Image classification with Neural Networks

Report submitted in partial fulfillment of the requirements for the degree

of

Master of Science in Computer Science

by

Gallage Don Yasiru Prabath Amarasinghe
885675

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITÀ DEGLI STUDI DI MILANO

2020/2021

DECLARATION

Project Title	Image classification with Neural Networks
Author	*Gallage Don Yasiru Prabath Amarasinghe*
Student ID	885675

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 us or any other person for assessment on this or any other course of study.

Date : February 24, 2021

Contents

List of Figures

List of Tables

Chapter 1

Abstract

The work of this paper is training neural networks for the classification of fruit and vegetable types and experiments with different network architectures and compares them according to their influence on the final predictive performance.

Recognising fruits and vegetables is important to create a more user friendly system .As an example to improve the identification process of fruit and vegetables in supermarkets .This can replace the requirement to enter the product code when the user scales the fruit or vegetable .

First I implemented simple feed forward neural networks and measured the performance by changing the number of neurons in the hidden layer and number of hidden layers. In the next section I implemented Convolutional Neural Network which is designed for image data analysis. In this section consider more training parameters and techniques such as pooling , dropout , batch normalization and earlystopping and their influence on the performance. In the final section discussed regarding modern CNN neural network architectures and their performance on this particular problem.

Chapter 2

The Dataset

The “Fruits 360” is a dataset publicly available on Kaggle containing 90483 images of fruits and vegetables belonging to the 131 classes. Image size is 100*100.Dataset divided test and train set.

To simplify the problem consider only the 10 most frequent types (apple, banana, plum, pepper, cherry, grape, tomato, potato, pear, peach) and used types opposed to the variety of the fruit and vegetable . So for this work dataset belonging to the above mentioned 10 classes.

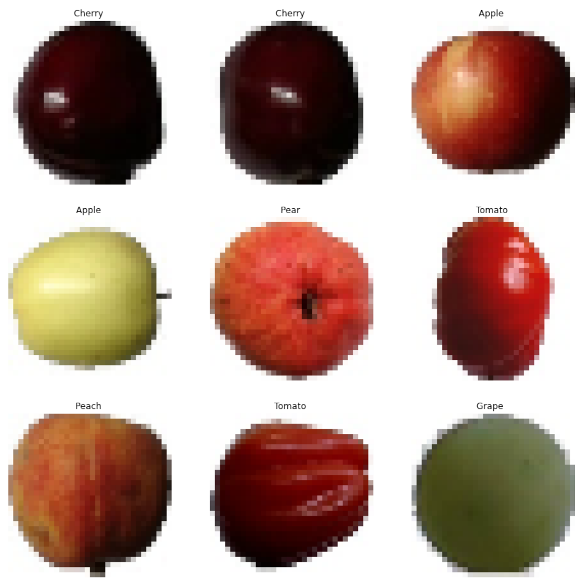


Figure 2.1: Random images of the dataset after the shuffle

Images are scaled down to 32*32 and transformed from JPG to RGB pixel values. Image data,the pixel values are integers with values between 0 and 255 .Normalize pixel values divide by 255.So that each pixel value has a value between 0 and 1.

The data is shuffled to prevent model learning from the order of training and prevent form any bias during the training.

After the end of the process there are 32607 train images and 10906 test images in the dataset.

Chapter 3

Method and Experimental Results

3.1 Feed Forward Neural Network

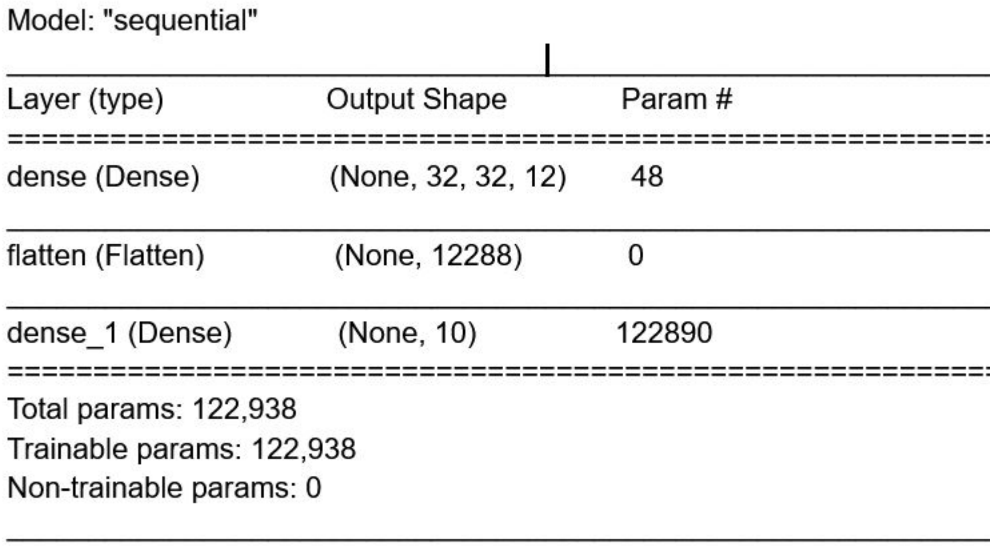
The feed forward model is the simplest form of neural network as information is only processed in one direction. While the data may pass through multiple hidden nodes, it always moves in one direction and never backwards.

Every neural network has three types of layers.input layer hidden layer and output layer. Every neural network has exactly one input layer and one output layer. Nodes of the input layer based on the input data and nodes of the output layer based on the number of classes in the data set. For hidden layers we can choose the number of hidden layers and nodes.

We will try to understand the effect of the number of nodes in hidden layers and number of hidden layers to the final outcome and performance .

3.1.1 FFN model 1 (the model with 12 hidden nodes)

In this experiment I started with a simple feed forward neural network .It define Sequential Multilayer perceptron model that accepts 32*32 images as input and one hidden layer that has 12 nodes with relu activation function then one hidden flatten layer then output layer with 10 nodes with sigmoid activation function.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32, 32, 12)        48
_________________________________________________________________
flatten (Flatten)            (None, 12288)             0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                122890
=================================================================
Total params: 122,938
Trainable params: 122,938
Non-trainable params: 0
_________________________________________________________________
```

Every Connection in the flatten layer connect to every connection in the dense layer.Which lead to higher numbers of trainable parameters.

The model learns fast and it shows good results starting from 2nd epoches. It was able to gain more than 85% validation accuracy after the first epoch due to less complexity of the problem. After the second epoch validation accuracy was increased to 96%

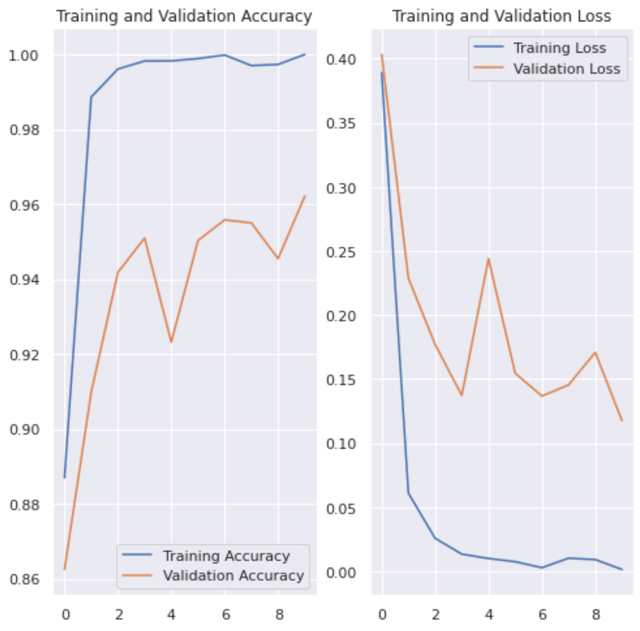


Figure 3.1: Accuracy and loss for training and validation

Figure shows 3.1.1 After the 10 epoches model was able to gain 95.621% . Zero one loss is 0.0438

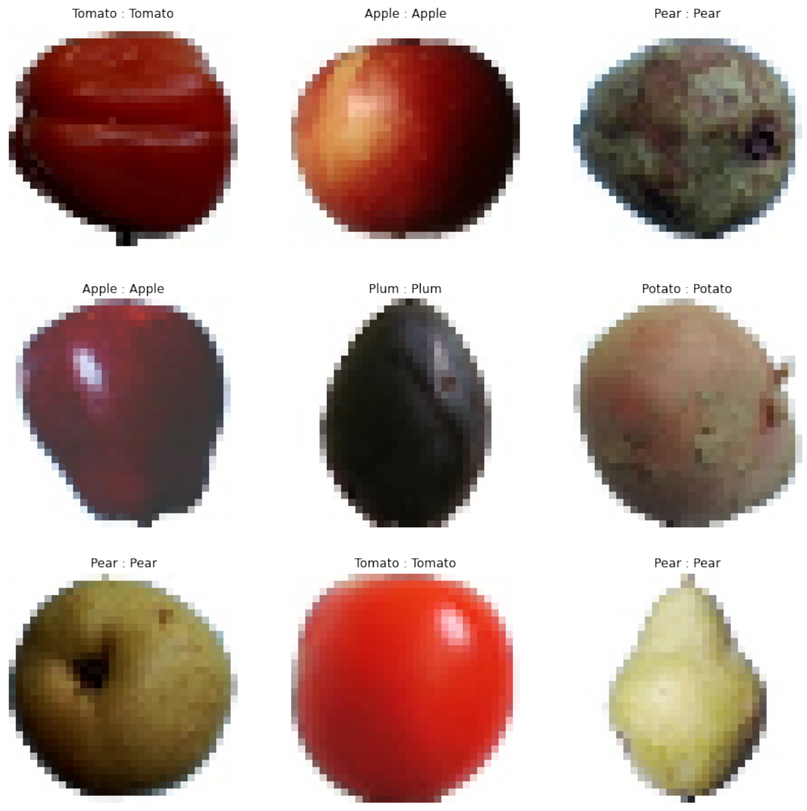
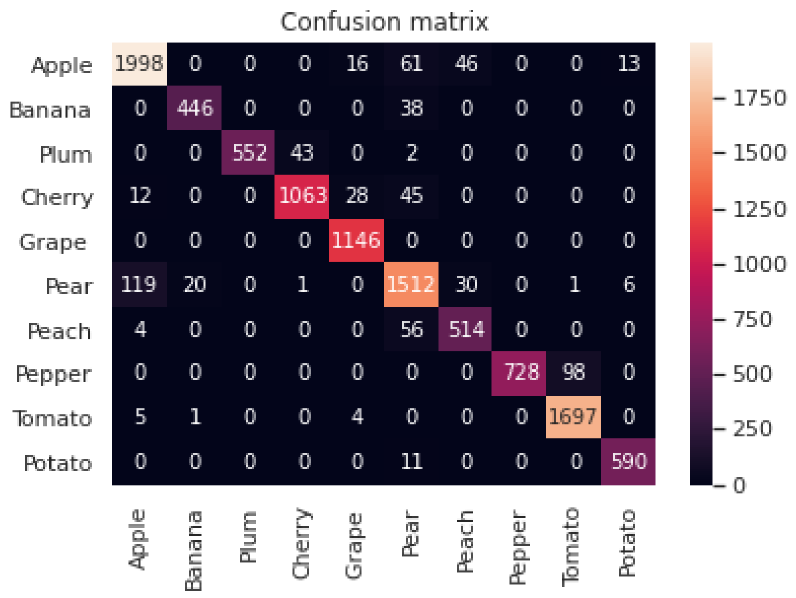


Figure 3.2: First 9 images of the test results

This is a really good result compared to its simplicity and performance .with machine with GPU training took only less than one minute.Zero one loss is 0.03787 and validation accuracy is 96.21%

Figure shows 3.3 confusion matrix we can describe the performance of our model on test data. Out of all 2152 apple images , the system predicted that 119 were pear 12 were cherry 5 potato 4 were peach and 1998 were Apple . Basically images of the Green apple and Green pear are difficult to identify distintinctle naked eye. Pepper and plum identified with 100% accuracy. All correct predictions are located in the diagonal of the table.



	Apple	Banana	Plum	Cherry	Grape	Pear	Peach	Pepper	Tomato	Potato
Apple	1998	0	0	0	16	61	46	0	0	13
Banana	0	446	0	0	0	38	0	0	0	0
Plum	0	0	552	43	0	2	0	0	0	0
Cherry	12	0	0	1063	28	45	0	0	0	0
Grape	0	0	0	0	1146	0	0	0	0	0
Pear	119	20	0	1	0	1512	30	0	1	6
Peach	4	0	0	0	0	56	514	0	0	0
Pepper	0	0	0	0	0	0	0	728	98	0
Tomato	5	1	0	0	4	0	0	0	1697	0
Potato	0	0	0	0	0	11	0	0	0	590

Figure 3.3: confusion matrix of the result

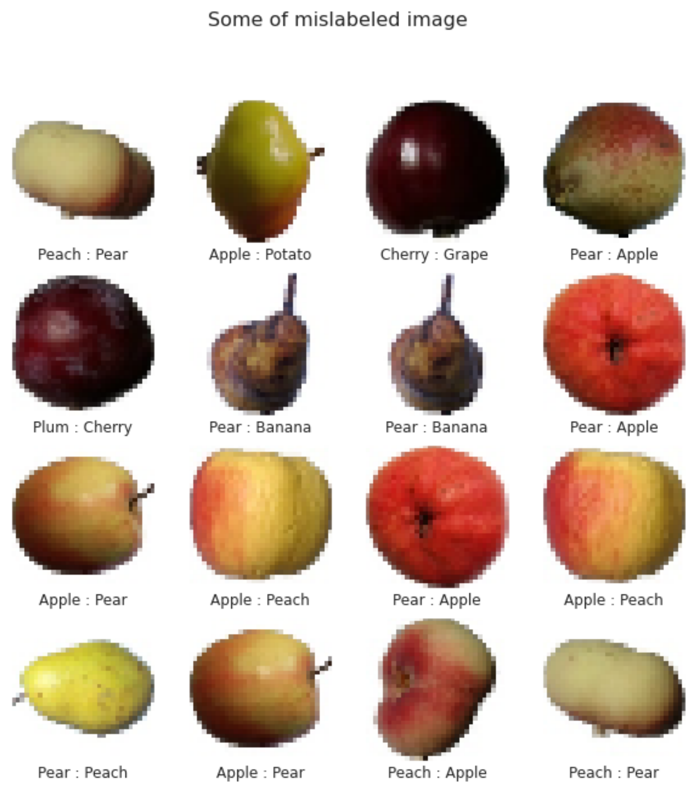
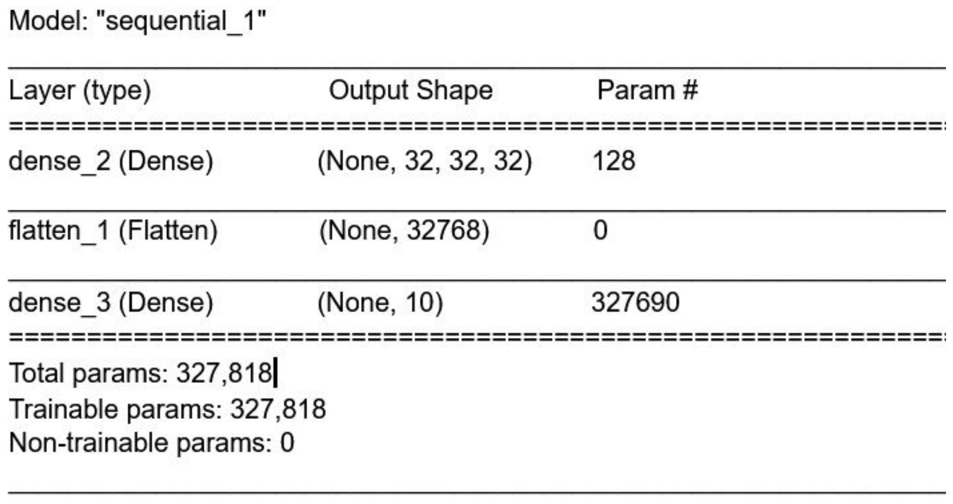


Figure 3.4: First 16 of mislabeled fruits

3.1.2 FFN model2 (the model with 32 hidden nodes)

For the next model increase nodes in the hidden layer to 32 from 12.This results to increase the trainable parameters in the model.It took more time to train the model due to complexity and which lead to better performance than previous model.

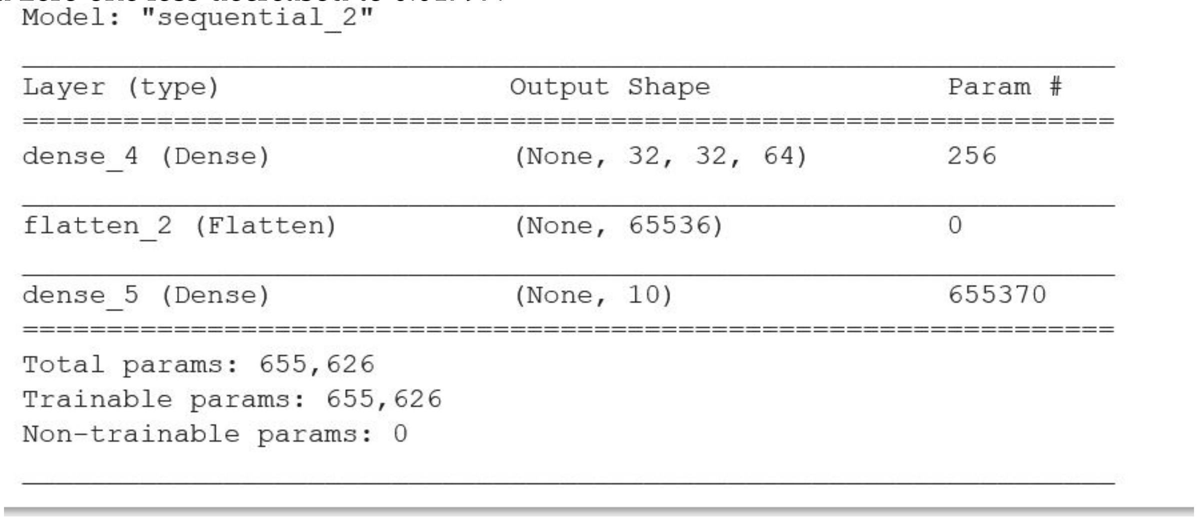
After increasing the node . I was able to get 97.71% accuracy and zero one loss decreased to 0.02292.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 32, 32, 32)        128
_________________________________________________________________
flatten_1 (Flatten)          (None, 32768)             0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                327690
=================================================================
Total params: 327,818
Trainable params: 327,818
Non-trainable params: 0
_________________________________________________________________
```

3.1.3 FFN model3 (the model with 64 hidden nodes)

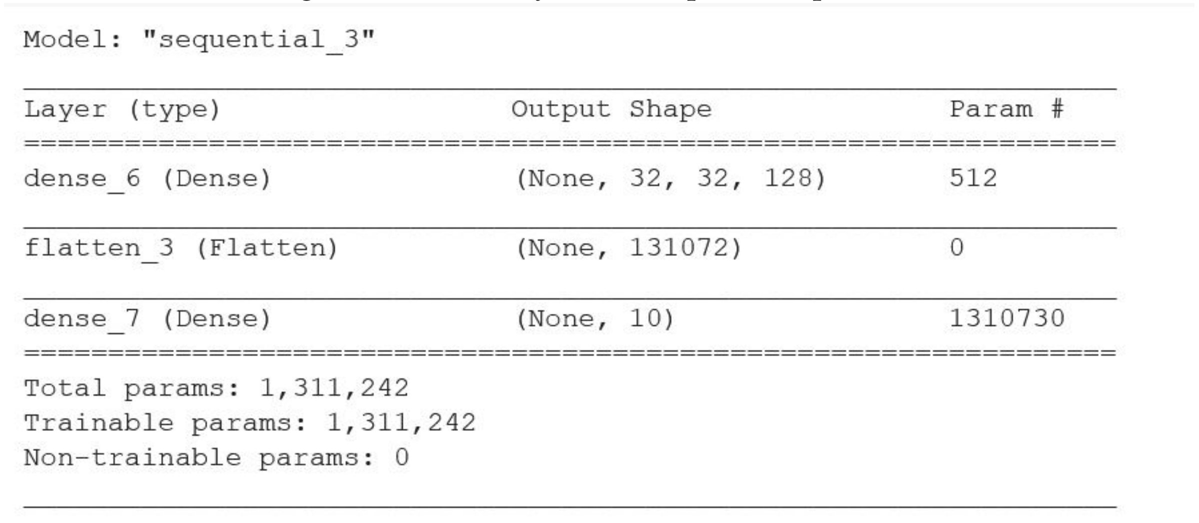
Increase hidden node to 64 which result to Validation accuracy increased to 98.20% and zero one loss decreased to 0.01797.



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 32, 32, 64)        256
_________________________________________________________________
flatten_2 (Flatten)          (None, 65536)             0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                655370
=================================================================
Total params: 655,626
Trainable params: 655,626
Non-trainable params: 0
_________________________________________________________________
```

3.1.4 FFN model4 (the model with 128 hidden nodes)

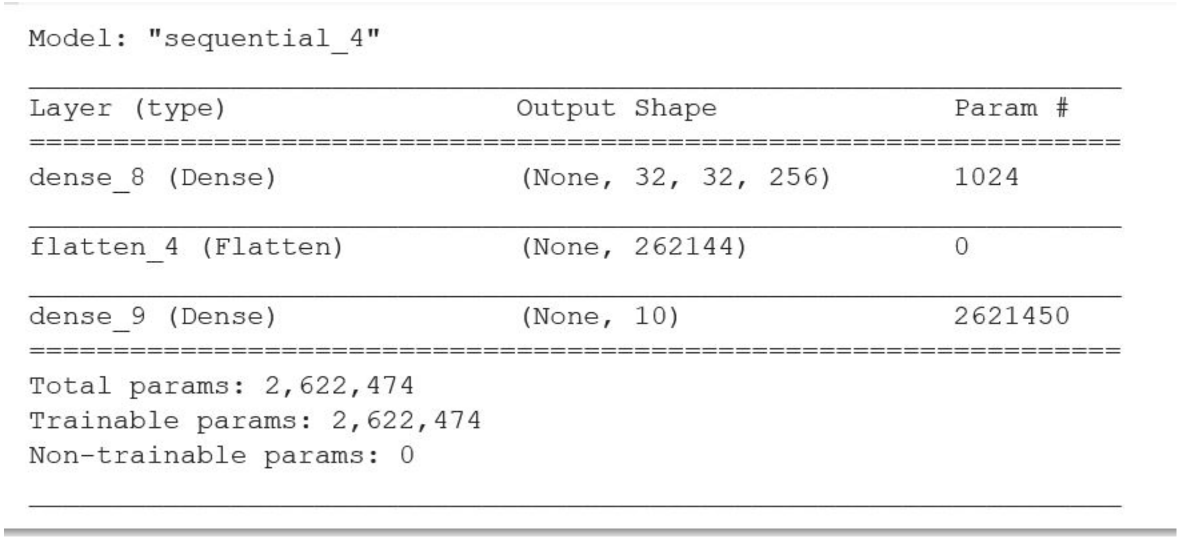
Validation accuracy is 98.28% and zero one loss is 0.01724. Slightly decreased the zero one loss.But training time increase by 50% compared to previous model



```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 32, 32, 128)       512
_________________________________________________________________
flatten_3 (Flatten)          (None, 131072)            0
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1310730
=================================================================
Total params: 1,311,242
Trainable params: 1,311,242
Non-trainable params: 0
_________________________________________________________________
```

3.1.5 FFN model5 (the model with 256 hidden nodes)

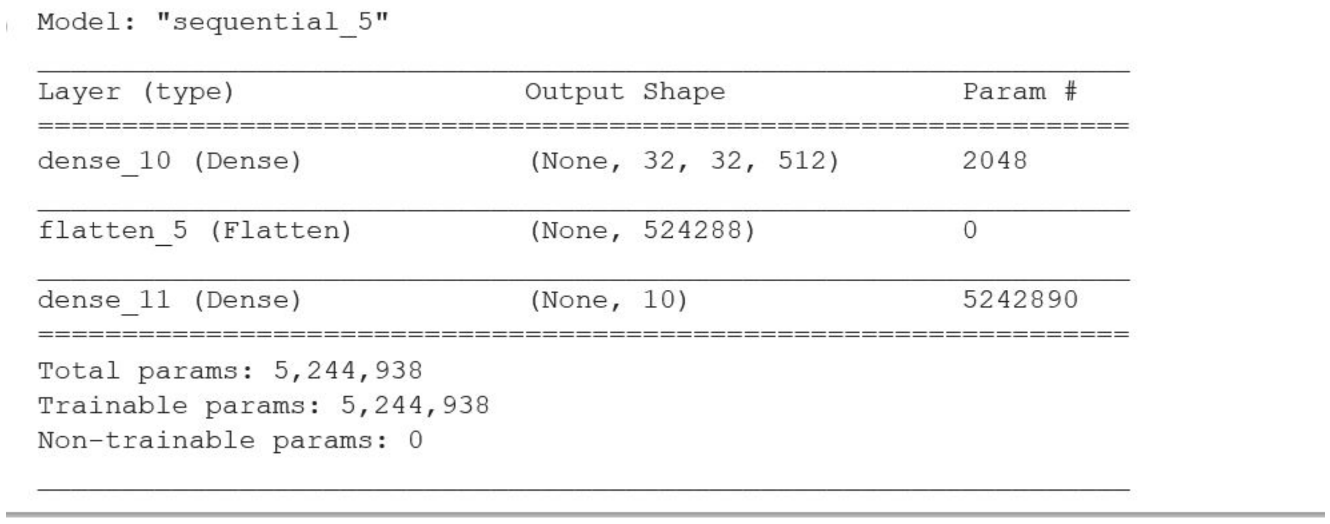
Validation accuracy is 98.11% and zero one loss is 0.01889



```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 32, 32, 256)       1024
_________________________________________________________________
flatten_4 (Flatten)          (None, 262144)            0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                2621450
=================================================================
Total params: 2,622,474
Trainable params: 2,622,474
Non-trainable params: 0
_________________________________________________________________
```

3.1.6 FFN model6 (the model with 512 hidden nodes)

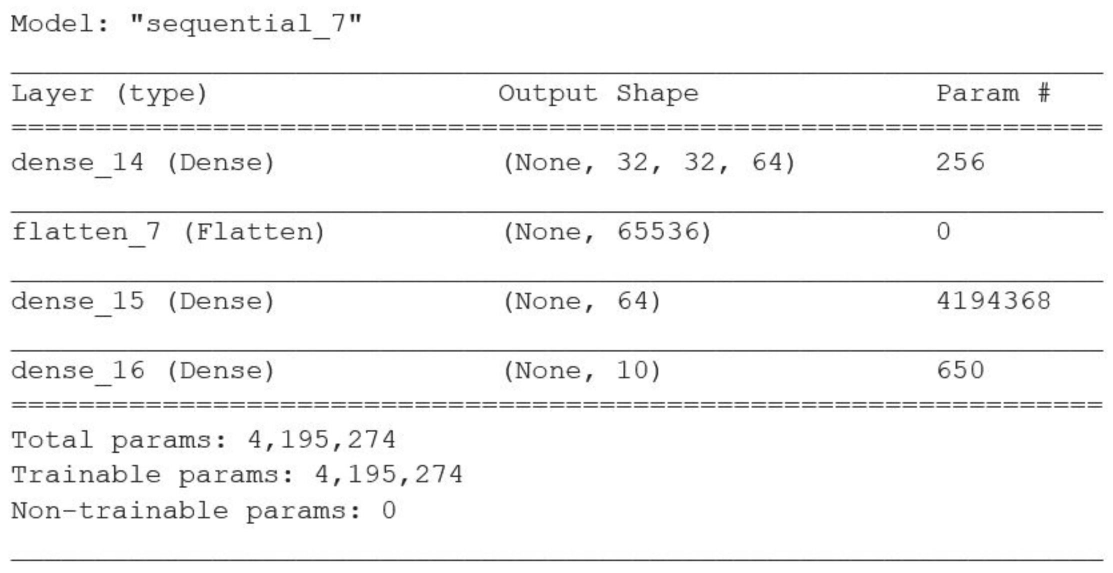
Validation accuracy is 97.80% and zero one loss is 0.02201. This shows little improvement in performance. But training time increase by 3 times compared to model4. Validation accuracy decreased and zero one loss increased.Therefore this good point to stop test by increasing nodes in the dense layer. Too many hidden neural network increase the complexity of the training algorithm.



```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 32, 32, 512)       2048
_________________________________________________________________
flatten_5 (Flatten)          (None, 524288)            0
_________________________________________________________________
dense_11 (Dense)             (None, 10)                5242890
=================================================================
Total params: 5,244,938
Trainable params: 5,244,938
Non-trainable params: 0
_________________________________________________________________
```

3.1.7 FFN model2.1 (Add extra dense layer)

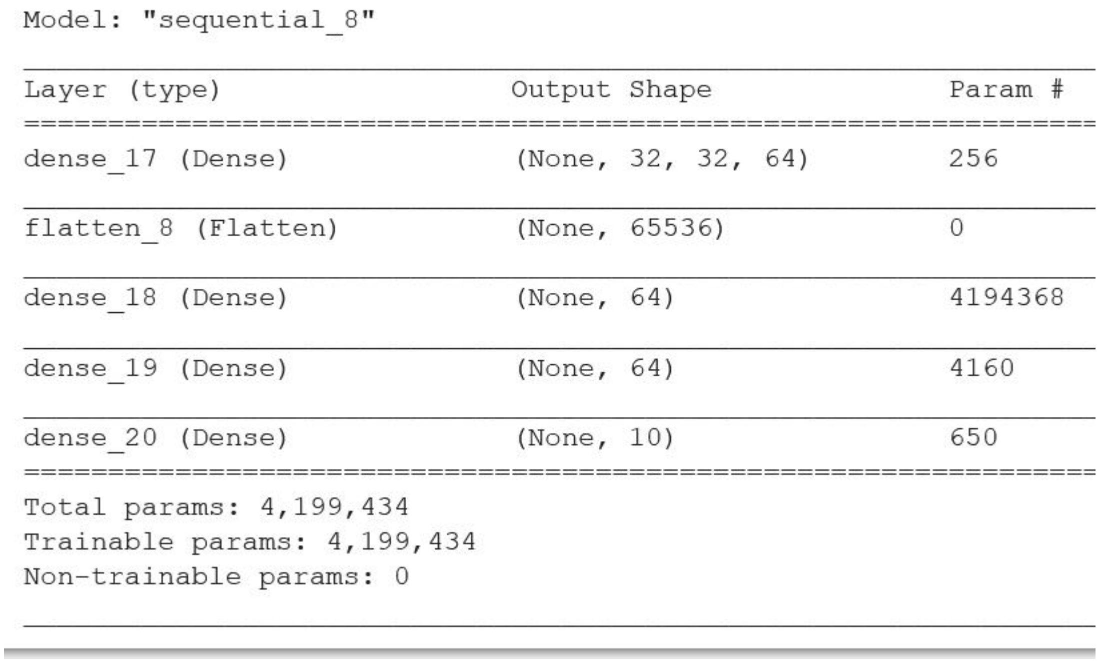
Here I am going to further fine tune the network with adding additional dense hidden layers. with additional dense layer total parameters increase by 8 times in the networks. Performance not increase as expected Validation accuracy is 98.40% and zero one loss is 0.03686.



```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_14 (Dense)             (None, 32, 32, 64)        256
_________________________________________________________________
flatten_7 (Flatten)          (None, 65536)             0
_________________________________________________________________
dense_15 (Dense)             (None, 64)                4194368
_________________________________________________________________
dense_16 (Dense)             (None, 10)                650
=================================================================
Total params: 4,195,274
Trainable params: 4,195,274
Non-trainable params: 0
_________________________________________________________________
```

3.1.8 FFN model2.2 (Add two dense layers)

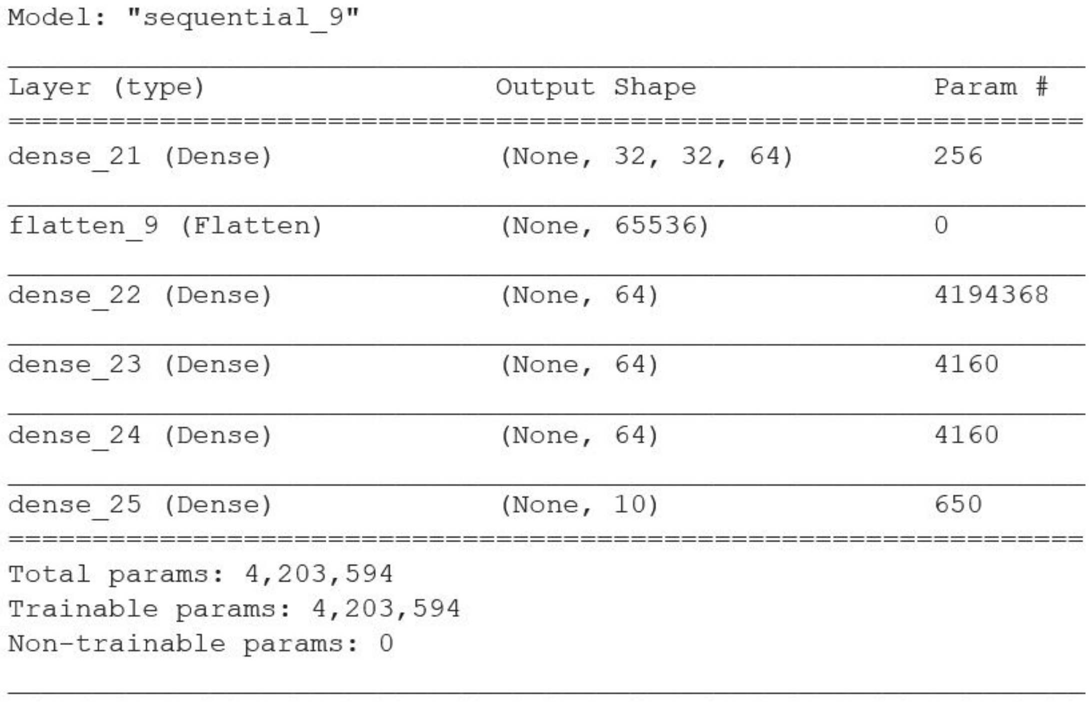
Test the model with adding another hidden dense layer Validation accuracy is 98.88% and zero one loss is 0.01861.



```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_17 (Dense)             (None, 32, 32, 64)        256
_________________________________________________________________
flatten_8 (Flatten)          (None, 65536)             0
_________________________________________________________________
dense_18 (Dense)             (None, 64)                4194368
_________________________________________________________________
dense_19 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_20 (Dense)             (None, 10)                650
=================================================================
Total params: 4,199,434
Trainable params: 4,199,434
Non-trainable params: 0
_________________________________________________________________
```

3.1.9 FFN model2.3 (Add three dense layers)

Test the model with adding another hidden dense layer Validation accuracy is 98.23% and zero one loss is 0.06657.



```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_21 (Dense)             (None, 32, 32, 64)        256
_________________________________________________________________
flatten_9 (Flatten)          (None, 65536)             0
_________________________________________________________________
dense_22 (Dense)             (None, 64)                4194368
_________________________________________________________________
dense_23 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_24 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_25 (Dense)             (None, 10)                650
=================================================================
Total params: 4,203,594
Trainable params: 4,203,594
Non-trainable params: 0
_________________________________________________________________
```

Model number	change	Zero one loss
1.1	12 nodes	0.03787
1.2	32 nodes	0.02292
1.3	64 nodes	0.01797
1.4	128 nodes	0.01724
1.5	256 nodes	0.01889
1.6	512 nodes	0.02201
2.1	add one layer	0.03686
2.2	add two layers	0.01861
2.3	add three layers	0.06657

Table 3.1: Comparison performance on FNN

simple FNN model with one hidden dense layer perform better than multiple hidden dense layer models.

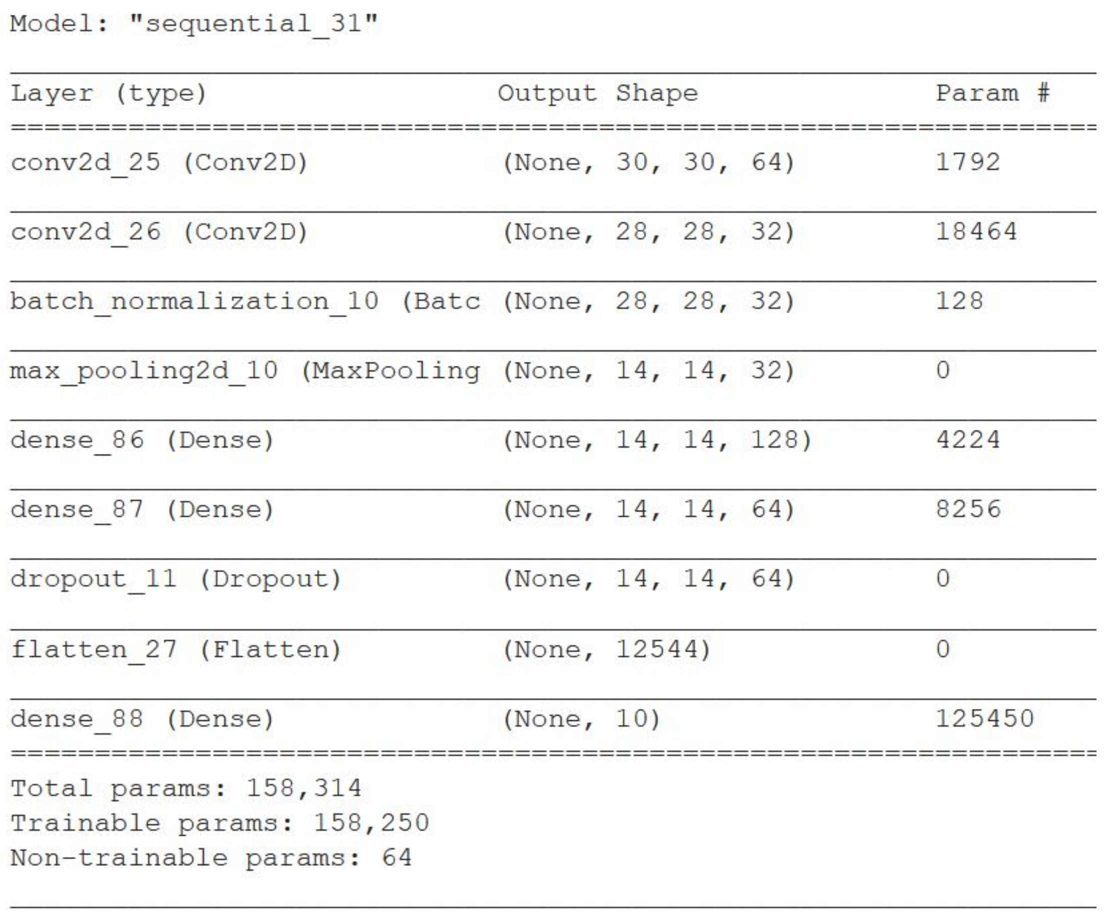
3.2 Convolutional neural network

Convolutional neural network (CNN) is a deep neural network originally designed for image analysis. The first layer is the convolution layer. Convolutional layer is the primary building block of CNN It extracts the high-level features (feature map) from the data set.The pooling layer is followed after the convolution layer. pooling layer used to down the sampling in the input image.The most used pooling layer is max pooling .

3.2.1 CCN model1: with one Convolutional layer

This model has fewer trainable parameters than the FFN discussed in the previous section. Convolutional layer reduces the dimensions of the input through the convolution operation which lead to fewer parameters.

With Feed forward network I did sufficient experiments with changing training parameter number of neurons and numbers of hidden layers. Therefore in here I did more experiment other factors such as batch normalization ,dropout , callback to increase the performance. Batch normalization used to normalize the output of the previous layers it result to increase the efficient of the neural network. Validation accuracy is 98.29% and zero one loss is 0.00743.



```
Model: "sequential_31"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_25 (Conv2D)           (None, 30, 30, 64)        1792
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 28, 28, 32)        18464
_________________________________________________________________
batch_normalization_10 (Batc (None, 28, 28, 32)        128
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
dense_86 (Dense)             (None, 14, 14, 128)       4224
_________________________________________________________________
dense_87 (Dense)             (None, 14, 14, 64)        8256
_________________________________________________________________
dropout_11 (Dropout)         (None, 14, 14, 64)        0
_________________________________________________________________
flatten_27 (Flatten)         (None, 12544)             0
_________________________________________________________________
dense_88 (Dense)             (None, 10)                125450
=================================================================
Total params: 158,314
Trainable params: 158,250
Non-trainable params: 64
_________________________________________________________________
```

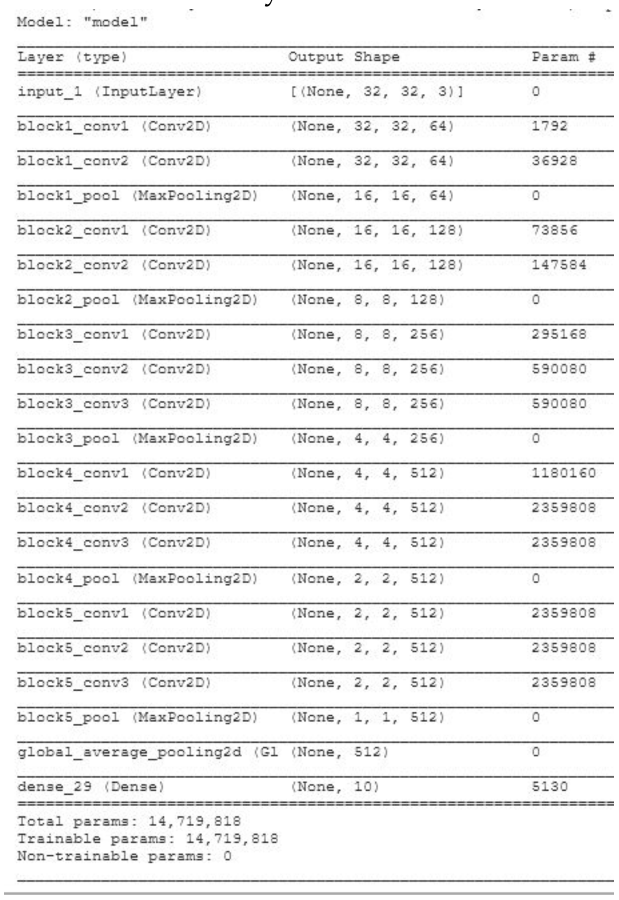
3.3 Transfer Learning with well modern CNN

Modern convolutional neural networks perform well with image classification tasks . But they required a lot of computing power and time to train. Also they required a large dataset to train without overfitting . Therefore it is impossible for this task to train from scratch with modern convolution neural networks such as VGG , Inception and ResNet. These models are trained with a well known “Imagenet” Dataset. It contains more than 14 million images in over 100 categories. With transfer learning I was able to save training time , resources and start using the weights of the trained “imagenet” model.

3.3.1 VGG

In 2014, VGG models were developed by the Researchers from Oxford. It is a convolutional neural network model developed for the image classification task. They released two CNN models with 16 layers (VGG-16) and 19 layers (VGG-19) .' VGG-19 is bigger than VGG-16 and both perform similarly for most of the image classification tasks.

Validation accuracy is 95.37% and zero one loss is 0.04630.



```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)         295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 4, 4, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0
_________________________________________________________________
global_average_pooling2d (Gl (None, 512)               0
_________________________________________________________________
dense_29 (Dense)             (None, 10)                5130
=================================================================
Total params: 14,719,818
Trainable params: 14,719,818
Non-trainable params: 0
_________________________________________________________________
```

3.3.2 MobileNet

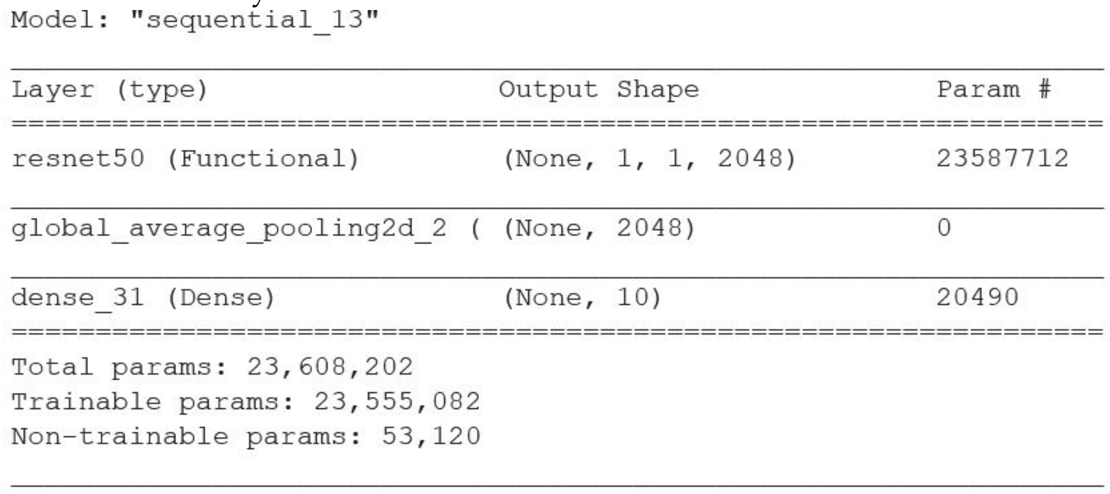
MobileNet is a simple but efficient convolutional neural network model developed by a team of Google engineers in 2017 .It is not very computationally intensive CNN. Therefore can use for mobile application

Validation accuracy is 93.68% and zero one loss is 0.06317.

3.3.3 Resnet

Residual network (ResNet) is a convolutional neural network model developed by Kaiming He et al in 2015. It is an extremely deep neural network with 152 layers.ResNet-50 that is a smaller version of ResNet 152.

Validation accuracy is 97.85% and zero one loss is 0.02155.



```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Functional)        (None, 1, 1, 2048)        23587712
_________________________________________________________________
global_average_pooling2d_2 ( (None, 2048)              0
_________________________________________________________________
dense_31 (Dense)             (None, 10)                20490
=================================================================
Total params: 23,608,202
Trainable params: 23,555,082
Non-trainable params: 53,120
_________________________________________________________________
```

Model name	Zero one loss
VGG16	0.04630
MobileNet	0.06317
Resnet50	0.02155

Table 3.2: Comparison performance on Modern CNN

Chapter 4

Conclusion

All the neural network architectures mentioned here provided good performance with this particular problem. The CNN model mentioned in the section 3.2 provided best performance. CNN perform better in the image processing and vision related task. Simple FNN with one hidden dense layers perform better than modern CNN discussed in here (VGG16 , MobileNet and Resnet50). Large dataset required complex model to learn a task and perform well with overffiting

In neural network we can increase the accuracy by tuning the parameters of the network. numbers of the nodes per layer and number of the layers are the most important parameters. we need to do the experiment and find the best for our specific dataset. Using too few nodes in the hidden layer will result underfitting and too many nodes will results in the overfitting.We need to consider more about overfitting with a small dataset like this. As well many nodes results in increasing training time. With FNN adding one or two additional dense hidden layers are sufficient for majority of the problem. Increasing complexity not always guarantee the best performance.

Bibliography

[1] Mengying Shu. Deep learning for image classification on very small datasets using transfer learning using transfer learning. pages 8–26, 2019.

[2] zero one loss. https://scikit-learn.org/, 2020.

[3] Dropouts and batchnormalization. https://analyticsindiamag.com/everything-you-should-know-about-dropouts-and-batchnormalization-in-cnn/, 2020.

[4] Learning rate. https://en.wikibooks.org/wiki/Artificial$_Neural_Networks$, 2021.