UNIVERSITÀ DEGLI STUDI DI MILANO

Master of Science in Data Science and Economics

Image classification with Neural Networks

Veronica Astorino

ID number: 942307

"Your brain does not manufacture thoughts. Your thoughts shape neural networks."
Deepak Chopra

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

TABLE OF CONTENTS:

LIST OF FIGURES:

INTRODUCTION

A Neural Network is the representation of what the human brain does, neurons interconnected to other neurons which forms a network. The identification of an image is natural for people, but for a computer is hard. Humans have been able to recognize the environment since they were born, differently, computers need effort and forceful computation in addition to complex algorithms to effort to understand in a correct way patterns and regions wherever a possible object could be.

The aim of this research is to explore several neural networks techniques that are used for image classification and establish the best architecture that is able to solves the problem of fruit and vegetables recognition.

For the exact purpose of answer to these questions, the project consisted of two parts. As the first approach I did some pre-processing for the aim of performing the CNN architecture models. Therefore, I created a training and test folders in order to classify images into 10 classes (i. e., apple, banana, plum, pepper, cherry, grape, tomato, potato, pear, peach), doing that I unified all the different varieties of a particular fruit into a single class. Subsequently, I converted all images to RGB, and I divided by 255, that is because RGB (Red, Green, Blue) are 8 bit each. The range for each individual color is 0-255. Dividing by 255, the 0-255 range can be described with a 0.0- 1.0 range where 0.0 means 0 and 1.0 means 255. Then, with the aim of having better results, I zoomed and resized images, proceedings that I reduced noise using flow to directory.

In the second part, I applied few network Architectures and inspected Overfitting and Underfitting by comparing accuracy and loss curves of train and test data; I turned Hyperparameters, and then I checked the performance of each predictor through zero one loss and image classification.

CHAPTER 1: "PRE-PROCESSING"

This data set contains images of fruits and vegetables. First, I reduced 131 classes into 10 by unifying all the different variants of fruits and vegetables into a single class.
Secondly, I created train and test generator (Figure 1) with flow from directory converted JPG images to RGB and rescaled them by dividing with 255. I selected batch size equal to 50. That is because it depends on different factors, larger batches provide a more accurate estimate of the gradient, but with less than linear returns.

```
[14] train_datagen = ImageDataGenerator(rescale = 1./255,
                        shear_range = 0.3,
                        horizontal_flip = True,
                        zoom_range = 0.3)
     test_datagen = ImageDataGenerator(rescale = 1./255)

     train_generator = train_datagen.flow_from_directory(train_path,
                                                         target_size = shape_of_image.shape[:2],
                                                         batch_size = 50,
                                                         color_mode = 'rgb',
                                                         class_mode = 'categorical')
     test_generator = test_datagen.flow_from_directory(test_path,
                                                         target_size = shape_of_image.shape[:2],
                                                         batch_size = 50,
                                                         color_mode = 'rgb',
                                                         class_mode = 'categorical')

     Found 5855 images belonging to 10 classes.
     Found 9192 images belonging to 10 classes.
```

Figure 1: Train and Test generator

After I zoomed to 0.3, due to this all the images of fruits are centered and do not have useful information on the sides (Figure 2 and Figure 3).

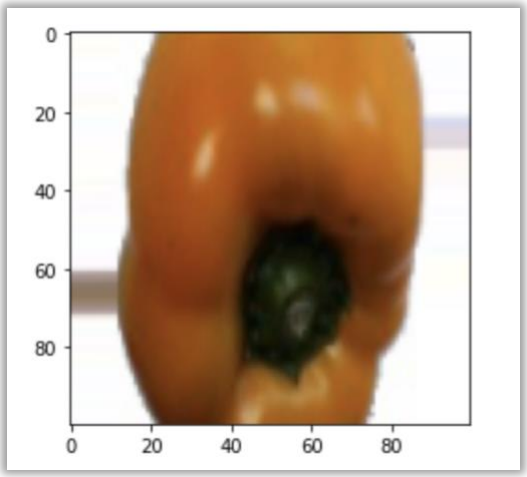

Figure 2: Fruit image

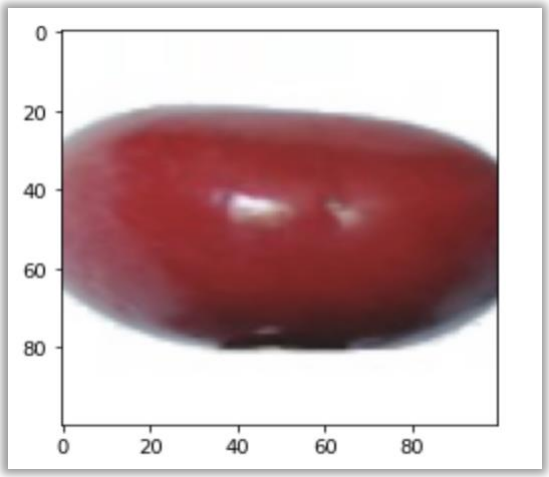

Figure 3: Fruit image

CHAPTER 2: "Loss, activation functions and Optimizers: Zero-one Loss, cross entropy, Adam and Relu"

The algorithm is trained using the categorical cross-entropy. The gradient of **cross entropy loss** function does affect the incorrect classes in another way relying upon how wrong they are. Thus, using cross-entropy, the score of the correct class is normalized by the scores of all the other classes to turn it into a probability. The most important part is that cross-entropy is a function that takes, as input, two probability distributions, q and p and returns a value that is minimal when q and p are equal. In this project, I know the actual label of the training data, therefore the true/ target distribution, that is p, has a probability of 1 in case of true label, and 0 otherwise. Through training a system to minimize cross entropy, I am telling the system that we want it to try and make the estimated distribution as close as it can to the true distribution. Hence, the class that the model thinks is most likely is the class corresponding to the highest value of q.

In order to evaluate the final performance of the test set, **the Zero-one loss** is used. The zero one loss function says that if the value is above zero, then classify as 0, otherwise 1. In this project, I created a zero-loss function by defining if predicted label is equal to test label, then give zero; else one. Then, I did the mean of all the values, if the value is closer to 0, then it is good.

The **Optimizer** used to train the network is **Adam** and the main activation function **Relu.** Relu improve neural networks by speeding up training and the main thing is that all the neurons are not activated concurrently. Gradients of logistic and hyperbolic tangent networks are smaller than the positive portion of the Relu. This implies that the positive portion is updated more rapidly as training progresses. I opt for the **AdamOptimizer** due to the fact that it has the advantage over the GradientDescent of using the running average of the gradients as well as the running average of the gradient squared. Adam exhibits marginal improvement with average, and it complies learning rate scale for distinct layers rather of handpicking by hand and performed very well with accuracy metrics.

CHAPTER 3: "Feed forward neural network"

A feed forward neural network is an artificial neural network in which the connections between nodes do not form a cycle. The feed forward model interconnection of perceptrons in which data and calculations flow in a single direction, from the input data to the outputs.
A succession of inputs gets into the layer and are multiplied by the weights. Subsequently, every value is added jointly to get a sum of the weighted input values. In the case in which the sum of the values is above a specific threshold, typically set at zero, so the value produced is 1, otherwise if the sum drops under the threshold, the output value is -1.
I fine-tuned the neurons of neural networks, I iteratively tuned the configuration during training by using techniques like pruning nodes based on (small) values of the weight vector after a certain number of training epochs, using other words, eliminating unnecessary, or redundant, nodes.

Input Layer
As regards to the number of neurons comprising the Input layer, the number of neurons comprising that layer is equivalent to the number of features in my data. Given that, it is completely and determined when I find out the shape of the training data.

Output Layer
In the Output Layer, as in the input layer, each NN has only one output layer. Figure out the number of neurons is determined by the selected model configuration. In the case of regressor, the output layer has a single node. With respect to the classifier, it also has a single node, except softmax is used in such a case in which the output layer has one node per class label in the model.

Hidden Layers
In relation to some empirically derived rules in order to establish the number of hidden layers are present. With the aim of to deal with that, the setting of the hidden layer configuration can be done by the usage of two rules: the first one fix the number of hidden layers equals to one; the other one the number of neurons in that layer is the mean of the neurons in the input and output layers. Additionally, the theoretical learning capacity of the model could be increased by adding more layer in the network and make more channels. Following the rule, I tried to play with the rates of 2x to 4x and see where it gets.

CHAPTER 4: "Convolutional layer"

The convolutional layer excerpts features form the image. In this layer each neutron is called a kernel or filter, this one extracts the features from the input image. The image is cut into smaller parts, and a kernel is usually a square with the dimension Ki × Ki × di. The formal definition of the convolutional layer can be expressed as followed, in which w is a tensor with three dimensions on the portion s, r, k and functioning as a kernel in the layer q as filter p. The feature is depicted by a three-dimensional tensor. In other words, the convolutional layer is the sum of the element wise product between the sliced block and the kernel.

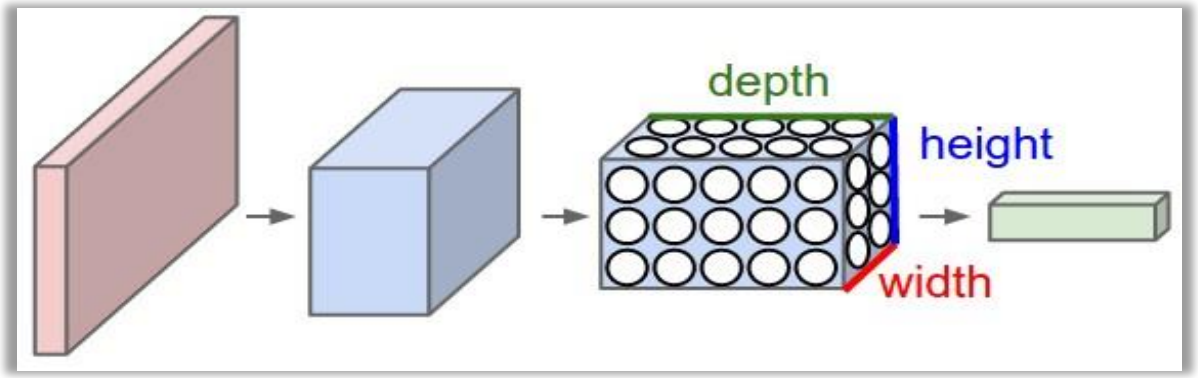

Figure 4: ConvNet in three dimensions

The usage of filters reduces the size of an image. To avoid this padding can be applied. Padding increases the size of an input image by adding additional values around the border of the image. Typically, zero values are used. The output size of an image can be calculated with the following formula, where I stand for the size of the image and k for the size of the filter.

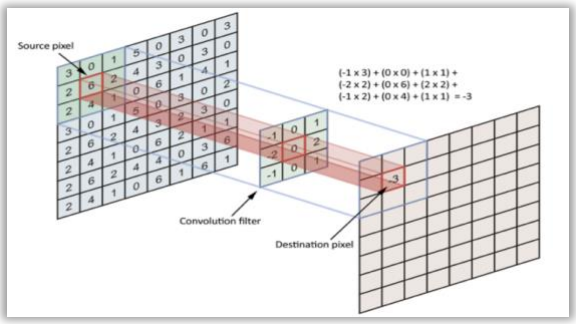

Figure 5: Convolutional Neural Networks

Paragraph 4.1: "Pooling"

The pooling layer extract a value inside a matrix m x n used on the input matrix j x i. The choice of the value relies upon a rule, that can be for instance the maximum value inside this matrix. Pooling has several steps, in general the pooling size is 2 x 2 with a stride of 2.

CHAPTER 5: “Network Architectures and Results”

I used these following Network Architectures in my project: VGGNet, LENET and Resnet.

Paragraph 5.1: “VGGNet”

VggNet exhibits that a crucial component for the purpose of having a good performance is the depth of the network. Their best networks include 16 convulsions layers and the features a highly homogeneous architecture that only carries out 3x3 convolutions and 2x2 pooling from end to end. A problem could be that the VGGNet is costly to evaluate, and the application need more memory and parameters. In the first fully connected layer are present mostly of these parameters, and this is due to the fact that these fully connected layers can be removed without decreasing the performance, lowering the number of necessary parameters in a considerable way.

The VGG network implements Max Pooling and Relu activation function. All the hidden layers use Relu activation, and the last Dense layers uses Softmax activation. Maxpooling is carried out over a 2x2 pixel window with a stride of 2.

Subparagraph 5.1.1: “VGG3 Blocks”

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 100, 100, 32)	896
conv2d_1 (Conv2D)	(None, 100, 100, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 50, 50, 32)	0
dropout (Dropout)	(None, 50, 50, 32)	0
conv2d_2 (Conv2D)	(None, 50, 50, 64)	18496
conv2d_3 (Conv2D)	(None, 50, 50, 64)	36928
max_pooling2d_1 (MaxPooling2	(None, 25, 25, 64)	0
dropout_1 (Dropout)	(None, 25, 25, 64)	0
conv2d_4 (Conv2D)	(None, 25, 25, 128)	73856
conv2d_5 (Conv2D)	(None, 25, 25, 128)	147584
max_pooling2d_2 (MaxPooling2	(None, 12, 12, 128)	0
dropout_2 (Dropout)	(None, 12, 12, 128)	0
flatten (Flatten)	(None, 18432)	0
dense (Dense)	(None, 128)	2359424
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290

Total params: 2,647,722
Trainable params: 2,647,722
Non-trainable params: 0

Figure 6: Results VGG3

The zero one loss with vgg3 is 0 (Figure 7), which depicts good performance. And if we see the lines of training loss and validation loss (Figure 8 and Figure 9), there are decreasing on similar trend, so there is not that much underfitting and overfitting, the same goes for the lines of accuracy.
We also see that accuracy and loss are gradually getting better in 10 epochs, as I selected these epochs after fine tuning and cross validating, with different Dense layer weights, Batch sizes and steps per epochs, and these settings, I chose gives me the best Training model.

The architecture contains:

- Dense layers equal to 128,
- Steps per epoch equal to 50,
- Epoch equal 10.

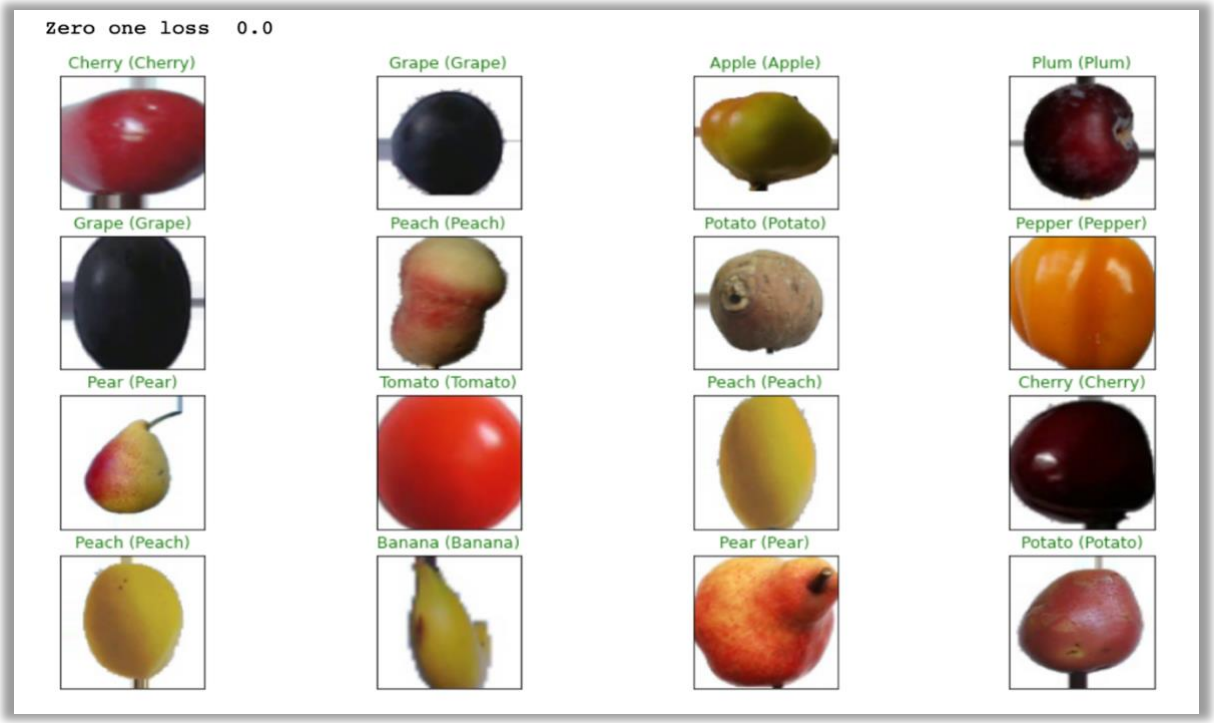

Figure 7: Prediction results VGG3

Figure 8: Overfitting and Underfitting check for VGG3

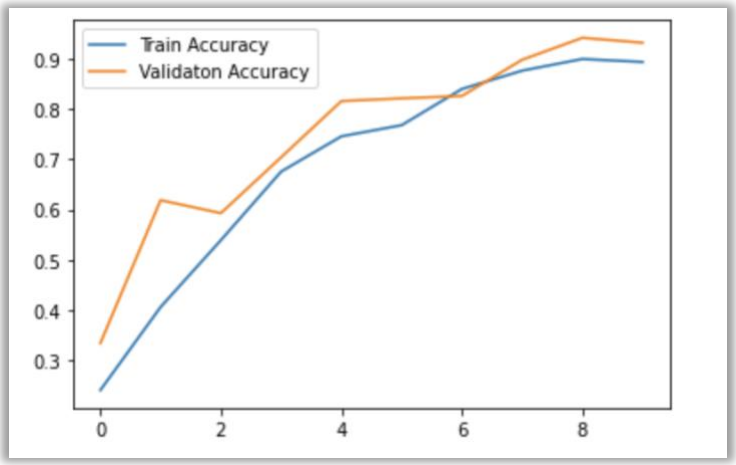

Figure 9: Overfitting and Underfitting check for VGG3

Subparagraph 5.1.2: "VGG13"

Layer (type)	Output Shape	Param #
conv2d_40 (Conv2D)	(None, 98, 98, 64)	1792
conv2d_41 (Conv2D)	(None, 96, 96, 64)	36928
max_pooling2d_16 (MaxPooling	(None, 48, 48, 64)	0
conv2d_42 (Conv2D)	(None, 46, 46, 128)	73856
conv2d_43 (Conv2D)	(None, 44, 44, 128)	147584
max_pooling2d_17 (MaxPooling	(None, 22, 22, 128)	0
conv2d_44 (Conv2D)	(None, 20, 20, 256)	295168
conv2d_45 (Conv2D)	(None, 18, 18, 256)	590080
conv2d_46 (Conv2D)	(None, 16, 16, 256)	590080
max_pooling2d_18 (MaxPooling	(None, 8, 8, 256)	0
conv2d_47 (Conv2D)	(None, 6, 6, 512)	1180160
conv2d_48 (Conv2D)	(None, 4, 4, 512)	2359808
conv2d_49 (Conv2D)	(None, 2, 2, 512)	2359808
max_pooling2d_19 (MaxPooling	(None, 1, 1, 512)	0
flatten_4 (Flatten)	(None, 512)	0
dense_8 (Dense)	(None, 6000)	3078000
dropout_4 (Dropout)	(None, 6000)	0
dense_9 (Dense)	(None, 10)	60010

Total params: 10,773,274
Trainable params: 10,773,274
Non-trainable params: 0

Figure 10: Results VGG13

The zero one loss with vgg13 is near to 0 (0.187), which depicts a good performance (Figure 11). Though we see the lines of training loss and validation loss (Figure 12 and Figure 13), they are decreasing on similar trend but with very bad information, due to that there is not that much underfitting and overfitting, the same goes for the lines of accuracy. However, this model was not able to reduce the parameters of pictures and extract information in an efficiently way to give good results. Additionally, after trying multiple hyperparameter tuning, the performance of this model does not improve.

The architecture contains:

- Dense layers equal to 6000;
- Steps per epoch equal to 50;
- Epoch equal 10.

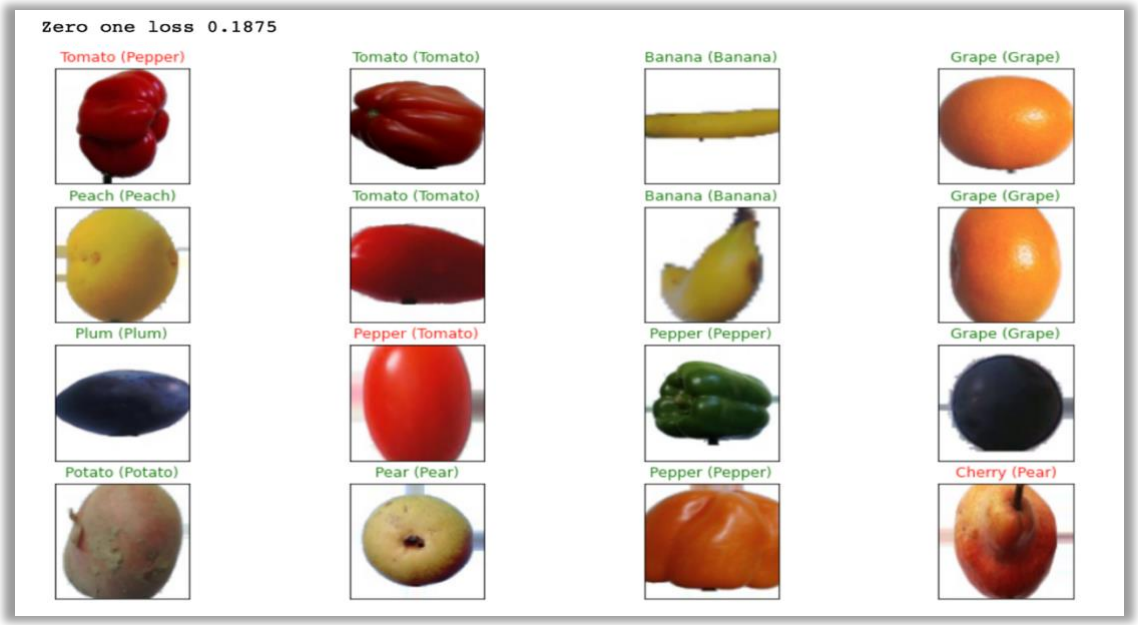

Figure 11: Prediction results VGG13

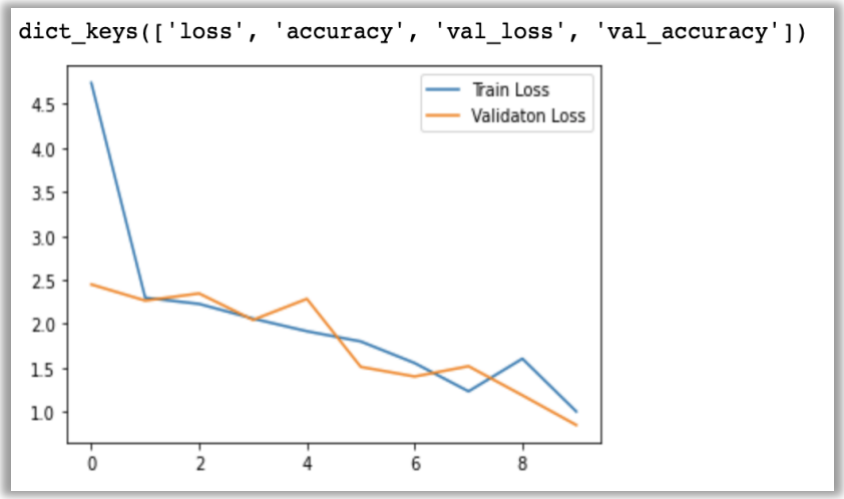

Figure 12: Overfitting and Underfitting check for VGG13

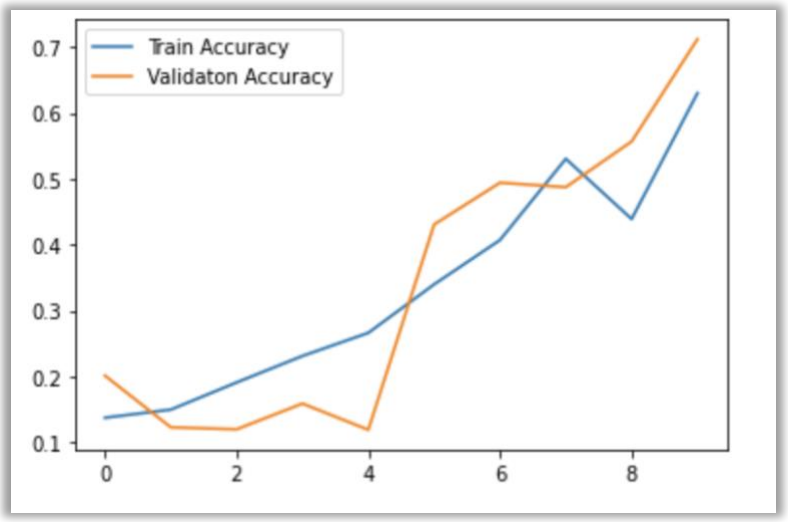

Figure 13: Overfitting and Underfitting check for VGG13

Paragraph 5.2: "Lenet"

The LeNet-5 architecture comprises two sets of convolutional and average pooling layers, tracked by a flattening convolutional layer, subsequently two fully connected layers and finally a softmax classifier.

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_50 (Conv2D)           (None, 100, 100, 6)       456
_________________________________________________________________
average_pooling2d (AveragePo (None, 50, 50, 6)         0
_________________________________________________________________
conv2d_51 (Conv2D)           (None, 46, 46, 16)        2416
_________________________________________________________________
average_pooling2d_1 (Average (None, 23, 23, 16)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 8464)              0
_________________________________________________________________
dense_10 (Dense)             (None, 120)               1015800
_________________________________________________________________
dense_11 (Dense)             (None, 84)                10164
_________________________________________________________________
dense_12 (Dense)             (None, 10)                850
=================================================================
Total params: 1,029,686
Trainable params: 1,029,686
Non-trainable params: 0
_________________________________________________________________
```

Figure 14: Results Lenet

The zero one loss with Lenet is zero, which depicts good performance (Figure 15). However, it could also be overfitting or underfitting with bias, but as we compare the lines of training loss and validation loss (Figure 16 and Figure 17), there are decreasing on similar trend, therefore there is not that much

underfitting and overfitting, the same goes for the lines of accuracy. Besides, the predictions of fruit class for the images are 100% accurate.

The architecture contains:

- Dense layers equal to 120;
- Steps per epoch equal to 50;
- Epoch equal 10.

Figure 15: Prediction results Lenet

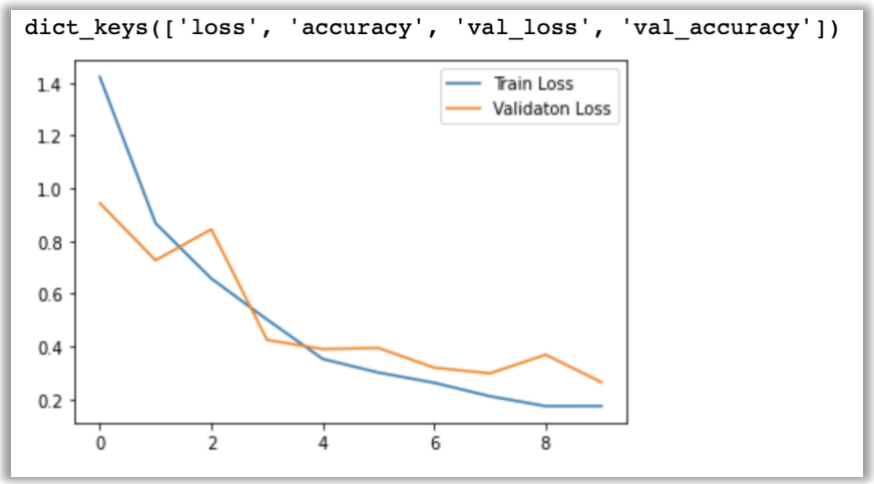

Figure 16: Overfitting and Underfitting check for Lenet

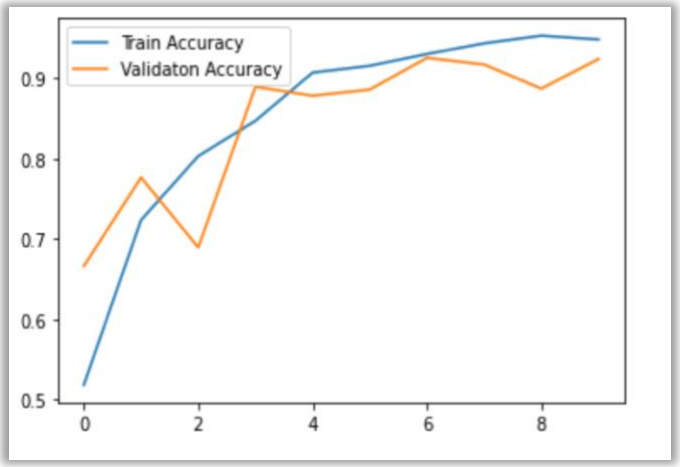

Figure 17: Overfitting and Underfitting check for Lenet

Paragraph 5.3: "Resnet"

Residual Network is a network in which the features leave out connections and a strong avail of batch normalization. Furthermore, the architecture is lacking totally connected layers toward the end of the network.

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_52 (Conv2D)           (None, 50, 50, 64)        9472
_________________________________________________________________
conv2d_53 (Conv2D)           (None, 50, 50, 32)        51232
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 25, 25, 32)        0
_________________________________________________________________
conv2d_54 (Conv2D)           (None, 25, 25, 16)        12816
_________________________________________________________________
flatten_6 (Flatten)          (None, 10000)             0
_________________________________________________________________
dense_13 (Dense)             (None, 256)               2560256
_________________________________________________________________
dense_14 (Dense)             (None, 10)                2570
=================================================================
Total params: 2,636,346
Trainable params: 2,636,346
Non-trainable params: 0
_________________________________________________________________
```

Figure 18: Results Resnet

The zero one loss with Resnet is zero, this one describes good performance (Figure 19). Nevertheless, it could also be overfitting or underfitting with bias, but as we compare the lines of training loss and validation loss (Figure 20 and Figure 21), there are decreasing on similar trend, hence there is not that

much underfitting and overfitting, likewise that goes for the lines of accuracy. The predictions of fruit class for the images are 100% accurate.

The architecture contains:

- Dense layers equal to 256;
- Steps per epoch equal to 50;
- Epoch equal 15.

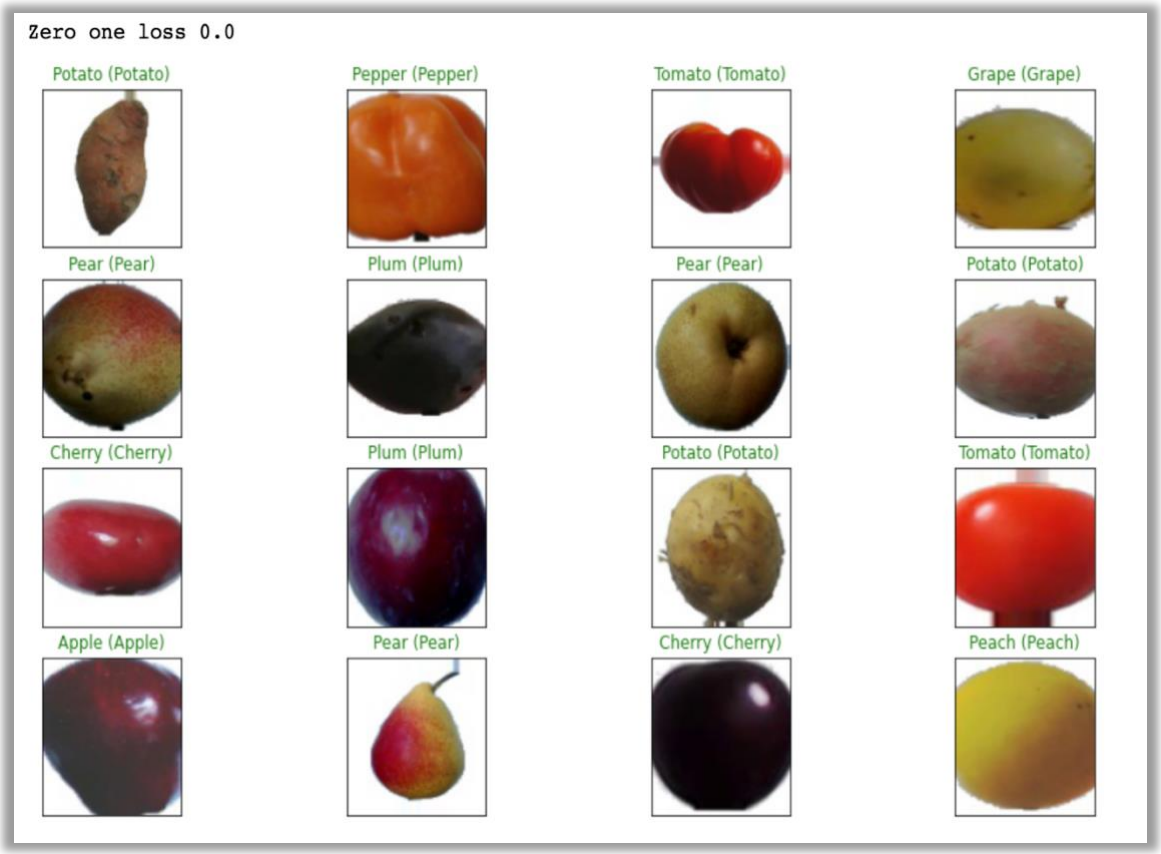

Figure 19: Prediction results Resnet

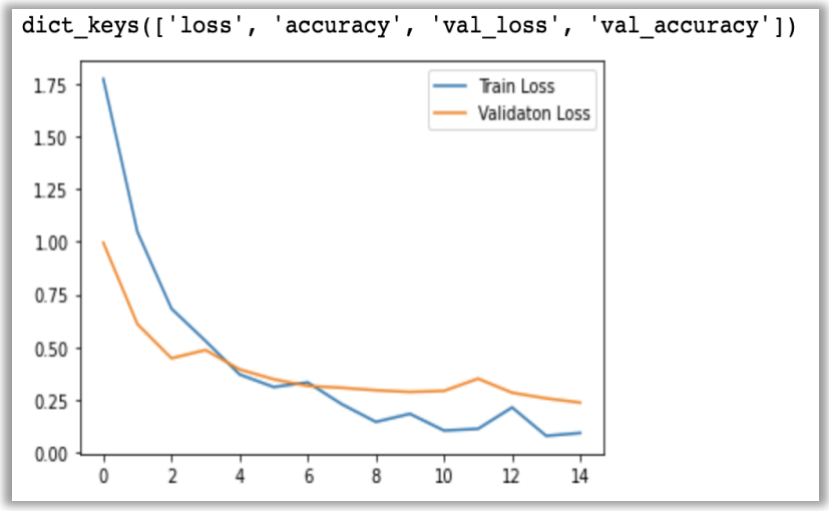

Figure 20: Overfitting and Underfitting check for Resnet

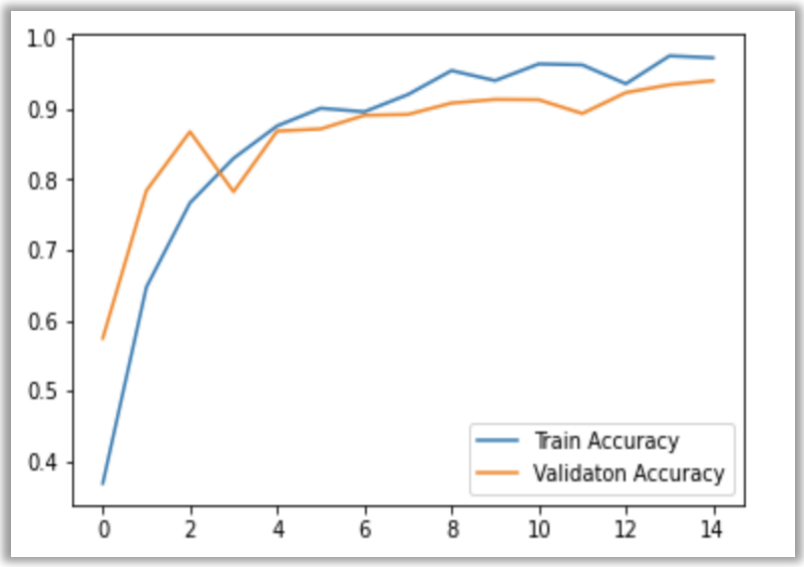

Figure 21: Overfitting and Underfitting check for Resnet

CONCLUSION

I found out that training a CNN model and fine-tuning hyperparameters is a random process. Owing to this doing experiment is tough, since every time you run the same experiments, you will get different results. Furthermore, I discover that many times my model was getting confused, probably because there were many types of particular fruit with different color and shapes, that are classified into a single class, for instance different types of apples under a single class apple. However, I was able to overcome the problem by adding weights in Dense layers and increasing neurons inside it, and additionally by adjusting steps for each iteration.

I discovered that I have to run the model and experiment with it several times and I had to take an average to find the best turning for hyperparameters. Moreover, I noticed that more simple models like Resnet50 and Lenet were having very good performance over this data, and more complex models like Vgg3 and Vgg13 were not so good in performance and accuracy. Furthermore, as I was adding more blocks in Vggnet model from Vgg3 to Vgg13 and increasing complexity, the performance was getting bad.

Considering parameters, they are weights that are learnt in the course of training. They are weight matrices that bring to model's predictive power, modified during back-propagation process.

In accordance with these experiments and model training, the following are the parameters/weights different models were able to train:

- VGG3 trained 2,647,722 weights from all the layers,
- VGG13 trained 10,773,274 weights from all layers
- Lenet trained 1,029,686 weights from all layers
- Resnet trained 2,636,346 weights from all layers.

BIBLIOGRAPHY:

https://www.kaggle.com/moltean/fruits - Dataset

https://cs231n.github.io/convolutional-networks/#conv – Stanford Vision and Learning Lab

http://cesa-bianchi.di.unimi.it/MSA/Notes/deep.pdf - Prof. Nicolò Cesa Bianchi, Professor of Computer Science, Università degli Studi di Milano

https://arxiv.org/pdf/1409.1556.pdf- Karen Simonyan∗ & Andrew Zisserman+
Visual Geometry Group, Department of Engineering Science, University of Oxford

http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf- Yann LeCun, Lèon Bottou, Yoshua Bengio, and Patrick Haffner

https://arxiv.org/pdf/1512.03385v1.pdf- Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun

https://becominghuman.ai/beginners-guide-cnn-image-classifier-part-1-140c8a1f3c12 -Lakshmanan Meiyappan