Classification of fruit images using Convolution Neural Network

The problem, potential clients and data

Image classification is increasingly being used in mobile applications. An example where classification of fruits will be helpful are applications which are used to track calorie intake. The first target of this project are such mobile applications:

Mobile applications that track calorie intake. Instead of manually entering the number of calories, users of such applications can just take a picture of the fruit that they are consuming and calories associated with that specific fruit can automatically be added. This will be a major attraction for users of such as applications.

Another example where fruit classification can be helpful is the identification of grocery items in a grocery store. Sorting of these items and placing them in their respective aisles is a problem that can be automated with applications that can identify these items. The second target of this project will be such stores:

One item that most if not all major grocery markets carry are fruits. An application which can sort fruits automatically will be of great benefit for these markets. In a larger context, this will be a first step in developing an application that can recognize a wide range of items carried by these stores.

Apart from this specific project of classifying fruit images, image classification in general, has applications across a wide range of businesses. From photo organization mobile applications to facial recognition used by social networking websites, image classification is being used to enhance business value and/or user experience. For example, identifying the type of damage a car has undergone can inform an insurance company about a potential claim and help them make an informed decision.

In summary, the skills required for image classification have a wide range of applicability. Therefore, an understanding of the process of image classification is crucial for any data scientist.

The data for this analysis were the images of more than 75 different classes of fruits. This data was acquired from the following data base link:

https://www.kaggle.com/moltean/fruits/data

Approach and Results

This classification was performed using convolution neural network. As this required a Graphical Processing Unit (GPU), the analysis was performed using Kaggle's GPU. The project was divided into three parts.

Part-1: Splitting data into training, validation and test set.

The data consisted of two folders: Training and validation. Within each folder there were subfolders containing images of single classes of fruits. A neural network is trained on training images and validated on validation set. Training progress is observed by examining the loss and accuracy of the network on validation set.

A major problem of the above-mentioned approach is that there is no data on which the model can be tested upon. The solution to this was splitting of the validation set into two sets: test set, containing 25% of the validation images, and validation set containing rest of the images.

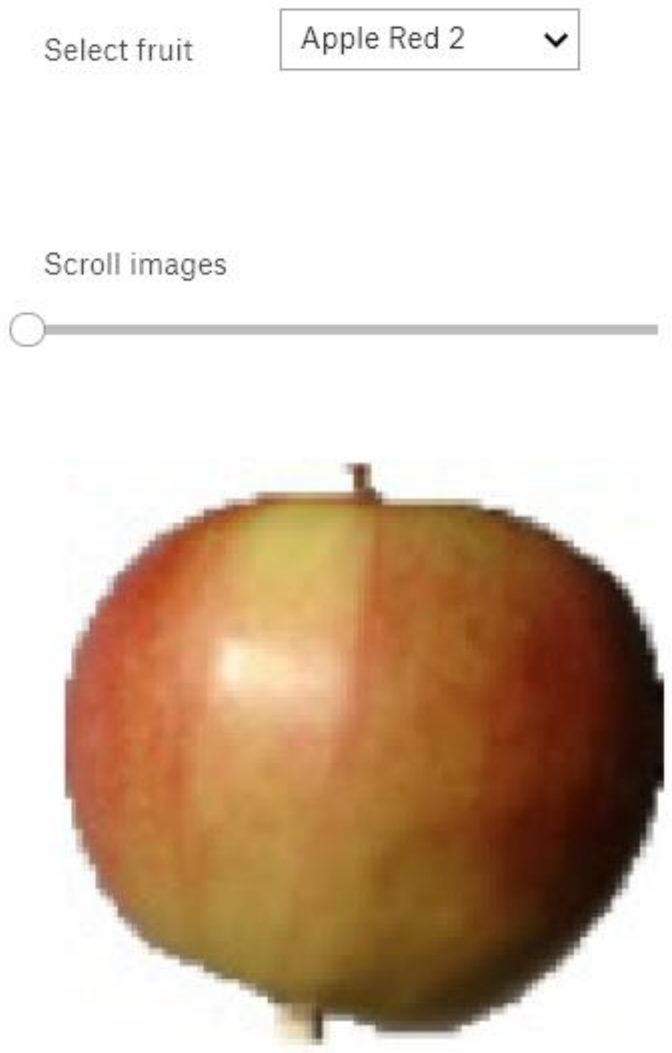
Once the data was split, the newly created data sets were compressed and saved. The datasets can be accessed via the following link:

https://www.kaggle.com/rehanali88/kaggle-for-deep-learning-p-1-getting-data-ready/output

Part-2: Exploratory data analysis and calculation of parameters required for training the neural network.

Data sets that were saved in the previous part were used to calculate certain parameters, that will be used in building and training the convolution neural network. Visual analysis of the data was also performed: the number of images in different classes were explored and an interactive plot to explore fruit images was generated.

The following image is a screen shot of the widget that can be used to explore the images in different classes.

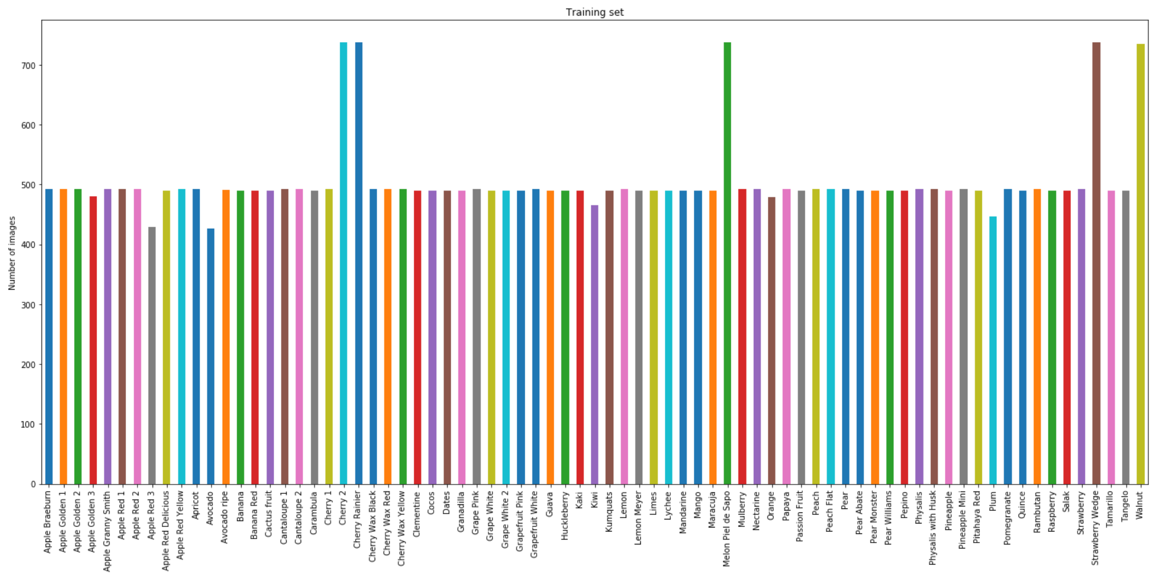


It contains a drop-down option which can be used to select a class of a fruit. The scroll bar can be used to scroll through the images in the selected class. Fruit images used by the widget are stored on Kaggle server. The widget can be used by running the following notebook on Kaggle:

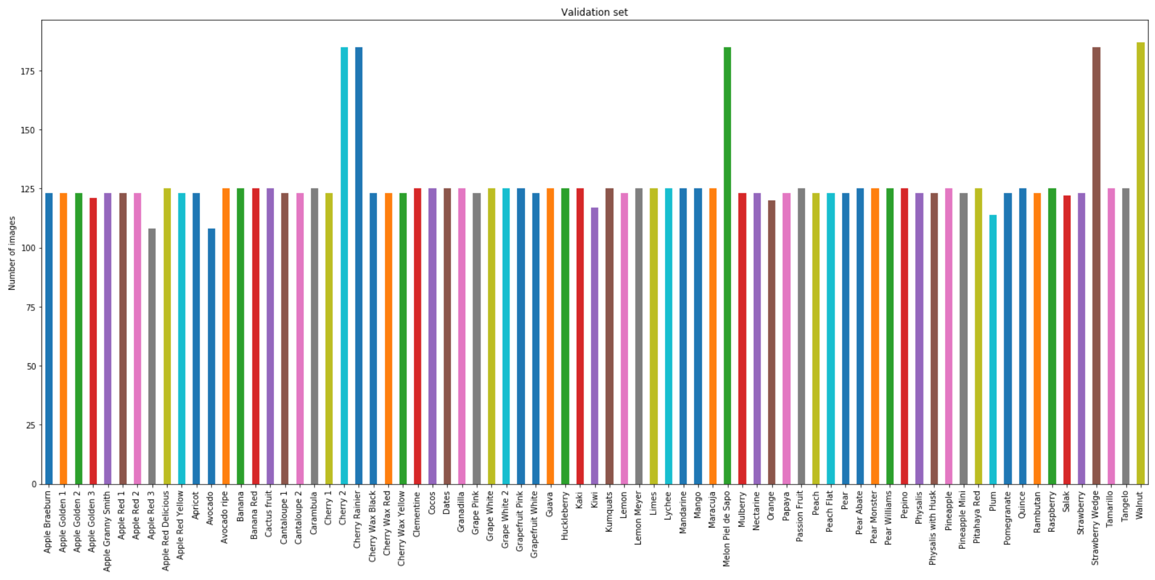
https://www.kaggle.com/rehanali88/kaggle-for-deep-learning-p-2-visualization-eda/output

As, the widget works only when run on Kaggle, the following image was generated for visual exploration of data on other platforms. It shows all the fruits in the data set.

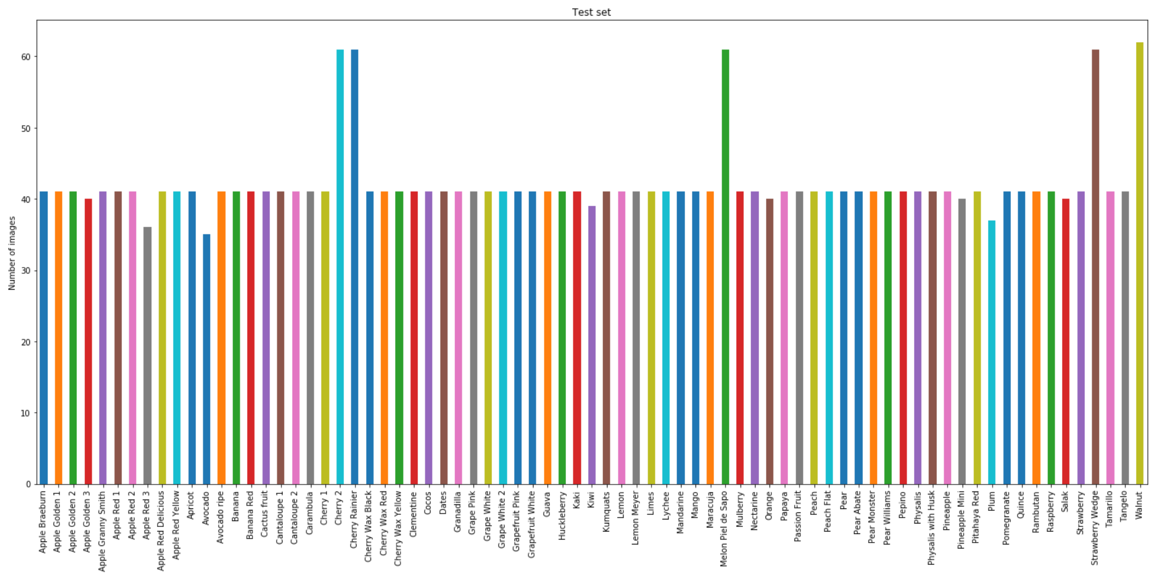
The data was explored further and bar plots of the number of fruits in each class for each set were generated.



The figure shows that in the training set there are approximately 500 images for each class with a few exceptions.



The figure shows that in the validation set there are approximately 125 images for each class. Like the training set there are a few exceptions.



The figure shows that in the training set there are approximately 40 images for each class. Like the training and validation set there are a few exceptions.

The important point to note is that there is no class imbalance. Excluding only a few classes, the number of fruits in all the classes is equivalent. This information will be important when we measure the performance of the neural network. But, before building and compiling the network, we need to calculate a few parameters.

The convolution neural network has several layers. The last layer of the network generates a prediction for the input image. For the output layer we need to specify the number of classes. For example, if we have 4 different classes, the network will output an array of length 4, containing the probability of the input belonging to each of the 4 classes. Similarly, we need to know the number of classes of fruits we have in our data set.

Apart from the number of classes, we need to calculate two other parameters: a batch size and steps per epoch.

Training images go into the model in batches. Based on the batch size we need to calculate another parameter, steps per epoch. As the name suggest, this is the number of steps required to go over all the images for each epoch. For example, if we have 100 images, and we use a batch size of 10, then the steps per epoch will need to be 10. The following formula relates the total images in a set, the batch size and steps per epoch:

$$Steps\ per\ epoch = \frac{Total\ number\ of\ images}{Batch\ size}$$

Therefore, the first calculation we performed is the number of images in each set.

$$\text{Number of training images} = 37836$$

$$\text{Number of validation images} = 9554$$

$$\text{Number of test images} = 3155$$

$$\text{Number of classes} = 75$$

Firstly, the output size for the last layer will be 75 as the number of classes is 75. Next, we calculated a number for training and validation images such that the total number of images divided by that number resulted in a remainder of zero. This number is the batch size. For training set the batch size was calculated to be 18 and for validation set the batch size was calculated to be 17. Then by using the formula for steps per epoch, mentioned above, the steps per epoch parameter was calculated.

The parameters calculated were stored in a pandas data frame and the data frame was saved as a '.csv' file. The file can be accessed via the following link:

https://www.kaggle.com/kernels/svzip/4504282

Part-3: Convolution neural network

The output from the previous part: parameters file, and the output from the first part: split data sets, was used to build a convolution neural network. The architecture of the network is as follows:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 98, 98, 32)        896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 49, 49, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 47, 47, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 23, 23, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 21, 21, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 10, 10, 128)       0
_________________________________________________________________
```

```
conv2d_4 (Conv2D)          (None, 8, 8, 128)        147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 128)        0
_________________________________________________________________
flatten_1 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)             (None, 512)               1049088
_________________________________________________________________
dense_2 (Dense)             (None, 75)                38475
=================================================================
Total params: 1,328,395
Trainable params: 1,328,395
Non-trainable params: 0
_________________________________________________________________
```

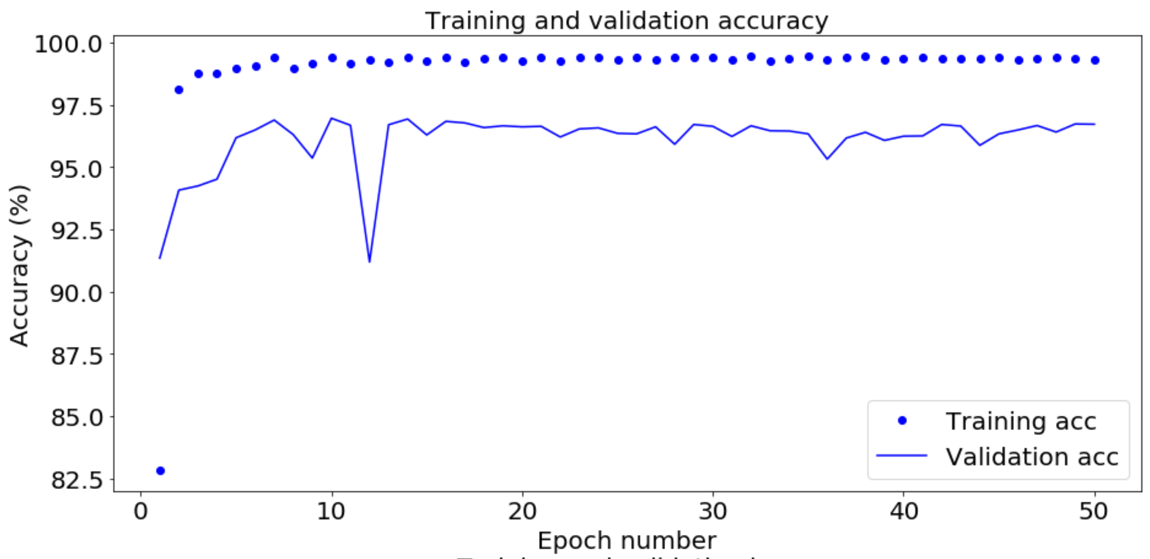
As shown in the architecture, the size of the output of the last layer is 75, equal to the number of classes of fruits.

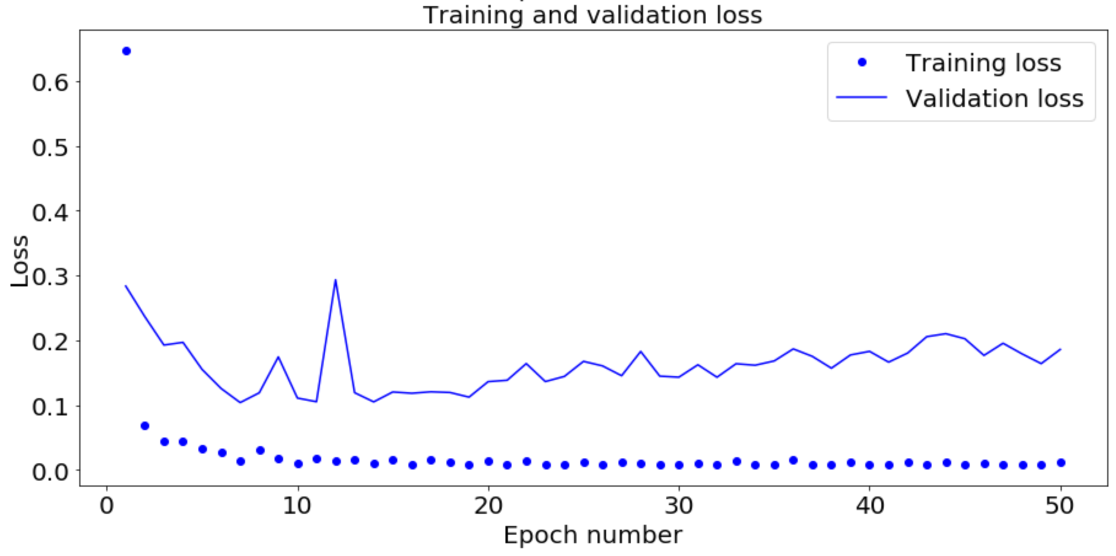
The model was compiled using categorical cross entropy as the loss function and accuracy as the measure of model performance. The optimizer used was Adam. The optimizer function’s role is to adjust the weights of the layer such that the loss is minimized. The Adam optimizer incorporates momentum in addition to other variables to determine the direction in which the weights should move.

An analogy for this is a ball rolling down a hill. Based on the momentum of the ball, it can get stuck in a region where the slope of the hill reverses. If the momentum is large enough for it to climb the height of the hill before reaching another downwards slope, the ball will not get stuck. Else it will come to a halt and this will be its stable state. A neural network aims to reduce loss such that it reaches its minimum value. This reduction is achieved by the optimizer.

The images that the network takes as the input need to be converted to tensors. In keras we can use the 'ImageDataGenerator' class to convert images to 3-Dimensional tensors. Not only are the images converted to 3D tensors they are also separated into batches. For this to happen the batch size needs to be known. This was already calculated in part 2.

Next the model was fit and the accuracy and loss of the model was monitored. The fitting needs 'steps per epoch' to be calculated. We had already calculated this in part 2. The network was trained for 50 epochs. Following plots show the network's progress.





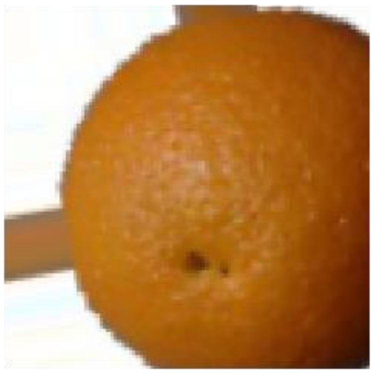
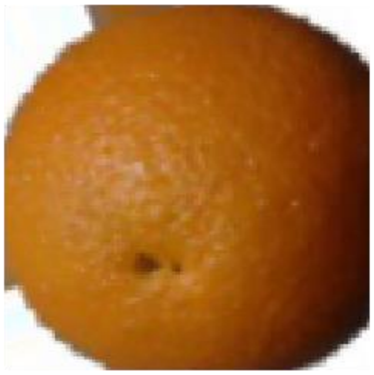
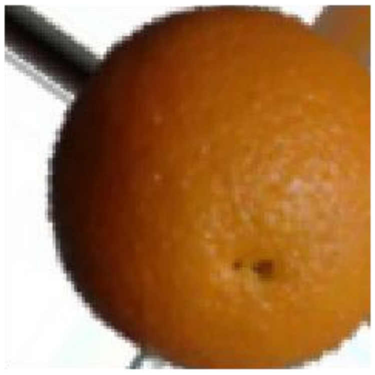
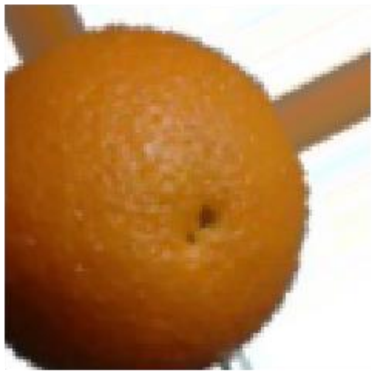
The above plots highlight the following points:

1. Validation loss shows a slight but steady increase as epoch number increases along with high variability.
2. Validation loss and accuracy never reach the values close to training loss and accuracy.
3. **We might be overfitting the data**.

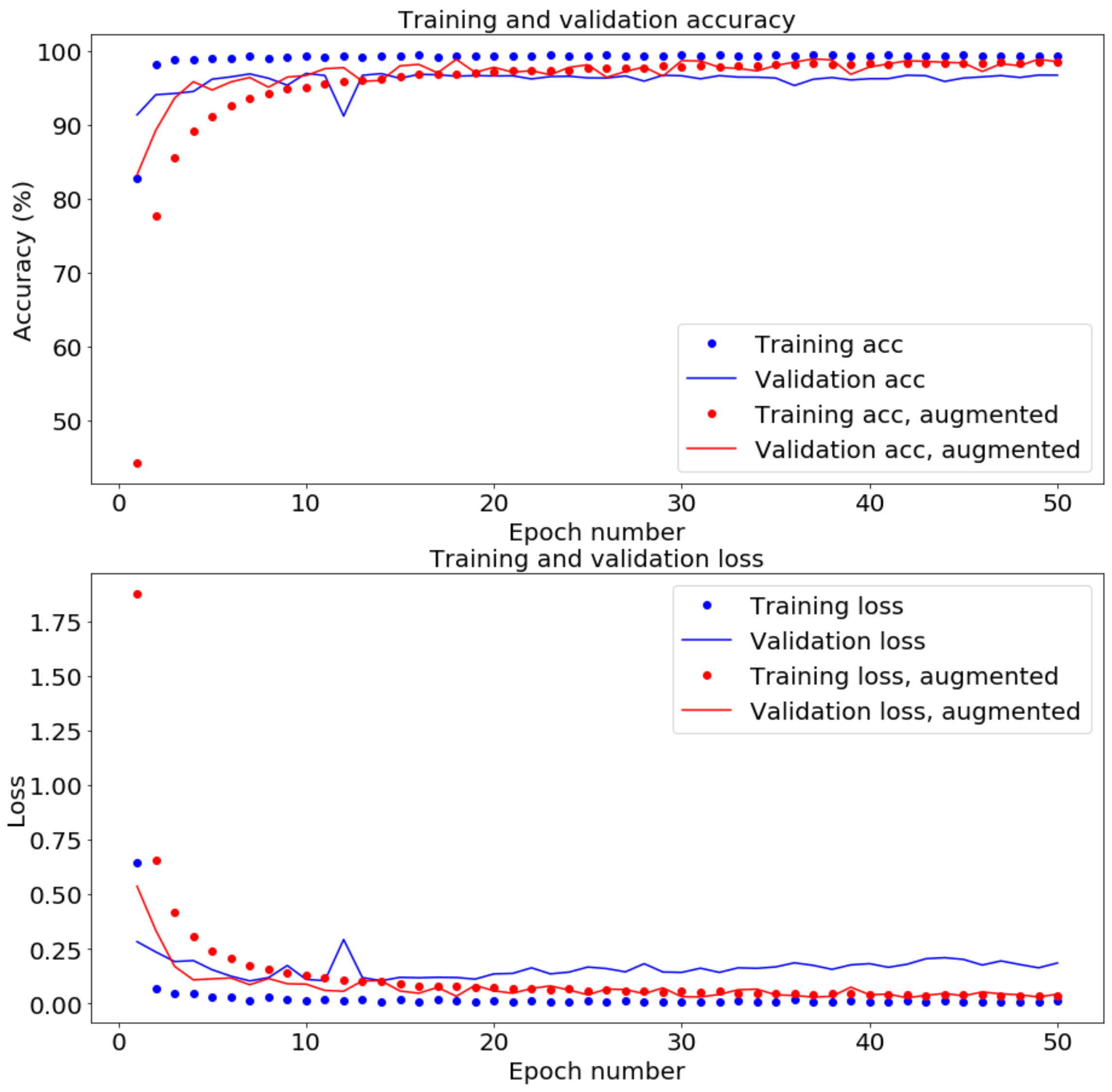
The major takeaway from the above plot’s is that the model might be overfitting the data. To overcome this, we tweaked the model architecture and used image augmentation. On the training data set model accuracy was around 96.7 %.

In the new model an additional layer was added: a drop out layer. This layer randomly removed half the values from the previous layer. The aim was to make sure that the model did not learn non-specific patterns.

We also used data augmentation. What this means is that new images from the same dataset were generated by applying certain transformations to the original images. These transformations involved image rotation, image shear, image shift etc. The aim of image augmentation is to reduce the probability of the network to encounter the same image twice. Following are four images that have been generated from a single image of an orange.

An important point to keep in mind is that augmentation was performed only for the training data. Everything else in the model was the same as the previous model. The results from the new model along with the results of the previous model are shown in the plots below.



The plots show that by tweaking the model architecture and using image augmentation we overcame overfitting. The following points support this conclusion:

1. Validation loss **DOESNOT** show a steady increase as epoch number increases. Also, the variability is low.
2. Validation loss and accuracy reach the values close to training loss and accuracy.

To compare the model performance, we evaluated model accuracy on test data. The accuracy increased from 96.7% to 98.6%. We used accuracy as a measure of model performance as there was no significant class imbalance.

The model architectures and model weights were saved. These can be accessed via the following link:

https://www.kaggle.com/rehanali88/kaggle-for-deep-learning-p-3-conv-net/output

Summary and current work

We used convolution neural network to perform multi class image classification. For easier management the project was divided into three parts. First the data was split into groups. Next, we performed visual exploratory data analysis. Lastly, we used convolution neural network to classify fruits. The accuracy for the model was 98.6%.

Not only can this project be used to classify fruits, but it also serves as a template for other image classification tasks. Moreover, it also highlights the use of Kaggle's kernels, which is crucial for numerous people who do not have access to a GPU.

Currently, we are building a web application for general use. In this application, users can upload an image of a fruit and the application will predict the class of the fruit. Moreover, the data set is being actively updated. The same model can be tweaked and used again on the updated data set.