In the name of God

University of Tehran
College of Engineering
Faculty of ECE

Intelligent Systems

HW4

Reza Jahani

810198377

Fall 1401

Table of Contents

Analytical Implementation of MLP	***3***
NN Applications in Classification	***8***
- ***MLP***	***8***
- ***CNN + MLP***	***17***
Transfer Learning for EfficientNet	***22***
Additional Section	***28***

Question 1) Multi-Hidden Layer Neural Network Perceptron

Part 1) Analytical Implementation

In this section we want to train a neural network with 3 hidden layers based on figure 1.1.1.

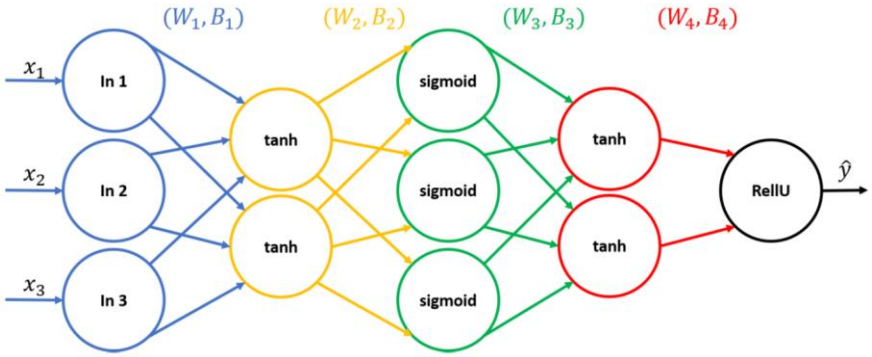


Figure 1.1.1: Neural Network Model

Below the equations used for forward propagation of the network is driven. Also the data which we are going to train the model with is presented.

$$X_1 = \begin{pmatrix} 7 \\ 7 \\ 7 \end{pmatrix} Y_1 = 7 \quad X_2 = \begin{pmatrix} 7 \\ 7 \\ 7 \end{pmatrix} Y_2 = 7$$

Technically the equations for each layer will be derived as:
$Z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$ where $a^{[0]}$ would be X(data) and other $a^{[l]}$ would be the value of $f^{[l]}(Z^{[l]})$ that $f^{[l]}(.)$ is the activation function of the $l'th$ hidden layer.

But in this example we name different variables after each layer's output and the following equations will be derived as below.

$$A_1 = W_1 X + B_1 \rightarrow \ Z = \tanh(A_1)$$
$$A_2 = W_2 Z + B_2 \rightarrow \ K = sigmoid(A_2)$$
$$A_3 = W_3 K + B_3 \rightarrow \ P = \tanh(A_3)$$
$$A_4 = W_4 P + B_4 \rightarrow \hat{y} = Relu(A_4)$$

Before we go any further the shape of these parameters stated in above equations are checked.

$$X\ (3,1)$$
$$W_1\ (2,3)\ \ B_1\ (2,1)\ \ A_1, Z\ \ (2,1)$$
$$W_2\ (3,2)\ \ B_2\ (3,1)\ \ A_2, K\ (3,1)$$
$$W_3\ (2,3)\ \ B_3\ (2,1)\ \ A_3, P\ (2,1)$$
$$W_4\ (1,2)\ \ B_4\ (1,1)\ \ A_4, \hat{y}\ \ (1,1)$$

Based on the instruction the initial parameters to start the training is presented below.

$$W_1 = \begin{pmatrix} 0.17 & 0.37 & 0.57 \\ 0.27 & 0.47 & 0.47 \end{pmatrix} \; W_2 = \begin{pmatrix} 7.15 & 7.45 \\ 7.25 & 7.55 \\ 7.35 & 7.65 \end{pmatrix} \; W_3 = \begin{pmatrix} 49.12 & 49.32 & 49.52 \\ 49.22 & 49.42 & 49.62 \end{pmatrix} \; W_4 = (0.16 \quad 0.36)$$

$$B_1 = \begin{pmatrix} 0.71 \\ 0.72 \end{pmatrix} \; B_2 = \begin{pmatrix} 7.15 \\ 7.25 \\ 7.35 \end{pmatrix} \; B_3 = \begin{pmatrix} 0.995 \\ 1.095 \end{pmatrix} \; B_4 = (0.26)$$

Now that the parameters are initialized with a starting value, we perform the forward propagation and then the backward propagation to update the parameters. This cycle will be done twice. First with (X_1, Y_1) training sample and then with (X_2, Y_2) sample. The cost function we want to minimize in each iteration is presented below.

$$E = \frac{1}{2}(\hat{y} - y)^2$$

Before starting the training procedure, we derive the backpropagation equations for derivatives and the updated parameters.
In each iteration (One forward propagation and one backward propagation) parameters will be updated as below.

$$W_n = W_n - \alpha D(W_n) \qquad D(W_n) = \frac{dE}{dW_n} \qquad n = 1,2,3,4$$

$$B_n = B_n - \alpha D(B_n) \qquad D(B_n) = \frac{dE}{dB_n} \qquad n = 1,2,3,4$$

$$n: Indicating\ number\ of\ Layer$$

Activation functions derivatives:

$$a = \sigma(z) \rightarrow \frac{da}{dz} = a(1-a) \;\Big|\; a = \tanh(z) \rightarrow \frac{da}{dz} = 1 - a^2 \;\Big|\; a = Relu(z) \rightarrow \frac{da}{dz} = u(z)$$

Some Notations:

$$* : Elementwise\ product \quad | \quad . : Broadcasting\ Product \quad | \quad u(z) = 1\ for\ z \geq 0\ , 0\ for\ z < 0$$

It can be illustrated that backward propagation equations are derived as below. Let l indicate the number of layer where the maximum value(4) stands for output layer and minimum value(0) stands for the input layer.

$$\frac{dE}{dz^{[l]}} = \frac{dE}{da^{[l]}} * \frac{da^{[l]}}{dz^{[l]}}$$

$$\frac{dE}{dW^{[l]}} = \frac{dE}{dz^{[l]}} \frac{dz^{[l]}}{dW^{[l]}} = \frac{dE}{dz^{[l]}} \cdot \left(a^{[l-1]}\right)^T$$

$$\frac{dE}{dB^{[l]}} = \frac{dE}{dz^{[l]}} . 1 = \frac{dE}{dz^{[l]}}$$

$$\frac{dE}{da^{[l-1]}} = \left(W^{[l]}\right)^T \frac{dE}{dz^{[l]}}$$

Backward Propagation Equations

$$D(W_4) = (\hat{y} - y)u(A_4)P^T \quad , \quad D(B_4) = (\hat{y} - y)u(A_4)$$

$$D(W_3) = \left[\left(W_4^T(\hat{y} - y)u(A_4)\right) * (1 - P^2)\right].K^T \ , \ D(B_3) = \left(W_4^T(\hat{y} - y)u(A_4)\right) * (1 - P^2)$$

$$D(W_2) = \left(W_3^T\left(W_4^T(\hat{y} - y)u(A_4)\right) * (1 - P^2)\right) * \left(K(1 - K)\right).Z^T$$

$$D(B_2) = \left(W_3^T\left(W_4^T(\hat{y} - y)u(A_4)\right) * (1 - P^2)\right) * \left(K(1 - K)\right)$$

$$D(W_1) = \left(W_2^T\left(\left(W_3^T\left(W_4^T(\hat{y} - y)u(A_4)\right) * (1 - P^2)\right) * \left(K(1 - K)\right)\right)\right) * (1 - Z^2).X^T$$

$$D(B_1) = \left(W_2^T\left(\left(W_3^T\left(W_4^T(\hat{y} - y)u(A_4)\right) * (1 - P^2)\right) * \left(K(1 - K)\right)\right)\right) * (1 - Z^2)$$

Let the learning rate $\alpha = 0.35$

In next page the hand implementation of the neural network is presented.

Since the initial values of the weights and biases (Trainable Parameters) are high (Which was out of my control because they were parametric with student ID) the network faces "Vanishing Gradient" problem and after first iteration, the gradients become zero and updated values will not be different with the beginning value. However, this statement doesn't apply for W_4 and B_4.

More detail can be observed in next page where the neural network is implemented.

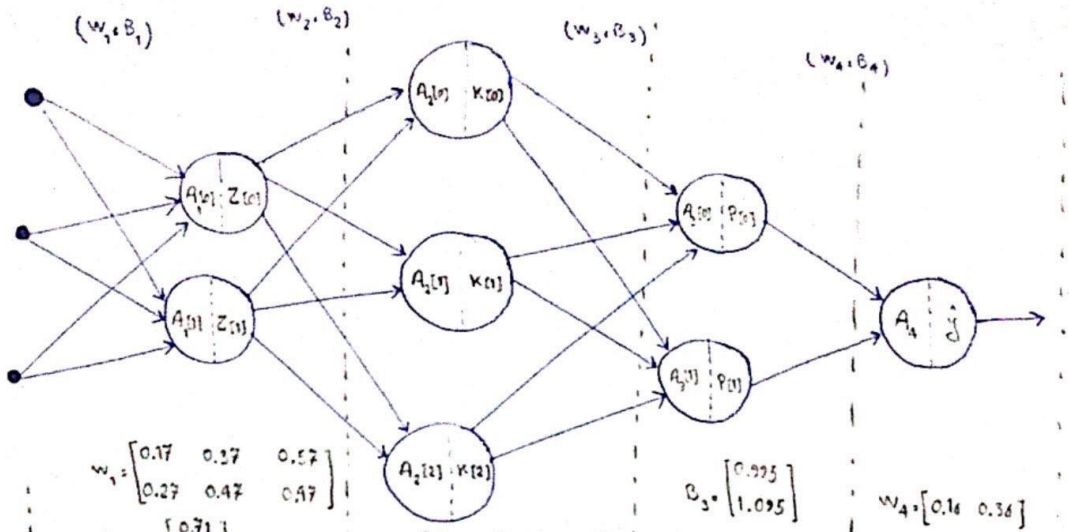
Final trained parameters would be as follow.

$$W_1 = \begin{pmatrix} 0.17 & 0.37 & 0.57 \\ 0.27 & 0.47 & 0.47 \end{pmatrix} W_2 = \begin{pmatrix} 7.15 & 7.45 \\ 7.25 & 7.55 \\ 7.35 & 7.65 \end{pmatrix} W_3 = \begin{pmatrix} 49.12 & 49.32 & 49.52 \\ 49.22 & 49.42 & 49.62 \end{pmatrix} W_4 = (2.33 \quad 2.43)$$

$$B_1 = \begin{pmatrix} 0.71 \\ 0.72 \end{pmatrix} B_2 = \begin{pmatrix} 7.15 \\ 7.25 \\ 7.35 \end{pmatrix} B_3 = \begin{pmatrix} 0.995 \\ 1.095 \end{pmatrix} B_4 = (2.32)$$

$$Cost\ Function\ after\ training:$$

$$E = \frac{1}{2}(7.31 - 7)^2 = 0.0481$$



#1 $\xrightarrow{FP}$

$X = \begin{bmatrix} 7 \\ 7 \\ 7 \end{bmatrix}$

$Y = 7$

$W_1 = \begin{bmatrix} 0.17 & 0.37 & 0.57 \\ 0.27 & 0.47 & 0.67 \end{bmatrix}$ $B_1 = \begin{bmatrix} 0.71 \\ 0.72 \end{bmatrix}$

$A_1 = \begin{bmatrix} 8.48 \\ 9.19 \end{bmatrix}$ $Z = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$W_2 = \begin{bmatrix} 7.15 & 7.45 \\ 7.25 & 7.55 \\ 7.35 & 7.65 \end{bmatrix}$ $B_2 = \begin{bmatrix} 7.15 \\ 7.25 \\ 7.35 \end{bmatrix}$

$A_2 = \begin{bmatrix} 21.75 \\ 22.05 \\ 22.35 \end{bmatrix}$ $K = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

$W_3 = \begin{bmatrix} 49.12 & 49.32 & 49.52 \\ 49.22 & 49.42 & 49.62 \end{bmatrix}$ $B_3 = \begin{bmatrix} 0.995 \\ 1.095 \end{bmatrix}$

$A_3 = \begin{bmatrix} 148.95 \\ 149.35 \end{bmatrix}$ $P = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$W_4 = \begin{bmatrix} 0.16 & 0.36 \end{bmatrix}$ $B_4 = 0.26$

$A_4 = 0.78$ $\hat{y} = 0.78$

$D(W_1) = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$ $D(B_1) = \begin{bmatrix} \cdot \\ \cdot \end{bmatrix}$

$W_1 \leftarrow \begin{bmatrix} 0.17 & 0.37 & 0.57 \\ 0.27 & 0.47 & 0.67 \end{bmatrix}$

$B_1 \leftarrow \begin{bmatrix} 0.71 \\ 0.72 \end{bmatrix}$

$D(W_2) = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$ $D(B_2) = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}$

$W_2 \leftarrow \begin{bmatrix} 7.15 & 7.45 \\ 7.25 & 7.55 \\ 7.35 & 7.65 \end{bmatrix}$

$B_2 \leftarrow \begin{bmatrix} 7.15 & 7.25 & 7.35 \end{bmatrix}^T$

$D(W_3) = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$

$D(B_3) = \begin{bmatrix} \cdot \\ \cdot \end{bmatrix}$

$W_3 \leftarrow \begin{bmatrix} 49.12 & 49.32 & 49.52 \\ 49.22 & 49.42 & 49.62 \end{bmatrix}$

$B_3 \leftarrow \begin{bmatrix} 0.995 \\ 1.095 \end{bmatrix}$

$D(W_4) = \begin{bmatrix} -6.22 & -6.22 \end{bmatrix}$ #1 BP

$D(B_4) = -6.22$ BP $\leftarrow$

$W_4 \leftarrow W_4 - \alpha D(W_4)$

$B_4 \leftarrow B_4 - \alpha D(B_4)$

$W_4 \leftarrow \begin{bmatrix} 2.237 & 2.537 \end{bmatrix}$

$B_4 \leftarrow 2.457$

#2 $\xrightarrow{FP}$

$X = \begin{bmatrix} 7 \\ 7 \\ 7 \end{bmatrix}$

$Y = 7$

$A_1 = \begin{bmatrix} 8.48 \\ 9.19 \end{bmatrix}$ $Z = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$A_2 = \begin{bmatrix} 21.75 \\ 22.05 \\ 22.35 \end{bmatrix}$ $K = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

$A_3 = \begin{bmatrix} 148.95 \\ 149.35 \end{bmatrix}$ $P = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$A_4 = 7.31$ $\hat{y} = 7.31$

$D(W_1) = 0$ $D(B_1) = 0$

$W_1 \leftarrow W_1^{(1)}$

$B_1 \leftarrow B_1^{(1)}$

$D(W_2) = 0$ $D(B_2) = 0$

$W_2 \leftarrow W_2^{(1)}$

$B_2 \leftarrow B_2^{(1)}$

$D(W_3) = 0$ $D(B_3) = 0$

$W_3 \leftarrow W_3^{(1)}$

$B_3 \leftarrow B_3^{(1)}$

$D(W_4) = \begin{bmatrix} 0.31 & 0.31 \end{bmatrix}$ #2 BP

$D(B_4) = 0.31$ BP $\leftarrow$

$W_4 \leftarrow \begin{bmatrix} 2.23 & 2.43 \end{bmatrix}$

$B_4 \leftarrow 2.32$

Final Trained Parameters :

$W_1 = \begin{bmatrix} 0.17 & 0.37 & 0.57 \\ 0.27 & 0.47 & 0.67 \end{bmatrix}$ $B_1 = \begin{bmatrix} 0.71 \\ 0.72 \end{bmatrix}$

$W_2 = \begin{bmatrix} 7.15 & 7.45 \\ 7.25 & 7.55 \\ 7.35 & 7.65 \end{bmatrix}$ $B_2 = \begin{bmatrix} 7.15 \\ 7.25 \\ 7.35 \end{bmatrix}$

$W_3 = \begin{bmatrix} 49.12 & 49.32 & 49.52 \\ 49.22 & 49.42 & 49.62 \end{bmatrix}$ $B_3 = \begin{bmatrix} 0.995 \\ 1.095 \end{bmatrix}$

$W_4 = \begin{bmatrix} 2.23 & 2.43 \end{bmatrix}$ $B_4 = 2.32$

So the final weights and parameters after training would be as follows.

$$W_1 = \begin{pmatrix} 0.17 & 0.37 & 0.57 \\ 0.27 & 0.47 & 0.47 \end{pmatrix} W_2 = \begin{pmatrix} 7.15 & 7.45 \\ 7.25 & 7.55 \\ 7.35 & 7.65 \end{pmatrix} W_3 = \begin{pmatrix} 49.12 & 49.32 & 49.52 \\ 49.22 & 49.42 & 49.62 \end{pmatrix} W_4 = (2.22 \quad 2.42)$$

$$B_1 = \begin{pmatrix} 0.71 \\ 0.72 \end{pmatrix} B_2 = \begin{pmatrix} 7.15 \\ 7.25 \\ 7.35 \end{pmatrix} B_3 = \begin{pmatrix} 0.995 \\ 1.095 \end{pmatrix} B_4 = (2.32)$$

After the analytical implementation of the neural network, the computations and calculations are implemented in python code and below results were obtained.

The following result illustrates the similarity of two methods of implementations.

```
Loss Function After 2 Iteration:0.05
Weights after Training
W1:[[0.17 0.37 0.57]
 [0.27 0.47 0.47]]
W2:[[7.15 7.45]
 [7.25 7.55]
 [7.35 7.65]]
W3:[[49.12 49.32 49.52]
 [49.22 49.42 49.62]]
W4:[[2.22815 2.42815]]
B1:[0.71 0.72]
B2:[7.15 7.25 7.35]
B3:[0.995 1.095]
B4:[2.32815]
```

Figure 1.1: Weights After Training

Question 2) Neural Network Application in Classification

- ## Section 1) MLP Network

In this section a neural network with a simple architecture is generated. Then the hyper parameters such as batch size for training, activation functions of each layer, loss function of the model and its optimizer and … are tuned. As we proceed, we show this tuning process with the following results.

1. Batch Size

For this part the network is optimized with stochastic gradient descent with a specific learning rate and loss function of categorical cross entropy. The structure of the model can be observed below.

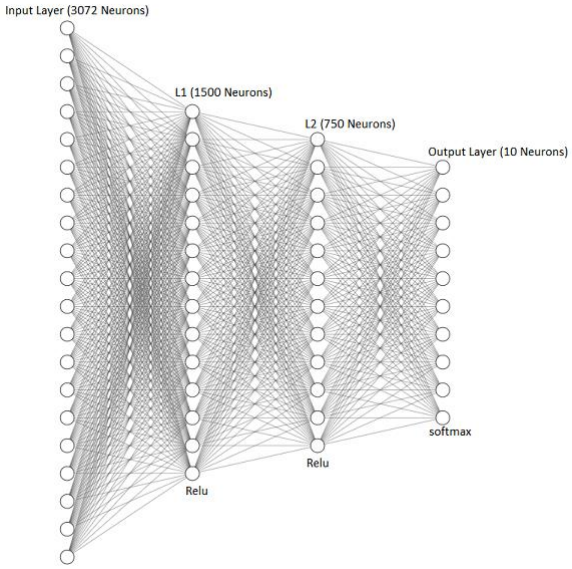
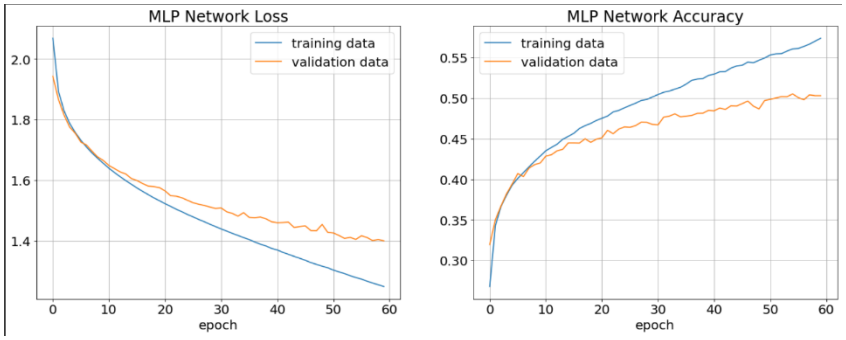


Figure 2.1.1: MLP Network first prototype

After this, the results will be presented in each of the batch sizes.

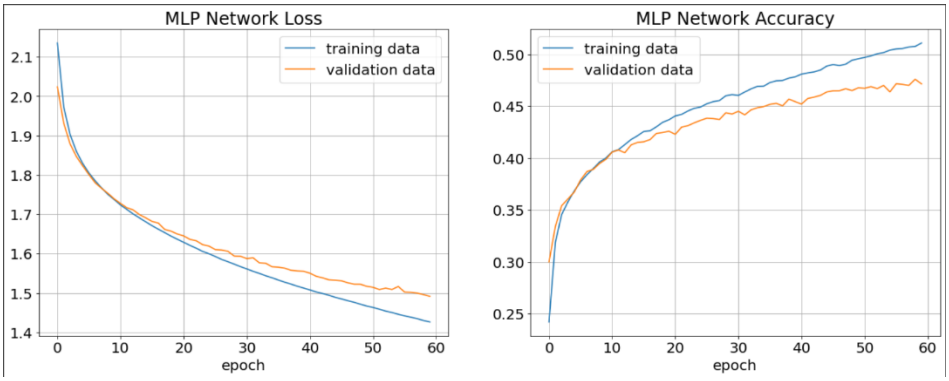
- Batch Size = 32




```
loss: 1.2485 - accuracy: 0.5738 - val_loss: 1.3992 - val_accuracy: 0.5030
```

Figure 2.1.2: Accuracy & Loss

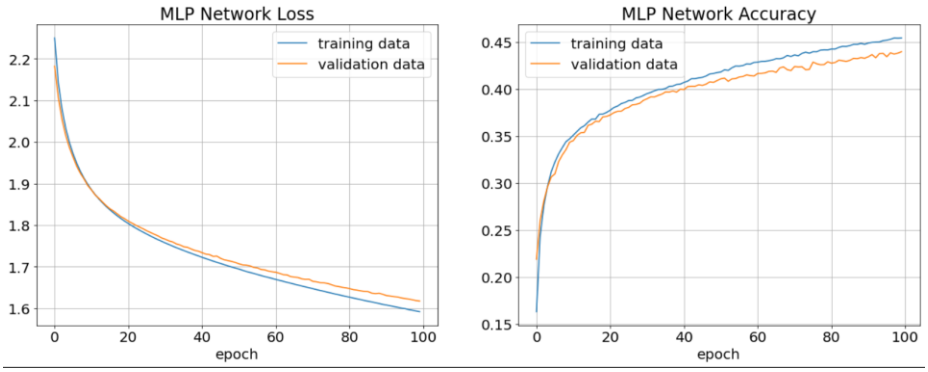
- Batch Size = 64




```
loss: 1.4262 - accuracy: 0.5109 - val_loss: 1.4913 - val_accuracy: 0.4716
```

Figure 2.1.3: Accuracy & Loss

- Batch Size = 256




```
loss: 1.5918 - accuracy: 0.4545 - val_loss: 1.6175 - val_accuracy: 0.4399
```

Figure 2.1.4: Accuracy & Loss

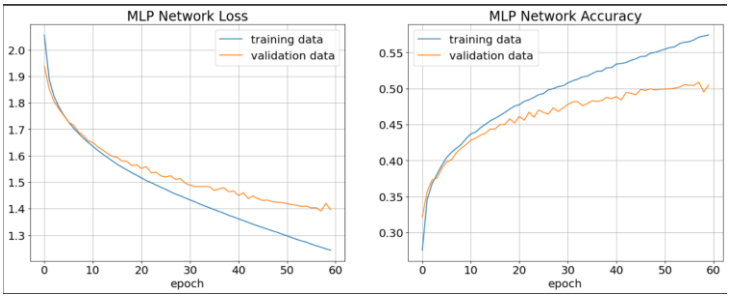
As we can observe in above figures, obtained final results for the model which was trained with mini batch with size of 32 performs better than other two options.

In case of consumed time for training of each network with different configurations, the network with batch size of 32 was trained 60 epochs which longed 5 seconds per epoch. The one with batch size of 64 was trained 60 epochs that each epoch longed 3 seconds per epoch. And the final model that had batch size of 256, was trained 100 epochs which each epoch lasted 1 second per epoch.

2. Activation Functions

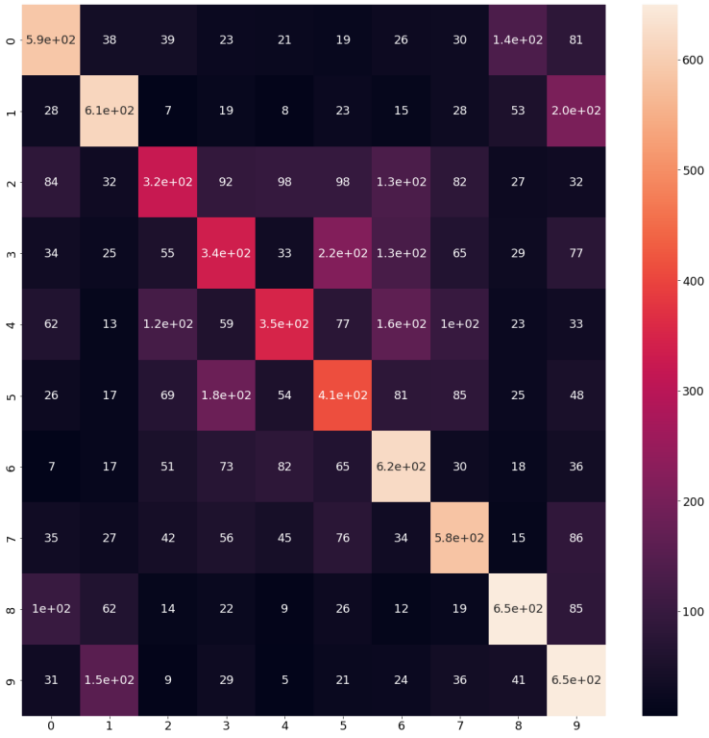
In this part activation functions of each layer was changed and the performance was evaluated throughout the training process. Obtained results won't perform better than the previous model and following results can be observed below.

- [RELU-RELU-SIGMOID]




```
loss: 1.2428 - accuracy: 0.5742 - val_loss: 1.3960 - val_accuracy: 0.5050
```

Figure 2.2.1: Activation Functions = [RELU-RELU-SIGMOID]



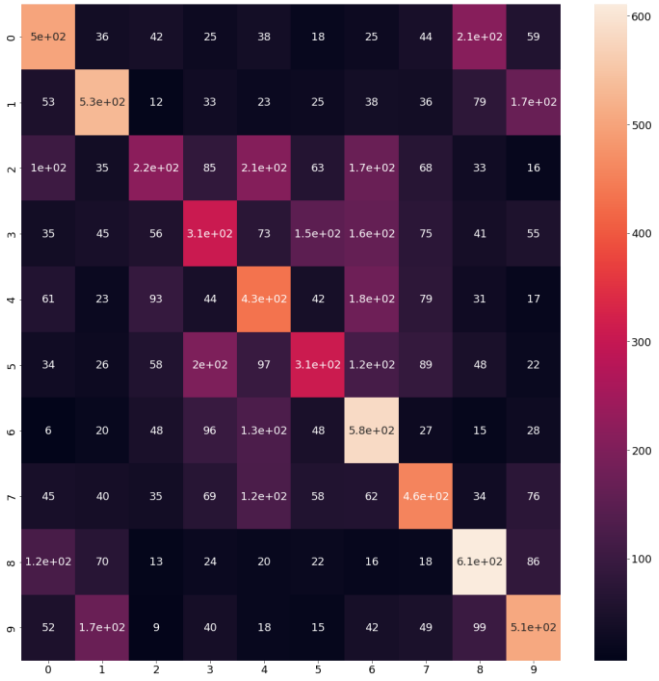

```
Model Accuracy=51.2%
Model Loss=1.38
```

Figure 2.2.2: Test Data on Model

- [RELU-SIGMOID-SIGMOID]


```
loss: 1.5809 - accuracy: 0.4502 - val_loss: 1.5999 - val_accuracy: 0.4434
```

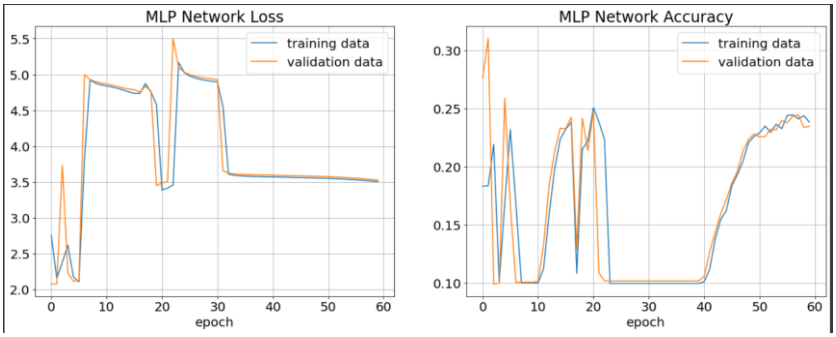
Figure 2.2.3: Activation Functions = [RELU-SIGMOID-SIGMOID]



```
Model Accuracy=44.6%
Model Loss=1.59
```

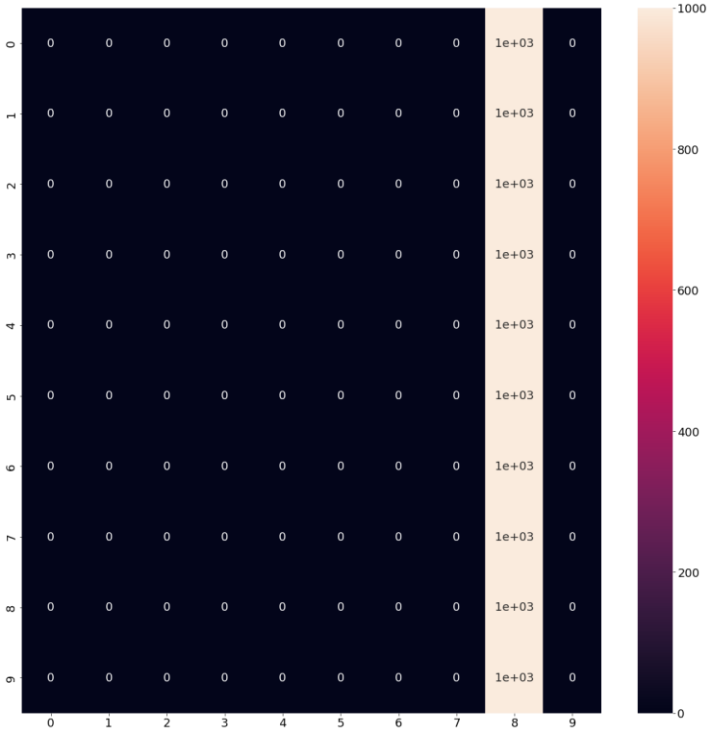
Figure 2.2.4: Model Evaluation

- [RELU-TANH-RELU]




```
loss: 3.5008 - accuracy: 0.2380 - val_loss: 3.5216 - val_accuracy: 0.2345
```

Figure 2.2.5: Activation Functions = [RELU-TANH-RELU]



```
Model Accuracy=10.0%
Model Loss=3.56
```

Figure 2.2.6: Model Evaluation

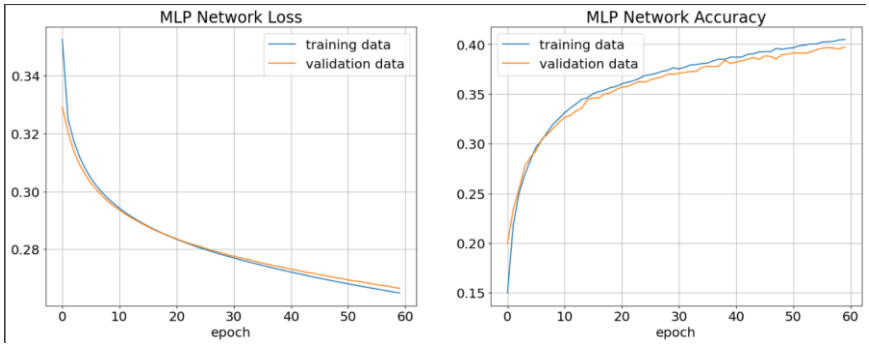
As it can be seen in results obtained above, no network performed better than the trained network in previous part. So we will continue with the first network for further parts.

Also as it can be seen in last part, RELU is not a good activation function because of its zero gradient and also we can deduce that when we switched from Soft Max to Sigmoid, the performance was no longer adequate enough since vanishing gradient steps up. It can be deduced based on experiment that Relu is a good activation function for layers before output layer while the optimal choice for output layer is Softmax.

3. Loss Functions

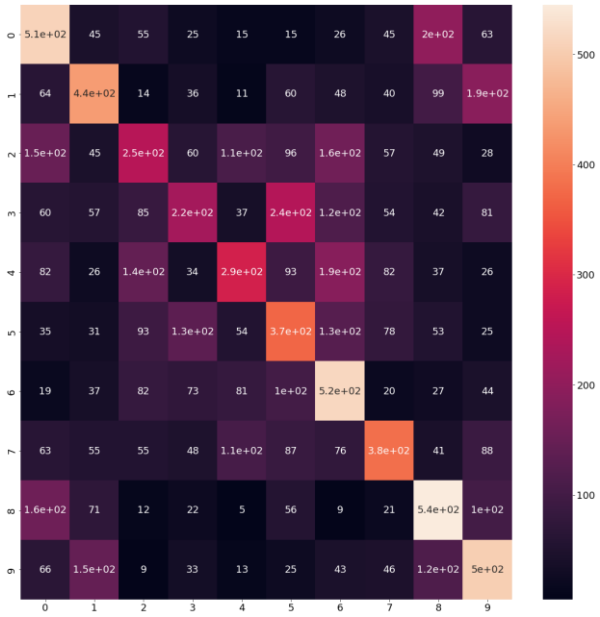
As we have proceeded so far, the optimal performance we achieved was with training with batch size of 32 and activation functions of [RELU, RELU, SOFTMAX]. As instructed we shall not change these hyper parameters anymore and move on to other hyper parameters. In this part we want to test 2 different loss functions and observe the performance.
So far we have been using “Categorical Cross Entropy” loss function.

- Loss Function = Binary Cross Entropy




```
loss: 0.2649 - accuracy: 0.4048 - val_loss: 0.2665 - val_accuracy: 0.3972
```

Figure 2.3.1

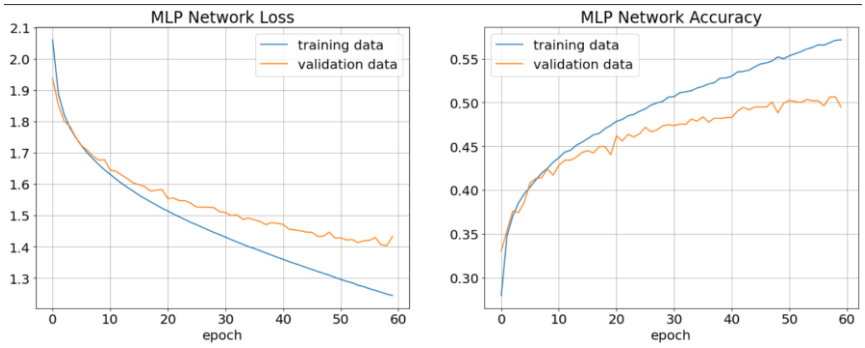


```
Model Accuracy=40.2%
Model Loss=0.27
```

Figure 2.3.2: Model Evaluation

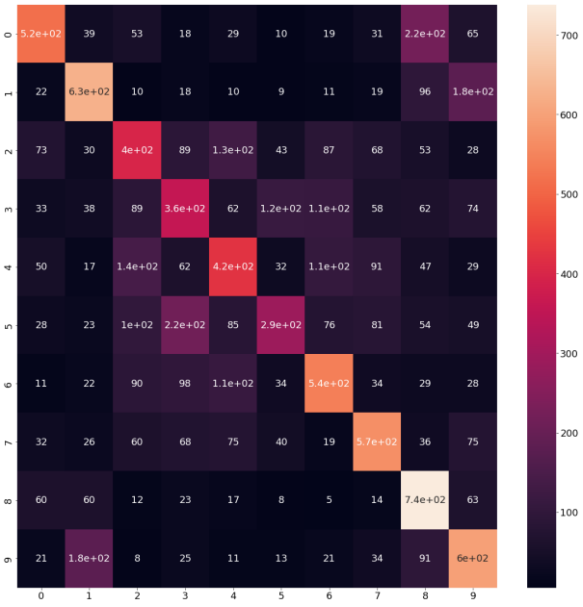
As it can be observed above in figure 2.3.1, achieved results are not better than the model trained with categorical cross entropy. Furthermore, one more loss function is tested and the results are presented.

- Loss Function = Kullback Leibler Divergence




```
loss: 1.2447 - accuracy: 0.5715 - val_loss: 1.4333 - val_accuracy: 0.4946
```

Figure 2.3.3



```
Model Accuracy=50.6%
Model Loss=1.39
```

Figure 2.3.4: Model Evaluation

As we can deduce, by changing the loss function to this specific one, no significant improvement can be observed. Previous models had better accuracies.

4. Optimizers

In this section we try 2 different optimizers instead of ***Stochastic Gradient Descent*** optimizer that we have been using so far. Following results will be presented.

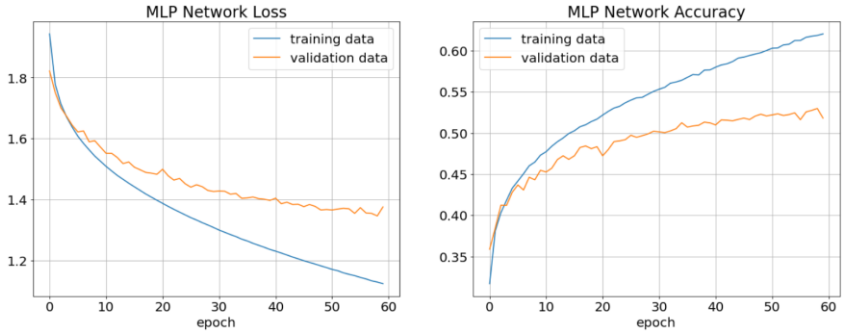
- Adam Optimizer


```
loss: 0.9850 - accuracy: 0.6445 - val_loss: 1.9406 - val_accuracy: 0.4801
```

Figure 2.4.1

The model gets over fit after 6 epochs and after that accuracy of validation set doesn't change and validation loss increases. So this optimizer should not be used hence the following result is not adequate enough.

- Adagrad Optimizer




```
loss: 1.1294 - accuracy: 0.6184 - val_loss: 1.3463 - val_accuracy: 0.5297
```

Figure 2.4.2

Employing this optimizer, our results will be improved and as we can see, we reached a higher accuracy and lower loss.

5. Optimal Model Hyper Parameters

Based on evaluations and our searches through different values of hyper parameters such as batch size for training, activation functions, optimizers, and loss functions, the best model can be chosen with the following hyper parameters.

$$Batch\ Size = 32$$
$$Activation\ Functions = [RELU - RELU - SOFTMAX]$$
$$Loss\ Function = Categorical\ Cross\ Entropy$$
$$Optimizer = Adagrad\ (Learning\ Rate = 0.001)$$

As the model is trained with above configuration, it is evaluated by the test data which was separated in the beginning.

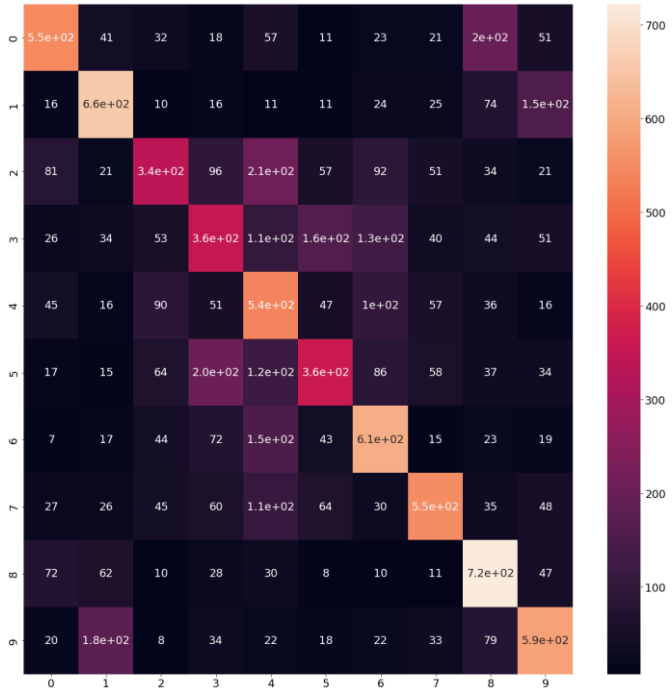


Figure 2.5.1: Confusion Matrix

```
Model Accuracy=52.7%
Model Loss=1.34
```

Figure 2.5.2: Accuracy & Loss of the Model

```
Recall Score=0.53
Precision Score=0.53
F1-Score=0.52
```

Figure 2.5.3: Other Evaluation Criteria

- Section 2) MLP + CNN Networks

Part 1)

After finding the optimum network with fully connected layers, 3 convolutional layers are added to the beginning of the network. First and second layer has 32 filters with kernels of size (5, 5) and stride value od (2,2) while the third convolutional layer has 16 filters of kernel size of (2,2) and stride value of (1,1). All convolutional layers have RELU activation function. After this procedure, the model's accuracy and loss would be as presented below.

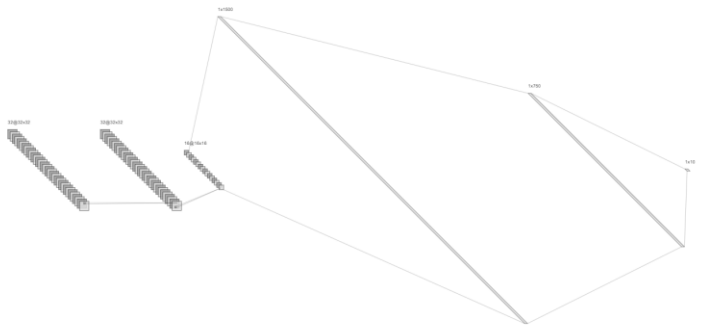
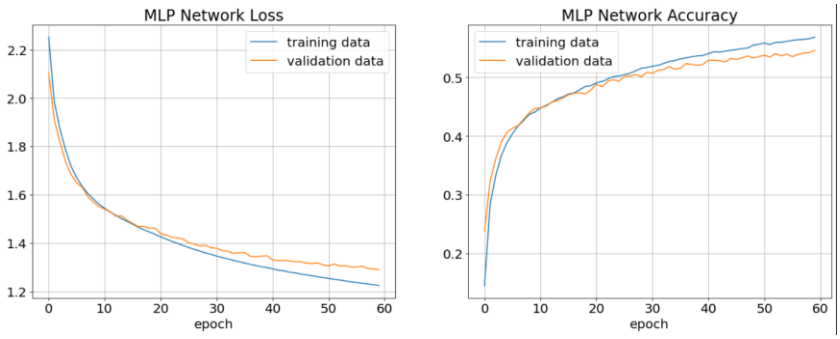


Figure 2.2.1.1: Network's Architecture

Below the accuracy and loss on training and validation data is presented.




```
loss: 1.2251 - accuracy: 0.5686 - val_loss: 1.2911 - val_accuracy: 0.5461
```

Figure 2.2.1.2

In next page, the test data is employed to check the model's performance. As it can be observed in figure 2.2.1.3, the performance of the model is improved.

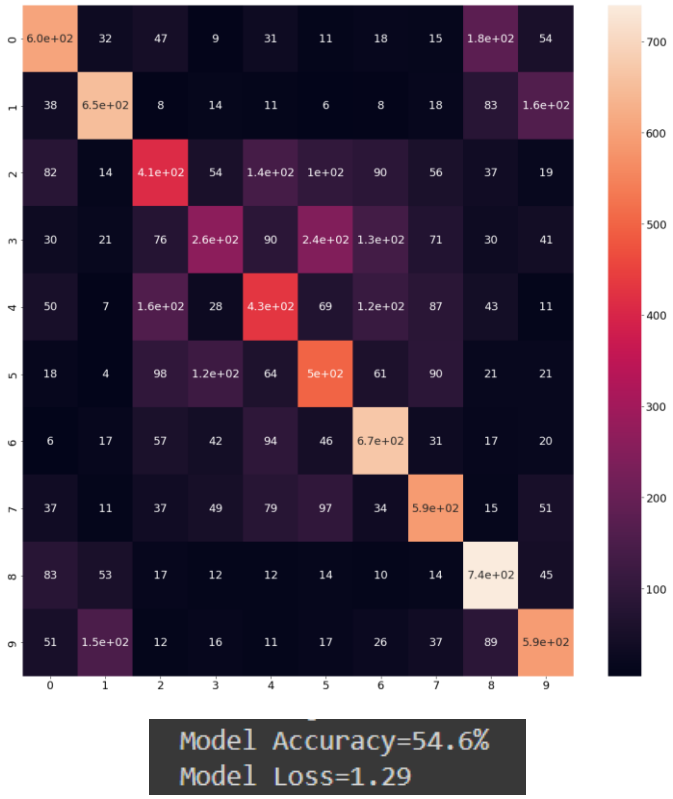


Figure 2.2.1.3: Model Evaluation

Part 2)

- Pooling

Pooling is an operation that is conducted on a square of 2D data. This operation is employed to down sample. For instance, making a 8 by 8 2D matrix to a 4 by 4 2D matrix. Taking every square of 2 by 2 and replacing it with one data. Max pooling takes the maximum of elements between 4 items and sets the replaced element with this value. Average Pooling takes the average and so on. The purpose of the pooling layers is to reduce the dimensions of the hidden layer by combining the outputs of neuron clusters at the previous layer into a single neuron in the next layer.
An instance of pooling is shown in figure 2.2.1.4.

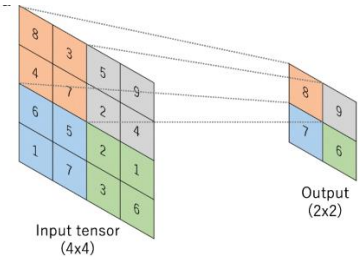
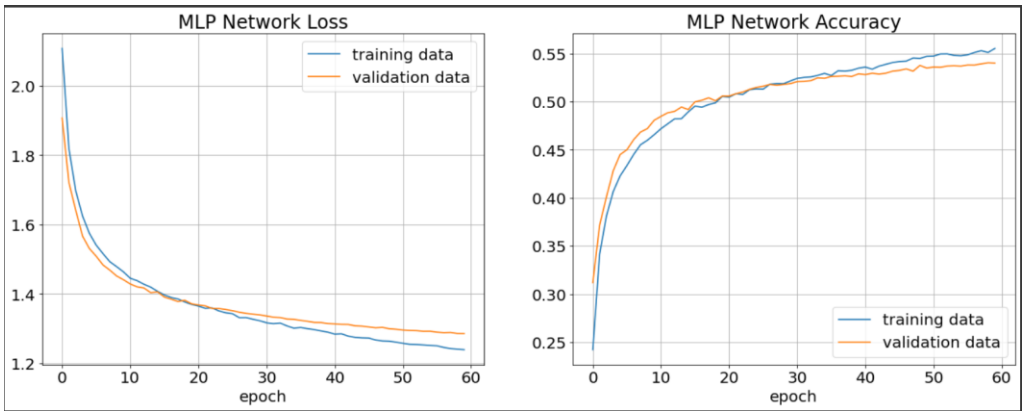


Figure 2.2.2.1

- Batch Normalization

Batch normalization is a technique employed to make the training of a neural network faster and more stable by normalizing the output values of a layer by re-centering and re-scaling.

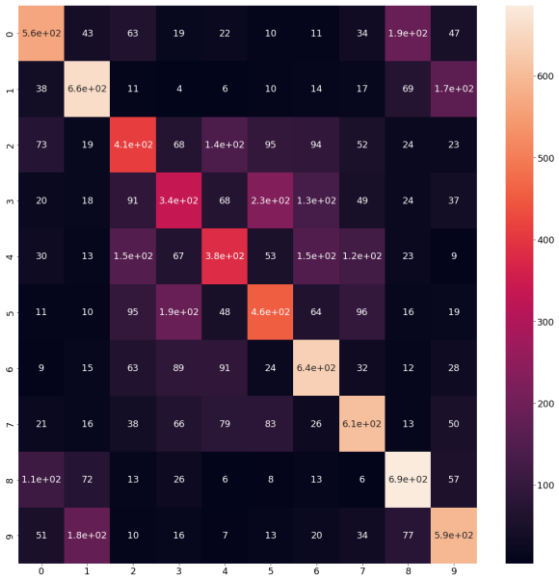
After adding batch normalization and average pooling layers to the network the accuracy and loss of the network will change the way observed in figure 2.2.1.5.




```
loss: 1.2393 - accuracy: 0.5554 - val_loss: 1.2857 - val_accuracy: 0.5402
```

Figure 2.2.2.2

Evaluating the model, we will reach below results.

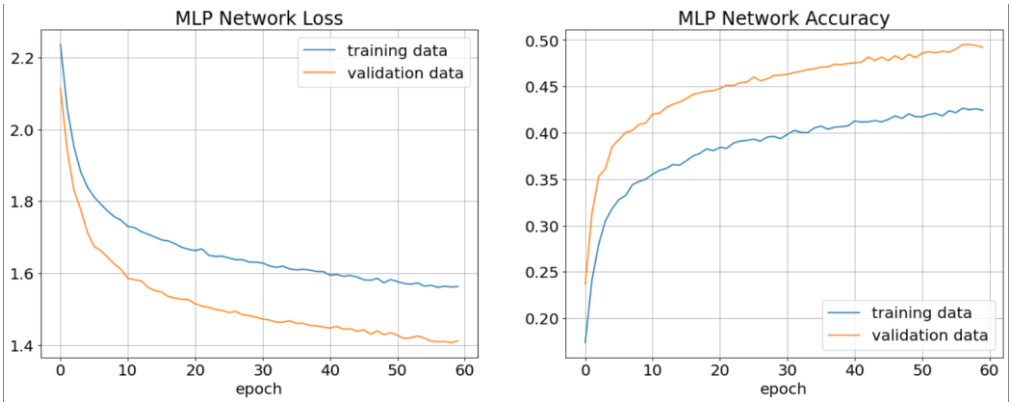


```
Model Accuracy=53.400000000000006%
Model Loss=1.28
```

Figure 2.2.2.3: Model Evaluation

Part 3)

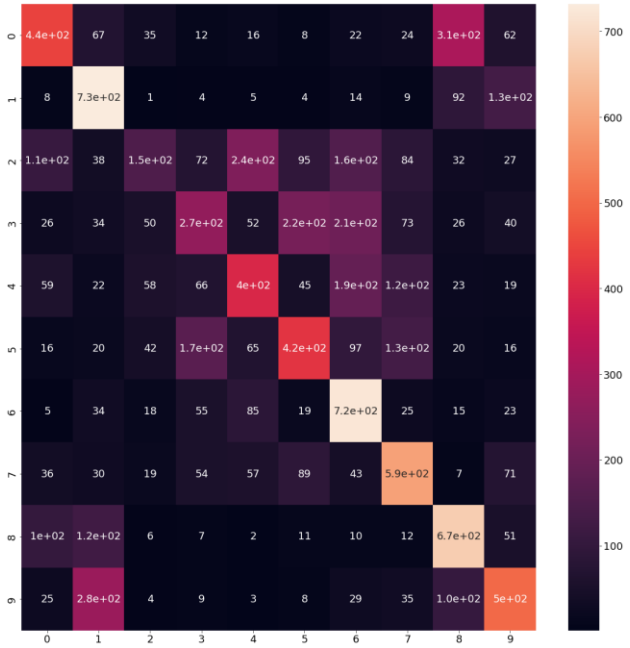
In this part we aim to improve the performance by adding drop out to the layers. Below the accuracy and loss of the model is presented while training. Drop out is used to drop some of neurons with a specific probability to avoid overfitting.




```
loss: 1.5627 - accuracy: 0.4242 - val_loss: 1.4115 - val_accuracy: 0.4918
```

Figure 2.2.3.1

Model is evaluated employing the test data and the result is observed below.



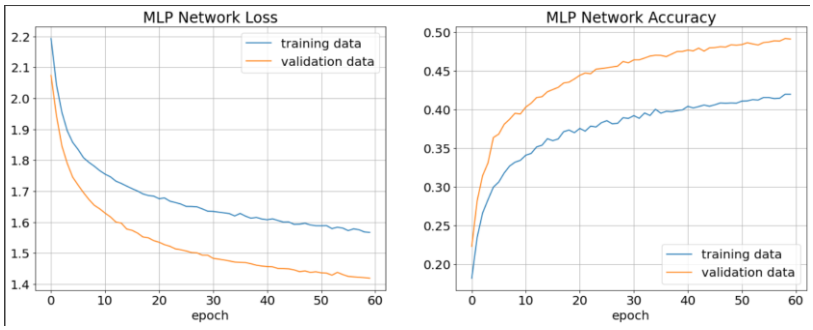
```
Model Accuracy=49.0%
Model Loss=1.41
```

Figure 2.2.3.2: Model Evaluation

Part 4)

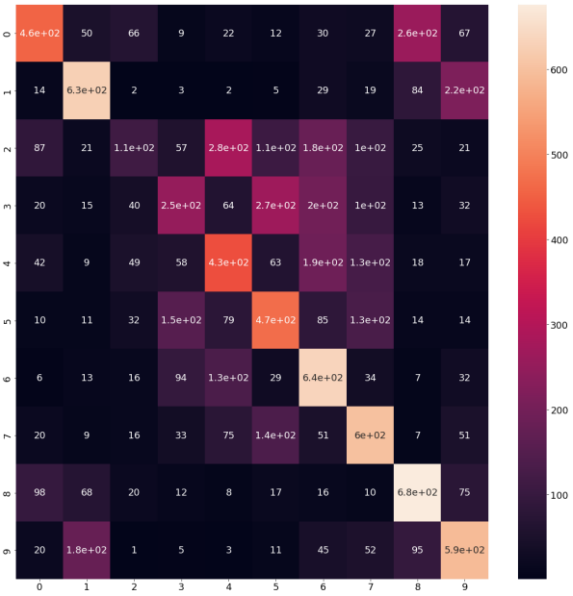
By adding early stopping, we check the performance again. Early stopping is mostly used to make sure the training of the model is efficient and the model is prevented from being over fit. As a matter of fact, by adding early stopping the training stops when the model is about to get over fit.

The result can be observed below. Both while training and testing.




```
loss: 1.5668 - accuracy: 0.4196 - val_loss: 1.4189 - val_accuracy: 0.4910
```

Figure 2.2.4.1



```
Model Accuracy=48.5%
Model Loss=1.41
```

Figure 2.2.4.2: Model Evaluation

Question 3) Transfer Learning for Efficient Net

Section 1) Introduction to EfficientNet

Generally, these models are developed models of Convolutional Neural Networks with added different types of layers to make the whole network wider, deeper, and increase its resolution. Below short summaries of different aspects of model EfficientNetB0 is presented.

- Network's Architecture

To shortly explain the architecture of this model, we briefly present the modules and different units which are employed in forming this network. EfficientNetB0 has 237 layers while this number for EfficientNetB7 is 813. In figure 3.1.1, two of the units employed at the beginning and end of this network can be observed.

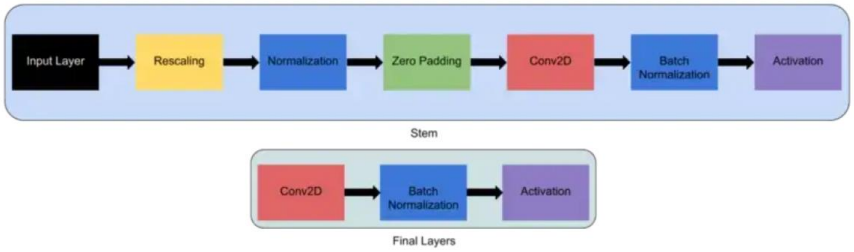


Figure 3.1.1: First & last unit of architecture

To make the architecture look more simple, the network is formed by 5 different modules in which different layers with different functionalities. In figure 3.1.2 these modules are briefly presented.

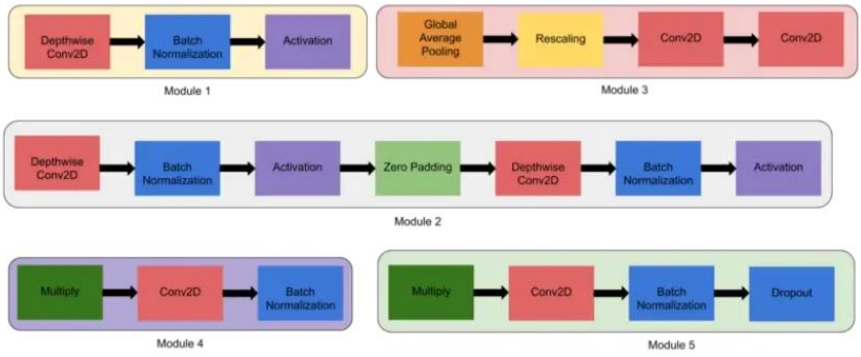


Figure 3.1.2: Modules with different functionalities

After being introduced to these modules and units and being familiar with their functionalities, the final architecture of the model can be presented.

In figure 3.1.3, the final architecture of the EfficientNetB0 model is presented.

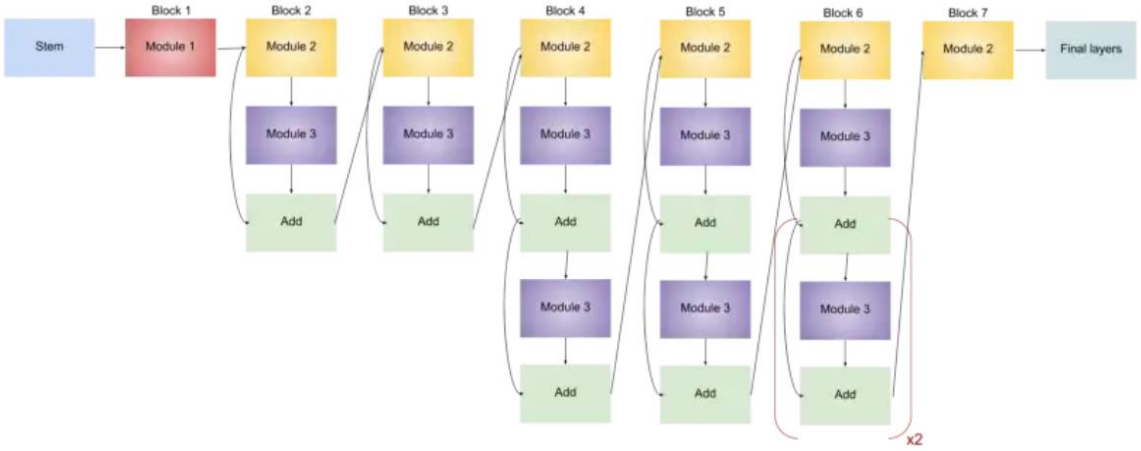


Figure 3.1.3: EfficientNetB0 Architecture

- Other Versions Architecture

As already presented above in figures 3.1.1 to 3.1.3, architecture of other versions of EfficientNet models are quite the same with a little difference of formation and number of units and modules employed in the network. For instance, EfficientNetB7 and EfficientNetB5 can be observed in figure 3.1.4 and 3.1.5.

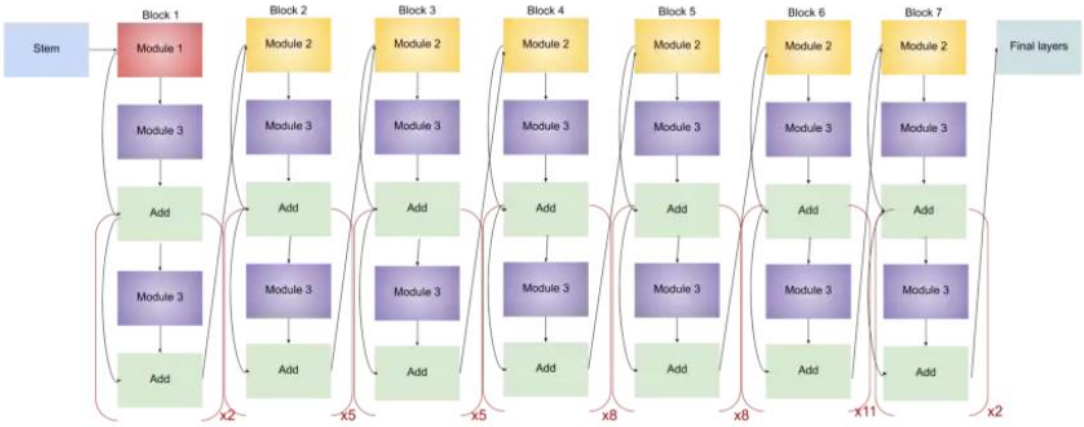


Figure 3.1.4: EfficientNetB7

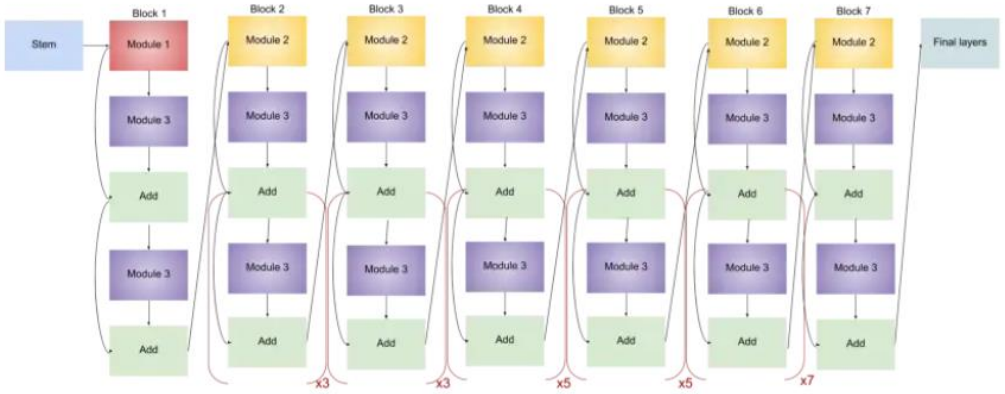


Figure 3.1.5: EfficientNetB5

As it can be observed, all of the models of this family has something similar and in common while the aspect that makes them different from each other, is mostly the formation and number of employed modules.

- Necessary Preprocessing of Input Image

As explained in the references the input of the model is 224 x 224 x 3 shaped image. So obviously when the picture is about to enter the network for classification, it should be reshaped to the following presented size so that the network performs well.

- Advantages & Disadvantages of the Model

In this part we present the results obtained from experiments on this family of models and state some of the advantages and disadvantages of this group of models.
As presented in figure 3.1.6, accuracy of different models per number of parameters can be seen.

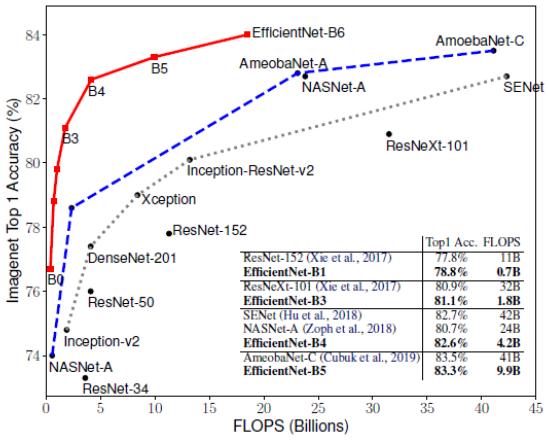


Figure 3.1.6

As observed above, with less trainable parameters, higher accuracy is obtained.

Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPS	Ratio-to-EfficientNet
EfficientNet-B0	**76.3%**	**93.2%**	**5.3M**	**1x**	**0.39B**	**1x**
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x
EfficientNet-B1	**78.8%**	**94.4%**	**7.8M**	**1x**	**0.70B**	**1x**
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M	7.6x	11B	16x
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M	4.3x	6.0B	8.6x
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M	3.0x	5.7B	8.1x
Xception (Chollet, 2017)	79.0%	94.5%	23M	3.0x	8.4B	12x
EfficientNet-B2	**79.8%**	**94.9%**	**9.2M**	**1x**	**1.0B**	**1x**
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M	5.2x	13B	13x
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M	6.1x	13B	13x
EfficientNet-B3	**81.1%**	**95.5%**	**12M**	**1x**	**1.8B**	**1x**
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M	7.0x	32B	18x
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M	7.7x	35B	19x
EfficientNet-B4	**82.6%**	**96.3%**	**19M**	**1x**	**4.2B**	**1x**
SENet (Hu et al., 2018)	82.7%	96.2%	146M	7.7x	42B	10x
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M	4.7x	24B	5.7x
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M	4.6x	23B	5.5x
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M	4.5x	23B	6.0x
EfficientNet-B5	**83.3%**	**96.7%**	**30M**	**1x**	**9.9B**	**1x**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
EfficientNet-B6	**84.0%**	**96.9%**	**43M**	**1x**	**19B**	**1x**
EfficientNet-B7	**84.4%**	**97.1%**	**66M**	**1x**	**37B**	**1x**
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-

We omit ensemble and multi-crop models (Hu et al., 2018), or models pretrained on 3.5B Instagram images (Mahajan et al., 2018).

Table 3.1.7

But not all good aspects are gathered in this family of models. As observed above the Ratios of other models are higher and this makes other models better than this family of models.

Section 2) Implementation Employing Transfer Learning Idea

In this section the main intention is to test the pre trained model with a test data. As obvious in the notebook, the picture which is given to the model is picture of a perfume as below.

Figure 3.2.1: Test Data

After the procedure of preprocessing which was resizing and rescaling the image, the data is fed to the model and based on the spectrum of the output (1000 neurons), highest 3 predictions are observed in figure 3.2.2.

	Label	Probability
0	perfume	0.609569
1	lotion	0.168529
2	soap_dispenser	0.024027

Figure 3.2.2: Model's prediction & their probabilities

As we can see, the model has successfully predicted the 'Perfume' label which is correct.

Section 3) Solving an Error

In this section we aim to check what is better to do when the test label is not among the labels which our model can predict. In this case the model's prediction is not trustable since the model has never been trained with that specific label.

To solve this problem the only way that we have is to change the output of the model from an specific label to the fact that model is not able to predict the label. For this purpose, we decide on a reasonable value for the probabilities (value of output neurons). If the highest probability after prediction is not higher than the specific value which we have chosen, instead of a specific label, the model states that "Model is not able to classify this image."

Let's move to the notebook and implement this solution.

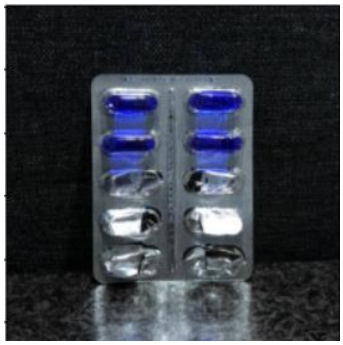


Figure 3.3.1: Pills

	Label	Probability
0	beaker	0.111231
1	vending_machine	0.060862
2	saltshaker	0.059228

Figure 3.3.2: Model's prediction

As it can be observed in figures 3.3.1 and 3.3.2, the model is not capable of distinguishing the picture's object and classification of the model is not trustworthy since the highest probability is 0.11.

The result after the correction can be observed in figure 3.3.3.

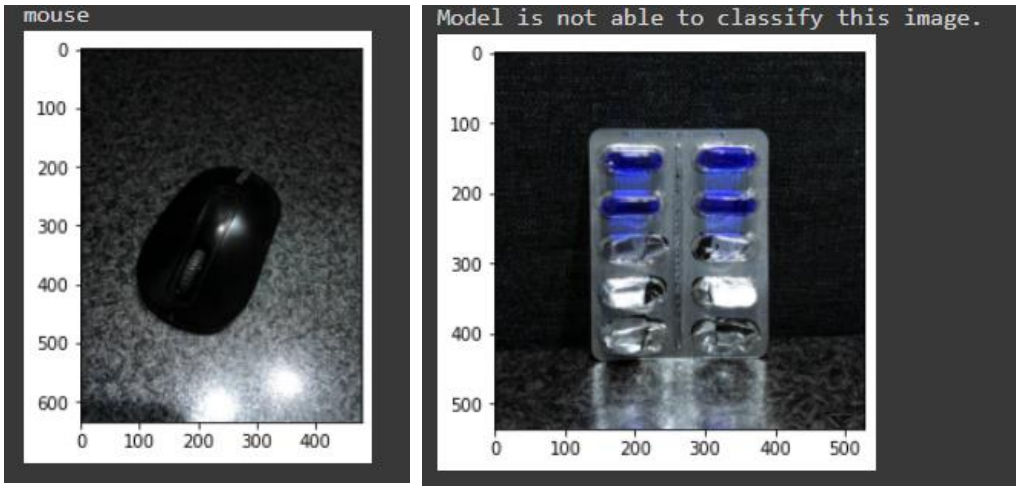


Figure 3.3.3

Presented above, the model's performance is fixed for not-trained cases.

Part 4) Training the Network with New Dataset

In this section we intend to train the network on new dataset, however the model is already trained on data with those labels. For this purpose, we employ the cifar-10 dataset but we only separate that partition of dataset that is related to only 2 labels of 'airplane' & 'Automobile'.
Idea of transfer learning is to apply a slight change on final layers of a network and retrain it. This is exactly what was conducted in this part.

In figure 3.4.1, the architecture of Transfer Learning based model and its training procedure can be observed. Due to the bad quality of the dataset the performance is not good enough. Unfortunately, I wasn't able to find a good dataset.

```
Model_New = EfficientNetB0(weights='imagenet')
TL_Model = Sequential()
TL_Model.add(Model_New)
TL_Model.add(Dense(500,activation='relu'))
TL_Model.add(Dropout(0.2))
TL_Model.add(BatchNormalization())
TL_Model.add(Dense(100,activation='relu'))
TL_Model.add(Dropout(0.4))
TL_Model.add(Dense(20,activation='relu'))
TL_Model.add(Dense(2,activation='softmax'))
TL_Model.compile(optimizer=Adam(learning_rate=0.1),loss='categorical_crossentropy',metrics=['accuracy'])

[20] history = TL_Model.fit(X__,Y,validation_split=0.3,epochs=10)

Epoch 1/10
14/14 [==============================] - 15s 444ms/step - loss: 1.8171 - accuracy: 0.4881 - val_loss: 0.6931 - val_accuracy: 0.5111
Epoch 2/10
14/14 [==============================] - 4s 309ms/step - loss: 0.7099 - accuracy: 0.4857 - val_loss: 0.6939 - val_accuracy: 0.4889
Epoch 3/10
14/14 [==============================] - 4s 310ms/step - loss: 0.6999 - accuracy: 0.4667 - val_loss: 0.6929 - val_accuracy: 0.5111
Epoch 4/10
14/14 [==============================] - 4s 309ms/step - loss: 0.6963 - accuracy: 0.5095 - val_loss: 0.6934 - val_accuracy: 0.5111
Epoch 5/10
14/14 [==============================] - 5s 330ms/step - loss: 0.6940 - accuracy: 0.4905 - val_loss: 0.6938 - val_accuracy: 0.4889
Epoch 6/10
14/14 [==============================] - 4s 317ms/step - loss: 0.6933 - accuracy: 0.5000 - val_loss: 0.6932 - val_accuracy: 0.5111
Epoch 7/10
14/14 [==============================] - 4s 316ms/step - loss: 0.6954 - accuracy: 0.5095 - val_loss: 0.6930 - val_accuracy: 0.5111
Epoch 8/10
14/14 [==============================] - 4s 314ms/step - loss: 0.6939 - accuracy: 0.4667 - val_loss: 0.6929 - val_accuracy: 0.5111
Epoch 9/10
14/14 [==============================] - 5s 338ms/step - loss: 0.6944 - accuracy: 0.5095 - val_loss: 0.6930 - val_accuracy: 0.5111
Epoch 10/10
14/14 [==============================] - 4s 311ms/step - loss: 0.6993 - accuracy: 0.4905 - val_loss: 0.6936 - val_accuracy: 0.4889
```

Figure 3.4.1

Additional Section

- All of the implementations for this HW are in the google colab file named ***HW4_810198377.ipynb***.
- All questions and their sections and parts are separated with the related headings and titles.
- Pictures for Question 3 are in a folder named ***Pic_Q3***.