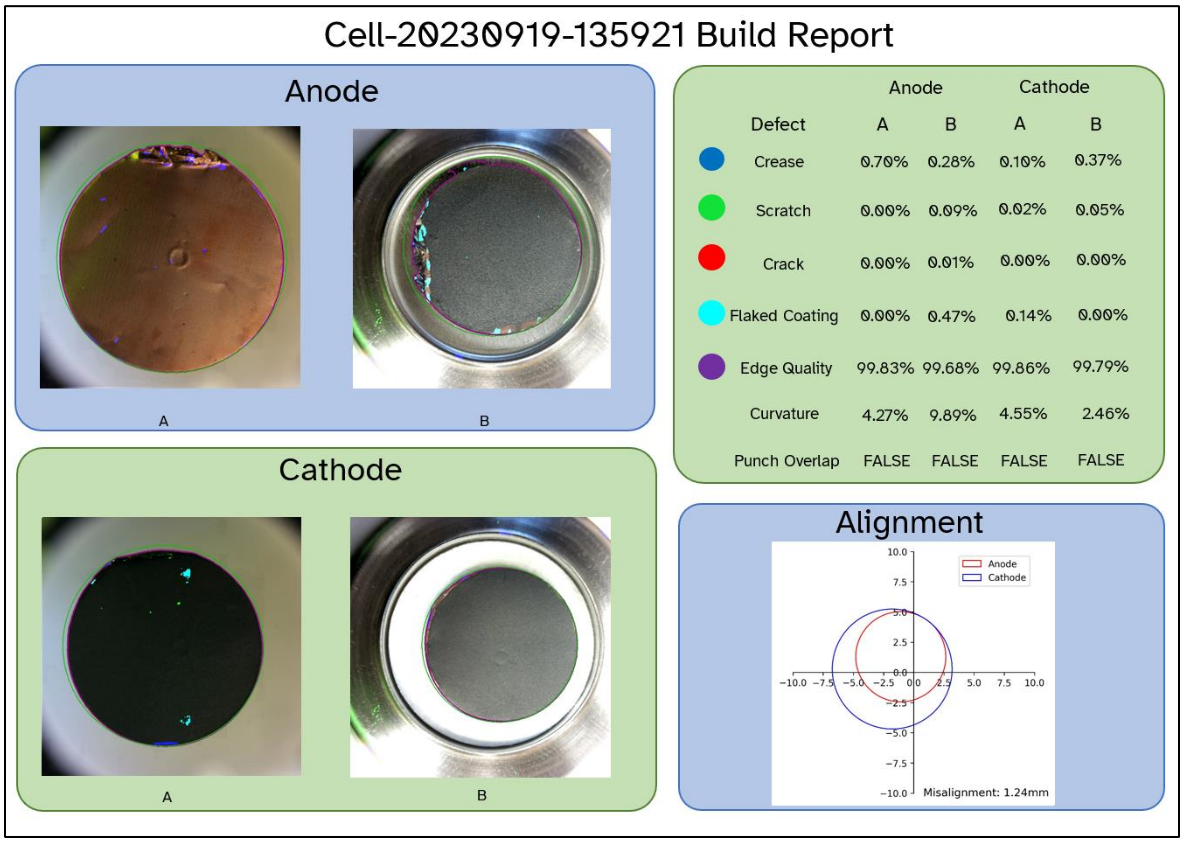
Fig.70 - Cell 3 Build Report



Cell-20230919-135336 Build Report

		Anode		Cathode	
	Defect	A	B	A	B
●	Crease	0.22%	0.59%	0.13%	0.55%
●	Scratch	0.00%	0.02%	0.00%	0.03%
●	Crack	0.00%	0.01%	0.00%	6.23%
●	Flaked Coating	0.00%	0.25%	0.00%	0.24%
●	Edge Quality	99.86%	99.78%	99.82%	99.75%
	Curvature	4.41%	2.48%	11.03%	9.56%
	Punch Overlap	FALSE	FALSE	FALSE	FALSE

Fig.71 - Cell 4 Build Report



		Anode		Cathode	
	Defect	A	B	A	B
🔵	Crease	0.70%	0.28%	0.10%	0.37%
🟢	Scratch	0.00%	0.09%	0.02%	0.05%
🔴	Crack	0.00%	0.01%	0.00%	0.00%
●	Flaked Coating	0.00%	0.47%	0.14%	0.00%
🟣	Edge Quality	99.83%	99.68%	99.86%	99.79%
	Curvature	4.27%	9.89%	4.55%	2.46%
	Punch Overlap	FALSE	FALSE	FALSE	FALSE

Fig.72 - Cell 5 Build Report

6.1.3 – Discussion

Firstly, the report builder module has been shown to successfully collate and present back data gathered during the cell build process. The system was consistently able to locate the component contour in the image and perform morphological analysis. These results reflect the success shown with the testing carried out in Section 4.0. Moreover, it could be argued that these results are quantitively better than those gathered previously. This can be seen in the purple line on the images which denotes the contour of the component. The accuracy of the morphological methods hinge on the accuracy to which the outer contour of the component is found, as the rationale of the numerical methods to ascertain curvature, edge detection and punch overlap have been validated.

This also demonstrates a shortcoming in the curvature algorithm. The cathode in the cell 4 build report demonstrates how a misshapen cell components can invalidate the curvature determination as the bounding circle is not representative of the component. As shown, the determined curvature of the cathode for both the top and bottom view are higher than expected. This is due to misshapen aspect of the edge defect causing the reference minimum bounding circle to misrepresent the component. The likelihood of this kind of defect being present in a true use case is low due to the higher quality of component preparation standards of actual end users compared to this experiment. However, other methods to determine this reference circle should be considered to mitigate this. One method that has been used previously is the Hough transform. As mentioned previously, this method uses a statistical approach using the relationships of points within an image

[17] to ascertain the presence of circles. This method could be resilient to these edge anomalies as a circle of best fit would be applied to the contour rather than encapsulating its entirety.

The accurate determination of the component contour allows for accurate location information about the components to be deduced after the component has been placed onto the build pad. This can be seen throughout the cell build reports in the form of the alignment plots. This information is extremely valuable to an end user as alignment of the active components within the cell is important for optimum performance. This alignment analysis gives clear feedback to the end user about the alignment of their active components, allowing them to monitor the effect this has on cell performance and to discard cells which don't meet a standard. It must be noted that the actual post-place alignments themselves are not optimal during this testing, most notably cell 1. This is due to the Cellerate software build used to complete these tests was going through a large transitional period and the alignment calibration was not completed optimally. However, this is not necessary to validate this systems ability to locate the component after placing and report back active component alignment.

Furthermore, the algorithm once again proved its ability to spot punching overlap on a component. Referring to cell build report 1, the system was able to determine the punching overlap present in the cathode from both the top and bottom cameras.

The most surprising and disappointing outcome of this testing was the performance of the machine learning component of the project. The high success rate and performance seen in the initial validation tests in Section 5.0 were not seen during these tests. Although defects were being detected, the activation on these defects and the success rate at which they are being identified was not acceptable to address the initial objectives.

This could be due to a number of factors. The most probable factor is a combination of an overfit model and a leak of similar data between the training sub dataset and the validation sub dataset. When a model overfits, it 'memorises' the training data into the model weights. This means it performs very well on the data it was trained on but it does not generalise well to unseen data [15]. This effect would be obvious when applied to the validation sub dataset as that data has never been seen by the model during training and would be respond poorly with the model. However, the initial dataset contains similar images that differ only with change in lighting conditions and contain the same defects in shape and class. This means an overfit model that had 'memorised' a defect would respond well to a copy of that defect in the validation data set, giving the illusion of great generalisation.

Another reason could be the context of the images within the dataset. The images were captured using a static camera and not using the Cellerate system with the LED ring. The LED ring gives a very niche and specific characteristic to images in terms of lighting, which is not reflected in the data used to teach the model.

A further reason could be underrepresentation or non-representation of certain defects contained in the images present in the build reports in Section 6.1.2, which is also a symptom of a small and non-diverse dataset. An example of this can be seen in build report 4, anode B. These types of defects are novel and not represented in the dataset. The defects would intuitively fall somewhere between flaked coating and scratch in the current dataset labels.

The overall reflection from the unsatisfactory results of the machine learning component is that, to address the shortcomings, more data is required to be added to the training dataset. This data should be captured from the system itself in the same use case context. More data on all defect subtypes is required to broaden the knowledge captured in the model's weights. The amount of datapoints on each defect is currently relatively small when compared to other sophisticated models, so therefore it understandable the model would not perform to the levels displayed in the validation testing in Section 5.

With this in mind however, the testing still proves the potential of this machine learning methodology. Defects are being detected just not at the reliability and accuracy required to truly satisfy viability at this stage. With additional data, there is optimism that this same methodology can produce a model that satisfies it's expectations to a higher level.

6.2 – Benchmarking

Due to the nature of the application, consideration had to be given to the execution time of the additional processing that this project adds to the Cellerate system. The system must be able to continuously build cells without delay, any delays introduced by this project would inherently devalue the Cellerate system as throughput is one of the most critical selling points.

An investigation into the processing and inference times accrued during runtime was conducted to understand the optimal processing pipeline, this would ultimately determine whether the processing of each cell would be done sequentially or in batches at a convenient time.

During runtime the only critical process that is required is the contouring of the cell by the bottom camera to determine the location of the cell component centroid within the image. The centroid is used to calculate additional offsets so the component is placed into the middle of the coin cell casing. The additional processing described throughout this project are able to be completed at a later point as none have bearing on the cell build process. This could be in batches at the end of build sessions, or as the autoloader is grabbing the next tray if the execution time permits.

6.2.1 – Methodology

An experiment was conducted to evaluate the execution times of various stages of processing required for both the morphological analysis and the machine learning components.

For the experiment, timign benchmarks were introduced throughout the code at critical points and the absolute time taken from the Raspberry Pi was output and collected. 5 electrodes were processed through the system and their execution times were saved. This process was repeated 5 times and the average was taken.

6.2.2 – Results

6.2.2.1 – Morphological Analysis Results

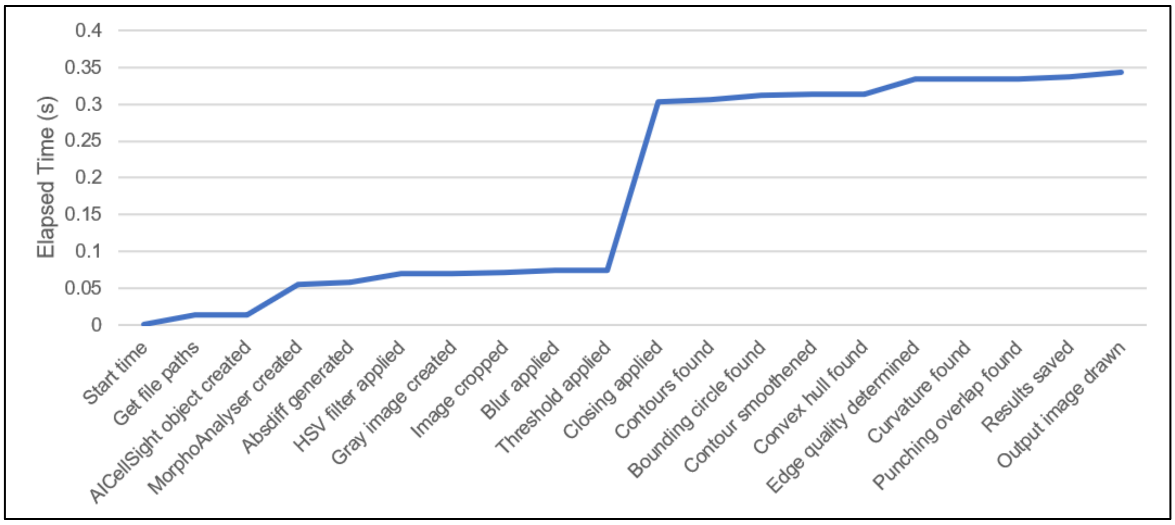


Fig.73 - Elapsed time of stage by stage execution of morphological analysis

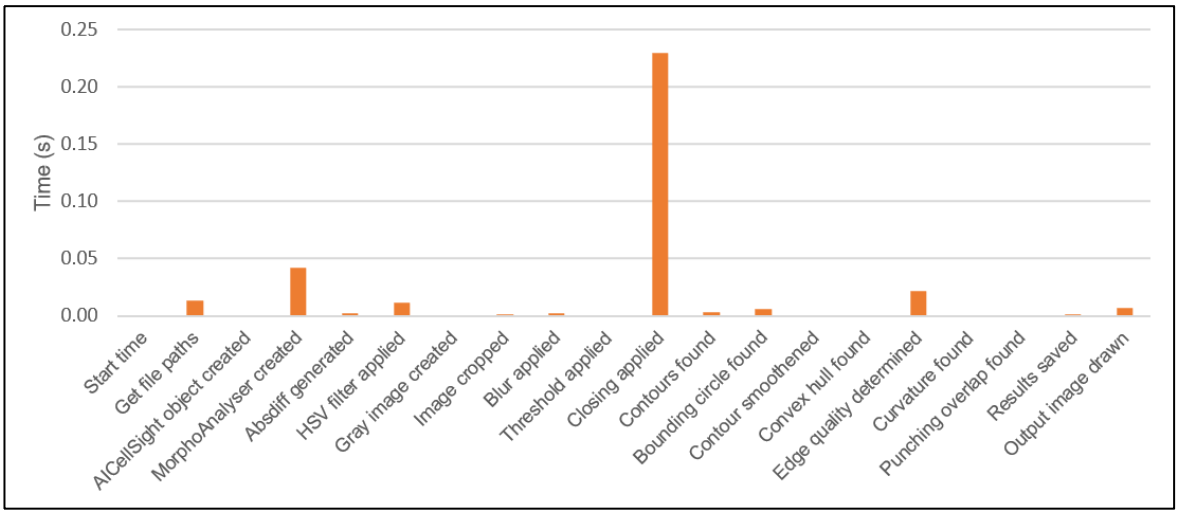


Fig.74 - Execution time of each stage of morphological analysis

Fig.73 and Fig.74 show the elapsed time and individual execution times by stage of the morphological analysis procedure respectively.

6.2.2.2 – Machine Learning Surface Inspection Results

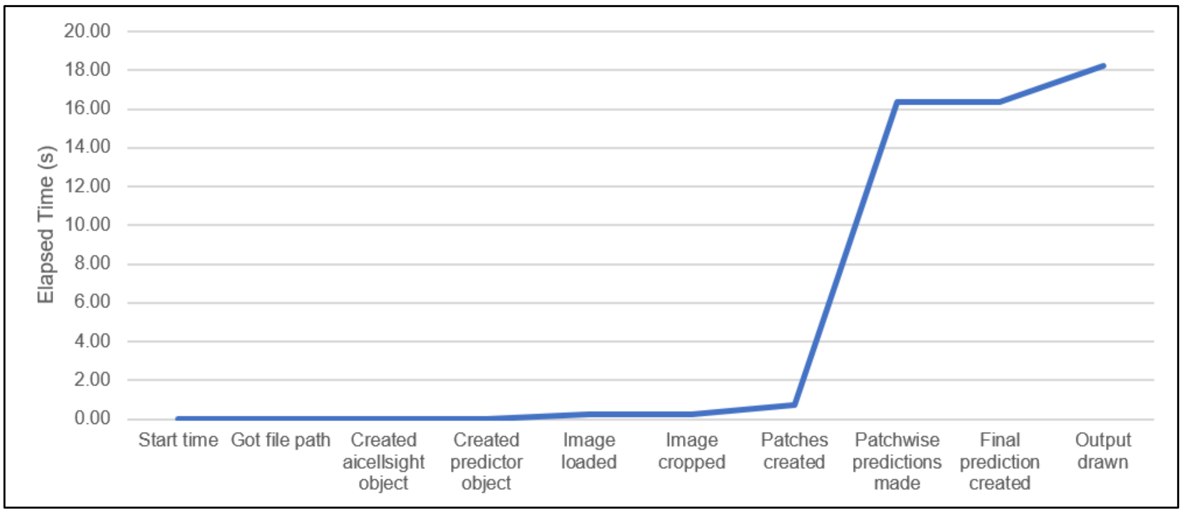


Fig.75 - Elapsed time of stage by stage execution of surface inspection

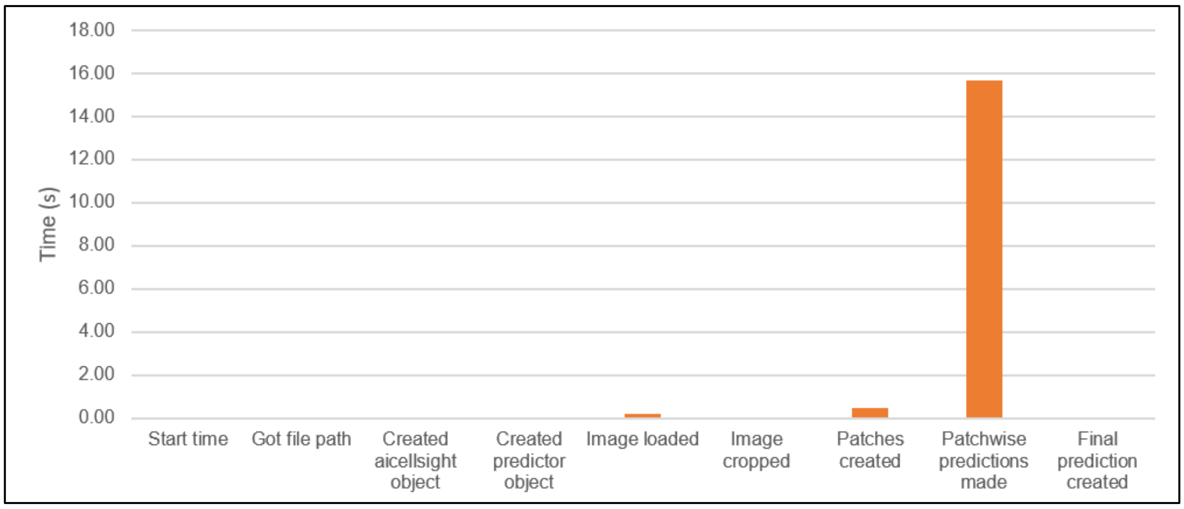


Fig.76 - Execution time of each of surface inspection

Fig.75 and Fig.76 show the elapsed time and individual execution times by stage of the surface inspection procedure respectively.

6.2.3 Discussion

Firstly, the average execution time of the morphological analysis is relatively quick for what is required for this system, the entire execution of the code took less than 0.35 seconds. As can be seen from Fig.73. The most processing heavy stage was the morphological closing operation, which is used to fill in gaps in the threshold mask and ensure the contour is an optimal representation of the outside of the cell component. It is no surprise that this is stage is the most processing heavy as kernel operations, such as opening and closing, require vast amounts of kernel convolutions and computation. Referring back, it was for this reason that the method for creating the smoothened contour was changed in Section 4.1. What can be confirmed is that this processing can be done in line with systems normal execution and does not present a requirement for batch processing.

With the machine learning surface inspection however, there is a much greater time demand. This processing time demand comes almost solely from the patch wise predictions stage. As the images taken by the system are large (2566x1944x3) they must first be reduced into a stack of images compatible with the input layer of the machine learning model (256x256x3) and inferenced through the model individually. This means that for a single image, even with a crop applied to isolate the useful part of the image containing the cell component, 25 separate model inferences are required before the image is stitched back together for output. Fig.76 shows that this stage incurred a nearly 16 seconds delay, which is somewhat expected for 25+ image inference through the model, but by far presents the most processor demand of the whole system. This is additionally impressive considering the limitations of the Raspberry Pi. The hardware is only compatible with Tensorflow lite which is a lightweight version of Tensorflow that strips away any parallel inferencing optimisations that GPUs are capable of and converts the model into a compatible CPU procedurally executable file.

A combined execution time of both the morphological analysis and surface inspection came to 18.45s. This only represents the execution time for one image though and under normal circumstances there will be at least 4 images to process per cell: one image for either surface of the activate components of the cell, the anode and cathode. This means that there is a minimum time requirement per cell of around 73s. Inserting this delay into the normal build routine of the assembler is of course not acceptable. Adding an additional minute to the cell build time would devalue the system significantly.

There are times during the cell build where the assembler is idle however, during crimping and during an autoloader operation. During these operations the assembler is waiting to receive the coin cell back from crimping or to receive the next tray from the autoloader. The delay incurred while crimping is around 45 seconds and the delay incurred while changing trays is around 30 seconds. This time can be used by the system to process the images in the cell which would cover the additional required processing time..

Fig.77 shows a pipeline diagram of the execution order of the system to accommodate or the extra processing time required. The Assembler module is never idle, using the time when the crimper and autoloader are controlling the flow of the cell components to perform the required processing to minimise hang ups in the operation flow.

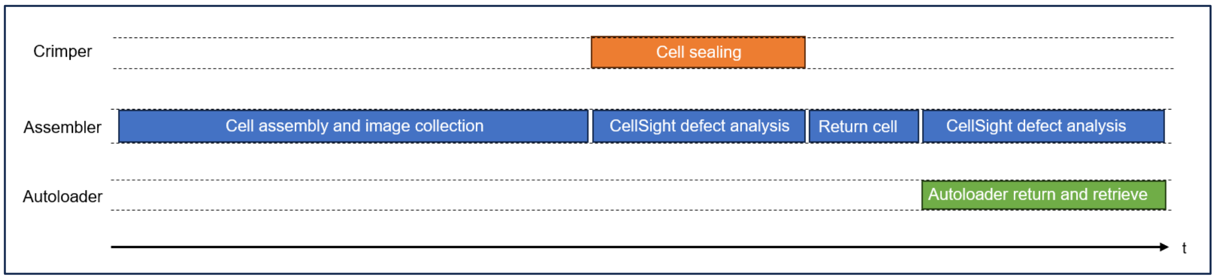

Crimper
Cell sealing
Assembler
Cell assembly and image collection
CellSight defect analysis
Return cell
CellSight defect analysis
Autoloader
Autoloader return and retrieve
t

Fig.77 – Pipeline diagram of the cell build process with stage descriptions

7.0 - Conclusions

The goal of the project was to develop software analysis tools that would provide the end users of the Cellerate system with more data to optimise and accelerate their battery research. This goal was to be realised by designing tools to provide data to the end user to isolate variables in the cell build process to build better batteries and bolster the work done to shift society over to greener and more effective energy storage methods. The data provided by the project would allow an end user to correlate shortcomings in coin cell battery production with cell test data to identify optimisations that can be made in the component preparation stages. The defined aspects to collect data on were defined as morphological defects to the cell component such as edge quality, curvature and punching overlap. As well as surface defects defined as scratches, creases, flaked electrode coating and coating cracks.

A discussion of the effectiveness of the methods applied can be broken, once again into two parts due to the vastly different implementations to satisfy the criteria of each part. Firstly, the morphological analysis component of the project looked to tackle defects related to the shape of the components specifically. A more traditional image processing approach was hypothesized to be the most suitable as methods to find and contour shapes in images have long been established with many effective implementations in past, work such as [20] [16].

The methods described in this work have been shown to be mostly effective at achieving the goals that were laid out at the beginning of the project. The method of using a stable background image as a reference point to subtract the foreground and isolate the cell component has been shown to be effective and with some additional filtering a good representation of the contour of the cell component could be extracted and analysed. This is particularly effective for determining the degree of curvature present in the cell component when placed into the cell and also the presence of any punching overlap. These two are particularly effective as the reliance on a hyper accurate representation of the cell contour is not required. The experiments described in Section 4 and Section 6 show that the methods to determine component curvature and punching overlap are successful and provide good accuracy. It has been shown that the relationship between the height of the component curvature and the area of the shape when looked down on as a 2D plan is solid and can be used to provide an indication of component curvature.

The results for the punching overlap are even stronger with 100% accuracy of the detection of punching overlap with no false positives or false negatives. It was also shown to be effective even in the presence of curvature in the component. These results validate the effectiveness of using the convex hull of the contour to determine any missing material.

In earlier testing in Section 4.0 there were shortcomings of the method to determine the cell edge quality. These were the reliance on a very accurate approximation of the cell contour in order to make high precision comparisons with the smoothened contour. The testing in Section 6.0

demonstrated good performance of the morphological analysis operations on the cell but still contained an instance in which noise made it through the filtering stages to invalidate the accurate component contour.

The validation experiment methodology also proved to have some flaws as it relies on subjective interpretation of what makes a ‘good’ and ‘bad’ edge as it is difficult to quantify. This subjective interpretation could conflict with the quantitative analysis method and skew results. Another shortcoming that must be reiterated is the requirement of a calibration. The HSV filter requires a calibration to work optimally which is ultimately a pitfall of the entire method. Different materials require separate calibrations and it is possible that moving the machine into a different area may cause the requirement for another calibration to be completed, due to varying lighting conditions. A method to remove the dependency on this calibration would be a huge next step for the system.

The remaining objectives to be detected and qualified by the analysis tools were scratches, cracks, creases and flaked coatings. It was hypothesised that these would lend themselves much more effectively to a machine learning approach. Throughout the literature, there have been many implementations of defect detection models that use various architectures and classification types to achieve this.

The classification methodology which stood out as the most appropriate for this application was semantic segmentation. This allows the user to see in great detail not only the presence of a defect but also the localised area in which it is present. It was decided that this approach would give the maximum amount of data back to the end user to optimise their cell preparation stages and identify and rectify shortcomings in how their components are handled.

The overwhelming majority of defect detection applications that use convolutional neural networks in the literature use the U-Net architecture [30] [31] [32]. This is due to its great response to small bespoke datasets, such as what would have been available for this application.

The methodology in Section 5 done to evaluate the effectiveness of various models revealed that there exists a fundamental unique relationship between the dataset and the model architecture to garner the best results. Many of the variations on the U-Net architecture boast improved results of the standard U-Net. The quoted improvements in this case were not seen during this investigation. Most performed worse and some performed marginally better, but at the expense of greatly increasing the complexity of the model.

It is clear from this investigation that each dataset should be treated as its own isolated instance and the promise of improved results seen when applied to other datasets should not be assumed. These results can point you in a general direction to guide the optimisation of the model but it is up to individual testing and tweaking of the model parameters and architectures to find the right architecture for the specific dataset.

The final model evidently performed well when referencing the output classifications when tested using the validation dataset. However, when applied to truly novel data, in Section 6.0, the model clearly did not perform as expected. It is hypothesized that the model was overfit during training and a leak of similar data between the training sub dataset and validation sub dataset gave an illusion of great performance and generalisation. It was determined that the testing, while disappointing, still demonstrates the potential of the technology given that a satisfactory amount of data is available for training which sufficiently represents the broad scope of potential defects. As the system is used by more customers and more cell data is collected the dataset will become more diverse and mature. As a starting point to something that will indefinitely improve the results gathered in this work are promising regardless.

Furthermore, the system integration testing in Section 6 showed that the collected data on the cell build inspection is collated and reported back to the user in a clear and concise manner. In addition, the benchmarking results gathered show that the system can be run on the lightweight Raspberry Pi device in reasonable time. With some scheduling design, the Raspberry Pi is capable of processing the neural network inferencing, which is the most time demanding process, during the downtime of the assembler system, during a crimping or autoloader process. A big consideration when setting out the design of the system was not to compromise the throughput of the Cellerate system, as this constitutes one of its greatest USPs. Not compromising the current operation of the Cellerate system whilst adding the additional functionality discussed in this project is a great achievement.

7.2 - Further Work

One of the most promising and encouraging developments that has been brought to the public eye whilst completing this project is Meta's Segment Anything Model (SAM). This open source model has been trained on an incredibly large dataset of over 11 Billion images. The model is able to segment the articles within an image to a great accuracy and reliability. If this model could be reasonably implemented into this system it could optimise many of the processes.

The most impact it could have is to improve the reliability of edge quality detection algorithm. It would remove the need for the background subtraction methodology, including the HSV filter. This advancement would put greater trust that the contour edge found as part of this process is accurate and that the edge quality calculations performed using it are representative and reliable. It would also remove the need to take an additional background image of the cell build plate before the component is placed to allow the background subtraction process to be performed, saving additional time. Moreover, the successful implementation of this technology would remove the need for any component calibration of the HSV filter. This was a great concern as the current method adds some further user dependency to find a good calibration

During the SAM model investigation, further testing and optimisation should be done to improve the robustness of the system. This is to gain further confidence the system will work as expected in various settings. This testing should be mainly focussing around lighting condition changes and whether this could disrupt the calibration. Of course, with the implementation of the SAM model this not need be a priority as the model is adaptable and resilient to changes in lighting.

Of course, the most natural further work for this project is to continue to collect and label data from the field to grow and improve the performance of the machine learning model. With a constant flow of actual use case data from the field the sophistication and performance of the model could become more viable within a short period of time.

7.3 - Final Statement

In conclusion, a great deal of personal development has been achieved in the past year completing this project. All the knowledge displayed in this report surrounding the machine learning topic was exclusively gained during the process of study. Both technical and managerial skills have been developed through working with the Cellerate team and managing personal deadlines and milestones. The technical results of the project are generally positive with clear direction of how these can be improved and built upon going forward. The work completed in this project is significant as there is currently nothing on the market that rivals what the Cellerate system can achieve, let alone with the additional functionality developed in this project. Therefore, the fusion of this work with the Cellerate product has an opportunity to really penetrate the battery research space and help advance the field.

8.0 - References

[1] UNFCC, “The Paris Agreement,” [Online]. Available: https://unfccc.int/process-and-meetings/the-paris-agreement/the-paris-agreement.

[2] J. Wood, “Batteries are a key part of the energy transition. Here’s why,” 15 September 2021. [Online]. Available: https://www.weforum.org/agenda/2021/09/batteries-lithium-ion-energy-storage-circular-economy/. [Accessed 12 January 2023].

[3] P. Stevens, “The battery decade: How energy storage could revolutionize industries in the next 10 years,” CNBC, 30 December 2019. [Online]. Available: https://www.cnbc.com/2019/12/30/battery-developments-in-the-last-decade-created-a-seismic-shift-that-will-play-out-in-the-next-10-years.html. [Accessed 12 January 2023].

[4] C. Crownhart, “What’s next for batteries,” MIT Technology Review, 4 January 2023. [Online]. Available: https://www.technologyreview.com/2023/01/04/1066141/whats-next-for-batteries/. [Accessed 12 January 2023].

[5] Cellerate, “The Cell Asseembly and Sealing System,” Cellerate, 2022. [Online]. Available: https://www.cellerate.co.uk/products. [Accessed 12 January 2023].

[6] Intel, “What Is Computer Vision?,” 2023. [Online]. Available: https://www.intel.co.uk/content/www/uk/en/internet-of-things/computer-vision/overview.html. [Accessed 28 June 2023].

[7] Zeiss, “Why good vision is so important,” 16 October 2021. [Online]. Available: https://www.zeiss.co.uk/vision-care/eye-health-and-care/health-prevention/why-good-vision-is-so-important.html. [Accessed 28 June 2023].

[8] Enoch J, L. McDonald, L. Jones, J. PR and D. Crabb, “Evaluating Whether Sight Is the Most Valued Sense,” *JAMA ophthalmology,* vol. 137, no. 11, pp. 1317-1320, 2019.

[9] T. O. Sharpee and R. J. Rowekamp, “How the brain recognizes what the eye sees,” Nature Communications, 8 June 2017. [Online]. Available: https://www.salk.edu/news-release/brain-recognizes-eye-sees/. [Accessed 28 June 2023].

[10] P. Ganesh, “Types of Convolution Kernels : Simplified,” Towards Data Science, 18 October 2019. [Online]. Available: https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37. [Accessed 28 June 2023].

[11] L. Shapiro and G. Stockman, “Filtering and Enhancing Images,” in *Computer Vision*, Washington, Michigan, Pearson, 2000, pp. 145-209.

[12] M. A. Ansari, D. Kurchaniya and M. Dixit, “A Comprehensive Analysis of Image Edge Detection Techniques,” *International Journal of Multimedia and Ubiquitous Engineering,* vol. 12, no. 11, pp. 1-12, 2017.

[13] J. Canny, “A Computational Approach to Edge Detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vols. PAMI-8, no. 6, pp. 679-698, 1986.

[14] OpenCV, “OpenCV,” 2023. [Online]. Available: https://opencv.org/. [Accessed 12 January 2023].

[15] N. Ryan and K. O'Shea, “An Introduction to Convolutional Neural Networks,” arXiv, Aberystwyth, Lancashire, 2015.

[16] M. Bastan and S. S. Bukhari, “Active Canny: Edge Detection and Recovery with Open,” *IET Image Processing,* vol. 11, no. 12, pp. 1325-1332, 2017.

[17] A. S. Hassaniem, S. Mohammed, M. Sameer and M. E. Ragab, “A Survey on Hough Transform, Theory, Techniques and Applications,” *IJCSI International Journal of Computer Science Issues,* vol. 12, no. 1, pp. 139-156, 2015.

[18] H. Tabkhi, R. Bushey and G. Schirner, “ALgorithm and Architecture Co-Design of Mixture of Gaussian (MoG) Background Subtraction for Embedded Vision,” IEEE, Boston, 2014.

[19] Pranjal, “Background Subtraction for multiple conditions,” Cronj, 2019. [Online]. Available: https://www.cronj.com/blog/background-subtraction/. [Accessed 29 June 2023].

[20] C. G. Spinola, J. Canero, G. Moreno-Aranda, J. M. Bonelo and M. Martin-Vasquez, “Real-Time Image Processing for Edge Inspection and Defect Detection in Stainless Steel Production Lines,” in *IEEE International Conference on Imaging Systems and Techniques*, Batu Ferringhi, 2011.

[21] P. Wang, X. Zhang, Y. Mu and Z. Wang, “The Copper Surface Defects Inspection System Based on Computer Vision,” in *2008 Fourth International Conference on Natural Computation*, Jinan, 2008.

[22] M. Jogin, M. M. S. Mohana, D. G. D, M. R. K and A. S, “Feature Extraction using Convolutional Neural Networks (CNN) and Deep Learning,” in *2018 3rd International Conference on Recent Trends in Electronics, Information & Comminication Technology*, Bangalore, 2018.

[23] V. Lakschmanan, M. Gorner and R. Gillard, “Object Detection and Image Segmentation,” in *Practical Machine Learning For Computer Vision*, Sebastopol, O'Reilly, 2021.

[24] H. Zhao, Y. Lv, J. Sha, R. Peng, Z. Chen and G. Wang, “Research on Detection Method of Coating Defects Based on Machine Vision,” in *2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, Dalian, 2021.

[25] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” arXiv, Washington, 2015.

[26] O. Ronneberger, P. Fischer and T. Brox, “U-Net: Convolutional Networks for Biomedical,” in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, Munich, 2015.

[27] Y. Hou, J. Yan and D. Zeng, “ELU-Net: An Efficient and Lightweight U-Net for Medical Image Segmentation,” *IEEE Access,* vol. 10, pp. 35932-35941, 2022.

[28] K. Li and G. Ding, “L-FCN: A lightweight fully convolutional network for biomedical semantic segmentation,” in *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, Madrid, 2018.

[29] J. B. Roerdink and A. Meijster, “The Watershed Transform: Definitions, Algorithms and Parrallelization,” *Fundamenta Informaticae,* vol. 41, pp. 187-228, 2001.

[30] X. Cao, B. Yao, B. Chen and Y. Wang, “Multi-defect detection for magnetic tile based on SE-U-Net,” in *2020 IEEE International Symposium on Product Compliance Engineering-Asia (ISPCE_SN)*, Chongqing, 2020.

[31] Z. Xinzi, “BSU-Net: A Surface Defect Detection Method Based On Bilaterally Symmetric U-Shaped Network,” in *2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*, Harbin, 2020.

[32] D. Amin and S. Akhter, “Deep Learning-Based Defect Detection System in Steel Sheet Surfaces,” in *2020 IEEE Region 10 Symposium (TEMSYMP)*, Dhaka, 2020.

[33] S. Sahoo, “Residual blocks — Building blocks of ResNet,” 27 November 2018. [Online]. Available: https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec. [Accessed 12 January 2023].

[34] G. Shang, G. Liu, P. Zhu, J. Han, C. Xia and K. Jiang, “A Deep Residual U-Type Network for Semantic Segmentation of Orchard Environments,” *Applied Sciences,* vol. 11, no. 1, pp. 322-335, 2021.

[35] S. Moon, “How data augmentation affects machine learning,” Datahunt, 23 May 2023. [Online]. Available: https://www.thedatahunt.com/en-insight/how-data-augmentation-impacts-machine-learning. [Accessed 29 June 2023].

[36] J. Wang and L. Perez, “The Effectiveness of Data Augmentation in Image Classification using Deep Learning,” arXiv, Stanford, 2017.

[37] TensorFlow, “Introduction to TensorFlow,” 2023. [Online]. Available: https://www.tensorflow.org/learn. [Accessed 6 June 2023].

[38] Keras, “About Keras,” [Online]. Available: https://keras.io/api/. [Accessed 6 June 2023].

[39] TensorFlow, “Deploy machine learning models on mobile and edge devices,” [Online]. Available: https://www.tensorflow.org/lite. [Accessed 20 June 2023].

[40] D. S. Bhattiprolou, “multiclass semantic segmentation using U-Net,” 11 March 2021. [Online]. Available: https://github.com/bnsreenu/python_for_microscopists/blob/master/208_multiclass_Unet_sandstone.py. [Accessed 9 May 2023].

[41] D. S. Bhattiprolou, “127_data_augmentation_using_keras.py,” 1 March 2020. [Online]. Available: https://github.com/bnsreenu/python_for_microscopists/blob/master/127_data_augmentation_using_keras.py. [Accessed 6 June 2023].

[42] OpenCV, “How to Use Background Subtraction Methods,” OpenCV, 29 December 2022. [Online]. Available: https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html#:~:text=OpenCV%3A%20How%20to%20Use%20Background%20Subtraction%20Methods&text=Background%20subtraction%20(BS)%20is%20a,scene)%20by%20using%20static%20cameras.. [Accessed 12 January 2023].

[43] OpenCV, “Morphological Transformations,” OpenCV, 29 December 2022. [Online]. Available: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html. [Accessed 12 January 2023].

[44] W. Benesova, “Curvature Scale Space in OpenCV,” 12 April 2013. [Online]. Available: https://vgg.fiit.stuba.sk/2013-04/css-%E2%80%93-curvature-scale-space-in-opencv/. [Accessed 12 January 2023].

[45] D. Pandey, “Contours and Convex Hull in OpenCV Python,” 19 April 2021. [Online]. Available: https://medium.com/analytics-vidhya/contours-and-convex-hull-in-opencv-python-d7503f6651bc. [Accessed 13 January 2023].

[46] J. Keifer and J. Wolfowitz, “Stochastic Estimation of the Maximum of a Regression Function,” University of North Carolina, Durham, 1952.

[47] H. Robbins and S. Mondro, “A Stochastic Approximation Method,” University of North Carolina, Durham, 1951.

[48] E. Hofesmann, “IoU a better detection evaluation metric,” Towards Data Science, 25 August 2020. [Online]. Available: https://towardsdatascience.com/iou-a-better-detection-evaluation-metric-45a511185be1. [Accessed 2 August 2023].

[49] J. Korstanje, “The F1 Score,” Towards Data Sciecne, 31 August 2021. [Online]. Available: https://towardsdatascience.com/the-f1-score-bec2bbc38aa6. [Accessed 2 August 2023].

[50] Jacomex, “Glove Boxes for Battery Applications,” 2023. [Online]. Available: https://www.jacomex.com/application/energy/batteries/. [Accessed 6 June 2023].

[51] Apple, “Blurring an Image,” [Online]. Available: https://developer.apple.com/documentation/accelerate/blurring_an_image. [Accessed 2 August 2023].

[52] TagX, “Data Augmentation for Computer Vision,” medium, 4 Febuary 2023. [Online]. Available: https://medium.com/@tagxdata/data-augmentation-for-computer-vision-b97cbe0bae9e. [Accessed 29 June 2023].

[53] MTI, “19 mm Dia Graphite ANode Electrode Disk for CR20XX Coin Cells 100 pcs / pack - bccff-cms19,” 2022. [Online]. Available: https://www.mtixtl.com/bc-cf-cms-68.aspx. [Accessed 12 January 2023].

[54] F. Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions,” *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 1800-1807, 2017.

[55] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, “Going deeper with convolutions,” *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 1-9, 2015.

[56] A. Rosebrock, “Intersection over Union (IoU) for object detection,” 7 November 2016. [Online]. Available: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. [Accessed 2 August 2023].

[57] Cronj, “Background Subtraction For Multiple Conditions,” Cronj, 2019. [Online]. Available: https://www.cronj.com/blog/background-subtraction/. [Accessed 2 August 2023].

[58] Naveen, “What is a ReLU and Sigmoid activation function?,” Nomidl, 20 April 2022. [Online]. Available: https://www.nomidl.com/deep-learning/what-is-relu-and-sigmoid-activation-function/. [Accessed 2 August 2023].

[59] S. Saha, “A Comprehensive Guide to Convolutional Neural Networks - The EL15 Way,” Towards Data Science, 15 December 2018. [Online]. Available: https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/. [Accessed 12 January 2023].

[60] C. Kevin, “Feature Maps,” Medium, 11 May 2018. [Online]. Available: https://medium.com/@chriskevin_80184/feature-maps-ee8e11a71f9e. [Accessed 2 August 2023].

[61] R. Bhatia, “Top 5 Image Classification Research Papers Every Data Scientist Should Know,” Analytics India Mag, 17 September 2017. [Online]. Available: https://analyticsindiamag.com/top-5-image-classification-research-papers-every-data-scientist-should-know/. [Accessed 12 January 2023].

[62] Signal and Image Processing Lab, “Creating Image Segmentation Maps Using GANs,” 2021. [Online]. Available: https://sipl.eelabs.technion.ac.il/projects/creating-image-segmentation-maps-using-gans/. [Accessed 6 July 2023].

Appendix A: Algorithm Flowcharts

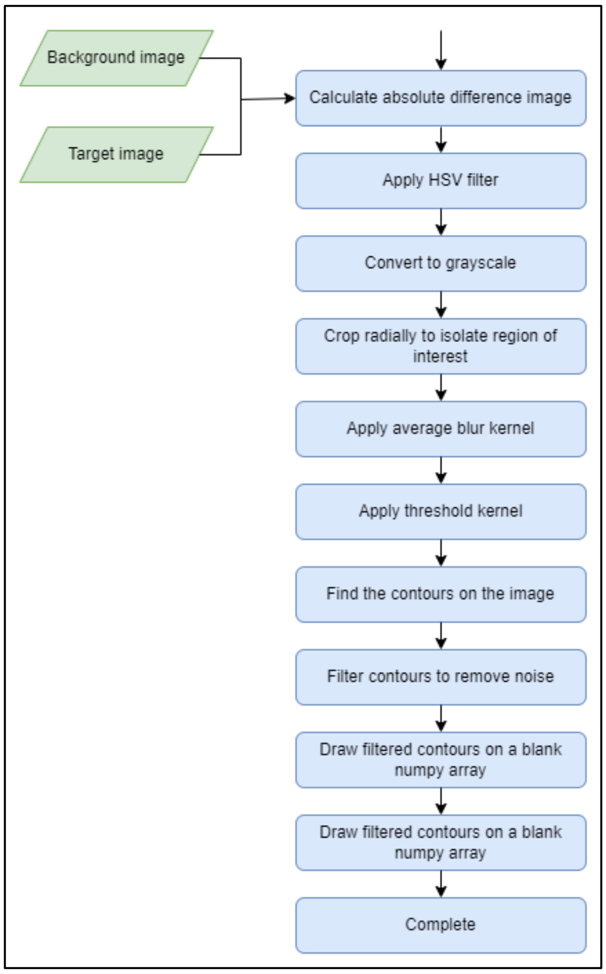

Background image
Target image
Calculate absolute difference image
Apply HSV filter
Convert to grayscale
Crop radially to isolate region of interest
Apply average blur kernel
Apply threshold kernel
Find the contours on the image
Filter contours to remove noise
Draw filtered contours on a blank numpy array
Draw filtered contours on a blank numpy array
Complete

Fig.78 – Image preprocessing flowchart

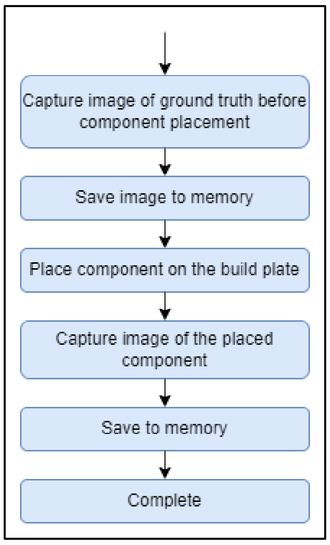

Capture image of ground truth before component placement
Save image to memory
Place component on the build plate
Capture image of the placed component
Save to memory
Complete

Fig.79 – Image collection flowchart

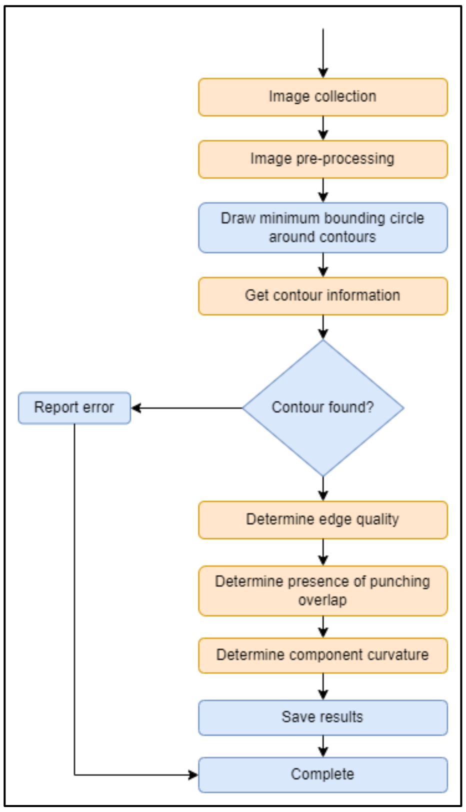


Fig.80 – Morphological analysis flowchart

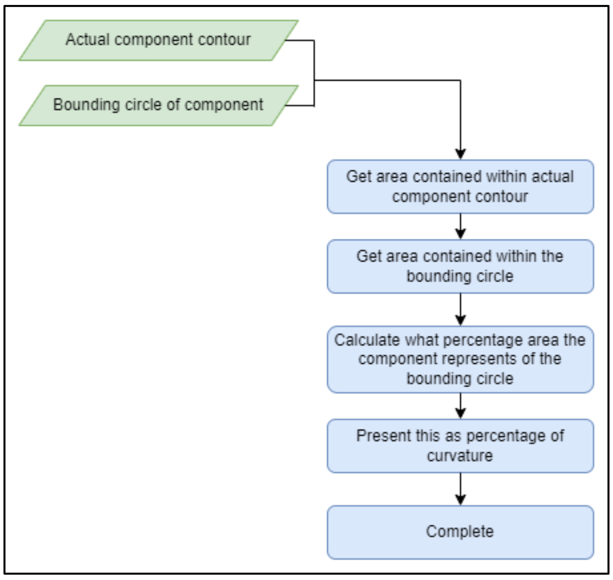

Actual component contour
Bounding circle of component
Get area contained within actual component contour
Get area contained within the bounding circle
Calculate what percentage area the component represents of the bounding circle
Present this as percentage of curvature
Complete

Fig.81 – Component curvature flowchart

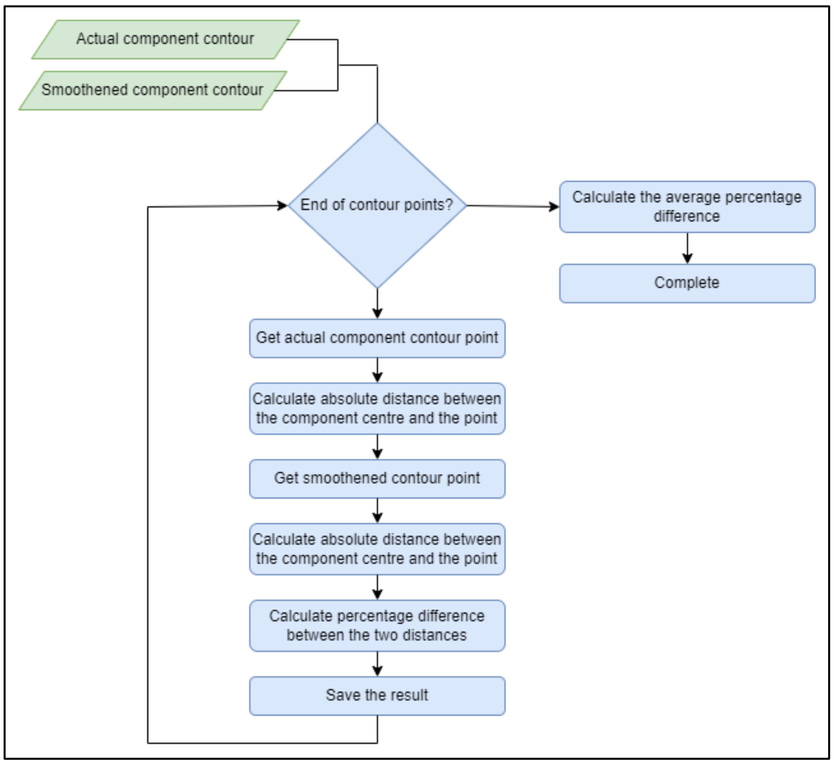

Actual component contour
Smoothened component contour
End of contour points?
Calculate the average percentage difference
Complete
Get actual component contour point
Calculate absolute distance between the component centre and the point
Get smoothened contour point
Calculate absolute distance between the component centre and the point
Calculate percentage difference between the two distances
Save the result

Fig.82 – Edge quality analysis flowchart

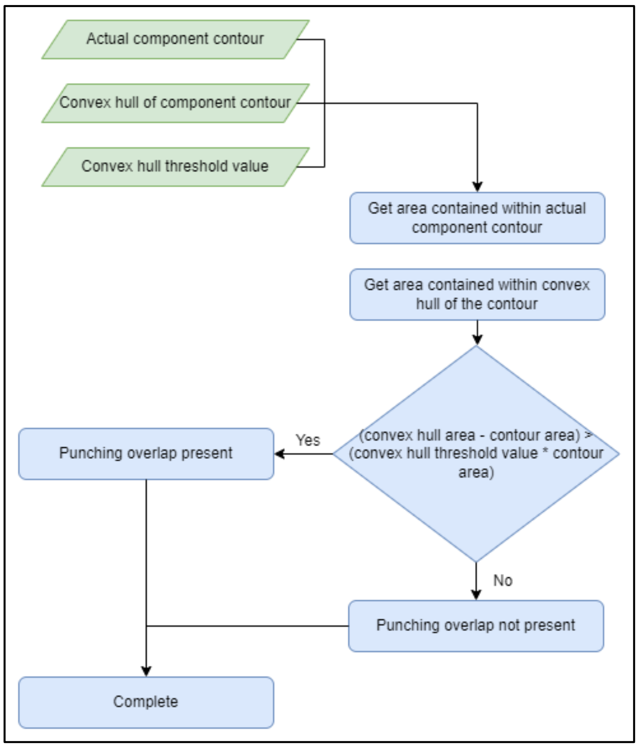

Actual component contour
Convex hull of component contour
Convex hull threshold value
Get area contained within actual component contour
Get area contained within convex hull of the contour
(convex hull area - contour area) > (convex hull threshold value * contour area)
Yes
Punching overlap present
No
Punching overlap not present
Complete

Fig.83 – Punching overlap flowchart

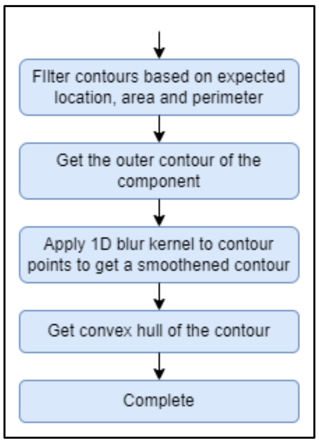

FIlter contours based on expected location, area and perimeter
Get the outer contour of the component
Apply 1D blur kernel to contour points to get a smoothened contour
Get convex hull of the contour
Complete

Fig.84 – Get contour information flowchart

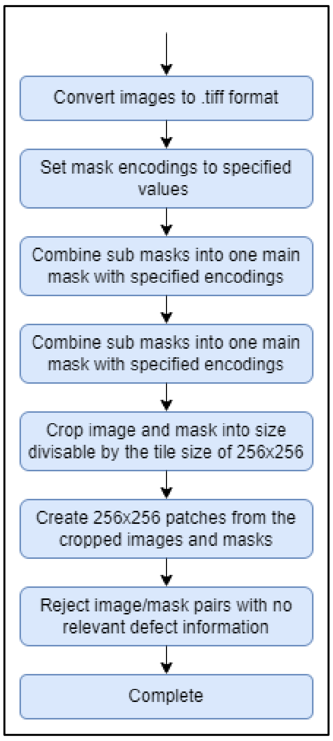

Convert images to .tiff format
Set mask encodings to specified values
Combine sub masks into one main mask with specified encodings
Combine sub masks into one main mask with specified encodings
Crop image and mask into size divisable by the tile size of 256x256
Create 256x256 patches from the cropped images and masks
Reject image/mask pairs with no relevant defect information
Complete

Fig.85 – Model generator pre-processing flowchart

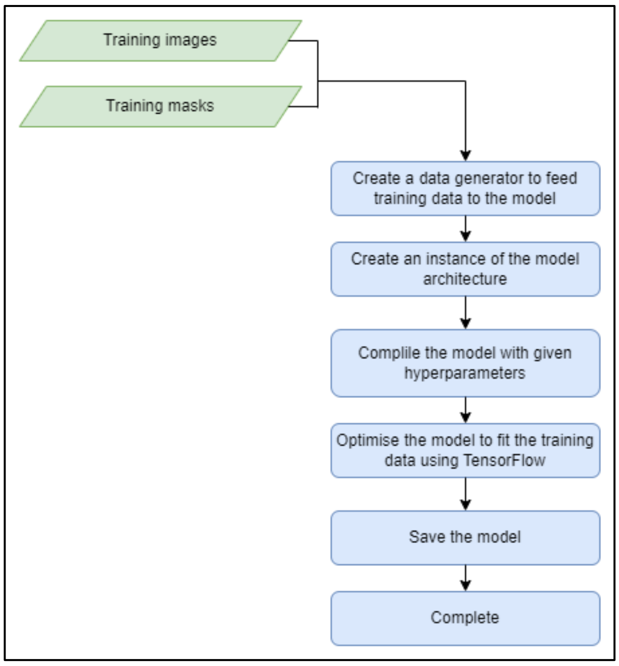

Training images
Training masks
Create a data generator to feed training data to the model
Create an instance of the model architecture
Complile the model with given hyperparameters
Optimise the model to fit the training data using TensorFlow
Save the model
Complete

Fig.86 – Model generator flowchart

Appendix B: Raw Results

	Pre Optimisations		After Smoothing Optimisation		After Smoothing and Edge Quality Optimisation	
Stage	Split(s)	Total Time(s)	Split(s)	Total Time(s)	Split(s)	Total Time(s)
Image Loading	0.24	0.24	0.24	0.24	0.23	0.23
Undistort Image	0.73	0.97	0.74	0.97	0.73	0.97
Background Removal	0.23	1.2	0.23	1.20	0.23	1.20
Locating Circle	0.18	1.39	0.18	1.39	0.18	1.38
Find Contour	0.02	1.41	0.02	1.41	0.02	1.40
Edge Quality Detection	164.56	165.96	14.18	15.59	0.10	1.50
Punching Overlap	0	165.97	0	15.59	0.00	1.50
Curvature Detection	0.01	166.09	0.01	15.59	0.10	1.50
Save Image	0.12	166.09	0.12	15.71	0.12	1.62

Table 10 - Raw results for Fig.41 and Fig.42

Electrode No.	Algo Determined Edge Quality %	Human assessed edge quality (/10)
1	99.7233%	8
2	99.7038%	7.5
3	99.6204%	5
4	99.6142%	5
5	99.7126%	7
6	99.7158%	7.5
7	99.7476%	8
8	99.7469%	7.5
9	99.6603%	6
10	99.6632%	6
11	99.6552%	7.5
12	99.7374%	7
13	99.7014%	7.5
14	99.6801%	7.5
15	99.6368%	6
16	99.6874%	6
17	99.6184%	9
19	99.6649%	8.5
20	99.7258%	8
21	99.6415%	8
22	99.7379%	8.5
23	99.7211%	9
24	99.7255%	9
25	99.6658%	8.5

26	99.6658%	8
27	99.7420%	9.5
28	99.7384%	9
29	99.6697%	8
30	99.6809%	8
31	99.6766%	8
32	99.6654%	9.5
33	99.6553%	9.5
34	99.6286%	7.5
35	99.7495%	10
36	99.6482%	8
37	99.6703%	10
38	99.6175%	6.5
39	99.6962%	10
40	99.7334%	8
41	99.7301%	10
42	99.6756%	9.5
43	99.6748%	8.5
44	99.6300%	6.5
45	99.6921%	10
46	99.6467%	7.5
47	99.6568%	9.5
48	99.5847%	6
49	99.7292%	9
50	99.7028%	8.5

Table 11 - Raw results for Fig.43

Electrode No.	Diameter (mm)	Measured Height (mm)	Algo Curvature (%)	Coating	Position
1	14	1.82	3.12%	Copper	Concave
2	14	1.42	2.85%	Black	Concave
3	14	2.52	7.73%	Black	Concave
4	16	1.68	1.27%	Copper	Concave
5	14	2.03	3.24%	Black	Concave
6	14	2.95	13.75%	Copper	Convex
7	14	1.36	4.44%	Black	Convex
8	16	2.82	10.73%	Copper	Convex
9	16	1.95	5.67%	Black	Convex
10	16	2.72	10.55%	Copper	Convex
11	14	2.28	7.48%	Black	Convex
12	14	2.71	8.11%	Black	Convex
13	14	2.33	5.58%	Black	Convex
14	16	3.32	13.94%	Copper	Convex
15	14	2.95	12.44%	Black	Convex
16	14	2.56	6.83%	Copper	Concave
17	14	1.61	3.55%	Black	Concave
18	16	1.17	2.02%	Copper	Concave
19	16	1.65	1.57%	Copper	Concave
20	16	2.13	6.72%	Copper	Concave

Table 12 - Raw results for Fig.47 and Fig.48

Electrode	Punch Overlap Category	Curvature	Detected?
1	Slight	No	Yes
2	Slight	No	Yes
3	Slight	No	Yes
4	Slight	No	Yes
5	Slight	No	Yes
6	Slight	Yes	Yes
7	Slight	Yes	Yes
8	Slight	Yes	Yes
9	Slight	Yes	Yes
10	Slight	Yes	Yes
11	Medium	No	Yes
12	Medium	No	Yes
13	Medium	No	Yes
14	Medium	No	Yes
15	Medium	No	Yes
16	Medium	Yes	Yes
17	Medium	Yes	Yes
18	Medium	Yes	Yes
19	Medium	Yes	Yes

20	Medium	Yes	Yes
21	Large	No	Yes
22	Large	No	Yes
23	Large	No	Yes
24	Large	No	Yes
25	Large	No	Yes
26	Large	Yes	Yes
27	Large	Yes	Yes
28	Large	Yes	Yes
29	Large	Yes	Yes
30	Large	Yes	Yes
31	Non	No	No
32	Non	No	No
33	Non	No	No
34	Non	No	No
35	Non	No	No
36	Non	Yes	No
37	Non	Yes	No
38	Non	Yes	No
39	Non	Yes	No
40	Non	Yes	No

Table 13 - Raw results for Table 1

Description	Delta Time	Elapsed Time
Start time	0	0
Get file paths	0.013080015	0.01308
AICellSight object created	7.43771E-05	0.013154
MorphoAnalyser created	0.041661625	0.054816
Absdiff generated	0.002587986	0.057404
HSV filter applied	0.01157836	0.068982772
Gray image created	0.00072443	0.069707202
Image cropped	0.001144791	0.070851992
Blur applied	0.002597208	0.0734492
Threshold applied	0.000414791	0.073863992
Closing applied	0.229787998	0.30365199
Contours found	0.003143187	0.306795176
Bounding circle found	0.006090832	0.312886008
Contour smoothened	0.000369568	0.313255576
Convex hull found	0.000154037	0.313409613
Edge quality determined	0.02155838	0.334967994
Curvature found	8.91876E-05	0.335057181
Punching overlap found	4.55856E-05	0.335102767
Results saved	0.001713648	0.336816415
Output image drawn	0.006940384	0.343756799

Table 14 - Raw results for Fig.73 and Fig.74

Description	Delta Time	Abs Time
Start time	0.000000	0
Got file path	0.024822	0.024822
Created aicellsight object	0.000104	0.024926
Created predictor object	0.000147	0.025072
Image loaded	0.203399	0.228471
Image cropped	0.000198	0.228669
Patches created	0.493908	0.722577
Patchwise predictions made	15.662192	16.38477
Final prediction created	0.007633	16.3924
Output drawn	1.807785	18.20019

Table 15 - Raw results for Fig.75 and Fig.76