Vegetable Image Classification using

Deep Learning Techniques

Sudalaiandi Raja Sudalaimuthu

Jayaraman Revathi

Jayasri Raghunathan

Sunil Varghese

National University of Singapore

Institute of Systems Science

Table of Contents

[1. Introduction: 3](#_Toc20775537)

[2. Problem Statement: 3](#_Toc20775538)

[3. Vegetable Classification Environment Setup: 5](#_Toc20775539)

[4. Data Set: 6](#_Toc20775540)

[**4.1** **Data Classification:** 7](#_Toc20775541)

[**4.2** **Steps to build the dataset:** 7](#_Toc20775542)

[**4.3** **Loading of dataset in the program:** 9](#_Toc20775543)

[**4.4** **Splitting of dataset in the program:** 10](#_Toc20775544)

[5. Modelling: 11](#_Toc20775545)

[**5.1** **Modelling Techniques** 11](#_Toc20775546)

[**5.2** **Design of Models** 14](#_Toc20775547)

[**5.2.1** **CNN (Veg\_Classifier\_CNN.py)** 14](#_Toc20775548)

[**5.2.2** **ResNet (Veg\_Classifier\_Resnet.py)** 16](#_Toc20775549)

[**5.3** **Processes and Techniques used to fine tune models** 17](#_Toc20775550)

[**5.3.1** **Determining Number of Convolutional layers:** 18](#_Toc20775551)

[**5.3.2** **Determining the number and size of filters, dropouts, number of neurons in the dense layer:** 19](#_Toc20775552)

[**5.3.3** **Determining optimizer and optimizer parameters:** 20](#_Toc20775553)

[**5.3.4** **Determining the epochs, batch size, initializer (He\_Normal):** 21](#_Toc20775554)

[**5.3.5** **Determining the epochs, batch size, initializer (Xavier):** 21](#_Toc20775555)

[**5.3.6** **Determining the regularizer:** 22](#_Toc20775556)

[6. Comparison analysis (Performance analysis of the models) 23](#_Toc20775557)

[**6.1** **CNN (Veg\_Classifier\_CNN.py)** 23](#_Toc20775558)

[**6.2** **ResNet (Veg\_Classifier\_ResNet.py)** 24](#_Toc20775559)

[**6.3** **Fine tuning comparison of the various models:** 25](#_Toc20775560)

[**6.4** **Veg\_Classifier\_TestingModelScripts** 25](#_Toc20775561)

[**6.5** **Ensemble (Veg\_Classifier\_Ensemble.py)** 27](#_Toc20775562)

[7. Challenges faced 28](#_Toc20775563)

[8. Findings and Conclusions 30](#_Toc20775564)

[9. Appendix 31](#_Toc20775565)

# **Introduction:**

The purpose of this project is to reliably recognize an Object (in this case a vegetable) in a shopping centre environment. Object recognition is a process of identifying a specific object in an image or video sequence. Despite the advancements in computer technologies this seemingly easy task is still a challenge for computer vision systems. On the other hand, humans tend to easily recognize an object in an image even though the object inside the image may vary in size, colour, orientation or even when partially visible.

Object recognition task is successful if the system is able to label the object based on models of known objects. For example, given an image containing an object, the system is capable of assigning the correct label to it. The recognition accuracy of the system can be calculated by comparing the result with a set of labels corresponding to a set of objects already known to the system.

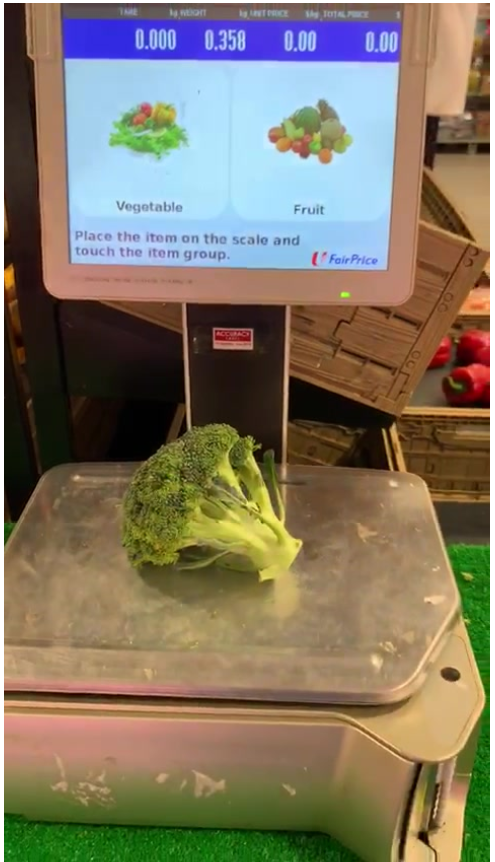
Many different approaches of object recognition exist including the traditional classifiers to deep neural network or deep learning. Deep learning is extension of machine learning algorithms that attempts to model higher level of abstractions of data by using complex architectures. The deep learning structures are typically composed of multiple layers of neurons (at least more than 3 layers) and multiple non-linear transformations.

The main objective of this project is to implement a deep convolution neural network for object classification.

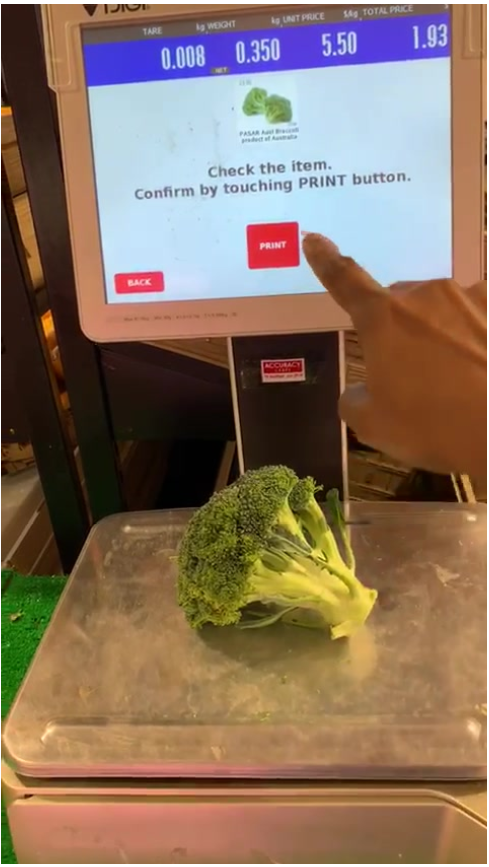
# **Problem Statement:**

If one has gone to shopping centre like NTUC FairPrice and bought a vegetable which needs to have a price sticker based on the weight then they must have come across the weighing machine set up as seen in the pictures 1.1 to 1.6 below.

After placing the vegetable on the weighing machine (1.2), one has to select the vegetable icon (1.3) and then choose the particular vegetable (1.4) followed by printing the price label(1.5 and 1.6).

1.1 1.2 1.3 1.4

1.5 1.6

We see that if we are able to correctly recognize the vegetable, we can remove the two steps (1.3 and 1.4) for price sticker printing. All then will be required will be to put a camera on the machine to take a picture once the vegetable is placed on the weighing machine. Then with our deep learning model we can identify the vegetable and print the sticker based on the weight of the vegetable.

The main objective of this project would be to implement a deep learning model which would be able to identify and classify the chosen vegetables correctly.

# **Vegetable Classification Environment Setup:**

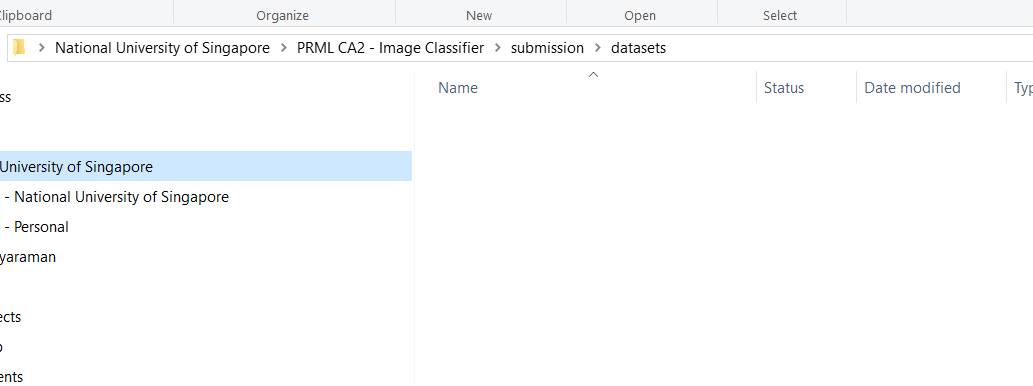
The following are the steps to run for executing the programs and it has to be done in following sequence:

Step1: The Setup Environment steps is provided in Conda setup.txt file.

Step2: Once the Conda environment is setup, extract the aivoyagers.zip to the local drive folder(aivoyagers)

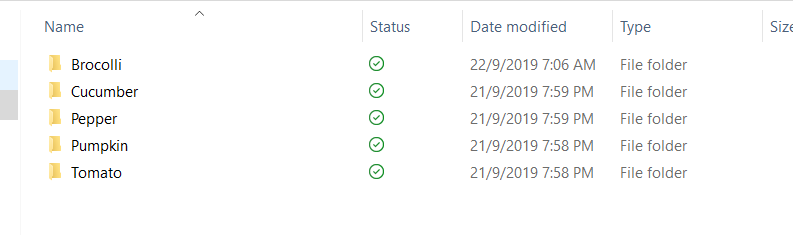
Step3: Once the file(aivoyagers.zip) is extracted, download the images from the below one drive URL to the datasets folders of extracted aivoyagers. Initially, datasets folder inside the aivoyagers will be empty.

Before dataset extraction



One drive URL: <https://nusu.sharepoint.com/sites/prmlca2/Shared%20Documents/Forms/AllItems.aspx?viewid=a9259cb6-6af2-4cb0-a60c-3d2b0686cf67&id=%2Fsites%2Fprmlca2%2FShared%20Documents%2FImage%20Classifier%2Fdatasets>

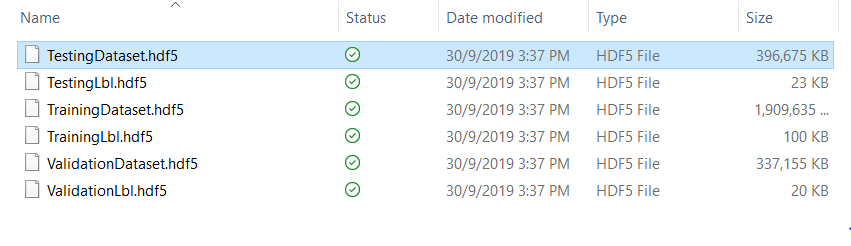
After dataset extraction, the dataset shall contain the below folder structures.



Step4:

Once the images have been extracted in datasets folders, run the following python programs in sequence for training the models:

1)For dataset preparation: Veg\_Classifier\_Dataset\_Preparation.py – It will create hdf5 files as below in the dataset\_models folder.



2) For CNN model: Veg\_Classifier\_CNN.py

3) For ResNet model: Veg\_Classifier\_Resnet.py

4) Fine Tuning model: Veg\_Classifier\_CNN\_Tuning\_\*.py

5) For Ensemble model: Veg\_Classifier\_Ensemble.py

Step5: For Testing, execute Veg\_Classifier\_TestingModelScripts.py

# **Data Set:**

The dataset (which consists of images) is used to train, validate and test our models. These are scraped from the internet as well as converted from videos we have physically taken from the various shopping centres around Singapore.

## **Data Classification:**

Our dataset images are classified under these 5 vegetables, namely:

1. Broccoli [1103]
2. Cucumber [1252]
3. Pepper (i.e. Capsicum) [1669]
4. Pumpkin [1088]
5. Tomato [1772]

The number of images each class has is given in square brackets above. In the initial stages we started of with 10 classes, but found a balance between the images required and the accuracy desired to be optimum around 5 classes.

As part of the pre-processing, the images were cleaned up by removing cross-contaminated images i.e. those images having two or more of the above vegetables in the same image. Also, judgement was exercised to remove images where excessive noise was perceived in the image.

## **Steps to build the dataset:**

Two methods to gather the images were adopted:

1. From the internet:

Images for vegetables were scraped from the internet primarily from the websites—

The software and codes used to scrape the imaqes are given below:

1. Programs for Image downloading from Internet

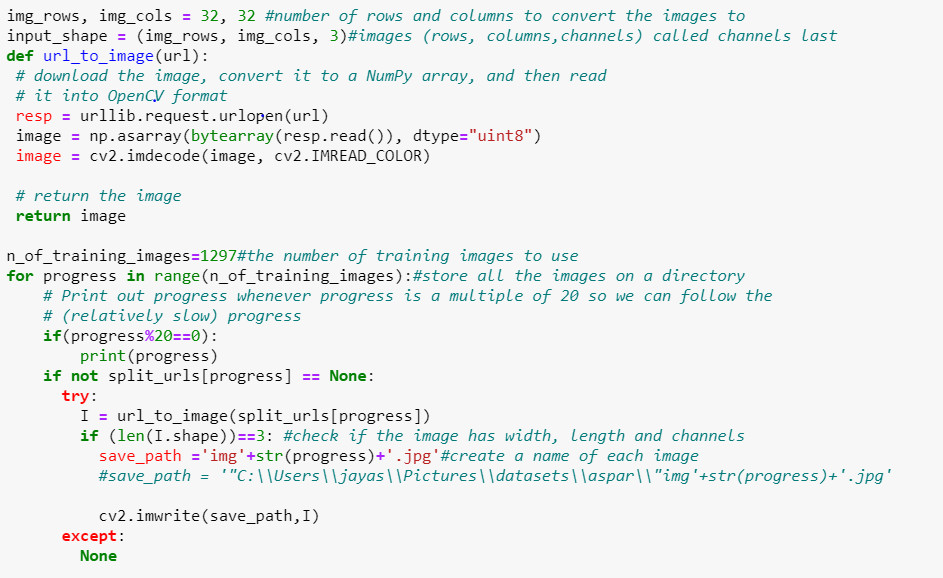
* Web scraping using PythonBeautifulSoup from PEXEL. This was partially successful as ‘Forbidden 403’ error was encountered after sometime.
* Web scraping from ImageNet enabled downloading images of vegetables
* Web scraping using Bing API was not successful, and ‘Access Denied’ was encountered

1. Tools used for Image downloading from Internet

* Google Image downloader
* Outwit Images



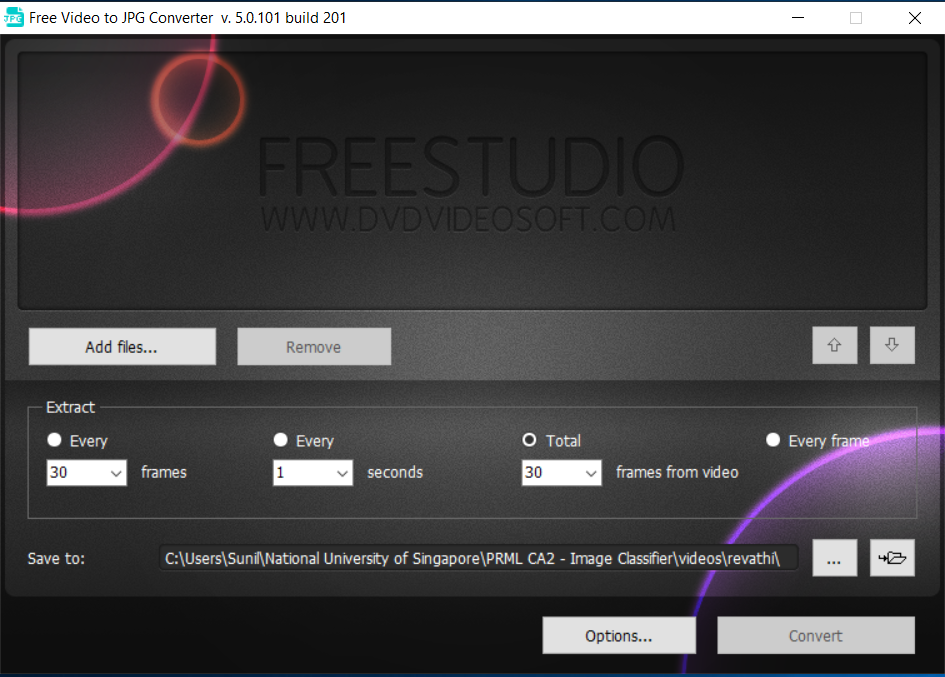




1. From videos taken from the vegetable shops:

The team members took videos and pictures from various shopping centres like Fairprice, Cold Storage, Sheng Siong, Mustafa, Vegetable shops at wet markets etc.

These videos were then converted into images by using Freestudio’s “Video to JPG converter” (shown below) tool with an optimum extraction rate.

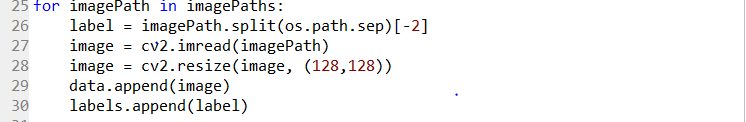


## **Loading of dataset in the program:**

To grab all the image paths from the original image directories (refer to Veg\_Classifier\_Dataset\_Preparation.py):



Next we loop over the image paths to extract the class label from the filename, load the image, and resize it to be a fixed 128x128 pixels, ignoring aspect ratio and then update the data and labels lists respectively.



Next convert the data into a NumPy array, then pre-process it by scaling all pixel intensities to the range [0, 1]

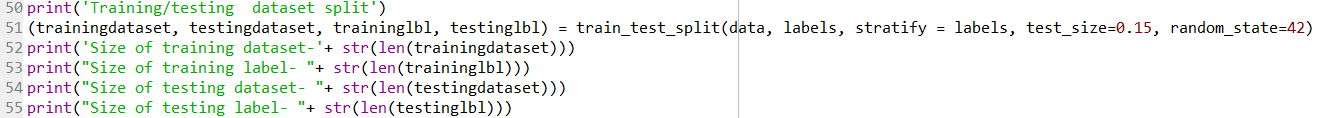


Encode the labels (which are currently strings) as integers and then one-hot encode them

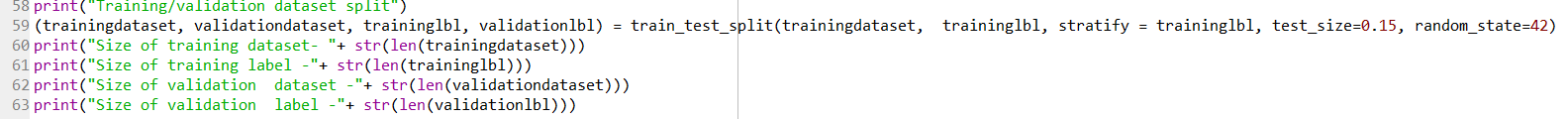


## **Splitting of dataset in the program:**

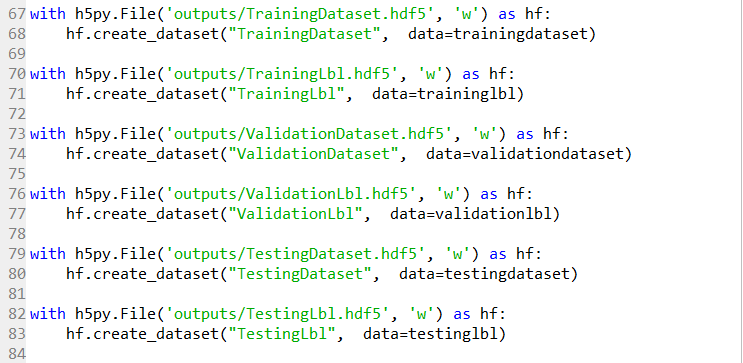
Partition the data into training and testing splits using 85% for training and 15% for testing



Partition the training data into a further split of training and validation using 85% for training and 15% for validation



The data is divided and stored as training dataset, validation dataset and testing dataset in h5 files for model generation purposes.



# **Modelling:**

For this project primarily 3 different modelling techniques have been tried:

1. Convolutional Neural Networks(CNN)
2. Residual Networks (ResNet)
3. Stack Ensemble

More details on these techniques are provided below.

## **Modelling Techniques**

1. Convolutional Neural Networks (CNN):

Computational models of neural networks have been around for a long time. Neural networks are made up of a number of layers with each layer connected to the other layers forming the network. In a typical neural network, the neurons are connected in a directed way having a clear start and a clear stop i.e., the input layer and the output layer. The layers between these two layers, are called as the hidden layers as shown in Figure 1. Learning occurs through adjustment of weights and the aim is to try and minimize error between the output obtained from the output layer and the input that goes into the input layer. The weights are adjusted by process of back propagation. The process of weight adjustment is repeated in a recursive manner until weight layer connected to input layer is updated.

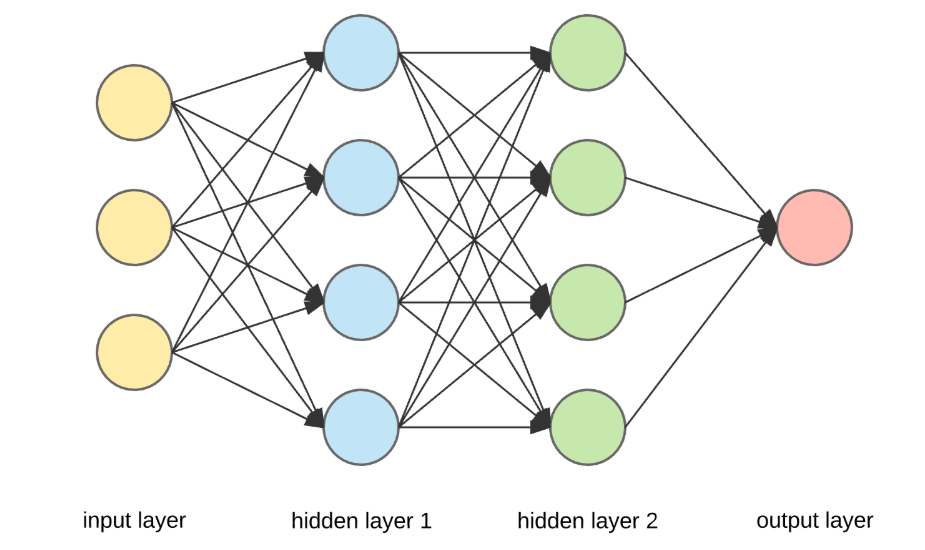


Figure 1: Typical network architecture

Convolutional Neural Networks (CNN) is a variant of Multilayer Perceptron (MLPs) which are inspired from biology. Convolutional neural networks are better suited to process two-dimensional (2-D) image.

A general CNN architecture looks like the one shown in Figure 2 and consist of distinct types of layers. The convolutional layer applies an array of weights to all of the input sections from the image and creates the output feature map. The pooling layers simplify the information that is found in the output from the convolutional layer. The last layer is the fully connected layer that oversees the gathering of the findings from former layers and provides an N-dimensional vector, where N stands for the total number of classes

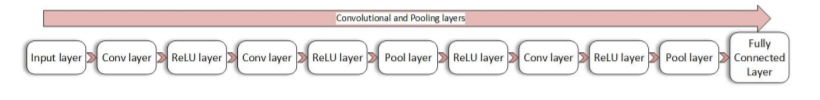


Figure 2: – A General CNN layer hierarchy

1. Residual Networks (ResNet)

Deep convolutional neural networks have led to a series of breakthroughs for image classification. So, over the years there is a trend to go more deeper, to solve more complex tasks and to also increase /improve the classification/recognition accuracy. But as we go deeper; the training of neural network becomes difficult and also the accuracy starts saturating and then degrades also. Residual Learning tries to solve both these problems.

In general, in a deep convolutional neural network, several layers are stacked and are trained to the task at hand. The network learns several low/mid/high level features at the end of its layers. In residual learning, instead of trying to learn some features, we try to learn some residual. Residual can be simply understood as subtraction of feature learned from input of that layer. ResNet does this using shortcut connections (directly connecting input of nth layer to some (n+x)th layer. The skip connection in the diagram in figure 3 is labelled “identity.” It allows the network to learn the identity function, which allows it pass the input through the block without passing through the other weight layers. It has proved that training this form of networks is easier than training simple deep convolutional neural networks and also the problem of degrading accuracy is resolved to some extent.

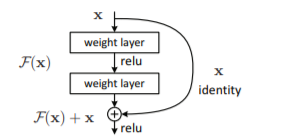


Figure 3. Residual learning: a building block

1. Stack Ensemble

[Ensemble modelling](https://www.sciencedirect.com/topics/computer-science/ensemble-modeling) is a process where multiple diverse models are created to predict an outcome, either by using different [modelling algorithms](https://www.sciencedirect.com/topics/computer-science/modeling-algorithm) or using different training data sets. The ensemble model then aggregates the prediction of each base model and results in once final prediction for the unseen data.



For Vegetable classification, Average Ensemble technique are used to produce a single desired output, in theory an ensemble of models performs better than any individual model. In Average Ensemble, the various errors of the models are "averaged out.".  It helps to reduces the variance in a final neural network model and helps in increasing the accuracy.

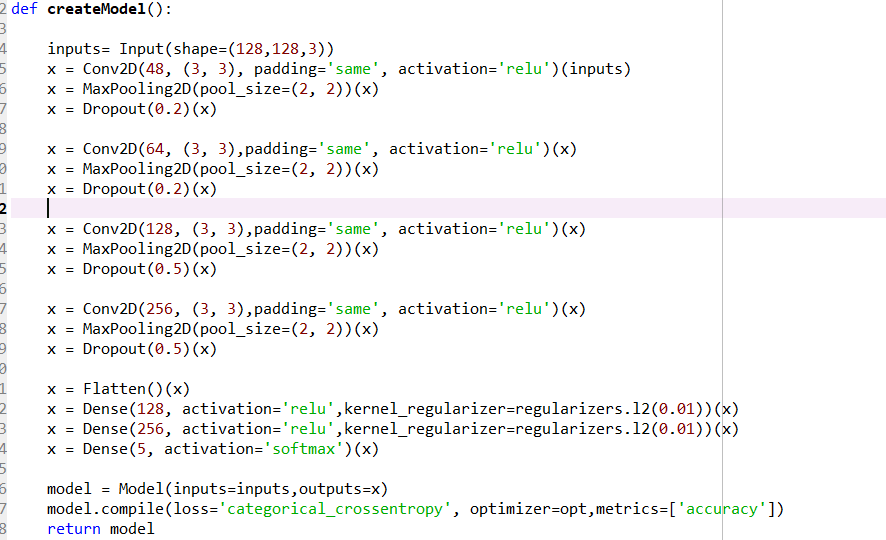
## **Design of Models**

In this section the various models in the programs are explained. Each of the below program creates the following artefacts. These artefacts are attached in the appendix in the following folders:

1. pdfmodels – To capture the model architecture
2. csvlogger – To log the validation loss and validation accuracy of each epoch
3. models – To capture the training models of each technique
4. html – To capture the console output

### **CNN (Veg\_Classifier\_CNN.py)**

**CNN** is used for feature extraction, which can detect basic patterns through edge detection, parts of the objects and complete objects detection at various stages. It has advantages of parameter sharing and extracts location invariant features in images. CNN is based on the concept of Local connectivity, as each neural connected only to a subset of the input image (unlike a neural network where all the neurons are fully connected). This helps to reduce the number of parameters in the whole system and makes the computation more efficient



**Max pooling** is used for progressively reducing the spatial size of the representation.

There are 3 blocks of CONV2D + Max Pooling layers + dropout in the model. The output of the CNN will be the input for the 3 fully connected layers. There are 3 Dense layers, the activation being relu for the first 2 layers.

The activation for the last layer is [SoftMax](https://en.wikipedia.org/wiki/Softmax_function), which is used for predicting a single class of 5 mutually exclusive classes.

**Dropout** is used to avoid overfitting, which will drop random neurons during each iteration

**Optimization** is performed using Adam, as it combines the advantages of both RMSprop and gradient descent with momentum. Adam implements the exponentially moving average of the gradients to scale the learning rate. It keeps an exponentially decaying average of past gradients. Adam is computationally efficient and has very little memory requirement. The default parameters of learning rate, beta1 and beta2 are used

The training is by **Mini batch** size of 64 samples. Since a subset of training examples is considered during each step, it can make quick updates in the model parameters and can also exploit the speed associated with vectorizing the code.

As the dataset is smaller, real time **Data Augmentation** is implemented using Keras Image Data Generator. This Image Generator randomly shifts the width horizontally and vertically, rotate the image by the given angle, horizontally and vertically flips the images

**L2 Regularizer**, which applies penalties on layer parameters during optimization, that helps in controlling the overfitting of the model to the training data, so that the model generalizes well to the test and production **data.**

**Activation** function choice are relu for better weight update. Tanh and Sigmoid are not used as the exponentially diminishing gradients can cause many static or dead neurons in the earlier layers.

### **ResNet (Veg\_Classifier\_Resnet.py)**

There are 4 set of ResNet blocks, the first set is a simple resblock and next 3 of them are down res block sets . In the simple resblock set, all the blocks are simple resblocks, while in the downresblock set, the first block is a downresblock and all the other blocks are simple resblocks.



Each block has 2 sets of conv2d layers followed by batch normalisation and Relu activation. In the down resblock, the stride is 2 for the first part of the block, while for the simple resblock, the stride is 1. The successive resnet blocks have 16,32,64 and 128 convolutional filters of size 3,3

Because of the passing of the identity function and batch normalization, Resnet performs better than the CNN.

## **Processes and Techniques used to fine tune models**

Only the CNN models were tuned in the project as ResNet models takes a long time to find tuning parameters. A better approach would be to find the tuning parameters from CNN and apply it to ResNet.

Dropout value is tuned in a such a way that dropout value is smaller in the initial layers and higher in the subsequent layers. CNN filter size is tuned with size choice of (3 & 5). The number of filters choices are used as 128 and 256 which are related to the size of the input image. The initializer choice is between Xavier and He\_normal.

The L2 regularizer is used for preventing overfitting and the lambda choices are given in the logarithmic scale (0.1, 001 and 0.001).

‘He\_normal’ and ‘Xavier’ are used for the initializer tuning. Expectedly the model chose Xavier initializer as it is the most suitable initializer for the chosen activation function

Optimizer choices are Adam and rmsprop are used.

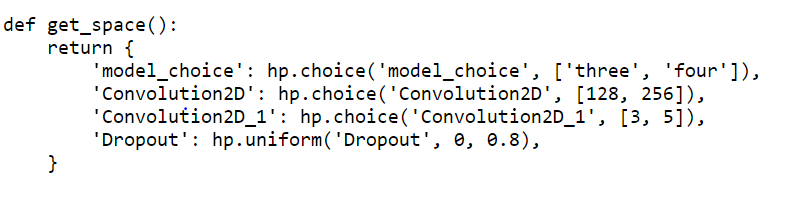
Learning rate decay has been defined as a step decay function in  [Learning RateScheduler](https://keras.io/callbacks/#learningratescheduler)  callback and return the updated learning rates for use in the optimizer.

### **Determining Number of Convolutional layers:**

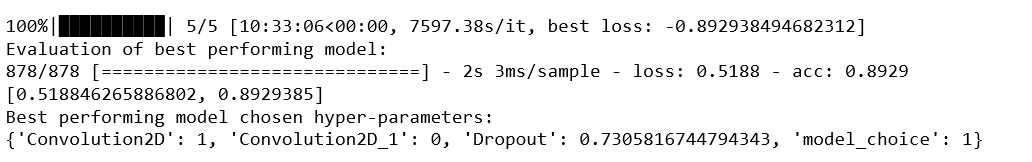
Program : Veg\_Classifier\_CNN\_Tunning\_Add\_Layer.py



The search space is as per the following:

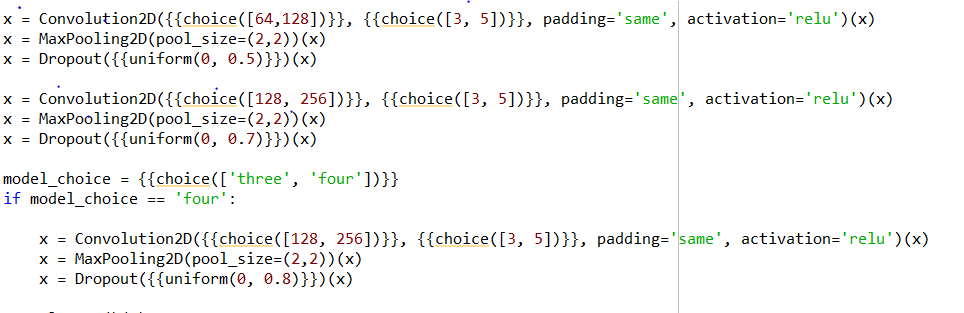


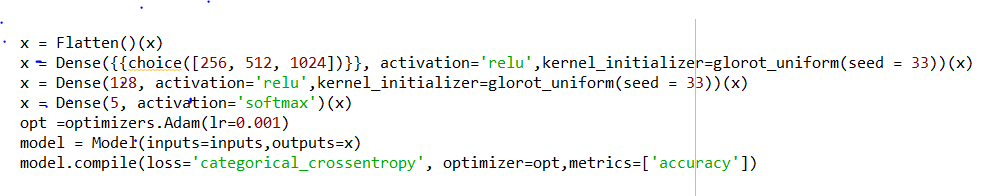
The output of the tuning program – best validation accuracy is 0.8929



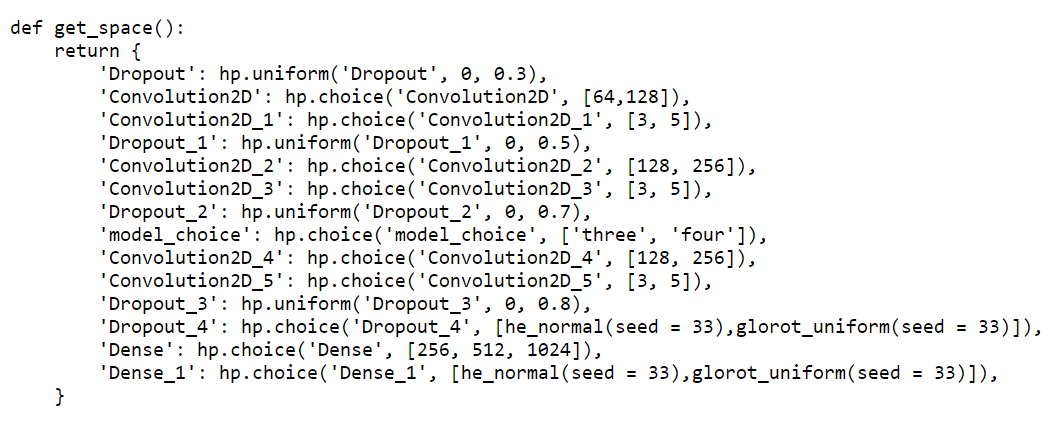
### **Determining the number and size of filters, dropouts, number of neurons in the dense layer:**

Program : Veg\_Classifier\_CNN\_Tunning\_Add\_Layer\_Conv\_Choice.py

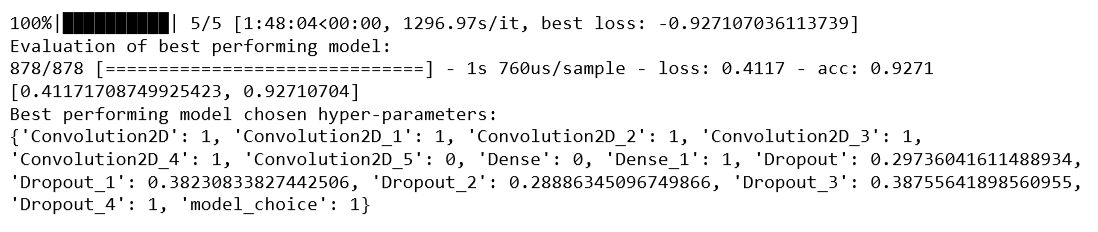




The search space is as per the following:

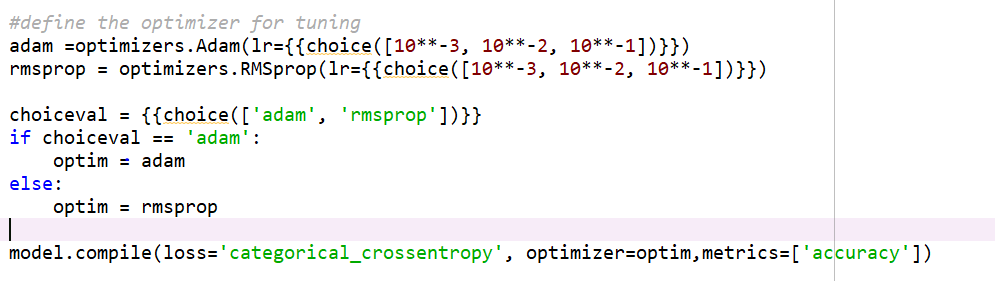


The output of the tuning program – best validation accuracy is 0.9271

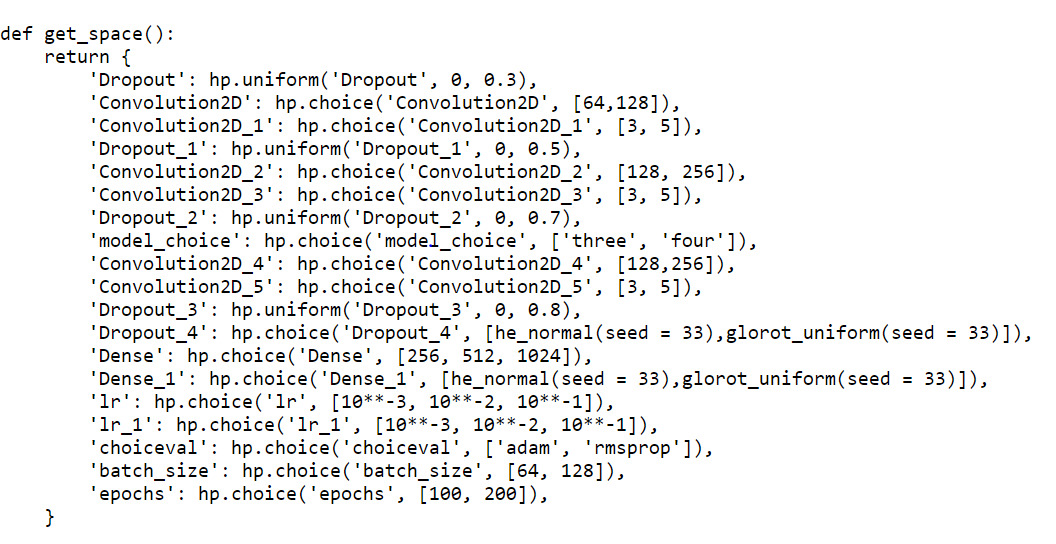


### **Determining optimizer and optimizer parameters:**

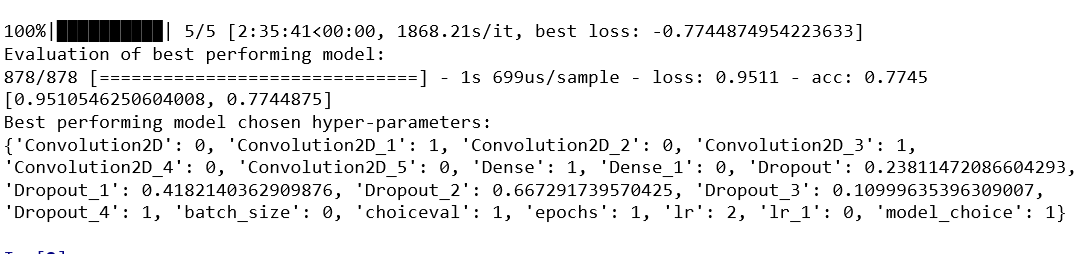
Program: Veg\_Classifier\_CNN\_Tunning\_Add\_Layer\_Conv\_Epoch\_optimizer.py



The search space is as per the following:

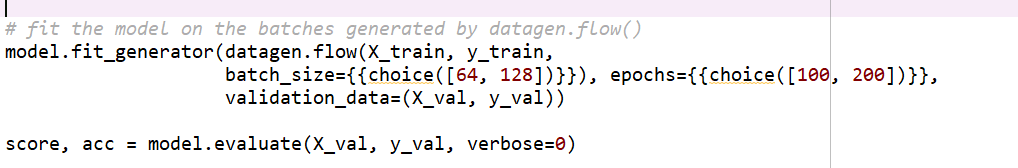
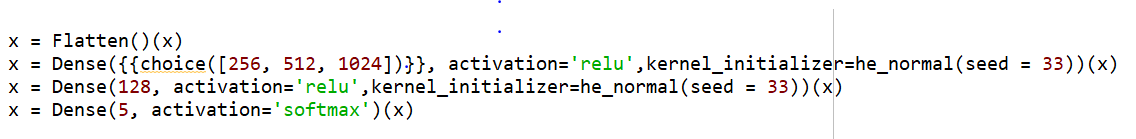


The output of the tuning program – best validation accuracy is 0.7745

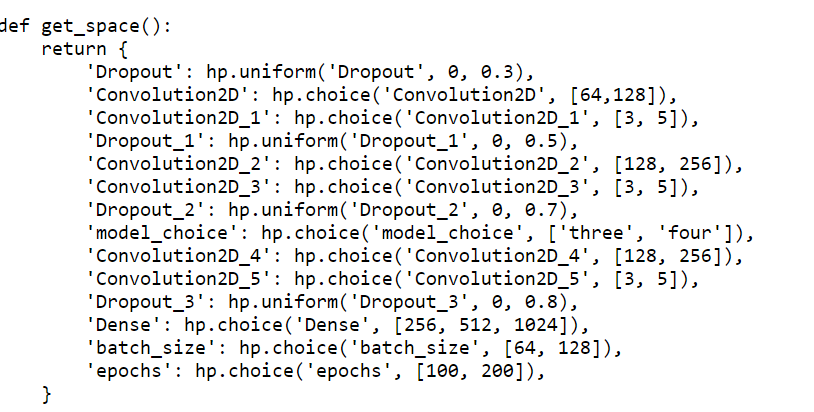


### **Determining the epochs, batch size, initializer (He\_Normal):**

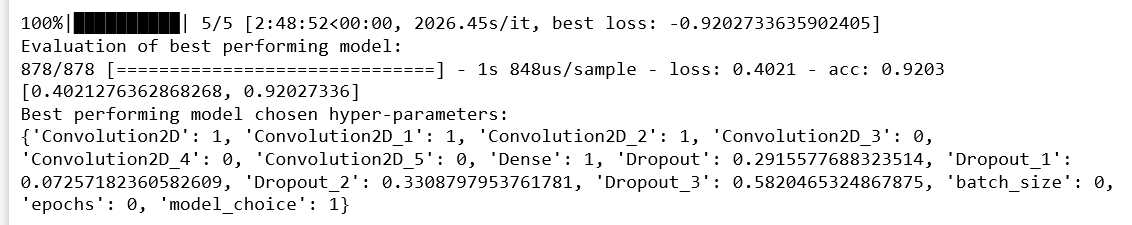
Program: Veg\_Classifier\_CNN\_Tunning\_Add\_Layer\_Conv\_Choice\_he\_normal.py



The search space is as per the following:

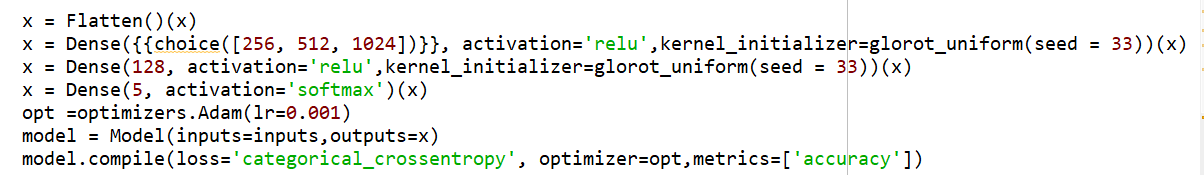


The output of the tuning program – best validation accuracy is 0.9203

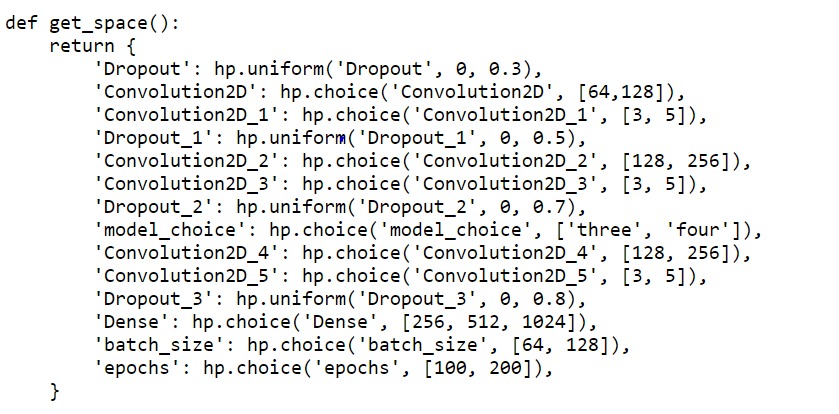


### **Determining the epochs, batch size, initializer (Xavier):**

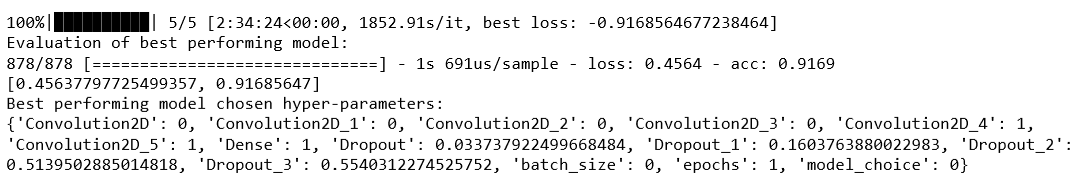
Program: Veg\_Classifier\_CNN\_Tunning\_Add\_Layer\_Conv\_Choice\_Xavier.py



The search space is as per the following:

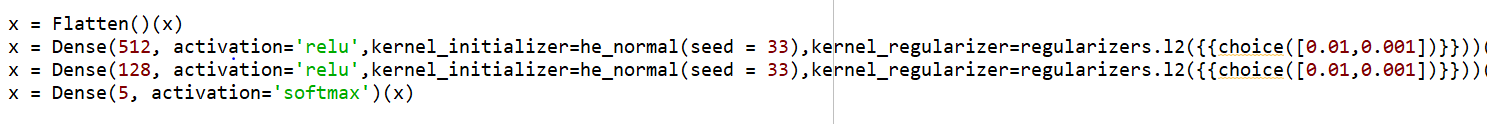


The output of the tuning program – best validation accuracy is 0.9169

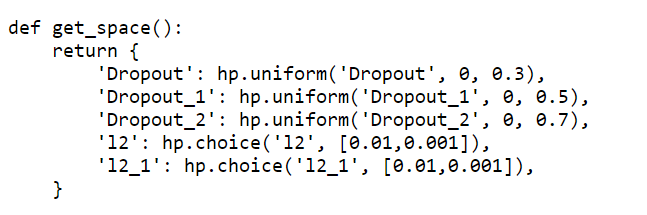


### **Determining the regularizer:**

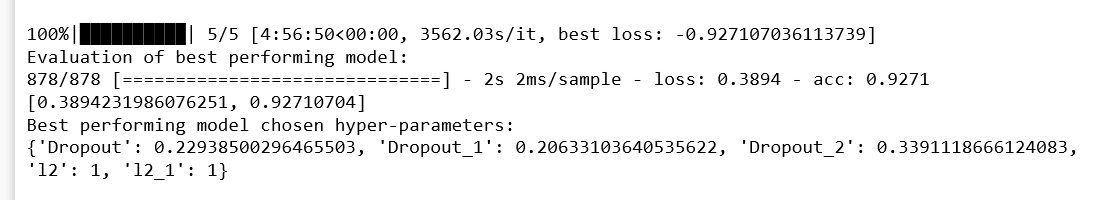
Program: Veg\_Classifier\_CNN\_Tunning\_Add\_Layer\_Conv\_Choice\_he\_normal\_regularizer.py



The search space is as per the following:



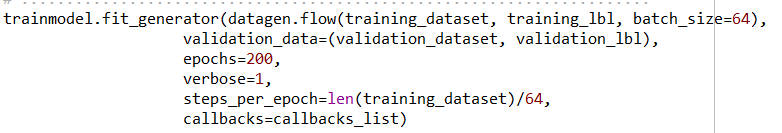
The output of the tuning program – best validation accuracy is 0.9271



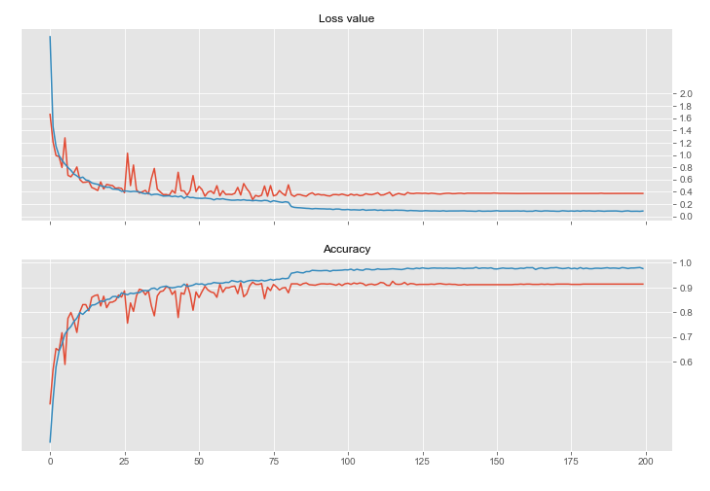
# **Comparison analysis (Performance analysis of the models)**

## **CNN (Veg\_Classifier\_CNN.py)**

Once the CNN model was ready, we trained the model with below parameters and validated against the validation dataset.



**On training the dataset, CSV logger saved the loss and accuracy scores in excel ((/csvlogger/** **Veg\_Classifier\_CNN) for the training and validation dataset. Using the generated csv data, the below graphs are plotted for the training and validation dataset.**

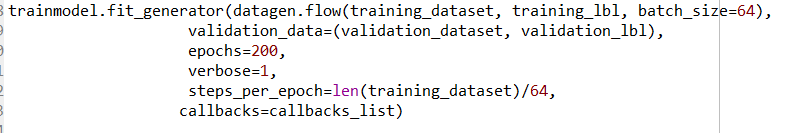


**On analysing the graph, initially the model shows some unstable behaviour but after around 75 epochs, the model shows good stability between training and validation accuracy as well as the loss.**

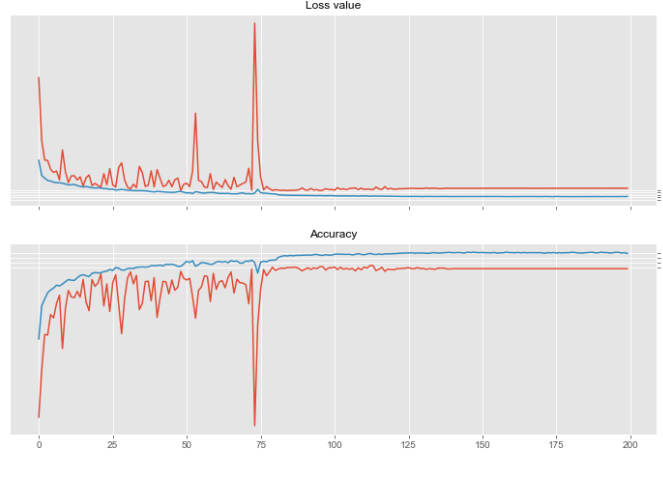
**After the run the training accuracy reached 0.976473 and training loss reached 0.085427. At the same time, the validation accuracy reached 0.91343963 and validation loss reached 0.372148274.**

## **ResNet (Veg\_Classifier\_ResNet.py)**

Once the ResNet model was ready, we trained the model with below parameters and validated against the validation dataset.



**On training the dataset, CSV logger saved the loss and accuracy scores in excel** **(/csvlogger/** **Veg\_Classifier\_Resnet) for the training and validation dataset. Using the generated csv data, graphs were plotted for the training and validation datasets and the results are as below.**



**On analysing the graph, initially the model shows very unstable behaviour compared to the CNN model, but around 75 epochs, the model shows good stability between training and validation accuracy as well as the loss.**

**After the run the training accuracy reached 0.9887392 and training loss reached 0.134056953. At the same time, the validation accuracy reached 0.9259681and validation loss reached 0.476334846.**

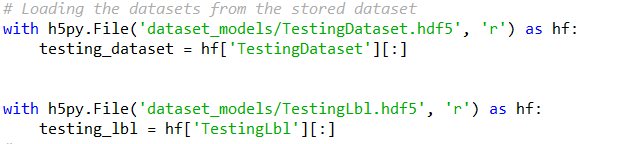
## **Fine tuning comparison of the various models:**

**A summary of the fine tuning done on the various model is captured in the excel sheet. This is appended in the Appendix.**

## **Veg\_Classifier\_TestingModelScripts**

Once all the models were generated, used the Veg\_Classifier\_TestingModelScripts.py to compare the accuracy score of each model.

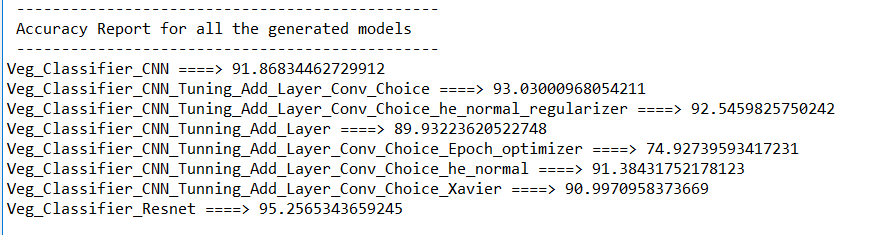
1. Loaded the testing dataset to predict the accuracy score of all the generated model including fine tuning of hyper parameters models also.



1. Loaded all the trained models to predict against the testing dataset.



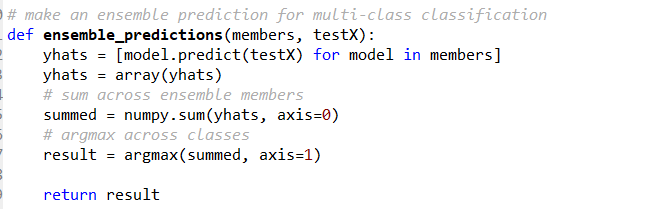
1. Generated the classification report for all the models and it is saved in the /html/ Veg\_Classifier\_TestingModelScripts.html
2. Accuracy report for testing dataset were generated as below.



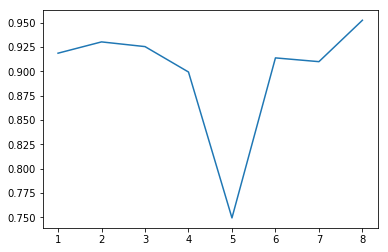
Accuracy report shows that the model generated by Resnet is higher for testing dataset compared to all the models.

## **Ensemble (Veg\_Classifier\_Ensemble.py)**

1. Make an ensemble prediction for all the models generated for our Vegetable Image Classification



1. Generated the accuracy report all the models and noticed that the variation in Veg\_Classifier\_CNN\_Tunning\_Add\_Layer\_Conv\_Epoch\_optimizer.py, which it has averaged by the average ensemble and gives the accuracy report as 94.58 as whole.





# **Challenges faced**

1. Datasets

Initially collected the dataset from the internet for all the vegetables. When we ran the scripts to downloaded images, some of the websites were authenticated and not allowing to download the images without using their APIS.

Tried with automatic tool download and able to receive lot of images. But noticed that cooked dishes with the vegetable were also downloaded. Noticed that the vegetables in the farms were also retrieved. It was painful to clean the above images manually.

We ran the simple CNN model for the collected model from the web and noticed that the accuracy came only around 50%.

Even after cleaning up the images we found the accuracy was still not good. Since our problem statement is restricted to vegetable classification in a supermarket environment. We decided to get clean images from the supermarket itself. We took the videos of vegetables from the shops and then converted the video in to images.

Tried again with CNN model and noticed that the accuracy went up to 70% only. After inspecting the images found that there was very less variation among the images.

So, we tried taking the videos by changing the angle and zooming and then converted the videos in to images.

Tried again with CNN model and noticed that the accuracy increased to 85%. Lesson learnt was that the dataset is very important for the accuracy of the model.

1. Hyperparameter tuning

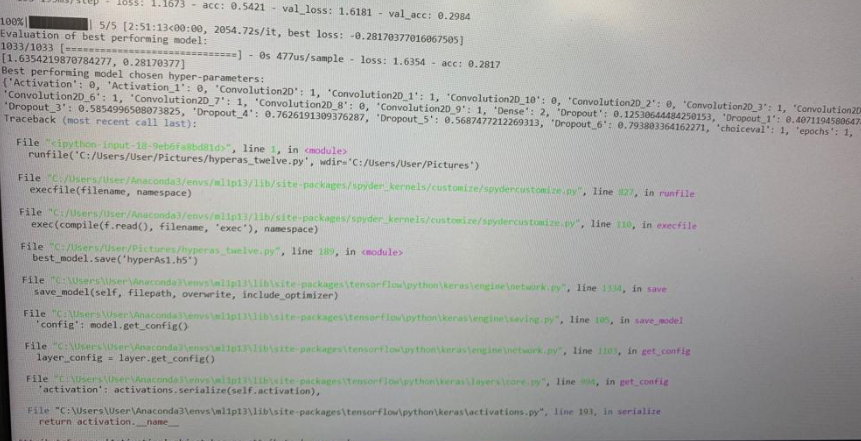
Searched in the internet for the hyperparameter tuning and it has mentioned that the Grid Search will give the best option. Hence tried with Grid search and noticed that the program was running for 2 to 3 days and still could not give the results.

As the Grid search does not make use of GPU acceleration it is not able to achieve higher speeds even after parallelizing.

Tried with Random search and it was able to give the results in lesser time. But again, the search parameter was taken randomly and the parameter may not be best.

So again, searched and found that the other method which we were using (Hyperas) which enables us to take advantage of GPU acceleration enables us to train models at least 10x faster and was also an easy way to approach hyper-parameter tuning.

When we used hyperas for parameter tuning, one more hurdle was waiting for us. “Model could not be saved because of the below error”.



Tried a lot and noticed that the Class Activation has to be serialised for saving and hence used “activation” as a parameter in the layer to avoid the model saving issue. Finally, all our tuning model were able to save.

1. Time taken for execution for hyper parameter tuning

Initially hyper parameter tuning program took 2-3 days to run a single run. In the end we were able to get the output of a single hyperparameter tuning program in around 2.5 to 4 hrs. If any error appeared, we needed to run this again and this was challenged we faced. Fortunately we all have laptops with GPU chip.

1. Stacking ensemble.

When we tried stacking ensemble, noticed that the sample program given was for Logistics Regression and it was giving bad input error because of multiclass classification. But because of time constraints, couldn’t explore much and used average ensemble program only.

# **Findings and Conclusions**

1. The testing accuracy increased after the dataset image numbers were suitably increased and cleaned. The testing accuracy of CNN model increased from the 1st iteration of 50% to almost 85% just by changing and increasing the number as well as the quality of the images.
2. In ResNet, overall validation accuracy is better than CNN. However, the time taken for ResNet to train the model is double than that of CNN. Noticed also the training model stability is lower for ResNet in early stages of training
3. For CNN and ResNet, noticed that around 75 to 100 epoch itself, the network is trained.

.

1. Accuracy report shows that the model generated by Resnet (95.25%) is higher for testing dataset compared to all the models.
2. Among CNN models after fine tuning process of the testing dataset, model generated by the Add\_Layer\_Convolution (93.03%) has given the best accuracy, but the he\_normal\_regularizer option was the best in validation dataset.

# **Appendix**

Folder structure of our submission are explained below -

1. Submission root folder - All the python (executable) and reports document for our Vegetable Image Classification were placed in the root folder.
2. datasets – Contains all the images used by the Vegetable Image Classification
3. html – Contains all the console outputs for all the executable python scripts
4. models – Contains all the training models of each the techniques used, including the fine-tuning hyper parameter programs.
5. others – Contains the tools used for web scrapping and scripts used to collect the images from the internet.
6. pdfmodels – Contains all the model architectures for each technique.
7. conda setup – Explains the environmental setup.
8. Summary of Fine-Tuning Analysis of CNN parameters.xls – Explains the summary of the findings for all options which we tried for fine tuning analysis.