School of Physics and Astronomy

Senior Honours Project

A Machine Learning Approach to Characterisation of Bacterial Flocculation in Wastewater Treatment

Diego M. Fernández
9 April 2021

Abstract

Over the course of the last decade, machine learning (ML) has proven an exceptional method in labelling and classifying a broad spectrum of images. We apply a novel ML approach to the characterisation of images of bacterial aggregation in wastewater treatment (WWT) via the development of a convolutional neural network (CNN). WWT is one of the most important biotechnological processes in the world. This phase of WWT, however, uses outdated biological and microscopic quality assessment methods, which have been challenged by the complex nature of the physics and biology of the system. In this project, we develop an effective prototype model that by far exceeds our pre-set specifications of 60% accuracy for proof of concept by consistently delivering >70% test accuracy. The success of this experiment will further our understanding of the science behind bacterial aggregation and may prove revolutionary in modernising global WWT process control.

Declaration

I declare that this project and report is my own work.

Signature:  Date: 9 April 2021

Supervisor: Prof. Cait MacPhee 10 Weeks

Contents

Acknowledgements

The author would like to thank advisors Prof. Cait MacPhee, Dr. Gavin Melaugh, and Dr. Ryan Morris of the Soft Condensed Matter and Biological Physics Groups in the Institute of Condensed Matter and Complex Systems at the University of Edinburgh for their support in this Senior Honours Project. We thank Paul Banfield, technical manager at Veolia, for his provision of images from wastewater treatment sites across Scotland, without which the project could not have been completed. Finally, we would like to thank the National Biofilms Innovation Centre for their funding and the Biotechnology and Biological Sciences Research Council for their grant.

1 Introduction

Wastewater treatment (WWT) is an essential biotechnological process required in ensuring the long-term conservation of usable water and in maintaining the health of the environment. This process is critical in preventing disease, protecting ecosystems and wildlife - both on land and in bodies of water – and generally reducing waste [1]. WWT plants have been around for over a century [2], and thus have developed multi-step, lengthy and complex procedures aiding in the release of cleaner effluent, the technical term for treated wastewater [3]. This project focuses on the biological phase of the process: the activated sludge process. In short, this phase consists of using certain bacteria to form flocs, clusters of organic polymers attached to bacteria [3], to settle the solid materials in wastewater. This flocculation process has the positive effect of gathering waste in the sewage and aggregating in such a way that it sinks to the bottom of the tank without requiring the expenditure of excessive energy. Though this interaction has been well-studied, unpredictable flocculation from the bacteria can prove costly to the efficiency of the system, suggesting a need for a modern approach to its characterisation. We aim to modernise this process, whilst bettering our understanding of its mechanisms via a novel machine learning (ML) approach written in Python.

The objective of this project is to provide proof of concept for the automation of the aforementioned characterisation process using a convolutional neural network (CNN) on microscopic images taken at WWT plants. The images involved in CNN training are courtesy of Paul Banfield at Veolia, an international company that works with WWT [4]. A CNN is a deep learning method used for image classification and analysis. Using this 'black box' approach, we are able to confront one of the big challenges posed by visual characterisation of bacterial aggregation: not knowing enough about the identifying features in the flocculation process. Automation of this kind also removes the need for out-dated, frequent, laborious and error-prone human inspection and allows for instantaneous results that inform the plant on its performance. The immediacy and effectiveness of virtual characterisation would enable rapid treatment corrections that maximise the amount of bacterial biomass in the flocs [3].

The use of image analysis on the biological structures found in wastewater provides a unique opportunity for a physics-based approach to the classification problem at hand. Here is where the science of soft matter colloidal physics enters the frame. The size of the bacteria and organic matter in the water is larger than atoms, but smaller than grains [5] in the range of 0.5 to 2 μm [3]; these floating substances are the focus of study for many researchers in soft matter physics. Rod-like bacteria and exopolysaccharides - sticky polymers produced by the bacteria, found outside the cells [6] - are the primary constituents of the substance and can arrange in numerous complex ways [7]. Our development of a successful CNN will enable further exploration of the physics and biology involved in the bacterial flocculation process.

With quantitative confidence we can say that the objective of creating a prototype, proof of concept bacterial aggregation characterisation CNN has been met. In this report we focus on the computational side of the project, discussing the development and effectiveness of the constructed CNN. We additionally consider how this tool can be refined and what steps can be taken to improve upon this work in the future. This novel combination of the emerging fields of soft matter physics and ML could be revolutionary

in the advancement of wastewater treatment on a global scale.

2 Background and Theory

In this section we will briefly touch upon the WWT process, the background biology and some simple soft matter physics concepts. The architecture and fundamentals of CNNs will also be explained in greater depth.

2.1 An Overview of the Wastewater Treatment Process

The WWT process typically consists of three phases, as depicted in Figure 1. The primary treatment involves the filtering of water through a screen, the removal of small solids in a grit chamber and the settlement of remaining solid matter in sedimentation tanks [8]. Secondary treatment typically consists of a biological phase in aeration tanks where solids stick to bacteria and settle; this is where the project is focused and will be further discussed in Section 2.2 [8]. The tertiary treatment involves the removal of nutrients, such as nitrogen and phosphates, as well as any remaining material, dependent on the location of the plant [3].

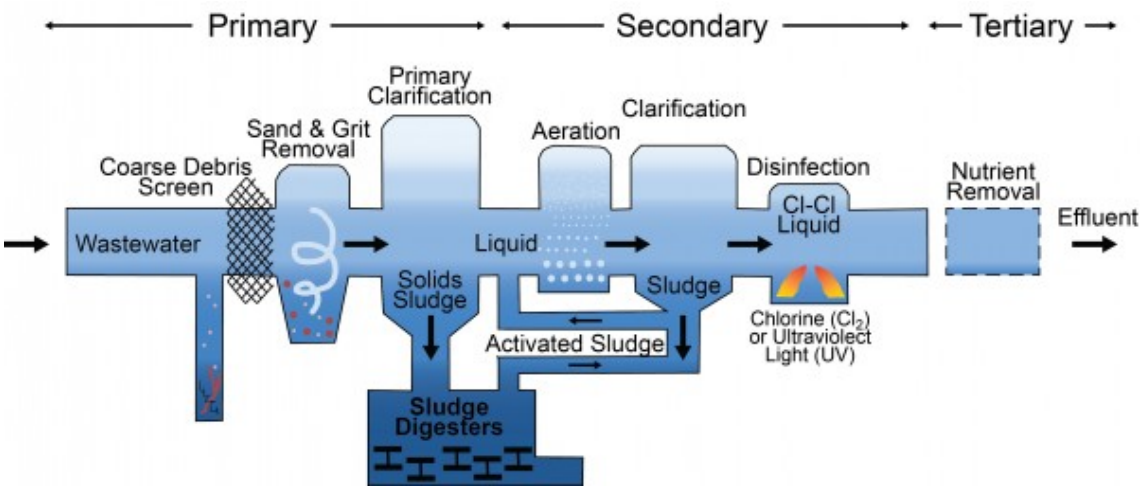


Figure 1: The wastewater treatment process in 3 phases [9].

2.2 The Secondary Phase: Flocculation and Biology

The secondary, or activated sludge (AS), phase in WWT consists primarily of aeration and clarification via sludge settlement [10]. AS is the term used to describe the aerated sewage containing bacteria, protozoa and other microbes in an aeration tank [9]. In these tanks, oxygen is provided to the system to increase respiration in bacteria, ideally maintaining a high proportion of flocs of size 10μm to 1mm [3].

WWT sites face a challenge in maintaining the right proportion of bacteria in AS. In [3], Dr. Davies notes that in the aeration tanks, "Settlement is affected ... by nutritional imbalance, changes to the microbial components of the activated sludge, and the presence

of toxic chemicals in the mixed liquor." In addition, filamentous bacteria, rod-like bacteria extending outward from a floc, are essential in determining whether a floc sinks or floats. If there are an excessive number of long filaments, the floc may take an unreasonable amount of time to settle [3].

Methodology in ascertaining the quality of flocculation in this phase of WWT is outdated, inexact, and inefficient. The overall settleability in aeration tanks is characterised by the Sludge Volume Index (SVI) or the Sludge Specific Volume Index (SSVI) [3]. The SVI is determined by collecting samples of the sludge in a 1 litre container, allowing it to settle for 30 minutes, and then inserting the resulting amount of sludge settled, as well as the concentration of suspended solids into Equation 1 [11]:

$$\text{SVI } (\frac{mL}{g}) = \frac{\text{Settled Sludge Volume } (\frac{mL}{L})}{\text{Suspended Solids Concentration } (\frac{mg}{L})} (\frac{1000mg}{g}) \tag{1}$$

The 'Suspended Solids Concentration' term comes from the Mixed Liquor Suspended Solids (MLSS) value, measured in tests [3]. To achieve this value, the floating solids are removed from the sample with filters. These solids are then dried, and their weight is used in the MLSS value for the equation [3]. In this project, however, the SSVI value will be used to classify the images. The SSVI is similar to the SVI, but requires a special cylinder, seen in Figure 2, in which the mixture is consistently stirred.

Figure 2: SSVI test cylinders [12].

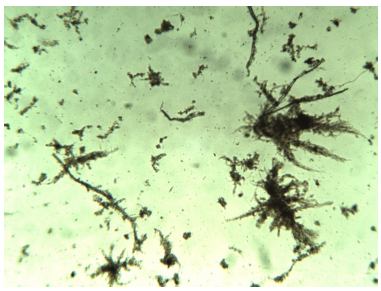
Process control corrections are dependent on knowledge of current SSVI levels. Ideal levels can be challenging to maintain given the amount of time and resources that must be spent on taking measurements [11]. SSVI values in the real-world data provided by Veolia come from WWT sites in Allanfearn, Lossiemouth and Fort William (see Figure 3 for locations).

These values are ideal in the range of 60 to 120 mL/g. An example of an ideal SSVI value image is depicted in Figure 4b. Effluent in this range is fairly clear, with most of the bacteria held in porous flocs capable of withstanding stress due to aeration; the AS also effectively traps matter in the settling process [11, 3]. A low SSVI value, depicted in Figure 4a, means the sludge is dense and settles too quickly, not gathering enough matter on the way down, which often occurs as a result of old, over-oxygenated sludge [11]. Finally, a high SSVI value, as seen in Figure 4c, means the sludge settles too slowly due to its low density, which is typically attributed to young bacteria leaving behind floc

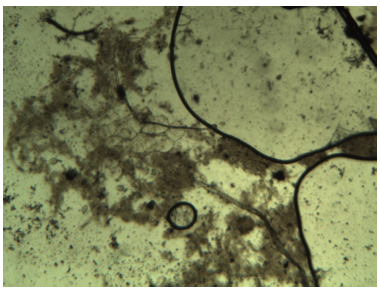
particles [11]. Clearly, bacteria stick together in different, inconsistent ways, and this can be challenging to explain, suggesting the need for an automated 'black box' approach. One of the overall aims in developing this ML algorithm for SSVI characterisation is to aid in rapidly making adjustments to process control that can save time, cost and energy.

Figure 3: The three locations from which the image data was collected [13].

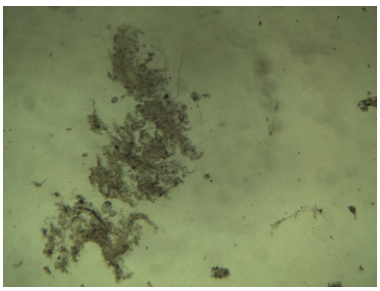
It is important to note that these samples of AS are merely a small portion of the system but are used to characterise it as a whole. In a similar fashion, the images used to train the CNN are only small samples of the system and are not taken within the SSVI measurement cylinders. We treat the aeration tanks as homogeneous and assume that the SSVI and training images represent the tank as a whole. The data from this project will be further discussed in Section 3 of the report.



(a) Low SSVI 57.9 mL/g



(b) Good SSVI 110.4 mL/g



(c) High SSVI 210.3 mL/g

Figure 4: Images from the Allanfearn WWT site depicting SSVI values for the three categories: (a) low, (b) good and (c) high.

Over the last 20 years, there has been an increase in characterisation of AS via microscopy [14]. In [15], Sezgin, Jenkins and Palm determine the existence of a relationship between filament length, the length of rod-shaped chains of bacterial cells extending out of the floc, and floc size with the settleability of the AS. The authors to this paper confirm that the process is exceptionally laborious and time-consuming. This is where the project could prove impactful in enhancing the efficiency of bacterial flocculation characterisation.

2.3 The Link to Soft Matter Physics

Soft matter colloidal physics can be defined as the physics of a system in which insoluble solids interacting with liquids are smaller than grains, but larger than the surrounding molecules [5]. These substances, such as ice cream, paint or biofilms [5], have viscoelastic properties [16]. Biofilms are communities of bacteria on the surfaces of liquids and are in the realm of soft matter physics [17]. The flocs in aeration tanks are biofilm-like, but instead of being on the surface, they are free floating. Here, the sticky polymers are produced by bacteria [7, 18, 19]. As the image in Figure 4 shows, we are looking at interactions between bacteria and polymers on a scale that is between microscopic and macroscopic. In the AS phase, as bacteria grow, they stick to surrounding polymers and each other in complex, non-equilbrium ways [7]. This makes it challenging to comprehend and classify using mathematics and models. These obstacles are the primary reason for our use of CNNs; they can characterise the images without needing to learn the background physics or biology.

2.4 Convolutional Neural Networks

Machine learning (ML) is a primarily statistics-based computational information processing method. This method can continuously and autonomously learn from data to produce models that aim to label and classify unseen data; in our case, the data consists of images [20]. Within this broad category of artificial intelligence, we focus on the subcategory of deep learning. Deep learning consists of layered functions in an artificial neural network (ANN); an ANN is a model designed to emulate the interconnection of neurons firing in the brain [20].

In this project, we decided to use ML algorithms because of their ability to learn about a system, or in this case an image, without needing any prior information [20]. As previously mentioned, current characterisation methods of the flocculation process are not as effective as they could be, and we intend to improve this using a CNN model.

2.4.1 What is a CNN?

A CNN is a deep learning ANN, using between 10^2 and 10^{10} parameters, which is the term for weights and biases of an algorithm [20, 21], to train a network on how to classify images. In short, the image data is sequentially passed through convolutional layers, which alter the data, and fully connected layers, which classify the data, in order to learn these parameters [22]. A CNN also contains layers that specifically aim to reduce the size of the image array, making this algorithm excellent at rapid image classification, once it has learned ideal parameters [23].

This sequential CNN algorithm is a type of supervised learning, in which we train the CNN on a set of labelled data so that when it learns, it can confirm whether it is learning correctly, or incorrectly [24]. In our case, the CNN will take an image, run it through a series of convolutional layers that hierarchically extract features, pass it through a fully connected, ANN-based layer, and then return a class or class probability [23]. The input in the case of a 2D colour image will be in the form of an array of pixels with a given width and height, and a set of red, green and blue (RGB) values as numbers.

A CNN's structure consists of three phases: convolutional layers, pooling layers, and fully connected layers. There can, and typically are, multiple of each of these layers [22]. The convolutional layer consists of passing the initial image through a series of layered filters, or kernels, which return a varied, same size array [22]. This is often done by subtracting the mean RGB value of the pixels from each point on the 2D grid [22]. The next phase, pooling, partitions this array in into smaller receptive fields (squares of a certain dimension within the array of pixels) whose values are determined via max pooling or average pooling [25]. Max pooling takes the largest value of each receptive field and assigns it that value; this essentially labels that region according to its most important feature [25, 23]. Average pooling follows in suit by taking the average of the field's features and assigning it that value [25, 23]. A fully connected layer is, on its own, an ANN and is the final step in reaching the output classification. Here, the array is 'flattened' into a 1D array, or vector, of weights and biases. The data goes through another set of layers - input, hidden and output – each of which seeks to alter the weights of the data by projecting it onto a series of neurons, the 'black-box' pieces of the algorithm [25]. An example CNN can be found in Figure 5.

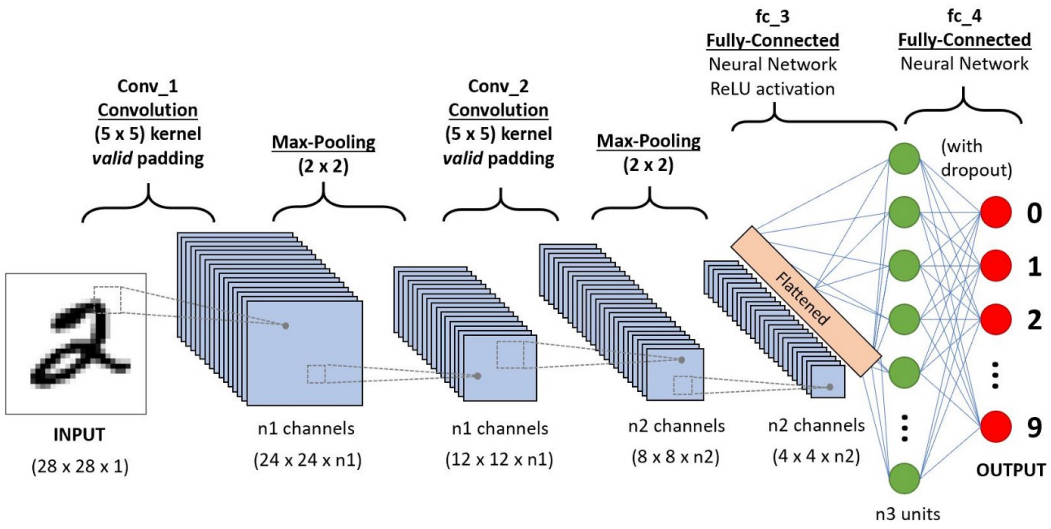


Figure 5: Example of a CNN with the three primary layers: convolutional, pooling, and fully connected [23]. As can be seen, the first convolutional layer is followed by a pooling layer, which essentially decreases the area of the image. This is followed by another convolutional layer and another pooling layer to further reduce the image. It is finally flattened and put into the fully connected layer before being classified via the softmax function. Not depicted in this image is the softmax activation function, which takes the output of the final fully connected layer to characterise an image.

Once an output 1D vector is achieved, an image is run through an activation function, which determines the output class [26]. In this project, we use the softmax classification function for our final output [27]. We can see this function in Equation 2; it returns the probability of an image j, given a set of weights $\Theta^{(i)}$, being in a class, y (e.g. 'low' = 0, 'good' = 1 or 'high' = 2 SSVI):

$$P(y = j|\Theta^{(i)}) = \frac{e^{\Theta^{(i)}}}{\Sigma_{k=0}^{l} e^{\Theta_k^{(i)}}} \tag{2}$$

Here, l is the number of possible classes. The numerator is a value of the exponential of the 1D vector for class j, and the denominator acts as the normalisation value summing all the classes and parameters j could take on [27]. The objective of this CNN is to classify the data rather than return probabilities of a given class, so the highest probability is used to characterise an image [27].

Here we develop two CNNs, both of which use the softmax function for the final output. Another activation function, called the Rectified Linear Unit (ReLU) activation function, is applied to grid points at the end of convolutional layers. This function returns the maximum of 0 and the values from each cell of a convolutional layer [26]. For example, once the mean RGB values are subtracted from each pixel, negative pixels passing through ReLU would become 0, and positive ones would remain the same [26].

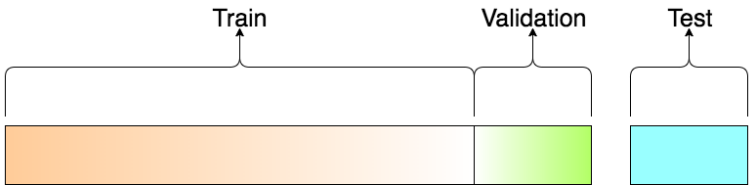


Figure 6: A visual representation of how the train, validation and test data sets are usually split [28]. These are typically split with a ratio of 80/10/10, respectively.

This process of developing the algorithm begins by splitting the data into three sets: a training set for learning, a validation set for tuning of the learned parameters and a test set in which the accuracy of the model is determined. Typically, the data is split with 80% train data, 10% validation data and 10% test data [28], as depicted in Figure 6.

The final important point that must be addressed when discussing CNNs is the error rate. We define error via the use of a loss function. In our CNN, we use the categorical cross entropy (CCE) loss function. CCE takes the negative log of the softmax function [29]. This can be seen in Equation 3, where $f_i(x)$ represents the softmax function for class i for all the target classes.

$$CE = -log(f_i(x)) \tag{3}$$

A batch is a portion of the train and validation images inserted into the algorithm, typically consisting of less than 5% of the images. Running all the batches through the CNN is defined by the term 'epoch' [30] . The number of epochs, iterations through all the layers in this sequential CNN algorithm, is critical in determining accuracy and loss. Typically, the more epochs a CNN runs through, the lower the loss. There is a high

risk of over-fitting the data if the number of epochs is too large; this can be figured out by looking at the training loss over time and finding the point at which it is no longer decreasing. Over-fitting the data would mean that the final model classifies the training data well, but typically not the test data, and vice versa for under-fitting [30].

It is important to take into account the loss function when selecting the learning rate of a CNN. The learning rate is a hyper-parameter, a parameter that dictates some of the structure of the CNN, that decides how much the weights of the model should change according to the loss at each epoch [31]. CNN's typically use stochastic gradient descent to define the direction in which the weight should shift at each epoch [31]. The learning rate is essentially the step size at each epoch; this determines how quickly the model learns. A small learning rate leads to slow training of the weights, while a large learning rate risks skipping over important features. This can be visualised by thinking about a polynomial in which steps are taken in a certain direction with the goal of finding a minimum. Too large of a step means the minimum could be skipped, while too small of a step means it will take too long to reach the minimum [31]. In Keras, the learning rate is a value between 0 and 1, with the typical optimal value lying at 0.0001 [21, 32].

2.4.2 Selecting an Appropriate API

The Python application programming interface (API) that is used in this project is Keras. When selecting the best neural network API, we took many features into consideration: user friendliness, performance, speed, complexity, debugging capability, readability, and ability to handle large data sets. As the goal here is a proof-of-concept prototype, the top priority is finding the least complex API. During the research phase, we considered using either Keras, PyTorch or TensorFlow. Of the three, Keras is the highest-level option, allowing for minimal direct work with data arrays [33]. Keras includes all the functionality of TensorFlow [34] with the downside of being the slowest of these APIs. The datasets being used for training, however, are not particularly large, and this should not override the need for a straightforward interface. PyTorch is faster and has more effective debugging tools, but this is outweighed by Keras' user friendliness and the project's lack of need for debugging. It is clear that PyTorch and TensorFlow have a better performance and speed than Keras; this means that if a prototype with Keras were to be successful, it might be more effective to move to another API.

3 Method

In this section, we will discuss the development of the project's two CNNs: a manually made sequential CNN and a modified sequential VGG16 CNN. We also dive into the topics of data preparation and methods of performance analysis.

3.1 Data Preparation

One of the most important parts of developing a CNN is ensuring the data is prepared in an effective fashion. This step in the process includes naming files correctly, placing them in appropriate directories and formatting images to match the specifications of the CNN.

As mentioned previously, the data provided by Veolia came from three Scottish WWT sites: Allanfearn, Fort William and Lossiemouth. This data included 1334 images spanning a period of 12 months from January 2020 to January 2021. Not all days of the year were included; many days in the summer did not have images. Temperature plays a role in the flocculation process [3], so it is important to note that the warmer times of year have a decreased representation in the data. This could mean that when the resulting CNN is applied to new summertime images, the accuracy might be lower than expected. Alongside the images, Veolia provided SSVI values for each day of 2020 at each site. The distribution of images per site can be seen here in Table 1:

Site	Number of Images
Allanfearn	780
Fort William	398
Lossiemouth	156

Table 1: Distribution of Images for the 3 WWT Sites

Among these files were duplicates, corrupted files, and improperly formatted images. The first step of the project required scanning through the files with a Python script and removing the duplicates. Next, we manually looked through the folders to check if there were any corrupted files, which were easy to identify because of the difference in the icons representing them. Finally, the files from 2021 had to be removed because we did not have SSVI values for those dates.

In the project, we decided to use bitmap (BMP), or '.bmp', files instead of joint photographic experts group (JPG), or '.jpg', files. JPG files are compressed in such a way that they lose more information than in the compression of BMP files [35], so we opted for the image type that contains more information. The images from Lossiemouth and Allanfearn were already BMP files, but the Fort William files were JPGs, so we wrote a Python script that converted these to BMP files. This meant that Fort William files contained less pixel data than the other two sites; this was taken into account when running the CNNs by reducing the number of pixels inserted into the algorithm. Files of pixel dimensions 1024x768 were reduced to dimensions of 224x224 using a Keras class called ImageDataGenerator [21]. In short, a Python class can be defined as a Python object that holds data and functions within [36]. This change in image dimension shortens the training time of the algorithm. The use of a square array also makes it more straightforward to pool the data. Once this portion of data preparation had been completed, roughly 8.3% of the images had been filtered out, leaving 1223 files, or 11.33 GB of information. Later in the project this would prove costly since more data would have meant improved training of the CNN.

One of the remaining key challenges of data preparation was ensuring that file names were consistent and represented all the data from each site. We wanted to include 4 pieces of information in the file names: SSVI classification, which is split into 'high', 'good', and 'low' categories, the date, the SSVI value, and the WWT site. The files, however, only included the date and a number labelling whether it was the 1st, 2nd, 3rd or 4th image taken at that location that day (though there were typically less than three images per day). The inconsistent formats of the file names, including difference in spacing, usage

of punctuation (e.g. using '.' instead of '_'), inclusion of additional labels, and date order made the re-labelling process a time-consuming challenge. The use of Python's 'os' module [37] made the file renaming possible, and the use of Python's 'pandas' module [38] allowed for simple extraction of data from the file containing SSVI values. The development of lengthy Python scripts eventually led to relabelled files, which were all placed in the same folder. Tips for renaming the files, accessing '.csv' files, and developing scripts can be found in Appendix E.

3.2 Building a CNN

In this project, we developed 2 CNNs based on existing models. The first CNN was a simple manually made sequential model, based on a series of tutorials found on 'deeplizard' [39]. The second model was based on a well-developed image analysis model that goes by the name of VGG16; 'deeplizard' tutorials were also used to develop this model [32]. We will discuss these in their own respective subsections.

Due to a lack of images with a 'low' SSVI value, the initial CNN development began with binary class classification. Once this model was working, we extended it to 3-class classification, though the results were not comparable due to the lack of training data, as is clear in Table 2. In this section of the report, we will only describe how the 2-class models were created. The 3-class models are created by changing an argument in the final layer of each model so that it returns 3 classes as opposed to 2.

Classification	Number of Images
High	399
Good	668
Low	156

Table 2: Distribution of Images for the 3 SSVI Characterisations

3.2.1 Image Formatting and File Distribution

Though we have already discussed the preparation of data, we must consider two final steps prior to developing the CNN. The first of these is splitting the data into train, validation, and test sets. In this step, we create three directories labelled 'train', 'validate', and 'test' each of which contains folders for 'high' and 'good' for binary classification or 'high', 'good', and 'low' for three class classification. The files were relocated using the Python 'glob' module [37], which can search for file names containing a provided Python string (e.g. 'high*' was the input for files with high SSVIs).

The files were then converted to a Keras-readable type of array via the ImageDataGenerator class [21] and linked to variables for training, validating and testing. The ImageDataGenerator class reduces the image size and then subtracts the mean RGB values from each cell in the array. 5 sample images can be seen in Figure 7. The splits of data for binary classification were: 300 per class for training, 60 per class for validating, and 36 per class for testing; for three-class classification: 100 per class for training, 35 per class for validating and 20 per class for testing. These splits were selected to maximise the number of images and to roughly approach the recommended 80/10/10 split [28].

As is apparent, there is not enough data for the multi-class CNN to be very effective; 100 training images will likely lead to statistical anomalies. These limitations shifted the primary focus of the project to binary classification, with the objective of achieving greater than 60% average test accuracy to establish proof of concept. If the project ran longer than 10 weeks, we could have asked Veolia for data from other sites in order to expand the data set and test 3-class classification.

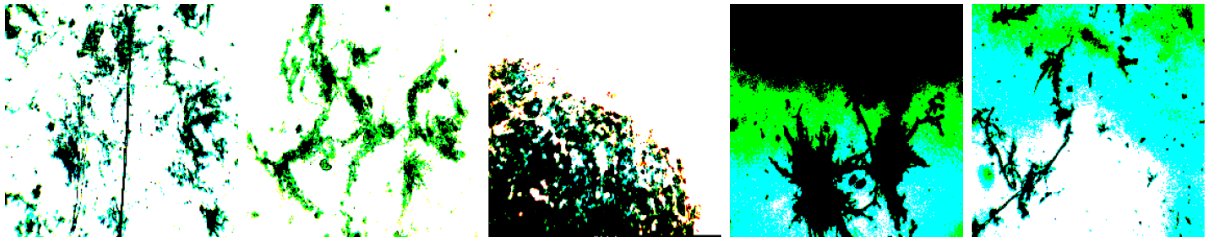


Figure 7: Example images from the ImageDataGenerator class [21]. From left to right the images are classified as 'high', 'good', 'good', 'low' and 'low'. The large amount of white space comes from the subtraction of mean RGB values at each point.

3.2.2 Building the Manually Made Sequential CNN

The first of the two CNNs consists of 293,954 trainable parameters across 6 layers. The model is built using Keras' 'Sequential()' class, which takes all the CNN layers as arguments. They are placed in order and are separated by commas. The 6 layers consists of 3 consecutive sets of 'Conv2D()', or 2D convolutional layers followed by 'MaxPooling2D', or 2D maximum pooling layers. Each layer learns a set of parameters. As mentioned in the background section, the resulting data must be flattened to a 1D array using 'Flatten()', before the 'Dense()' layer uses the softmax function for final classification. The code for this simple model can be seen in Listing 1 with origins based in a 'deeplizard' model from [39].

Listing 1: Manually Made Sequential CNN Code

```
simple_model = Sequential([
    Conv2D(filters = 32, kernel_size = (3,3), activation = 'relu', padding = 'same',
           input_shape = (im_size[0], im_size[1], 3)),

    MaxPooling2D(pool_size = (2,2), strides = 2),

    Conv2D(filters = 64, kernel_size = (3,3), activation = 'relu', padding = 'same'),

    MaxPooling2D(pool_size = (2,2), strides = 2),

    Conv2D(filters = 128, kernel_size = (3,3), activation = 'relu', padding = 'same'),

    MaxPooling2D(pool_size = (2,2), strides = 2),

    Flatten(),
    Dense(units = 2, activation = 'softmax')
])
```

The arguments (inputs) for the convolutional and pooling layers are using mostly standard values as recommended on Keras' documentation [21]. The meaning behind each these arguments has been commented on in the background section of the report.

In the convolutional layers, the 'filters' argument typically increases with the layer number and is composed of a matrix with the provided number of output square filters that are applied to the array [40]. The 'kernel_size' argument of (3, 3) is typically used for 2D RGB images and is the size of the window in which the convolution filters are applied [41]. We use small kernel sizes in our CNNs because we want to identify small (i.e. floc-sized) features of each image. The 'activation' argument is set to 'relu', a function we also discussed in the background section of this report. The 'padding' argument is used to assert that the output from the layer will be in the same size and shape as the input. Finally, the 'input_shape' argument determines the shape of the array, in this case including the width, height and number of colours in the tuple. In the pooling layers, the 'pool_size' argument defines the size of the receptive fields and the 'strides' argument is an integer for how far the receptive field moves for each step. We have selected small values for these arguments to maximise the amount of learning [21, 31].

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 224, 224, 32)      896
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 112, 112, 64)      18496
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 56, 56, 64)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 56, 56, 128)       73856
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 28, 28, 128)       0
_________________________________________________________________
flatten_5 (Flatten)          (None, 100352)            0
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 200706
=================================================================
Total params: 293,954
Trainable params: 293,954
Non-trainable params: 0
_________________________________________________________________
```

Figure 8: Keras summary of the manually made model [21].

Using Keras' summary function, we can see the architecture of the model, including information about the number of parameters per layer and the shape of the array at each layer in Figure 8. As is expected, the dimensions of the image are reduced, but more parameters are learned in each convolutional layer.

3.2.3 Building the VGG16 Sequential CNN

Listing 2: The Code Building the VGG16 Sequential CNN

```
v_model = vgg16.VGG16()
vgg16_model = Sequential()

for layer in v_model.layers[:-1]:
    vgg16_model.add(layer)
vgg16_model.add(Dense(units = 2, activation = 'softmax'))
```

The building process of the VGG16 CNN is similar to that of the manually made model. First, the model is loaded into a Python variable 'v_model' using the 'vgg16.VGG16()' class. Then, a sequential model is created and saved to a different variable 'vgg16_model'. The VGG16 model layers are then inserted into the sequential model using a 'for' loop, but excluding the final 'Dense()' layer. That final 'Dense()' layer is replaced by another 'Dense()' layer that sets the classification to 2 class (by changing the 'units' argument to 2) rather than keeping the original 4096 classes of the VGG16 model [21]. This simple code can be seen in Listing 2. A layout of the architecture of the VGG16 CNN model, prior to changing the final 'Dense()' layer, can be seen Figure 9.

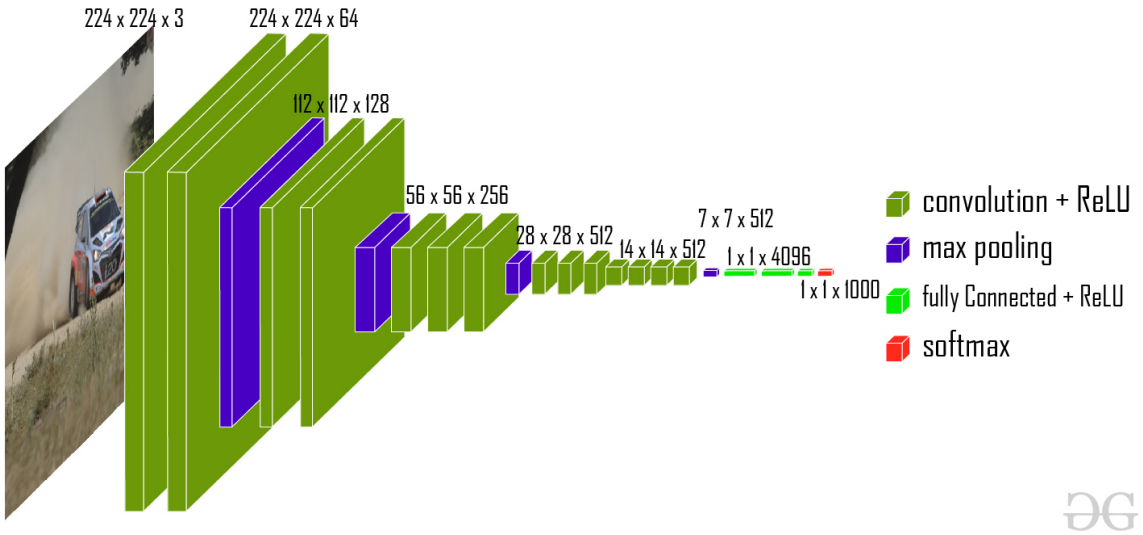


Figure 9: The architecture of the VGG16 model depicting the 16 layers [21].

3.3 Training, Predicting, Testing, and Analysing Sequential Models

The training of both the manual and VGG16 models is nearly identical, so we will only explain the VGG16 model here. The code for the VGG16 model can be found in Listing 5 of Appendix A. The training of the model is done with two lines of code, as seen in Listing 3. The first line compiles the code. In the compiling process, we first establish a learning rate. We have selected 0.0001 as was commented in the background section. The selected learning rate is inserted into an algorithm that optimises the model; we

have selected the Adam algorithm, but there are many other choices in the Keras library which we did not have time to explore [21]. We selected the 'categorical_crossentropy' loss function, and the 'accuracy' metric to analyse the training and validation processes. Accuracy and loss are the defining characteristics in our analysis of the effectiveness of the CNN.

The next step is fitting the model. This is where the actual training process occurs, and running this function begins the iterations through each epoch. The second line of code in Listing 3 ensures the use of 'training_batches' and 'validate_batches' as the labelled input training and validation data, respectively. The epochs parameter is then established, and the verbose argument is set to 2, allowing us to view the accuracy and loss at the end of each epoch [21]. Depending on the number of trainable parameters, 1 epoch can take between 200 and 500 seconds, making a 15-epoch binary classification training session last upwards of 2 hours on a standard quad-core laptop.

Listing 3: The Code Training the VGG16 Sequential CNN

```
vgg16_model.compile(optimizer = Adam(learning_rate = 0.0001), loss = 'categorical_crossentropy',
    ↪ metrics = ['accuracy'])

vgg16_model.fit(x = train_batches, validation_data = validate_batches, epochs = 15, verbose = 2)

vgg16_model_predictions = vgg16_model.predict(x = test_batches, verbose = 0)
```

Predictions were then made using the '.predict()' function on the testing data and plots of accuracy were produced. The accuracy plots are based on confusion matrices, which are explained in the next subsection.

3.4 Objective of the Method

With the CNN developed, we now want to optimise the algorithm by fine-tuning hyper-parameters. Due to the limitation of time and the length of an individual training session, we could only tweak two hyper-parameters during this project: the number of epochs and the splits of data. In the project, we trained the VGG16 model 4 times using 5, 10, 15 and 20 epochs. Early testing indicated that any number higher than 20 would likely lead to over-fitting of the model, rendering it inadequate on unseen data. We took all the images and placed them in three separate, identical folders. Each folder was randomly split into train, validation and test sets of the sizes mentioned previously. Comparing the runs on 4 different epochs of three randomised data splits reduces the risk of a favourable or unfavourable chance occurrence. The weights of the models are then saved so they can be tested on additional real-world data in the future.

Once run, predictions were made, and a confusion matrix of the results was plotted using a modified Python 2 function. A confusion matrix plots the correct, or 'true', labels against the predicted labels of each image [32]. The accuracy can be determined via these confusion matrices. Overall analysis was performed by plotting the means of the accuracy for the 3 models against the number of epochs. We then took the range between the two highest means as the ideal range of number of epochs that should be used. We would have increased the number of splits trained, but more than 20 hours of computer time were already spent on these runs. The accuracy of 'good' and 'high' classifications were also taken individually. We note that on average, the models classified 'good' SSVI images

more effectively than 'high' SSVI images. This test was not applied to the 3-class model, but simple runs on 10 epochs indicated that the manually made model classified the data better than a random guess. We will further explore this, alongside comparisons between binary and multi-class models in the discussion.

One final piece of analysis that was performed was determining whether the model is over-fit or under-fit. Plotting the accuracy and loss of training and validation sets over a period of 35 epochs helped determine the ideal number of epochs. We use this metric because we know that at the point where training loss stops decreasing, we have over-fit the data. Training accuracy reaching 100% also indicates an over-fit because it cannot easily be applied to unseen data. We also explore the validation loss because fluctuations can mean the batch size is incorrect [42].

4 Results & Discussion

The objective of creating a prototype, proof of concept CNN that classifies images of bacterial aggregation in WWT has been met. The results over the course of the project have overwhelmingly shown that the developed CNNs work better than a random classification guess. This was true on average and in every model and set of hyper-parameters used. This applies for both binary and 3 class classification for the manually made model and the VGG16 model. The success of the project indicates that the CNN development method was practical and effective. Although proof of concept is hereby established, further developments can be attempted by fine-tuning the hyper-parameters of the model. In this discussion we consider the performance of the CNNs, the assumptions made in the process, and what steps can be taken for the fine-tuning and improvement of the model in the future.

4.1 Assumptions and Targets

4.1.1 Assumptions

We assume that the images and SSVI values provided by Veolia are representative of each other and the aeration tanks themselves, despite being only minor portions of the generally homogeneous system. In analysis, however, this information must be considered a natural limitation to the effectiveness of the CNNs. The labels (SSVI values) attached to each image may not be entirely 'correct', and thus the model should statistically be unable to reach maximum accuracy. The level of accuracy reduction due to the uncertainty in SSVI representation of an image is unknown, and at present cannot be determined. Hence, final testing accuracy will likely not approach 100%.

We make two other major assumptions. Firstly, we have reasoned that the VGG16 model will likely perform better than the manual model. This assumption is made so that we can focus on thoroughly testing one model; this is because there is a limit to the amount of available computational time over the short period of this project. Secondly, we have assumed that the test images in each data split will be representative of any new and unseen images fed to the model. These assumptions are acceptable because the goal of the project was proof of concept and does not encompass larger-scale CNN development requirements (e.g. fine-tuning hyper-parameters). If we were to aim for an

ideal 97% accuracy model, we would need more training data, greater confidence in the training data, and potentially a more effective API, such as PyTorch or TensorFlow.

4.1.2 Targets

If the binary CNN models classify the test images with greater than 60% accuracy on average, we will treat the project as a success (i.e. we will have developed a prototype, proof of concept model). The choice of 60% rules out statistical anomalies, especially given that we are selecting a mean value of accuracy as opposed to the best value. We additionally aim to show that at least some 3 class models perform with greater than 43% accuracy with the same reasoning of the 10% buffer in relation to statistical anomalies.

4.2 Analysis of CNN Performance

The objective of producing VGG16 models with greater than 60% accuracy was overwhelmingly surpassed, with an average value of 72.41% test accuracy when combining all 12 assessed runs. The lowest test accuracy among these runs was 64%, stipulating that we can confidently establish proof of concept. The accuracies and mean for the 12 runs are depicted in Figure 10. The mean of this plot indicates the ideal range of epochs is between 10 and 15. At 10 epochs, the mean value was 74% and at 15 epochs the mean value was 74.33%, both of which were higher than the 70% at 5 epochs and 71.33% at 20 epochs. This is likely due to under-fitting of the model at 5 epochs and over-fitting at 20. As mentioned in the background section of the report, too few epochs means the model has not had enough time to train, implying under-fitting. Too many epochs, on the other hand, indicates that the model is only ideal for the training set, which reduces test accuracy, implying over-fitting of the model. We do not include error bars in this figure because the plot already contains the data from each split. The accuracy data, including 'total', 'high', and 'good' accuracy, can be found in Appendix C.

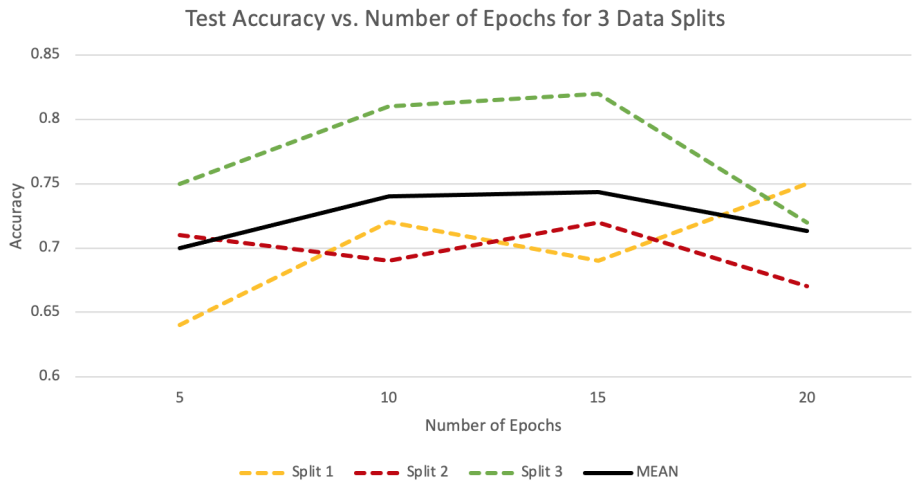


Figure 10: The accuracies for the 3 different data splits at epochs 5, 10, 15 and 20. The mean of the three models is included in black. The numeric values can be found in Appendix C.

Furthering our analysis, we aim to determine an ideal, single number of epochs to use for this model. The training accuracy in the 15 epoch models, tended to get too close to 100%, whereas an ideal value would be around 95-98%. In the 10 epoch models, the training accuracy was in the 90-93% range, meaning it could have been trained for longer. Judging by these values and a plot of training loss and accuracy over the span of 35 epochs on Figure 11, a 13-epoch training run is deemed the ideal value. Epoch 13 of the 35-epoch run is the final step in which the accuracy is in the ideal range, and in which the training loss is still characteristically decreasing. The 35-epoch run did produce a test accuracy of 83%, but we have excluded this value from our results because the model was clearly over-fit at 100% train accuracy and 0 train loss by the end of the run. Per these metrics, any VGG16 training sessions that run under 10 epochs are likely under-fit, and over 15 epochs are likely over-fit. In future development the value of 13 may be ideal in fine-tuning other hyper-parameters of the model. As mentioned previously, the same tests were not carried out with the manually made model in part due to lack of accuracy in preliminary trials. We knew in advance that the VGG16 model was already a proven and effective classifier and due to time limitations, we selected only this model for our testing.

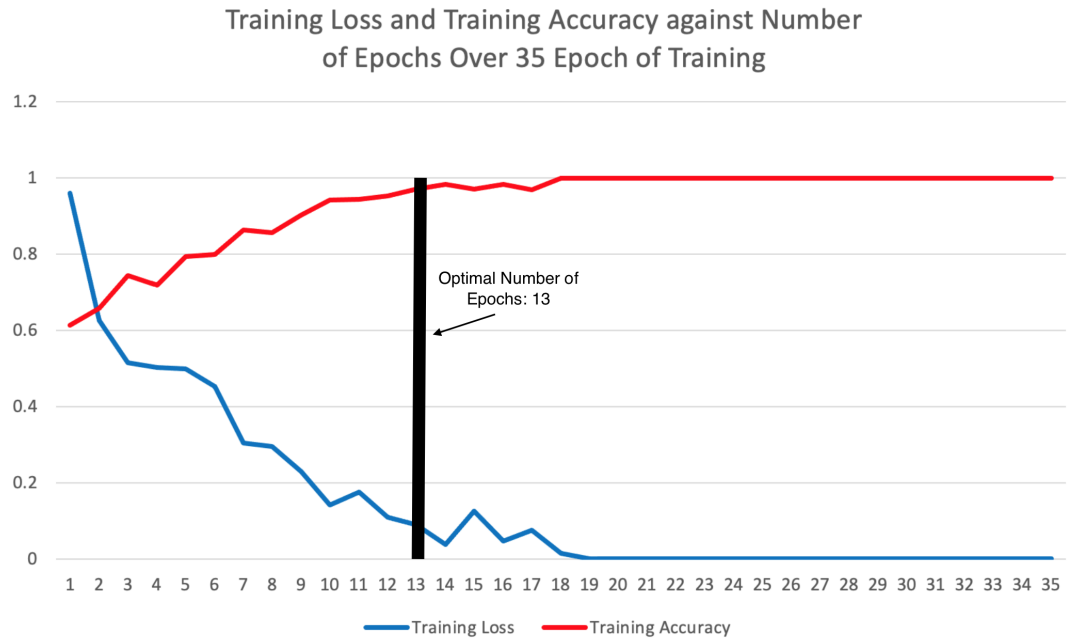
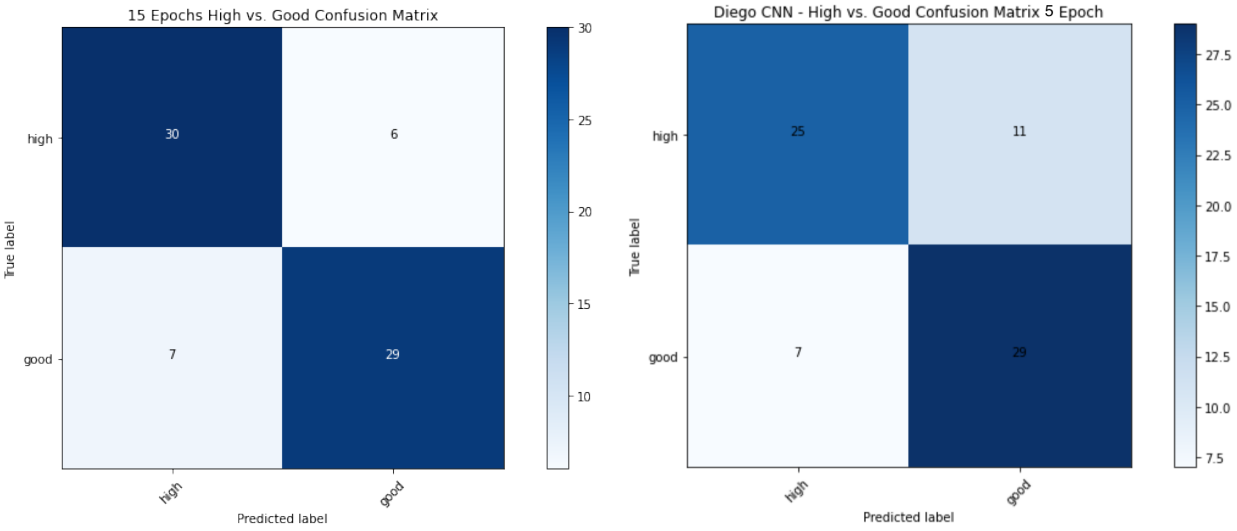


Figure 11: VGG16 training accuracy and training loss plotted against the number of epochs for a run of 35 epochs. The optimal value of 13 epochs occurs where the training accuracy is 97% and the loss is at a mere 0.0889. The raw data can be found in Appendix D in Figure 18.

An interesting, although not concerning, remark is that the VGG16 model classified 'good' values (79.17% accuracy) with greater accuracy than 'high' values (65.74% accuracy). The highest test accuracy CNN for the VGG16 model reached an astonishing 82% over a 15-epoch training period. This model predicted the 'good' label with an accuracy of 80.56% and the 'high' label with an even better 83.33%, as seen in the confusion

matrix in Figure 12a. This helps reduce concerns over the 'high' versus 'good' accuracy discrepancy. The manually made model also showed potential with one run managing a test accuracy of 75% over the course of 5 epochs. The 'good' classification accuracy was 80.56%, but the 'high' accuracy was 69.4%, as seen in the confusion matrix in Figure 12b. The manual model took less than 2 minutes to run and still produced a comparable accuracy to the 1 hour run of the VGG16 model. This shows that fine-tuning the manual model may be more efficient in producing an ideal model than fine-tuning the VGG16 model. This was not expected when developing the method for the project and will be left as a challenge for future developers.



(a) VGG16 model - 82% accuracy - 15 epochs. (b) Manual model - 75% accuracy - 5 epochs.

Figure 12: Confusion matrices for (a) the best VGG16 model with 82% accuracy over 15 epochs and (b) the best manual model with 75% accuracy over 5 epochs. This VGG16 model had the highest accuracy of all the trialled models. The CNN predicted values are labelled on the x-axis and actual (True) labels are found on the y-axis.

4.3 Additional Parameter Tweaks

One important variable that can be changed is the number of trainable layers in the VGG16 CNN. The VGG16 model includes 16 layers containing weights [21], so the addition of two lines of code, as seen in Listing 4, made it so that VGG16 would use its pre-developed weights for the first 'number_of_trainable_layers' number of layers.

Listing 4: Changing the Number of Trainable Layers in the VGG16 CNN

```
for layer in vgg16_model.layers[:number_of_trainable_layers]:
    layer.trainable = False
```

4.4 Discussion of the 3 Class Models

Proof of concept for 3-class classification was also achieved. Ten attempts at running the manual and VGG16 models on 5 and 10 epochs, respectively, yielded a maximum testing

accuracy of 50% each, topping a random guess by 17%. Of all these runs, the lowest test accuracy was 36%, so we are confident that there is promise of success when larger training sets are available. We note that both models classified 'high' more frequently than 'good' and 'low' and depict the best VGG16 confusion matrix in Figure 13.

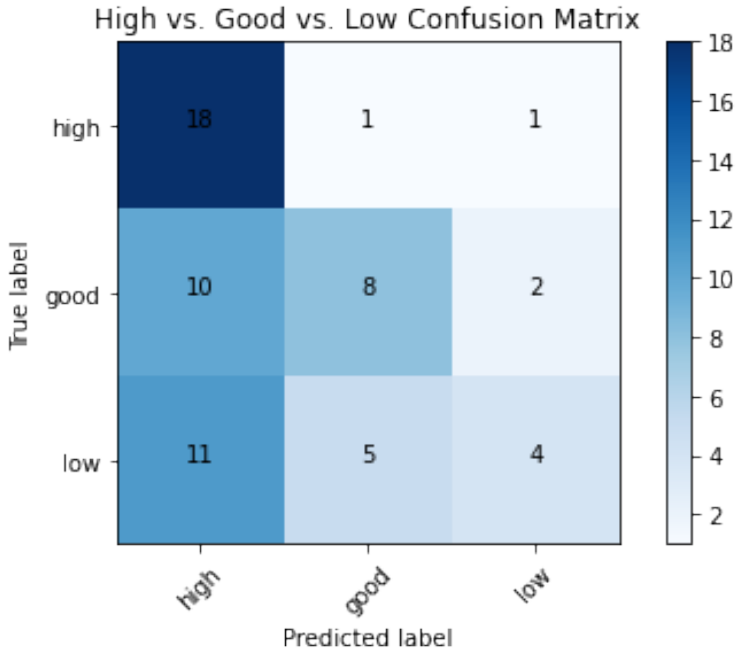


Figure 13: The confusion matrix for the best 3 class VGG16 model. This model achieved 50% accuracy and was run over the course of 10 epochs.

On average, runs of the 3-class models were much faster than the binary class models because of the decrease in number of available training images per class. Interestingly, the manually made model was significantly faster in the training process and yielded identical accuracy over the course of fewer training epochs. For the manual model, it took roughly 11 seconds to run each epoch and for the VGG16 model, it took just under 250 seconds per epoch.

We were surprised to see that the manual model performed with identical accuracy to the VGG16 model during some simple 3-class runs. In fact, the greater speed in training of the manual model suggests that it may be even better than the VGG16 model. Simply tweaking hyper-parameters of the manual model, as well as adding, removing, or re-ordering layers could prove more effective than using the established structure of the VGG16 model.

4.5 Improvements to the Method & Next Steps

One of the biggest problems faced by the training of the models was that validation accuracy and loss fluctuated inconsistently from epoch to epoch, both in the manual and VGG16 models [30]. This suggests a need for further hyper-parameter adjustments [42]. There are three primary options in resolving this issue: altering batch size, changing

the learning rate, and increasing the size of the training set [42]. In hindsight, we could have improved the performance of the CNN by focusing on fine-tuning the batch size and learning rate parameters, as opposed to the number of epochs. [32].

In this project, we could have also attempted to use different models. The Keras library contains 26 pre-trained models (including the VGG16 model) [43] and any one of these could have been selected for training. We chose the VGG16 model because of its proven ability to classify images [32] and because of the large number of parameters available for training [43]. This said, the VGG16 model primarily classified images of objects with a clear structure (e.g. cats and dogs); this may not be the best model for us due to the abstract nature of our images.

In terms of the manually made model, it could also have been improved by re-ordering, adding, or removing layers; this would likely have been the biggest source of improvement to our testing. Using a different model could potentially allow for the production of probabilistic models instead of classification models. This would mean that the output of an image being run through the CNN would yield a Gaussian prediction that would provide an ideal range of SSVI values as opposed to a classification of 'high', 'good' or 'low' [21]. Analysis of this type of CNN is far more complex but could prove useful in more specific characterisation of the data.

The primary focus of the project was merely proof of concept, but there are numerous ways in which the hyper-parameters of the model could have been improved. Time and computational power would be critical in this process. For example, the scripts could have been run using the university's cluster, allowing for greater speeds and potentially parallel processing. This would have allowed us to focus on creating effective, new scripts that tweak individual hyper-parameters, or even multiple parameters at once. In other words, larger amounts of accuracy and loss data for varied, individual hyper-parameters could have been collected if loops, iterating over the code for a period of weeks, had been developed. This is highly recommended for future fine-tuning of the model. Future architects of this CNN should take this into consideration and test the following hyper-parameters: batch size, data split, activation function, number of epochs, learning rate, optimiser, and number of trainable layers. Finally, and necessarily, the API will need to be changed to one that can train more quickly and effectively. PyTorch or TensorFlow, as discussed in the background, may be the ideal APIs going forward now that proof of concept has been established.

4.6 The Future: Global Impact and what Comes After this Project

The bacterial aggregation process in WWT faces limitations in classification of flocculation process to its time-consuming nature and unreliability in accuracy. The success of this project could prove revolutionary in how the characterisation process takes place. In partnership with Veolia, the code could be formally developed by software professionals and distributed to WWT sites across the globe. This would solve the aforementioned problems, reducing human labour hours and potentially improving accuracy. The speed of classification would also benefit WWT plants in enabling quick adjustments to aeration [3]. Finally, the development of highly accurate CNNs provides potential for in-depth analysis of the physics and biology within the system.

5 Conclusion

Convolutional neural networks have proven reliable in image analysis over the past decade and continue to show their strengths in our novel implementation on the crucial biotechnological process of wastewater treatment. Given the analysis of results for our developed CNNs, we can confidently say that the objective of developing a proof of concept, prototype bacterial flocculation characterisation CNN has been met. Our goal of 60% test accuracy for proof of concept has been significantly surpassed by our assessed binary classification CNN, which had a 72.4% average test accuracy and a promising maximum of 82%.

The methodologies employed in developing our CNN and in determining the ideal number of epochs were both effective, though there is room for further fine-tuning of the hyper-parameters of the model. Future developers of this model should be able to improve its accuracy by using significantly larger data sets, ideally on the scale of thousands per class. The development of our CNN shows great potential in providing future physicists and biologists critical data in understanding bacterial flocculation. Further implementations of these novel models could prove revolutionary to global WWT treatment.

References

[1] A. Cressler. [Online]. Available: https://www.usgs.gov/special-topic/water-science-school/science/wastewater-treatment-water-use?qt-science_center_objects=0\#qt-science_center_objects

[2] L. Metcalf and H. P. Eddy, *American sewerage practice*. McGraw-Hill, 1935.

[3] P. S. Davies, *The Biological Basis of Wastewater Treatment*. Strathkelvin Instruments, 2005.

[4] "Veolia home page," Apr 2021. [Online]. Available: https://www.veolia.co.uk/

[5] "Soft matter physics," Jul 2020. [Online]. Available: https://www.ph.ed.ac.uk/icmcs/research-themes/soft-matter-physics#::text=In\%20the\%20Edinburgh\%20Soft\%20Matter,applications\%20ranging\%20from\%20foods\%20to

[6] O. Holst and S. Müller-Loennies, "1.04 - microbial polysaccharide structures," in *Comprehensive Glycoscience*, H. Kamerling, Ed. Oxford: Elsevier, 2007, pp. 123–179. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780444519672000040

[7] G. Dorken, G. P. Ferguson, C. E. French, and W. C. Poon, "Aggregation by depletion attraction in cultures of bacteria producing exopolysaccharide," *Journal of The Royal Society Interface*, vol. 9, no. 77, p. 3490–3502, 2012.

[8] T. O. of Water, *How wastewater treatment works ... the basics*. EPA, 1998.

[9] "U.s. wastewater treatment factsheet," 2020. [Online]. Available: http://css.umich.edu/factsheets/us-wastewater-treatment-factsheet

[10] M. Scholz, "Chapter 18 - activated sludge processes," in *Wetland Systems to Control Urban Runoff*, M. Scholz, Ed. Amsterdam: Elsevier, 2006, pp. 115–129. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780444527349500219

[11] K. Tesh, "How to calculate sludge volume index - svi," Feb 2020. [Online]. Available: https://www.waterandwastewatercourses.com/calculate-sludge-volume-index-svi/#:~:text=Sludge\%20volume\%20index\%20(\%20SVI\%20)\%20is\%20calculated\%20by\%20dividing\%20the\%20settleability,always\%20expressed\%20in\%20mL\%2Fg.

[12] "Stiro-settlometer," Feb 2021. [Online]. Available: http://mcrpt.com/product/stiro-settlometer/

[13] "Locator map." [Online]. Available: https://app.datawrapper.de/

[14] D. P. Mesquita, A. L. Amaral, and E. C. Ferreira, "Activated sludge characterization through microscopy: A review on quantitative image analysis and chemometric techniques," *Analytica Chimica Acta*, vol. 802, pp. 14–28, 2013.

[15] M. Sezgin, D. Jenkins, and D. Parker, "A unified theory of filamentous activated sludge bulking," *Journal (Water Pollution Control Federation)*, vol. 50, no. 2, pp. 362–381, 1978. [Online]. Available: http://www.jstor.org/stable/25039548

[16] "Viscoelasticity." [Online]. Available: http://soft-matter.seas.harvard.edu/index.php/Viscoelasticity

[17] B. Hollmann, "Biofilms and their role in pathogenesis." [Online]. Available: https://www.immunology.org/public-information/bitesized-immunology/pathogens-and-disease/biofilms-and-their-role-in

[18] "Structure of water." [Online]. Available: http://butane.chem.uiuc.edu/pshapley/Enlist/Labs/WaterStruc/Teachers.html#:~:text=The\%20water\%20molecule\%20is\%20composed,diameter\%20of\%20about\%202.75\%20angstrom.

[19] K. Landfester, "Structured fluids.polymers, colloids, surfactants. by thomas a. witten and philip a. pincus." *ChemPhysChem*, vol. 6, no. 4, pp. 747–747, 2005. [Online]. Available: https://chemistry-europe.onlinelibrary.wiley.com/doi/abs/10.1002/cphc.200400537

[20] IBM Cloud Education, "Machine learning." [Online]. Available: https://www.ibm.com/cloud/learn/machine-learning

[21] F. Chollet *et al.*, "Keras," 2015. [Online]. Available: https://keras.io

[22] Arc, "An introduction to convolutional neural networks," Dec 2018. [Online]. Available: https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05#:~:text=Fully\%20Connected\%20Layer\%20is\%20simply,into\%20the\%20fully\%20connected\%20layer

[23] S. Saha, "A comprehensive guide to convolutional neural networks - the eli5 way," Dec 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[24] C. Sammut and G. I. Webb, Eds., *Supervised Learning*. Boston, MA: Springer US, 2010, pp. 941–941. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_803

[25] J. Vigueras-Guillén, B. Sari, S. Goes, H. Lemij, J. Rooij, K. Vermeer, and L. Van Vliet, "Fully convolutional architecture vs sliding-window cnn for corneal endothelium cell segmentation," *BMC Biomedical Engineering*, vol. 1, 01 2019.

[26] S. Sharma, "Sigmoid, tanh, softmax, relu, leaky relu explained," Feb 2019. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[27] H. Mahmood, "How a regression formula improves accuracy of deep learning models," Nov 2018. [Online]. Available: https://towardsdatascience.com/softmax-function-simplified-714068bf8156

[28] T. Shah, "About train, validation and test sets in machine learning," Jul 2020. [Online]. Available: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7

[29] "Categorical crossentropy loss function: Peltarion platform." [Online]. Available: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy

[30] M. A. Nielsen, "Chapter 3: Overfitting and regularization," Jan 1970. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap3.html#overfitting_and_regularization

[31] J. Brownlee, "Understand the impact of learning rate on neural network performance," Sep 2020. [Online]. Available: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

[32] Mandy, "Build a fine-tuned neural network with tensorflow's keras api," Aug 2020. [Online]. Available: https://deeplizard.com/learn/video/oDHpqu52soI

[33] "Keras vs tensorflow vs pytorch: Deep learning frameworks," Jun 2020. [Online]. Available: https://www.edureka.co/blog/keras-vs-tensorflow-vs-pytorch/#:~:text=Keras\%20is\%20a\%20high\%2Dlevel,of\%20TensorFlow\%2C\%20CNTK\%20and\%20Theano.&text=TensorFlow\%20is\%20a\%20framework\%20that,direct\%20work\%20with\%20array\%20expressions

[34] Keras Team, "Keras documentation: About keras." [Online]. Available: https://keras.io/about/

[35] S. Singh, "Difference between jpeg and bitmap," Mar 2020. [Online]. Available: https://www.geeksforgeeks.org/difference-between-jpeg-and-bitmap/

[36] Python Team, "9. classes." [Online]. Available: https://docs.python.org/3/tutorial/classes.html

[37] G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[38] The pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[39] Mandy, "Build and train a convolutional neural network with tensorflow's keras api," Jul 2020. [Online]. Available: https://deeplizard.com/learn/video/daovGOlMbT4

[40] Keras Team, "Maxpooling2d layer." [Online]. Available: https://keras.io/api/layers/pooling_layers/max_pooling2d/

[41] ——, "Conv2d layer." [Online]. Available: https://keras.io/api/layers/convolution_layers/convolution2d/

[42] I. Kandel and M. Castelli, "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset," *ICT Express*, vol. 6, no. 4, pp. 312–315, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405959519303455

[43] F. Chollet *et al.*, "Keras applications," https://keras.io/api/applications/, 2015.

Appendix

A Single Function that runs the VGG16 Model

This function only works once the images have been prepared and the input variables have been set. The only input required is the number of epochs. The output is a saved, trained model and a confusion matrix of the predictions.

Listing 5: Function that runs all of the VGG16 model at once

```
def runAllVGG16(epochs):

    ## create and compile the VGG16 model ##
    v_model = vgg16.VGG16()
    vgg16_model = Sequential()
    for layer in v_model.layers[:-1]:
        vgg16_model.add(layer)
    vgg16_model.add(Dense(units = 2, activation = 'softmax'))
    vgg16_model.compile(optimizer = Adam(learning_rate = 0.0001), loss = '
        ↪ categorical_crossentropy', metrics = ['accuracy'])

    ## train the model ##
    vgg16_model.fit(x = train_batches, validation_data = validate_batches, epochs = epochs,
        ↪ verbose = 2)

    ## make predictions ##
    vgg16_model_predictions = vgg16_model.predict(x = test_batches, verbose = 0)


    ## save and plot the confusion matrix ##
    cm_vgg16_model = confusion_matrix(y_true = test_batches.classes, y_pred = np.argmax(
        ↪ vgg16_model_predictions, axis = -1))
    cm_plot_labels = ['high', 'good']
    plot_confusion_matrix(cm = cm_vgg16_model, classes = cm_plot_labels, title = 'VGG16 CNN
        ↪  High vs. Good Confusion Matrix for ' + str(epochs) + ' Epochs')

    ## save accuracy values to variables ##
    high_correct = cm_vgg16_model[0][0]
    high_incorrect = cm_vgg16_model[0][1]
    good_correct = cm_vgg16_model[1][1]
    good_incorrect = cm_vgg16_model[1][0]
    correct = cm_vgg16_model[0][0] + cm_vgg16_model[1][1]
    total = np.sum(cm_vgg16_model)
    test_accuracy = (high_correct + good_correct)/total
    high_accuracy = high_correct/(high_correct + high_incorrect)
    good_accuracy = good_correct/(good_correct + good_incorrect)

    ## print accuracy values ##
    print("Total test accuracy: " + str(round(test_accuracy,2)))
    print("High accuracy: " + str(round(high_accuracy,2)))
    print("Good accuracy: " + str(round(good_accuracy,2)))

    ## save text file containing confusion matrix and accuracy information ##
    with open('vgg16_2_class_' + str(epochs) + 'epochs.txt', 'w') as f:
        f.write("Total test accuracy: " + str(round(test_accuracy,2)) + '\n')
```

```
    f.write("High accuracy: " + str(round(high_accuracy,2)) + '\n')
    f.write("Good accuracy: " + str(round(good_accuracy,2)) + '\n')
    f.write("Confusion matrix: " + str(cm_vgg16_model))
f.close()

## save the model weights ##
vgg16_model.save_weights('vgg16_2_class_' + str(epochs) + 'epochs_weights')

## return the trained model and the confusion matrix ##
return vgg16_model, cm_vgg16_model
```

B The VGG16 Model Summary

Layer (type)	Output Shape	Param #
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
dense_32 (Dense)	(None, 2)	8194

Total params: 134,268,738
Trainable params: 134,268,738
Non-trainable params: 0

Figure 14: Keras summary of the VGG16 made model [21].

C VGG16 Split Accuracy

Here we show the total, 'high', and 'good' accuracies for each of the 3 splits on 5, 10, 15 and 20 epochs. We additionally include the mean and ideal values in yellow and red, respectively.

Epochs	5	10	15	20
Split 1	0.64	0.72	0.69	0.75
Split 2	0.71	0.69	0.72	0.67
Split 3	0.75	0.81	0.82	0.72
MEAN	0.7	0.74	0.7433333	0.7133333

Figure 15: Total accuracy. Mean = 72.42%

	5	10	15	20
Split 1	0.4444444	0.6666667	0.5833333	0.6944444
Split 2	0.6111111	0.6388889	0.6944444	0.5555556
Split 3	0.7222222	0.8055556	0.8333333	0.6388889
MEAN	0.5925926	0.7037037	0.7037037	0.6296296

Figure 16: High accuracy. Mean = 65.74%

	5	10	15	20
Split 1	0.8333333	0.7777778	0.8055556	0.8055556
Split 2	0.8055556	0.75	0.75	0.7777778
Split 3	0.7777778	0.8055556	0.8055556	0.8055556
MEAN	0.8055556	0.7777778	0.787037	0.7962963

Figure 17: Good accuracy. Mean = 79.17%

D 35 Epoch Run Training Loss and Accuracy Table

Epoch	Training Loss	Training Accuracy
1	0.9597	0.6133
2	0.6259	0.6583
3	0.5141	0.7433
4	0.5024	0.7183
5	0.4994	0.7933
6	0.4529	0.7983
7	0.3034	0.8633
8	0.2954	0.8567
9	0.2285	0.9033
10	0.1408	0.9417
11	0.1754	0.9433
12	0.1092	0.9533
13	0.0889	0.9717
14	0.0382	0.9833
15	0.1252	0.97
16	0.0463	0.9833
17	0.0748	0.9683
18	0.0144	1
19	0.00069247	1
20	0	1
21	0	1
22	0	1
23	0	1
24	0	1
25	0	1
26	0	1
27	0	1
28	0	1
29	0	1
30	0	1
31	0	1
32	0	1
33	0	1
34	0	1
35	0	1

Figure 18: Raw data for the 35-epoch training run.

E File Renaming Functions & Tips

Listing 6: File Renaming Tips

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 3 15:08:12 2021

@author: Diego
"""


import os # os is used to access directories and change file names
import pandas as pd # pandas can access csv files and extract data (ideally with a dataframe)


################ THE GENERAL METHOD ###################
# 1) move to the working directory
# 2) iterate through the file names and edit the strings for the new file names
# 3) access corresponding data from csv files and insert in the strings
# 4) use os.rename() to change the file names

##### PLEASE NOTE RUNNING THIS CODE LEADS TO ERRORS! EDIT DIRECTORIES
    ↪ AND STRINGS
##### TO MAKE IT WORK. I HAVE ATTACHED CODE I USED TO RENAME THE
    ↪ ALLANFEARN DIRECTORY


############################## SIMPLE SET UP
    ↪ ##################################
# go to the selected directory
directory_name = ''
os.chdir(directory_name)


# get file names in alphabetical or numberic order from current working directory
files = sorted(os.listdir())


# iterate through files names in the directory (here we print the file name)
for x in files:
    print(x)


################## TIPS FOR CHANGING STRINGS OF FILE NAMES
    ↪ ###################
# It will only be helpful to work with file names in Python if they share some
# similarities (i.e. dates and times included)
# Scroll through file names and find key differences, such as different spacing, . vs. _, etc.
# Once this is known, a good technique to change the file names is to get the file names
# as strings and use if statements to determine which type of difference is there.
```

```
################## METHODS/FUNCTIONS FOR STRING EDITING
    ↪ ######################
example_file_names = ['file.1.test', 'file2test', 'file 3 test']

# 1) Turn string into list and index to replace characters or using
# string.replace(selected characters/string, new characters/string)


list_of_chars = list('file_name')
print(list_of_chars) # returns ['f', 'i', 'l', 'e', '_', 'n', 'a', 'm', 'e']
print('string'.replace('i', 'o')) # returns 'strong'


# 2) 'character'.join(list_of_strings) is useful for linking a list of str or char
# together via some selected character or string ->

new_string = '.'.join(list_of_chars)
print(new_string) #returns 'f.i.l.e._.n.a.m.e'

# 3) string.split('character_type') to split the string into a list of strings
# that are split by character_type (i.e. '.' or '_') ->

split_string = 'x.y.z'.split('.')
print(split_string) # returns ['x', 'y', 'z']

# 4) lstrip and rstrip funtions get rid of characters from the left side and
# right side of a string ->

stripped_string = '...string'.lstrip('.')
print(stripped_string) #returns 'string'


######################## ACCESSING DATA FROM CSV FILE
    ↪ ##########################
# Make sure the current working directory is the directory in which the csv file
# is held. Save the file to a dataframe as such:
csv_file_name = 'the file name.csv'
df = pd.read_excel(io=csv_file_name)

### IF DATES ARE NEEDED
# If the csv file contains dates and is an Excel file, you will need to use a
# pandas timestamp to search the dataframe for a date. I have already made a
# function for this:
# The input string format is 'two_digit_day.two_digit_month.two_digit_year'
# (e.g. 15.01.21 for 15th of January 2021)

def datereformat(string_time): #formats date into pandas timestamp
    spl = string_time.split('.')
    timestamp = pd.Timestamp(2000+int(spl[2]), int(spl[1]), int(spl[0]), 0)
    return timestamp

print(datereformat('15.01.21')) #returns the timestamp 2021-01-15 00:00:00


### SEARCHING THE DATAFRAME
```

```
# The best way to search for data in a dataframe is using the .loc method.
# For dataframe df, we use .loc and within the brackets can search for a specified
# value within a selected column and then get corresponding values in another column
# In the case below, the csv file has a columne named 'Date' and another named
# 'Allanfearn SSVI', and I extracted the first value at the given date. This
# might be clearer if you look at the example code below


# to test this, input your own csv file and changed the column names to your column names
# if you get rid of the index at the end you get a list values instead of the first one
date = datereformat('15.01.21')
first_allanfearn_value_at_date = df.loc[df['Date'] == date]['Allanfearn SSVI'].values[0]


########################## USING OS TO RENAME FILES
    ↪ ############################
# The simple function to rename a file is os.rename(original_name, new_name).
# The name must include the whole path of the file and the future path.
# Be careful to have a copy of the original folder before performing this operation
# because you could have an incorrect path name and ruin the whole thing
# Here is a simple example of how I have done it

for x in files:
    old_path = os.join([directory_name, x])
    new_file_name = 'the_new_file_name'
    new_path = os.join([directory_name, new_file_name])
    os.rename(old_path, new_path)
```