I Abstract (Introduction)

The computer vision task of classifying three different types of fruit (Cherries, strawberries and Tomatoes) which are the same color would have seemed like a rather difficult task before the advent and improvement of the Convolutional Neural Network (CNN) in 2011. CNN is now considered the premier machine learning algorithm for computer vision tasks. This report explores why and how the CNN towers over its predecessor such as the MLP and other tools alike, and what settings and parameters are particularly influential in creating a great model for this specific classification problem.

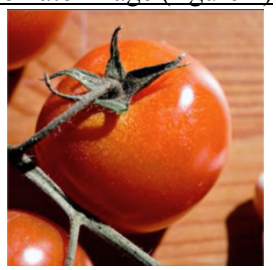
II Background (Problem and Investigation)

A Initial Data Analysis (problem identification)

The data set has 4,500 image instances spread across three classes with 1500 instances each, namely: Cherries, Tomatoes, and Strawberries. Below are some very standard images from their respective classes, which will be used to understand the dataset and the classification problem.

Cherry image (figure 1)

Tomato image (figure 2)



Strawberry image (figure 3)

This may seem like a rather redundant initial observation, however when having regard to the unseen test set of 1500 fruit instances it is obvious that this training set is inferior due to its instance count. We need significantly more instances so that the CNN can be trained rigorously for predicting new images. Because of this the dataset must be preprocessed with significant data augmentation in order to maximize the datasets ability to train a model for unseen data. This may include rotating, flipping or scaling etcetera.

Another *prima facie* observation is that all the images have the same dimensions of 300x300. However, the images themselves vary significantly in terms of the images composition. This may be in relation to the images background, light condition, number of objects, resolution etc. Accordingly, we must suit the model to an array of different types of images, unless for example we diminish all images uniqueness by filtering the images. Furthermore, because of this observation, the dataset contains outliers and noisy images, which will need to be addressed in preprocessing but we must first identify them and a method for dealing with them. The issues I have identified above will be explored further and answered in the Exploratory Data Analysis Section.

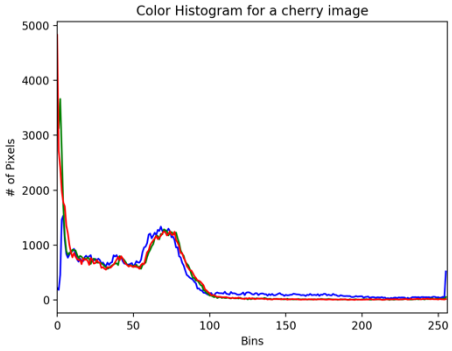
B Exploratory Data Analysis

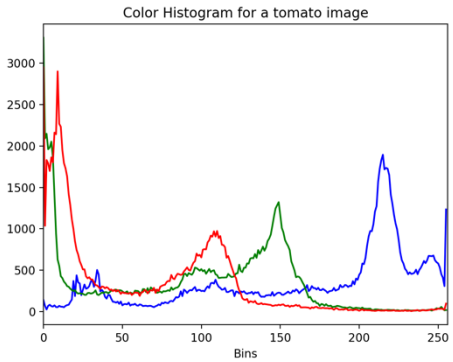
Important features for classification of these three red fruit are not as intuitive for a machine as they are for a human. The human brain can almost instantaneously recognise and identify what objects are presented in an image. Because of this I have had to ask myself what does the human brain subconsciously extract from an image of fruit so that we can inevitably classify what is in it? Is it the size, colour, composition or something I have not even thought of yet? For this reason I will use several well-known feature extracting/engineering tools to analyse and look into what are the best features for a machine to classify red fruit, namely:

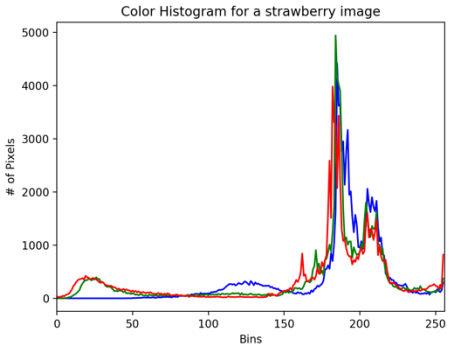
- Splitting RGB Channels
- ORB and Feature Matching
- Filters (Edge Detectors)
- Noise

The code which implemented all these feature extraction methods will be available in the **Appendices section 1.**

Splitting RGB Channels:
Splitting an images RGB channels can provide insight into the characteristics of an image in relation to colour. Furthermore if we compare the RGB channels of an image from each corresponding class we may be able to see what discerns the classes. Resultingly the model may be able to use this feature we have extracted to increase its capabilities as a classifier for fruit. Below are the split RGB values for figure 1, 2 and 3.

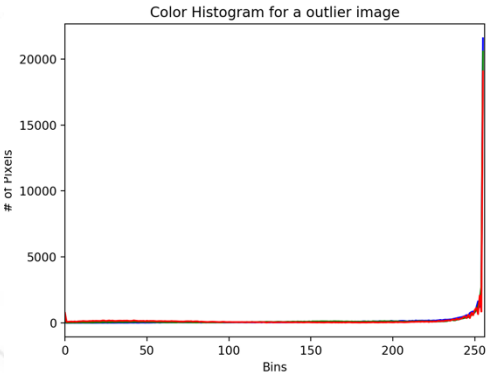






By measuring the proportions of red, green and blue we are able to find images with similar colour distributions. However, all the fruit are red and the shades of red which they produce in an image histogram depends on how ripe they are and what lighting is present in the image. Because of this it is nearly impossible for a model to pull any useful patterns from this feature extraction method unless we have only images of the fruit, which disregards the background. Furthermore we can see that this feature is not very helpful as it takes the large backgrounds into account more so that the actual colour of the fruit. By having regard to the histograms above, the cherry histogram displays large proportions of black, whereas the strawberry image displays large amounts of white, yet to the human eye the cherry and the strawberry are very similar colours. **Appendices section 1 figure 3** shows how this was implemented.

Outlier image

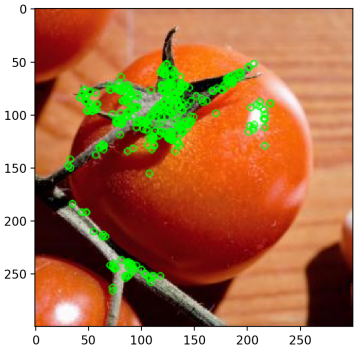


Splitting RGB channels in an image may not be useful to help classify images for this particular problem but it is still a useful tool for identifying outliers in the dataset. If we have regard to the images and their RGB channels to the right, we can see that the image is an outlier in the sense that it contains far too much white and little red. If we did not remove this outlier it may skew the models ability to classify fruit as it may distort the models perception of what a normal fruit is. It is also interesting to note that only by exploring what normal image channels look like are we able to deduce that the above is an outlier. And of course the outlier image is one which we want to delete, it has no benefit to classifying real fruit… maybe cartoon fruit?

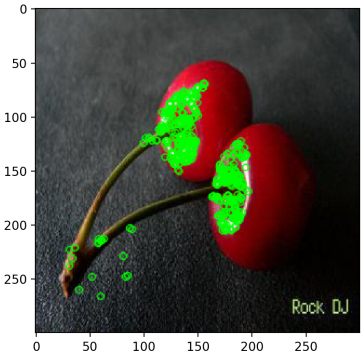
ORB and Feature Matching:

ORB is a combination of FAST (Features from Accelerated Segment Test) and BRIEF (Binary Robust Independent Elementary Features) which is an efficient local feature detector. It aims to extract important features from an image by utilising FAST's ability to recognise corners/edges by comparing any given pixels brightness to the surrounding 16 pixels. BRIEF will then takes all the key points found and binarize them into a vector object which uses a Gaussian kernel so that noise does not affect the feature points significantly. A CNN model will be able to use these features to create an array of important features for a particular class and will consequentially be able to classify new images more efficiently by matching these features. In **Appendices Section 1 figure 2** the code for implementing this is portrayed.

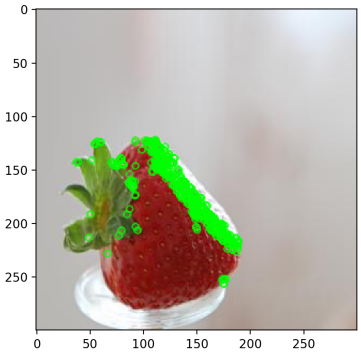
ORB on figure 1 (Tomato)



ORB on figure 2 (Cherry)



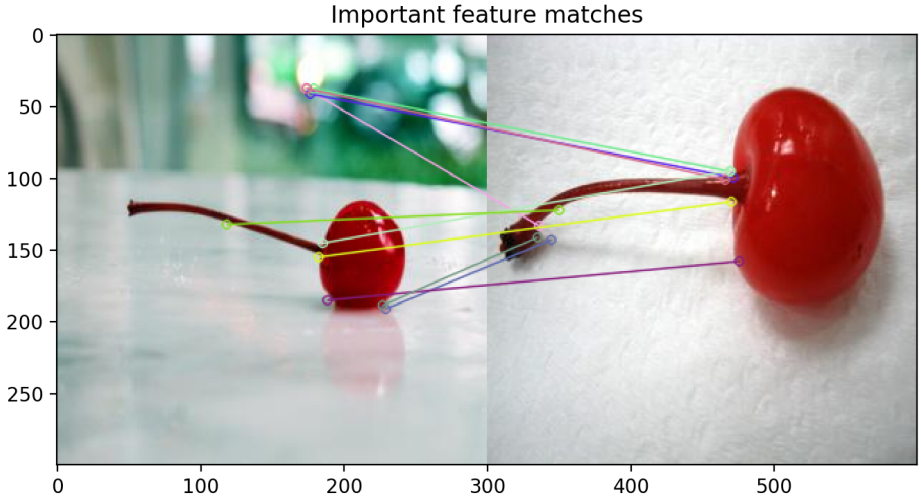
ORB on figure 3 (strawberry)



By having regard to the above portrayals we can see that ORB impressively discerns the background from the object which is the main focus of the image. Because of this the CNN will be able to disregard backgrounds which do not contribute to the important features of the fruit in the image and hence will increase the models accuracy. Furthermore ORB identifies the important features for each fruit, namely:

- We can see that the strawberries straight edge has been identified as one of its key features and is a point of difference to the round cherries and tomatoes.
- The cherries straight stem and lack of leaves at the top of the fruit has been identified as a key feature and is interestingly the main point of difference between the cherry and tomato
- The tomatoes web like stem which connects itself to other tomato and the leaves at the top of the fruit has been identified as its important features

Together these important features identified will help the CNN model distinguish each class. It will also help a model to predict its own class through matching the features which are present above. For example a unknown image of a fruit has been identified as having a straight edge, by having regard to the important features identified in training for the strawberry class the model can match the features and resultingly classify it correctly as a strawberry.



By having regard to the 'important feature matches' figure we can see how feature matching would work when a trained model sees a image it needs to classify. It will analyse the unknown images key features and try map them onto patterns which it has seen before. We can see that the features do not align everywhere. However some crucial matches have been made in areas which distinguish a cherry from the other classes e.g. the stem and the base of the stem.

Filtering for edge detection:

Filtering is the process of applying convolutional kernels to the original image, which in turn will drastically change what the image looks like for our eyes and the CNN model. They pronounce and embolden some of the more important features in an image, while disregarding the redundant information. It does this looking for sharp changes in intensity or colour, which is then reflected by high values below (what you can actually see).

Image filtering for tomato image

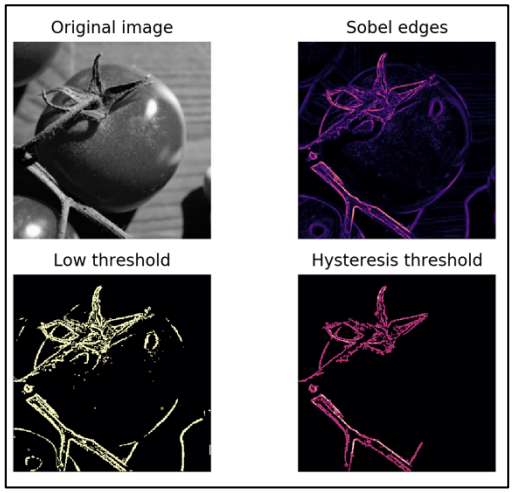


Image filtering for cherry image

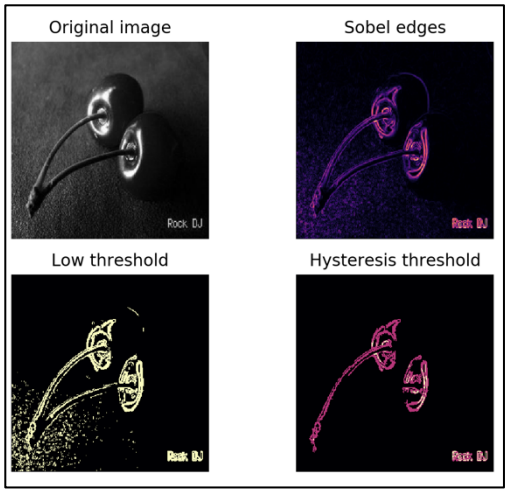
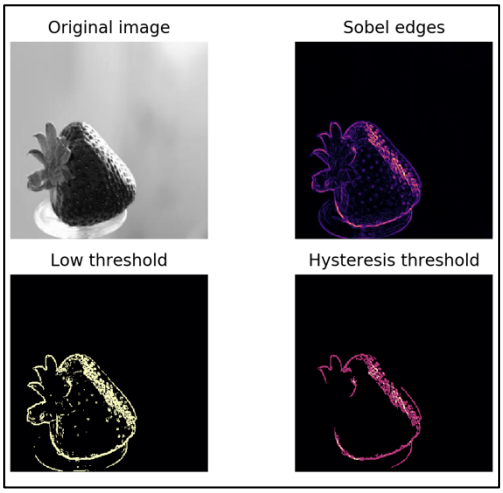


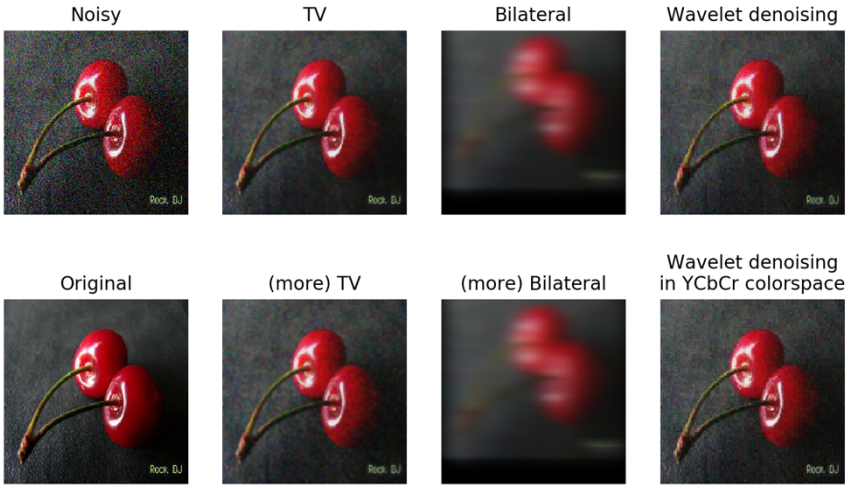
Image filtering for strawberry image



From the representations above we can see how the noise and the background are processed out by using edge detection. The Sobel Edge operator does this by trying to find the difference between pixels by using a gradient matrix to check each pixel in an image. Because of this we can see the extraction of some very useful features from the images which will be useful for the CNN to classify the fruit. Only the most relevant edges are detected by the Sobel edges operation. This is opposed to the low threshold operator and the Hysteresis threshold operator, which seem to have incorporated background/noise into its estimations and removed some key features respectively.

Noise:
The dataset at a first glance does not contain much noise in the usual sense, however there are some images with very low resolutions which I would consider somewhat 'salt and pepper noise'. Images with noise in their current form will cause more harm to the CNN than being completely left out.[1] This is because the noisy images confuse the model by feeding it information which is not similar to what class it proclaims to be from. The human brain can instantly see through the noise, however the CNN cannot. Because of this we must find a way to mitigate the influence of noise on the dataset by looking at methods of 'denoising' data. We can then apply these methods to certain instances in preprocessing which have been identified as noisy. By having regard to the analysis of different noise methods of Koziarski, Michał, and Cyganek, Bogusław I was able to identify which types of methods for filtering noise are the best for image classification.[2] Resultingly, the methods I have chosen as potential suitors for denoising the dataset are a total variation filter, bilateral filter and wavelet denoising filter.

Denoising methods



By having regard to the 'Denosing methods' representation to the right we can observe that noise can manifest itself in a way which seems harmless to the human eye. Furthermore we can also see that TV and Wavelet denoising in YCbCR colorspace are the best filters for denoising the 'Noisy' image presented. I may apply one of these filters during preprocessing to all images. However, I will balance this proposition against the idea that perhaps filtering all the images will lead to a net loss for the CNN. This is because the filter will decrease the quality of normal images which compose the majority in the dataset. If however I outsource some data I will likely denoise it using one of the aforementioned filters.

C Preprocessing

Function for removing image with low red pixel values
I have identified outliers by plotting images respective RGB channels in a histogram and then comparing them to 'normal' images. In order to remove some of the outlier images on a large scale I have created the function (below) which runs through each class of images and checks all the pixels present in a respective image. Following this it increments the variable (r,g or b) which has the strongest RGB value present in that particular pixel. If the percent of dominant red pixels is below a certain threshold in relation to all pixels, which I set at 13% I delete the image from the folder. This is because an image which contains so little red either means the image is an outlier e.g. contains an unripe fruit or the image is completely unrelated to the classification task at hand.

```
def picture_to_arr(image):
    r=g=b=0
    arr = imageio.imread(image)
    arr_list = arr.tolist()

    for row in arr_list:
        for col in row:
            p_b = col[0]
            p_g = col[1]
            p_r = col[2]

            if p_r > (p_g & p_b):
                r+=1
            elif p_g > (p_b & p_r):
                g+=1
            elif p_b > (p_g & p_r):
                b+=1

    minimum_req = 0.13 * (300 * 300) #pixels per image

    if r < minimum_req:
        print(image)
        print ("the of no* of red dominant pixels in this image=",r)
        print ("requirement is",minimum_req)
        os.remove(image)
```

Function for removing outliers
This function has removed some images from the dataset which may not look like they will be a problem at a first glance. However, because they have disproportionate levels of green and blue pixel values, they may skew the CNN's training capability. The inset image is an example of what this function identified and removed from the dataset. **Appendices Section 2 figure 1** shows the output results for this image in relation to the requirements I set. Because of this function the dataset will be optimise the CNN's training and pattern identification. This will also consequentially improve the CNN's ability to predict new instances as it will remove instances which distort its understanding of the classes .

As a preprocessing step I did not think that cropping was one which would provide benefit for training a CNN. Some images would benefit from cropping as it reduce the scope of an image and hence remove some irrelevant background. However on the whole, images largely vary in where the object for classification is placed in the image. Hence, If we were to crop all images we may cut and remove some key features from the red fruit which the CNN can use to optimize its prediction algorithms.

[1] Koziarski, Michał, and Cyganek, Bogusław. "Image Recognition with Deep Neural Networks in Presence of Noise – Dealing with and Taking Advantage of Distortions." *Integrated Computer-Aided Engineering* 24, no. 4 (January 1, 2017): 337–349. http://search.proquest.com/docview/1993972632/.
[2] Ibid.

Data Augmentation implementation

```
train_datagen = ImageDataGenerator(
    rescale = 1./255,
    shear_range=0.2,
    zoom_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2,
    rotation_range=20,
    horizontal_flip=True)
```

One of my initial observations about the dataset was in regard to the low instance count to help train the model. To mitigate this issue I have adopted a method which will multiply the amount of images which the CNN can train on exponentially. It involves augmenting the data in various way such as scaling or rotating. I implemented this through the 'Image Data Generator' method. This method takes every image and then does many combinations of augmentations selected from the types of changes labelled below (rotation_range, height_shift_range…). This will allow for the CNN to be trained more rigorously for predicting unknown data as it will identify more patterns from the larger dataset.

Resizing and manual image removal

I have chosen to resize all the images in the classes from 300x300 to 100x100. This is because it will be more efficient and less time consuming when training the CNN. Furthermore some images which are detrimental to training a CNN are still present after running the function 'for removing outliers' above. Because of this I have had to skim over the remaining images and delete the one which I identified as being outliers. **Appendices section 2 figure 2** shows the implementation for how I resized images and **Appendices Section 3** shows what images I deleted manually.

II Methodology

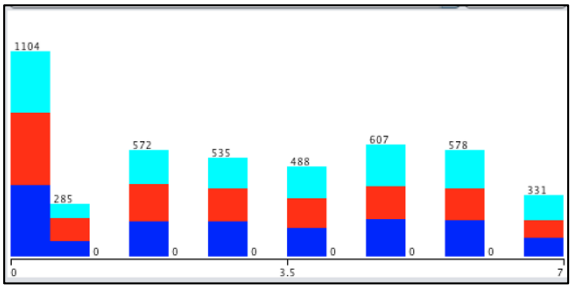
A Building a simple baseline model (for comparison)

Building a baseline model is essential for understanding both how the CNN works and how to evaluate the performance of a CNN. Furthermore a CNN's performance is only impressive when you have knowledge of how its predecessors is composed and consequentially understand why MLP is an inferior model for image classification.

Features created for baseline MLP

```
74  MPEG-7 Edge Histogram73
75  MPEG-7 Edge Histogram74
76  MPEG-7 Edge Histogram75
77  MPEG-7 Edge Histogram76
78  MPEG-7 Edge Histogram77
79  MPEG-7 Edge Histogram78
80  MPEG-7 Edge Histogram79
81  class
```

Example edge histogram feature values



To create a simple MLP model I simply added all instances of images from each class into an excel sheet. I then gave them a feature called 'class' which held the value of what the image is. I then loaded it to Weka and converted it to an arff file. Following this I installed a new Weka library called 'Image Filters' which allowed me to extract features from the images in relation to their class. I chose to extract the Edge Histogram's using one of the provided filters to create many features which could be used for training an MLP. The diagram above show us the features which were created and the values the feature holds, respectively. The filter identifies important edges of classes which can be used to identify its class. I ran a tenfold cross validation to train the model on the extracted features. Discussion about why the MLP performs the way it does and how it compares to the CNN will be presented in Results and Discussion.

Results for Baseline Model (MLP)

```
Time taken to build model: 104.41 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        2103               46.7333 %
Incorrectly Classified Instances      2397               53.2667 %
Kappa statistic                          0.201
Mean absolute error                      0.3581
Root mean squared error                  0.5617
Relative absolute error                 80.5696 %
Root relative squared error            119.1582 %
Total Number of Instances             4500

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0.435     0.301     0.419      0.435     0.427     0.132     0.594     0.414     CHERRY
               0.492     0.242     0.504      0.492     0.498     0.252     0.675     0.493     STRAWBERRY
               0.475     0.256     0.481      0.475     0.478     0.220     0.649     0.473     TOMATO
Weighted Avg.  0.467     0.266     0.468      0.467     0.468     0.201     0.639     0.460

=== Confusion Matrix ===

   a   b   c   <-- classified as
 652 399 449 |   a = CHERRY
 443 738 319 |   b = STRAWBERRY
 461 326 713 |   c = TOMATO
```

By having regard to results diagram to our left we can see that the accuracy of this model through tenfold cross validation is **46.7%**. This model will be tested on the unknown test set in the results section. It is interesting to note that the model struggled to classify the 'CHERRY' class most and received an accuracy score well above random which is not what I expected. Perhaps the features extracted by the edge histograms provided great insight and information to the MLP so that it could classify the fruit rather well. Because the dataset has balanced classes it is fine to evaluate the models performance using accuracy as opposed to the F1 score.

B Building the CNN

The CNN's initial architecture will be based on a model articulated by Ashrafi and Zahidun in a journal article written in 2011.[3] It will consist of two convolutional layers, which is then followed by pooling. Max pooling will be used with a pool size of (2,2), a stride length of 2 and the padding used will be the same. Both convolutional layers will have the parameters of 64 filters, kernel size of (3,3) and Rectified Linear Units (ReLU) as their activation function. A regularization layer dropout of 0.25 will be applied after both layers. After this, a flattening layer will be used to morph the 2-D filter matrix into a 1-D feature vector. Finally, two more fully connected layers will be applied with the first and second having activation functions of ReLU and softmax classifier respectively. The network will be tuned and refined from this base model given by Ashrafi and Zahidun using the specifications of this particular classification problem. The summary for implementing this model can be seen in the **Appendices Section 4 figure 1**. Similarly the final CNN model summary can be seen in **figure 7**, below are the justifications for the settings of the final model.

C Tuning the initial CNN Model

Cross validation and Evaluation:

I have decided not to split the training dataset and do a k-fold cross validation. This is because if I were to split the dataset the pre-processing step of data augmentation would make each training set for a particular epoch vastly different to other rotations. As a result I have simply just decided to apply the same data augmentation pre-processing steps to the supplied test set of 15 images plus another 20 random images and cross validate the CNN against this collection of data at the end of every epoch. I believe this decision will allow me to evaluate the CNN's hyperparameters, ability to acquire the underlying patterns and understand the dataset holistically. In the **Appendices Section 4 figure 4 and 5** there is an example of the cross validation which occurs at the end of every epoch and how it is implemented through training.

Loss functions:

Loss functions are algorithms for evaluating how well a given neural network models the dataset. A high loss function would entail that the models predictions deviate largely from the actual result. The standard loss functions for classification problems is Categorical Cross Entropy Loss. Cross entropy loss heavily penalises predictions which are wrong but confident. This is a feature which other loss function such as Hinge loss or Soft Margin Classifier do not take into account. I believe that this factor will be extremely important to have regard to when evaluating models performance and then selecting a final model. This is because it will make clear whether a model is giving strong weights to the underlying patterns which allow a model to generalise well on new unknown data. Because of the aforementioned reasons I have decided to use Categorical Cross Entropy Loss as the loss function.

Optimisation techniques:

Optimisation algorithms play an important role in helping the CNN decrease the error function for the model. They also play a role in diminishing training costs, which interestingly may cost the convergence rate of the CNN (time the model takes to reach its limit accuracy). They also decide what hyperparameters might need tuning as some are intuitively manipulating them to meet the particularity of the dataset.

Adaptive Moment Estimation (**Adam**) will continue to be the gradient descent optimization algorithm for this classification problem. This is because "The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters."[4]. Furthermore, Adam computes adaptive learning rates for hyper-parameters, therefore they require little fine tuning as you will likely receive the best results using the default value.[5] All of these reasons and benefits of using Adam is advantageous to the given classification problem in the sense that it will save time (training cost) and suit a large dataset with many hyperparameters (which we have after data augmentation). Adam has also show remarkable success in computer vision (image classification problems) tasks. Therefore it makes it the logical choice for the task of classifying red fruits.

---

[3] Ashrafi, Zahidun. "Implementation of Fruits Recognition Classifier Using Convolutional Neural Network Algorithm for Observation of Accuracies for Various Hidden Layers." *arXiv.org* (June 11, 2019). http://search.proquest.com/docview/2209781708/.
[4] Kingma, Diederik, and Ba, Jimmy. "Adam: A Method for Stochastic Optimization." *arXiv.org* (January 30, 2017). http://search.proquest.com/docview/2075396516/.
[5] Ruder, Sebastian. "An Overview of Gradient Descent Optimization Algorithms." *arXiv.org* (June 15, 2017). http://search.proquest.com/docview/2076187906/.

I had regard to other techniques such as **RMSProp + Momentum** which is one of the optimization algorithms that Adam is based off and therefore is rather similar. However there is one considerable difference between the two algorithms; Adam has a bias correcting mechanism, whereas RMSProp does not. Therefore RMSProp cannot mitigate the bias which some models innately create. This would present a rather large problem to any data scientist trying to sell their model if they used RMSProp. In this particular case a bias model could be sold an orchard owner, where rely on information which is both bias and incorrect to inform their business strategy and consequentially lose money/efficiency. I also had regard to **AdaGrad** when selecting the optimisation technique however when comparing it to Adam is had the same shortcomings as RMSProp.

Regularisation strategy:

Regularisation is a strategy employed by data scientist to mitigate overfitting of a neural network i.e. a CNN. It is an essential feature for a successful CNN as their tendency is to overfit as they become extremely well trained on the particular training set. Because of this there are multiple strategies which are given to stop over fitting and two of the most common are dropout and L2 regularization. L2 regularization penalises nodes with large weights, which in turn reduces the values which are given to the activation function.[6] Consequentially a less particular function will be fitted to the data, and thus reducing overfitting. Whereas dropout regularization simply places a probability of removing certain nodes for each layer that employs the method. The optimum threshold (probability of dropout) for any given data set is different for each different model and dataset. Dropout reduces overfitting by disallowing the model to give large weights to any given feature as there is a chance it will be removed, instead weights are evenly distributed across all features. For this reason dropout increases the generalisation capabilities of the model.[7] Because of this I have decided to employ both strategies to mitigate overfitting for my model. An incremental dropout layer will be added after every second layer starting at 0.2 and ending at 0.4. The weight decay for the L2 regularization method with be **1e^-4**.

Interestingly Koziarski and Cyganek from UST in Poland thought that giving some images noise would improve generalization abilities of the model.[8] This is because it would disallow a neural network to weight any given feature highly as it must have regard to the images which are blurred and do not have the sharp feature which was identified previously. This consequentially would in theory improve the generalisation abilities of a model if done correctly.

Activation function:

The activation function essentially calculate the output (weighted sum) of a particular node or neuron in a neural network given its inputs. In deciding which activation function I would use for the particular classification problem I weighed up three different functions, namely: Linear, Sigmoid and ReLU. A linear activation function is simply a straight line function which will weigh up a neurons inputs to give the output. I have not chosen to use this function as it simply would defeat the purpose of using a CNN with multiple stacked layers. This is because when you use a linear activation function each layer is activated by the previous linear function and acts as input for the layer. Consequentially your layers will act as a single layer because it is essentially just one large linear function, which continue on from each other.

That leaves the sigmoid and ReLU activation functions. Firstly, it is important to note that both of these function are not linear in nature and therefore we can stack layers for a CNN model, without it being a ruse. This is because between the sigmoid function (Appendices Section 4) X values of -2 and 2 the gradient is very steep, which means the output (Y) will tend to fall on either side of the curve with a tiny change in X. This means the CNN will be able to make clear distinctions at each neuron, which will result in better prediction. However there is a tradeoff with this advantage, because if the X value falls outside the bounds described earlier the gradient is very shallow, which means the output (Y) will not vary largely.[9] This means the network will refuse to learn or will at an extremely slow pace. ReLU simplifies the sigmoid function and essentially outputs a value of 0 if X is below 0 and X if it is above zero (Appendices section 4). This means that there is fewer sparse activations from neurons. This resultingly makes the neural network much more cost efficient compared to the sigmoid function as it has fewer 'connections' between neurons. This may also help reduce overfitting on the dataset as ReLU can be able to recognise the more underlying patterns present in the dataset by not allowing every neuron to add information to the neural network. However, there is one issue with the ReLU function and it relates to large parts of a neural network becoming passive as a result of many X value being close to zero (dying ReLU problem)[10].

---

[6] Xu, Bingxin, Guo, Ping, Chen, Clphilip, and Xu, Bingxin. "An Adaptive Regularization Method for Sparse Representation." *Integrated Computer-Aided Engineering* 21, no. 1 (January 1, 2014): 91–100. http://search.proquest.com/docview/1475533511/.
[7] Wager, Stefan, Wang, Sida, and Liang, Percy. "Dropout Training as Adaptive Regularization" (July 4, 2013).
[8] Koziarski, Michał, and Cyganek, Bogusław. "Image Recognition with Deep Neural Networks in Presence of Noise – Dealing with and Taking Advantage of Distortions." *Integrated Computer-Aided Engineering* 24, no. 4 (January 1, 2017): 337–349. http://search.proquest.com/docview/1993972632/.
[9] Oostwal, Straat and Biehl. "Hidden Unit Specialization in Layered Neural Networks: ReLU vs. Sigmoidal Activation" (October 16, 2019). https://arxiv.org/pdf/1910.07476.pdf
[10] Ibid.

Because of the aforementioned reasons, I have decided the CNN model will continue to use and explore the ReLU activation function. I have decided this primarily because I want the CNN to be as computationally expensive as possible so I can explore many variations.

Data:

I outsourced data for the three fruit classes from the publicly available dataset 'fruits-360'.[11] It contains over 86,000 instances of different fruit and the images collected by a recording of a singular piece of fruit revolving. All images are 100x100 pixels so they will integrate well into the preexisting dataset and CNN training. Below are some of the images sourced from the dataset.

Images from the dataset 'fruits-360'

The addition of these images is going to enrich each respective classes training. This is because these images provide the CNN with the object alone and its multiple variations. This will allow the CNN to extract the exact features from each respective class to help predict and classify new and unknown instances. Furthermore the CNN will be able to identify exactly parts of an new/unknown image which align closely to some of this new training data, which consequentially will increase its prediction capabilities dramatically. Furthermore the new data will increase the CNNs ability to generalise on new data and decrease overfitting. Because of this the Cherry, Tomato and Strawberry classes have 1055, 1245 and 885 new images respectively. Another reason why this data enriches the dataset is because the data contained different variations of the fruit. For example we can see that there are both two variations/species of cherry and strawberry above and there were more which were added. This I believe will allow the model to identify more obscure species of the fruit. In the **Appendices section 4 figure 6** you can see how the 'fruits-360' dataset was structured in terms of how it had multiple species of the same fruit.

Setting hyper-parameters:

Hyperparameters are parameters which are set before the learning process begins (before you begin to train the model). I will firstly discuss the following hyper parameters in depth, which relate to optimization and training the model: the learning rate, batch size and number of epochs. For the learning rate value there is a golden threshold. This is because if the learning rate is to low it will take a vast amount of rotations (epochs) to reach a level which we would consider competent. However if it is too high the optimum values (least error) will be missed as it jumps over it. Therefore I have decided to set a learning rate of **0.007** as this value will provide rigorous training to optimise loss while not having to do many rotations. Furthermore, this is the learning rate which is used by Sakib, Ashrafi and Siddique to classify images of fruit and vegetables.[12] Batch size refers to the amount of data fed into one training step. The optimum batch size has a tradeoff, if you have a large batch size it increases training calculations at the cost of needing more memory whereas if you have a low batch size it induces more variation in the models error calculations and stops the model from settling at the minimum. Because of this I have decided to use a **batch size of 32**, which lands somewhere in the middle of the tradeoff and consequentially will give us a little of everything with regard to the benefits described above. How many epochs a CNN does when training a model defines how many rotations of the dataset occur during training. The optimum amount of epochs is correlated to the convergence rate of the error function. When the error function starts to plateau is where the amount of epochs should end. Because of this I have decided to use **50 epochs** as the models I create seem to converge on 0.22 around 50 epochs. In **Appendices section 4 figure 2 and 3** this observation is justified.

I will now discuss and justify the settings of some hyperparameters which relate to the structure of the model, namely: number of hidden units and number of layers. The number of hidden units is the paramount feature which defines a models learning capabilities of a model. If you provide a model with too many hidden units the model will tend to overfit the data rather than understand the underlying patterns. Whereas if it is too small then the model will pick up on the complexity of the classification problem. Because of this I have decided to have 32 hidden units (filters) on the first two hidden layers which are used, which will then increase to 64 and then 128 as hidden layers increase. I believe these values will ensure that neither of the aforementioned problems occur as they will provide well rounded learning capabilities to the final CNN model **(Appendices section 4 figure 7**). The deeper a CNN the better they perform. Therefore the amount of **hidden layers will be set at 6** because this is the amount which gave me the best results with regard to loss and accuracy. The stride of the model is influenced by the size of the pooling window (2x2) which consequentially means the stride is 2 and after each pooling step the window will slide 2 pixels.

[11] https://www.kaggle.com/moltean/fruits

[12] Ashrafi, Zahidun. "Implementation of Fruits Recognition Classifier Using Convolutional Neural Network Algorithm for Observation of Accuracies for Various Hidden Layers." *arXiv.org* (June 11, 2019). http://search.proquest.com/docview/2209781708/.

Ensemble learning:

Ensemble learning is a technique adopted by data scientists to optimise their predictive model. How ensemble learning works: it uses several model which each take a portion of the dataset and then averages those models to given one final model for predicting new instances. The method I will employ to show ensemble learning for multiclass classification problems is by training two CNN models and then loading their best weight and then average the outputs for prediction. I will expect a lower error rate than any single model I have created.

Pre-trained model:

Transfer learning is the process of using a pre-trained model which has been trained to solve a similar problem which can be used for the current one. So we are looking for a model which has been trained on a large dataset and would be suitable classifying fruit. Transfer learning is popular because it creates successful models in a relatively shorter amount of time.[13] Because we are using Keras we have access to an array of pretrained models such as ResNet5 and VGG. We can then set their weights as something which align with the particular classification problem, which in this case would be **'ImageNet'** because it has a vast array of images of fruit as well any many other things. I have decide to used ResNet5 because it was the winner of the imageNet challenge in 2015, which makes it the best available model for the given weights.

III Results Discussion

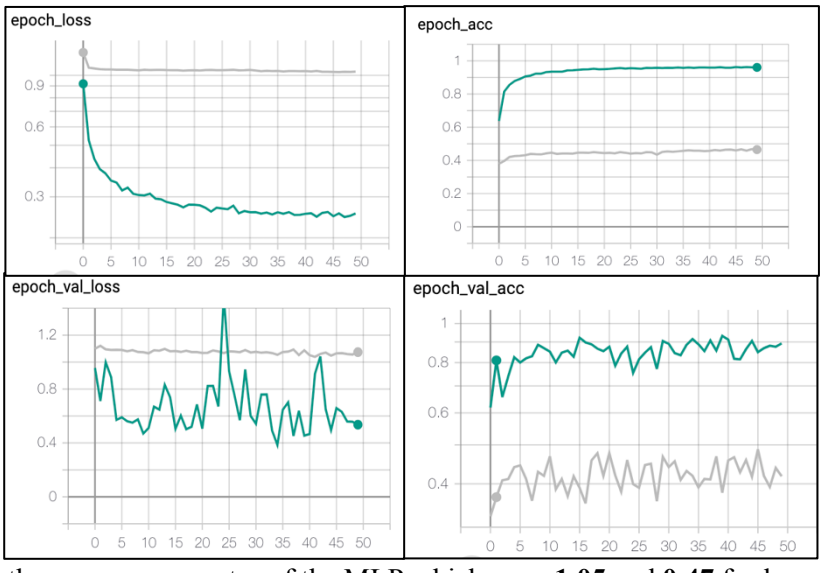
A MLP Settings

The Multilayer Perceptron model created for classifying fruit will be then evaluated against the same test set which the CNN is evaluated on. The Multilayer Perceptron models setting are discussed above in the "building a simple baseline model" section extensively, however the main setting are:

- The model trains on Edge histograms of the images, which are features extracted using a outsourced filter.
- The model will be evaluated against a tenfold cross validation.

B Results

TensorFlow results from training and validation sets (Green = CNN, Grey = MLP)



Results from final models (CNN and MLP) being tested on new unknown data

Metric	*MLP*	*CNN*
Accuracy	0.3238	0.4285
Loss	*1.7410*	1.1974

We can see from the results that the CNN Model virtually outclass the MLP in every category. However it is interesting to note how consistent the MLP's score for validation loss in training is compared to the CNN's. Furthermore we can see that the CNN's accuracy and loss improves drastically faster than the MLP and as a result reaches a better convergence rate of around **0.98** and **0.27** respectively. This we can compare with the convergence rates of the MLP which were **1.05** and **0.47** for loss and accuracy respectively. One thing which I may criticise the CNN of is inconsistency between accuracy loss scores from training to testing on new unknown data. Because of this it is likely the steps I took to mitigating overfitting in the methodology section did not go far enough. We must also not that the MLP on unknown data actually produces a result which is worse than picking at random. This is interesting because the MLP's loss is not horrible, which potentially means the test set I chose to evaluate the models on is not similar to the training data set.

[13] "Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review." *Neural Computation* 29, no. 9 (September 2017): 2352–2449.

C Analysis of Results

The above results further the well-established idea that MLP lags behind the CNN when it comes to real world applications of computer vision like the one at hand. This is because The CNN is superior to the MLP in terms of identifying underlying patterns and features in the computer vision tasks. The primary reason for this is the excessive number of parameters the MLP produces for each node as a result of being fully connected. Consequentially node connection are dense and their weights are heavy which causes inefficiencies in both identifying patterns and computational cost. CNN layers are sparsely connected as opposed to fully connected, which means weights are shared and nodes are not connected to every other one. This means that (1) the neural network will not overfit as much as MLP and (2) it will identify and delineate patterns better than MLP.

Another reason why CNN produces superior accuracy results is because it can have regard to spatial information. This is because a CNN can take matrices as well as vectors as input. The CNN takes in a 100x100x64 in this particular data matrix whereas an MLP can only receive a flattened vector. Images are spatially ordered data so it makes sense that because the CNN has regard to this property it has a superior prediction ability. This is a fundamental difference between CNN and MLP. Furthermore, the number of filters (32/64/128) adds to the depth of the matrix and as a result increases its ability to classify images compared to the MLP as it looks at the image as a whole instead of breaking it up.

IV Conclusions

A Pros and cons of my approach

A positive aspect about my approach was the use of call backs of checkpoints and TensorBoard when training the model. The checkpoint callback allowed me to test multiple iterations of the model created. This is because every time the validation loss would decrease in an epoch the method would save the weights for that particular epoch. This meant that I could try many different weights from the same training process instead of one. It also allowed me to plug in weights from previous models to newer ones if they had the same makeup. The TensorFlow callback allowed me to log the respective results of the models I created at the end of every epoch. This allowed me to make the observations and inferences above in the results analysis and the methodology. The only negative with regard to the TensorFlow call back is that I did not use it earlier as it would have been extremely interesting to see the progress I made from using 3 layers to 7 layers, which gave me the final model for comparing against the MLP.

Another positive aspect of my approach was the time which I spent researching the different aspects of this assignment. Whether it be what activation function I should use for the particular classification problem or just trying to extract some of the features which the CNN may draw upon in training. I never skipped the process of understanding what I was actually implementing. As a result I believe this process has given me a very well rounded understanding of CNN's intricacies as well as using the CNN to solve the classification problem.

B Future work

Since the CNN is universally accepted as an almost perfect mechanism for tackling computer vision problems such as that one at hand. I think perhaps where researchers must now spend their focus is trying to diminish the CNN's tendency to overfit the data it is trained on. This is because in the real world a CNN which is slightly overfitted will introduce bias into societies processes, which may have unintended consequences. For example, the dataset which is used by machine learning models to identify cancers may in turn lead to biases against certain races.

C Main findings

Although I was not able to replicate a model which performed this particular classification problem as well as human could. I was able to implement and explore a vast amount of settings and processes which allow a computer generated model to carry out tasks which we would consider a human task. The accuracy of the final selected model produces results far above random, which shows that the model created understood and identified the underlying patterns in the fruit images classification problem. Furthermore we have understood why and how a CNN is superior to an MLP for computer vision tasks. These findings together are very interesting to society at large. This is because as the human species decides to allow automated processes into society to carry out essential functions, lay people give up their right to compete for work and understand how some of the most mundane tasks are carried out e.g. how a fruit picker (not necessarily human) decides what fruit to pick. For this reason it is important for people in the Machine learning industry to understand their implementations/creations and what that means for society. This is because often normal people are adversely affected and exploited by the creations of data scientists.

VI References

Koziarski, Michał, and Cyganek, Bogusław. “Image Recognition with Deep Neural Networks in Presence of Noise – Dealing with and Taking Advantage of Distortions.” *Integrated Computer-Aided Engineering* 24, no. 4 (January 1, 2017): 337–349.

Ashrafi, Zahidun. “Implementation of Fruits Recognition Classifier Using Convolutional Neural Network Algorithm for Observation of Accuracies for Various Hidden Layers.”

Kingma, Diederik, and Ba, Jimmy. “Adam: A Method for Stochastic Optimization.”

Ruder, Sebastian. “An Overview of Gradient Descent Optimization Algorithms.”

Oostwal, Straat and Biehl. “Hidden Unit Specialization in Layered Neural Networks: ReLU vs. Sigmoidal Activation” (October 16, 2019).

Wager, Stefan, Wang, Sida, and Liang, Percy. “Dropout Training as Adaptive Regularization” (July 4, 2013).

Xu, Bingxin, Guo, Ping, Chen, Clphilip, and Xu, Bingxin. “An Adaptive Regularization Method for Sparse Representation.” *Integrated Computer-Aided Engineering* 21, no. 1 (January 1, 2014): 91–100.

“Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review.” *Neural Computation* 29, no. 9 (September 2017): 2352–2449.

“Fruit-360 dataset” https://www.kaggle.com/moltean/fruits

VII Appendices

Section 1 (Code for Exploratory Data Analysis and preprocessing)

Implementation for Noise methods (**figure 1**)

```
def Noise(img):
    img = img

    sigma = 0.155
    noisy = random_noise(img, var=sigma**2)

    fig, ax = plt.subplots(nrows=2, ncols=4, figsize=(8, 5),
                           sharex=True, sharey=True)

    plt.gray()

    # Estimate the average noise standard deviation across color channels.
    sigma_est = estimate_sigma(noisy, multichannel=True, average_sigmas=True)
    # Due to clipping in random_noise, the estimate will be a bit smaller than the
    # specified sigma.
    print(f"Estimated Gaussian noise standard deviation = {sigma_est}")

    ax[0, 0].imshow(noisy)
    ax[0, 0].axis('off')
    ax[0, 0].set_title('Noisy')
    ax[0, 1].imshow(denoise_tv_chambolle(noisy, weight=0.1, multichannel=True))
    ax[0, 1].axis('off')
    ax[0, 1].set_title('TV')
    ax[0, 2].imshow(denoise_bilateral(noisy, sigma_color=0.05, sigma_spatial=15,
                    multichannel=True))
    ax[0, 2].axis('off')
    ax[0, 2].set_title('Bilateral')
    ax[0, 3].imshow(denoise_wavelet(noisy, multichannel=True))
    ax[0, 3].axis('off')
    ax[0, 3].set_title('Wavelet denoising')

    ax[1, 1].imshow(denoise_tv_chambolle(noisy, weight=0.2, multichannel=True))
    ax[1, 1].axis('off')
    ax[1, 1].set_title('(more) TV')
    ax[1, 2].imshow(denoise_bilateral(noisy, sigma_color=0.1, sigma_spatial=15,
                    multichannel=True))
    ax[1, 2].axis('off')
    ax[1, 2].set_title('(more) Bilateral')
    ax[1, 3].imshow(denoise_wavelet(noisy, multichannel=True, convert2ycbcr=True
                                    ))
    ax[1, 3].axis('off')
    ax[1, 3].set_title('Wavelet denoising\nin YCbCr colorspace')
    ax[1, 0].imshow(img)
    ax[1, 0].axis('off')
    ax[1, 0].set_title('Original')
```

Implementation for **ORB and Feature Matching (fig2)**

```
def image_detect_and_compute(detector, img_name):
    """Detect and compute interest points and their descriptors."""
    img = cv2.imread(os.path.join(dataset_path_cherry, img_name))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    kp, des = detector.detectAndCompute(img, None)
    return img, kp, des


def draw_image_matches(detector, img1_name, img2_name, nmatches=10):
    """Draw ORB feature matches of the given two images."""
    img1, kp1, des1 = image_detect_and_compute(detector, img1_name)
    img2, kp2, des2 = image_detect_and_compute(detector, img2_name)

    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(des1, des2)
    matches = sorted(matches, key = lambda x: x.distance) # Sort matches by distance.  Best

    img_matches = cv2.drawMatches(img1, kp1, img2, kp2, matches[:nmatches], img2, flags=2)
    plt.figure(figsize=(8, 8))
    plt.title("Important feature matches")
    plt.imshow(img_matches); plt.show()
```

Implementation for RGB histograms **(figure 3)**

```
def colour_histogram(img):
    chans = cv2.split(img)
    colors = ("b", "g", "r")
    plt.figure()
    plt.title("Color Histogram for a outlier image")
    plt.xlabel("Bins")
    plt.ylabel("# of Pixels")
    features = []

    # loop over the image channels
    for (chan, color) in zip(chans, colors):
        hist = cv2.calcHist([chan], [0], None, [256], [0, 256])
        features.extend(hist)
        plt.plot(hist, color = color)
        plt.xlim([0, 256])

    plt.show()
```

Section 2 (Preprocessing)

Output for img which does not meet requirements **(fig 1)**

```
/Users/harryrodger/Desktop/data/cherry/cherry_0100.jpg
the of no* of red dominant pixels in this image= 10630
requirement is 11700.0
```

Implementation for resizing the images **(figure 2)**

```
def resize_img():
    for filename in os.listdir('/Users/harryrodger/Desktop/data/tomato/'):
        if filename.endswith('.jpg'):
            img = ('/Users/harryrodger/Desktop/data/tomato/%s' % filename)
            OGimg = Image.open(img)
            size = (100,100)
            resized_image = ImageOps.fit(OGimg,size,Image.ANTIALIAS)
            resized_image.save(('/Users/harryrodger/Desktop/dataNEW/tomato/%s' % filename))
```

Section 3 (Example images which were manually removed and resizing function)

Section 4 (Methodology)

Figure 1: CNN original model summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 64, 64)        1792
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 64)          36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 500)               512500
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 1503
=================================================================
Total params: 626,579
Trainable params: 626,579
Non-trainable params: 0
```

Figure 2: Convergence rate for accuracy

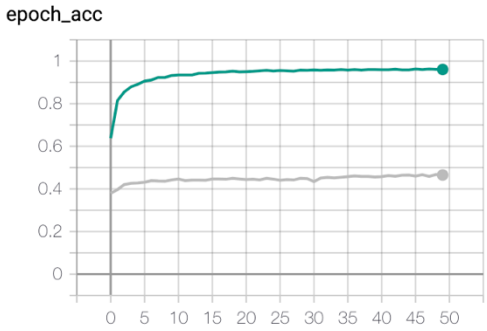


Figure 3: Convergence rate for loss

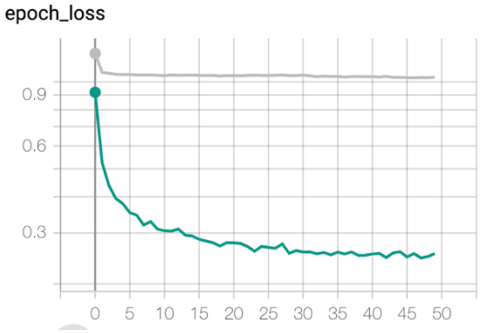


Figure 4: Example of the output for cross validation accuracy.

```
8000/8000 [==============================] - 3172s 396ms/step - loss: 0.0690 - acc: 0.9792 - val_loss: 1.1956 - val_acc: 0.8000
Epoch 39/50
8000/8000 [==============================] - 1191s 149ms/step - loss: 0.0636 - acc: 0.9809 - val_loss: 6.1919 - val_acc: 0.8667
Epoch 40/50
8000/8000 [==============================] - 1190s 149ms/step - loss: 0.0723 - acc: 0.9793 - val_loss: 1.0066 - val_acc: 0.8000
```

Figure 7:the final model summary.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 64, 32)        896
_________________________________________________________________
batch_normalization (BatchNo (None, 64, 64, 32)        128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 64, 64, 32)        9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 64, 64, 32)        128
_________________________________________________________________
dropout (Dropout)            (None, 64, 64, 32)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
batch_normalization_4 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 128)         147584
_________________________________________________________________
batch_normalization_5 (Batch (None, 4, 4, 128)         512
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 2, 2, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 2, 2, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 3)                 1539
=================================================================
Total params: 290,339
Trainable params: 289,443
Non-trainable params: 896
_________________________________________________________________
```

Figure 6: Structure of the fruits-360 dataset

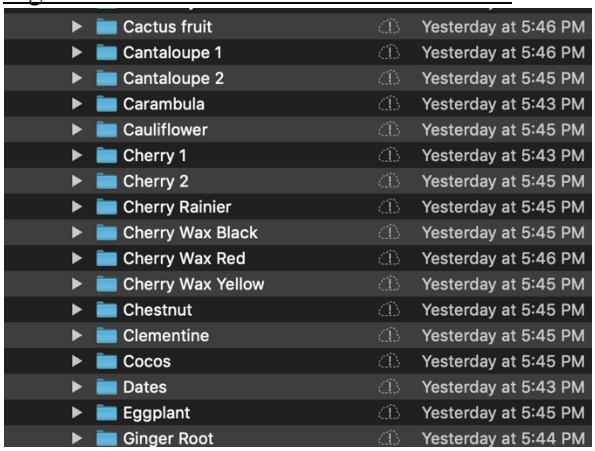


Figure 5: How the cross validation is implemented

```
model.fit_generator(
    training_set,
    steps_per_epoch=8000,
    epochs = 50,
    validation_data=test_set,
    validation_steps = 15)
```