CSE 4/574: Introduction to Machine Learning
Spring 2023

Assignment 2
Building Neural Networks and Convolutional Neural Networks

Madhu Babu Sikha, Zeming Zhang
msikha@buffalo.edu, zemingzh@bufalo.edu

Abstract

In this report, we discuss the modeling of fully connected neural networks (NN) and convolutional neural networks (CNN). The initial segment involves conducting data analysis and developing a fundamental NN. The subsequent part entails understanding how to optimize and enhance the NN by employing various techniques. In the third section, we will construct a basic CNN, followed by applying optimization and data augmentation techniques in the fourth segment. We discuss the implementation of AlexNet CNN architecture for doing image classification task. We also discuss optimization of the AlexNet architecture to improve the accuracy by using Batch Normalization, Learning Rate Scheduler and Early stopping techniques. The same architecture is then used to identify Street View House Numbers (SVHN) in the next part.

1 Part-I: Building a basic NN

In this section, we discuss the details of the dataset, the basic architecture of the NN and its training in detail.

1.1 Details about Dataset

A short description about the dataset is given below:

- The given dataset has 7 features namely f1, f2, f3, f4, f5, f6, and, f7 and one target variable 'target'.
- The dataset has 766 rows.
- The target variable has values 0 and 1, hence this is a binary classification problem.
- All the seven features are numerical but except feature 'f3', every row has one character randomly in a row, because of this the columns are being shown as categorical. By looking in to the data, we have removed those characters and converted the features into float type. Now the statistics of those features are calculated and given below:

	f1	f2	f3	f4	f5	f6	f7
count	760.000	760.000	760.000	760.000	760.000	760.000	760.000
mean	3.834	120.970	69.120	20.508	80.234	31.999	0.473
std	3.365	32.023	19.446	15.958	115.581	7.900	0.332
min	0.000	0.000	0.000	0.000	0.000	0.000	0.078
25%	1.000	99.000	63.500	0.000	0.000	27.300	0.244
50%	3.000	117.000	72.000	23.000	36.000	32.000	0.376
75%	6.000	141.000	80.000	32.000	128.250	36.600	0.628
max	17.000	199.000	122.000	99.000	846.000	67.100	2.420

Table 1: Summary of main statistics of the dataset

1.2 Visualization Graphs

Figure 1 displays a set of scatter plots that show the pairwise relationships between each combination of features and the target variable. The scatter plots are color-coded based on the target variable's value (0 or 1). By examining the pairplot, we can identify the strength of the correlation between each pair of variables, allowing us to determine which variables have a strong relationship with each other.

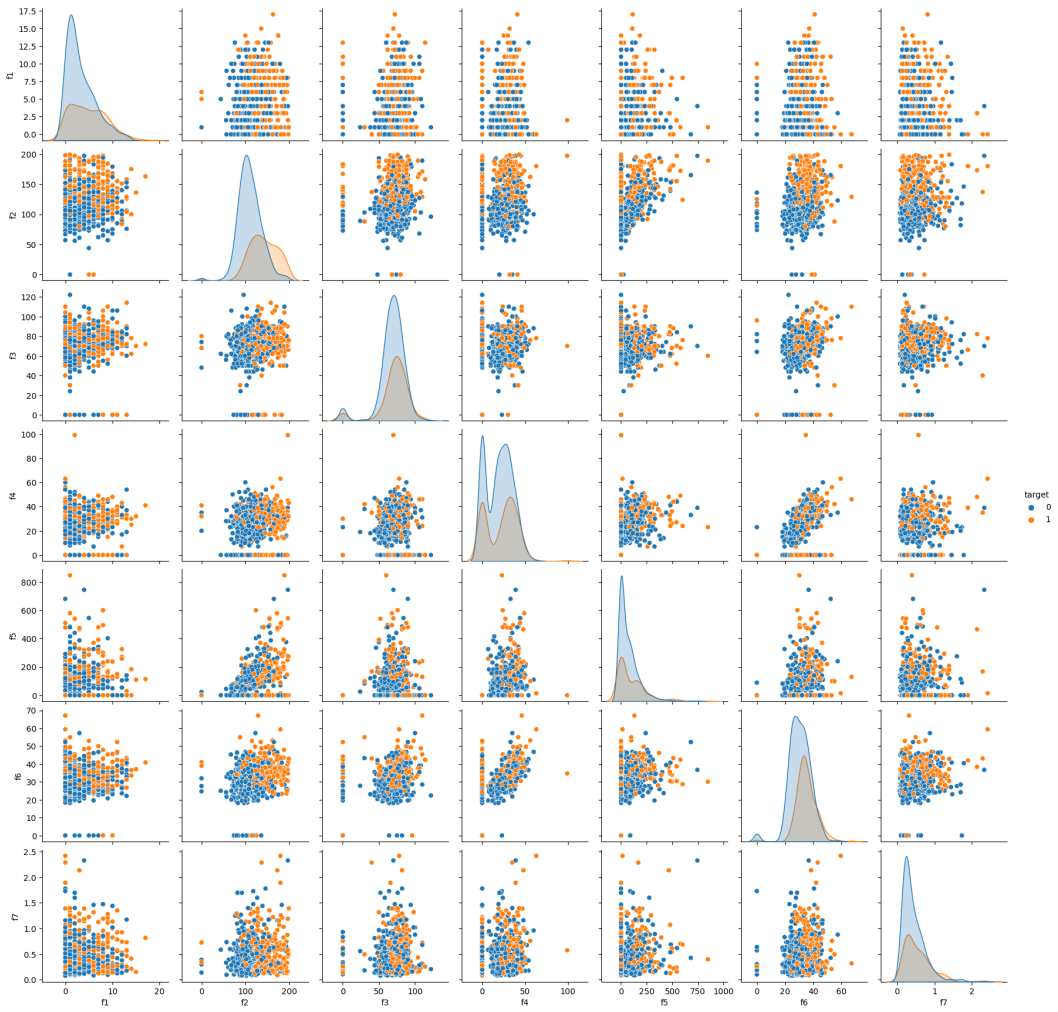


Figure 1: Pairplots of all features and target

We plot each variable against every other variable in the dataset. However, since the target variable is binary, we will only plot the pairwise relationships between the numerical features (f1 to f7) and the target variable.

The resulting pairplot will show a matrix of scatter plots with each row and column representing a different feature. The diagonal of the matrix will show the distribution of each feature, while the off-diagonal cells will show the pairwise scatter plots between each pair of features.

For example, the scatter plot in the cell at row 1, column 2 will show the relationship between features f1 and f2. Similarly, the scatter plot in the cell at row 2, column 1 will show the relationship between f2 and f1. The scatter plots will be color-coded based on the value of the target variable (0 or 1).

By examining the pairplot, we can identify any trends or correlations between the variables. For instance, we might observe that higher values of f2 tend to be associated with a higher likelihood of target = 1.

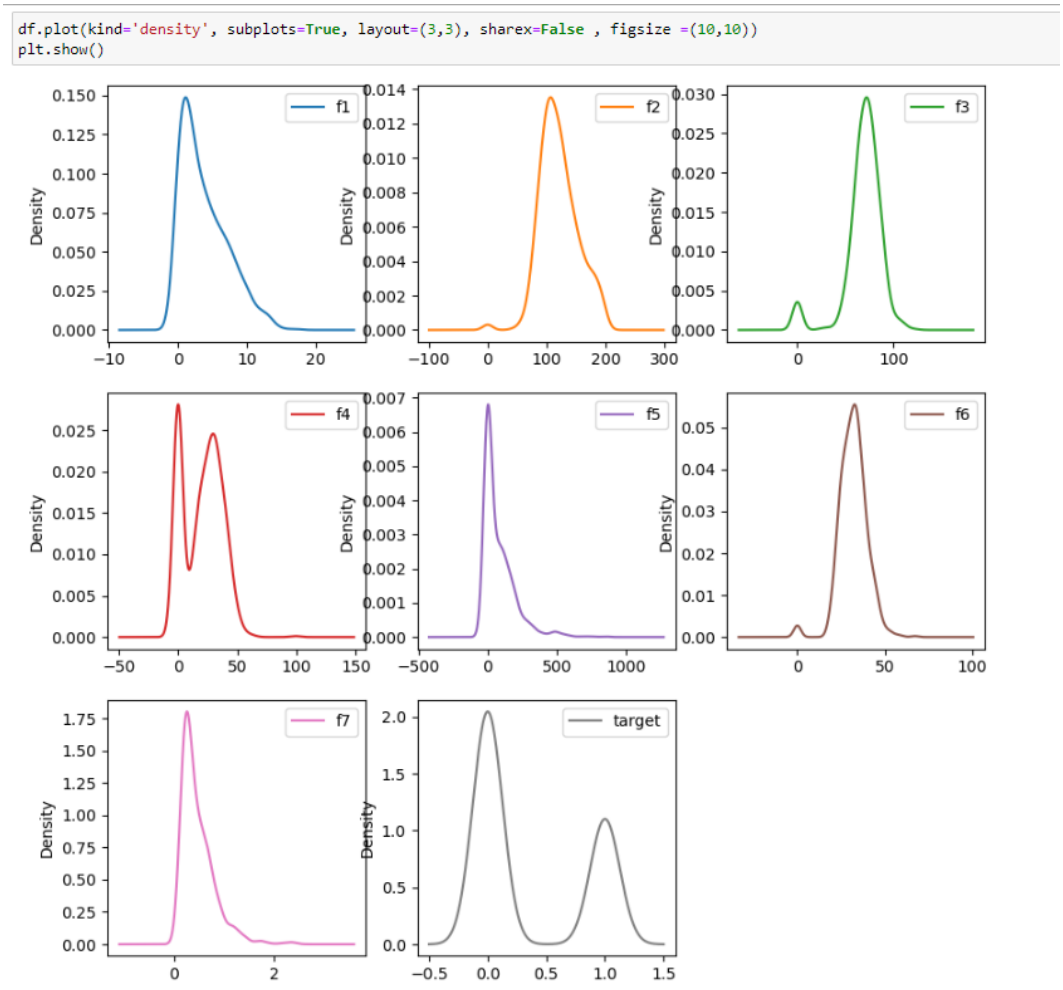


Figure 2: Density plots of Features

The density plots shown in Figure 2 are smoothed versions of a histogram that uses a kernel density estimate to plot the probability density function of the variable. The resulting plot shows the shape of the distribution of the variable, with higher peaks indicating regions where the variable is more commonly observed.

By examining the density plots of the features in the dataset, we can gain insight into the distribution of each variable. For instance, we might observe that the distribution of f2 is approximately normal, while the distribution of f5 is skewed to the right. We can observe that most of the variables are not following normal distribution.

Figure 3 shows the correlation matrix, wherein we can see that the highest positive correlation with the target variable (denoted as "target" in the table) is found with f2, with a correlation coefficient of 0.46. This suggests that as f2 increases, the probability of target being 1 also tends to increase.

```
fig, ax = plt.subplots(figsize=(10, 8), dpi=100)
sns.heatmap(df.corr(), cmap="BuPu", annot=True, fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```

Figure 3: Heatmap showing Correlations

Figure 4 shows the countplot of target variable, which has two bars one for each value of the "target" variable, showing the count of observations target being 0 and 1. We can observe that there is an imbalance in the dataset as the number of 1s and 0s are not equal.

```
plt.figure(figsize=(8,6), dpi=80)
sns.countplot(x="target", data=df)
plt.title("Countplot of Target Variable")
plt.show()
```

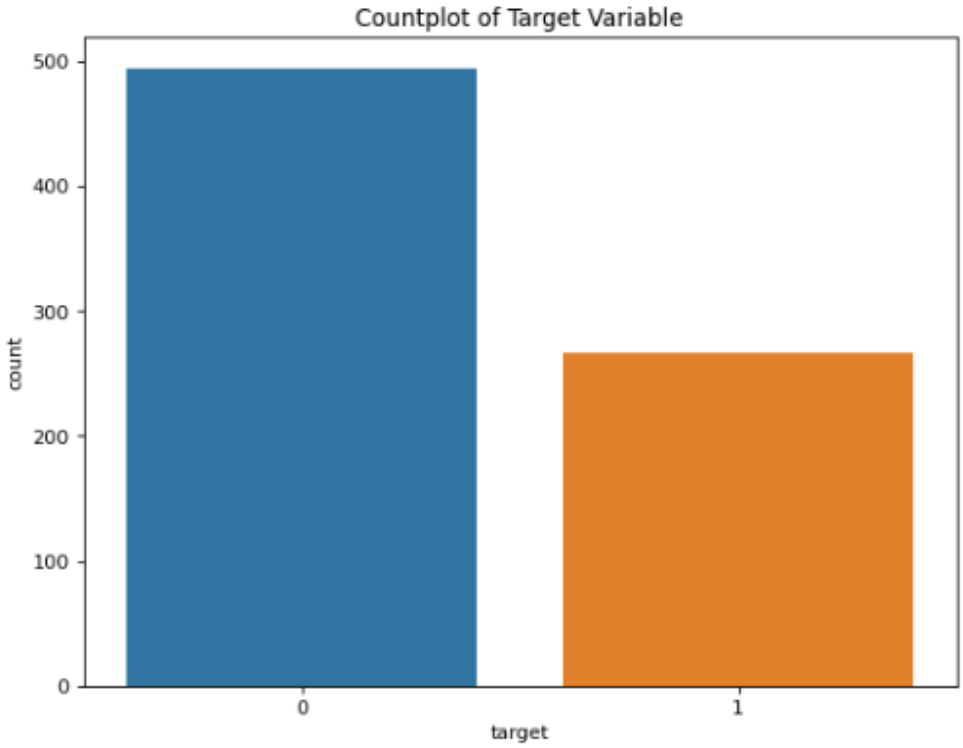


Figure 4: Countplot of Target

1.3 Preprocessing Steps

For reading the dataset, preprocessing (removing symbols/characters from features), scaling and train-test split, we have used numpy, pandas and scikit-learn libraries in Python. The following preprocessing steps are performed on the given dataset.

1. **Removal of garbage characters:** As discussed earlier, except feature 'f3', the other features have a character (like 'f1' has character 'c', 'f2' has 'f', etc.) in one of the rows. All these characters were removed and converted the features into numerical, otherwise the NN cannot be trained on this data.

2. **Scaling:** We also did normalize the data as the dynamic range of each feature is different and hence it would become for the NN to get trained on this un-scaled data.

3. **Train-Validation-Test Split:** After removing the random characters (total 6 in number) from the dataset, we are left with 760 rows. Now, the dataset is divided into training, validation and testing sets in 64%, 16% and 20%, proportion, hence the number of samples in training, validation and testing sets are 486, 122 and 152 respectively.

Definitely, these preprocessing steps helped the NN for better training and ultimately improving the accuracy the NN.

1.4 Architecture of NN

We have used Pytorch framework to build the NN, the basic architecture of the NN that we considered is described as follows. Note here that after trying out various combinations of the number of hidden layers and number of neurons in each layer, we have decided on this particular architecture and is described below:

- The NN architecture consists of three fully connected layers with Tanh activation functions and a single output neuron with a Sigmoid activation function as shown in Figure 5.
- The input layer has 7 neurons, which take in a tensor of 7 input features.
- The first hidden layer is a fully connected (dense) layer with 64 neurons, which applies a Tanh activation function.
- The second hidden layer is another fully connected layer with 32 neurons, which also applies a Tanh activation function.
- The output layer has a single neuron with a Sigmoid activation function, which outputs a single value in the range of 0 to 1, representing the probability of a binary classification.
- Dropout regularization is not applied in this network, as the dropout_rate parameter is set to 0.0.
- The weights of the linear layers are initialized using the xavier_normal_ initializer.

This architecture has a total of 101 trainable parameters (64 weights in the first hidden layer, 2048 weights in the second hidden layer, and 33 weights in the output layer). This NN is used for binary classification task on given dataset with 7 input features.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1, 64]             512
              Tanh-2                [-1, 1, 64]               0
           Dropout-3                [-1, 1, 64]               0
            Linear-4                [-1, 1, 32]           2,080
              Tanh-5                [-1, 1, 32]               0
           Dropout-6                [-1, 1, 32]               0
            Linear-7                 [-1, 1, 1]              33
           Sigmoid-8                 [-1, 1, 1]               0
================================================================
Total params: 2,625
Trainable params: 2,625
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.01
Estimated Total Size (MB): 0.01
----------------------------------------------------------------
```

Figure 5: Summary of the NN architecture

1.5 Performance of the basic NN

In this section, we discuss the performance of the basic NN that is build in terms of the accuracy and loss of the training and validation sets.

1.5.1 Simulation Setup:

To perform the simulation some hyperparameters of the NN is set through out the simulation, which are given in the Table 2. The other hypermaraters (optimization function, activation function, weights-initializer and dropout rate) are varied in different setups.

Table 2: Training Parameters

Parameter	**Value**
Learning Rate	0.0001
Total Epochs	150
Batch Size	64

We ran the NN for different combinations of the hyperparameters and fixed the hyperparameter values as shown in Table 3.

The hyperparameters for the basic NN model for part-I is given Table 3:

Table 3: Hyperparameters for the Basic NN Model in part-I

Dropout	Optimizer	Activation Function	Initializer	Learning Rate	Accuracy
0	AdamW	Tanh()	xavier_normal_	0.0001	0.75

The loss and accuracy values of training and validation sets are monitored for every epoch and plotted as shown in Fiugre 6. From the figure, we can observe that both the training and validation loss are decreasing as the number of epochs increases, which shows that there is no overfitting of the NN. Also, the training and validation accuracies are in increasing trend as the number of epochs increases.

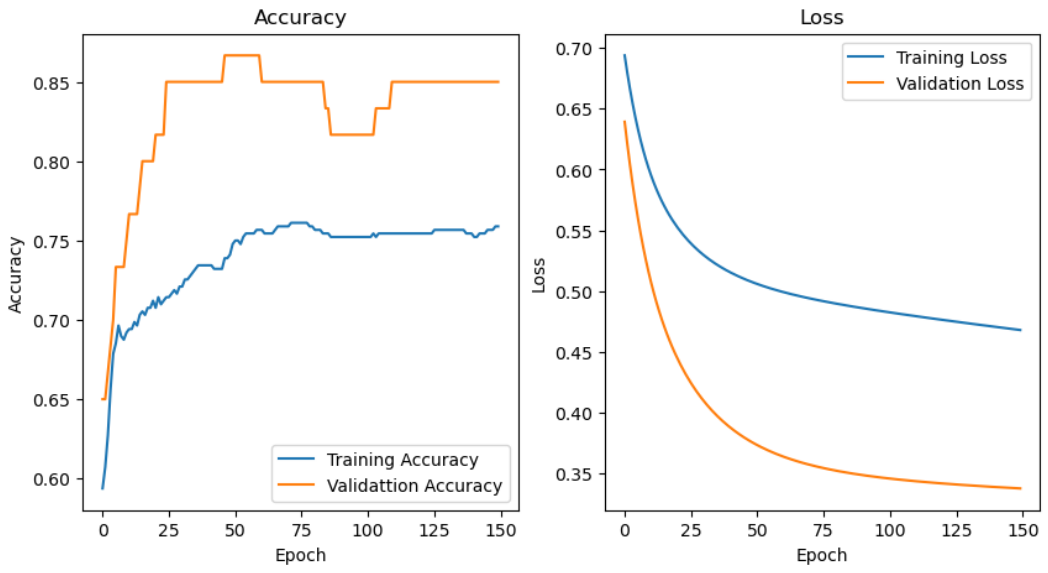


Figure 6: Accuracy and Loss of Training and Validation sets for the Basic NN model

The performance of the basic NN model is also good, we have achieved 0.75 accuracy on the test set.

2 Part-II: Optimizing NN

We describe the results of hyperparameter tuning for a neural network model, where we varied one hyperparameter at a time while keeping the others constant.

2.1 Activation Function Tuning

Figures 8, 9 and 10 are obtained when we varying the activation functions by keeping all the other parameters same as basic model NN of part-I.

The test accuracy results for activation function tuning are tabulated in Figure 7. Analysis of different hyperparameter tuning setups is discussed in detail in section 2.5.

	Setup 1	Test Accuracy	Setup 2	Test Accuracy	Setup 3	Test Accuracy
Dropout	0	0.7566	0	0.7632	0	0.7632
Optimizer	AdamW		AdamW		AdamW	
Activation Function	ReLU()		LeakyReLU(negative_slope=0.01)		ELU(alpha=1.0)	
Initializer	xavier_normal_		xavier_normal_		xavier_normal_	
Learning rate	0.0001		0.0001		0.0001	

Figure 7: Test accuracy for different Setups when Tuning the Activation Function

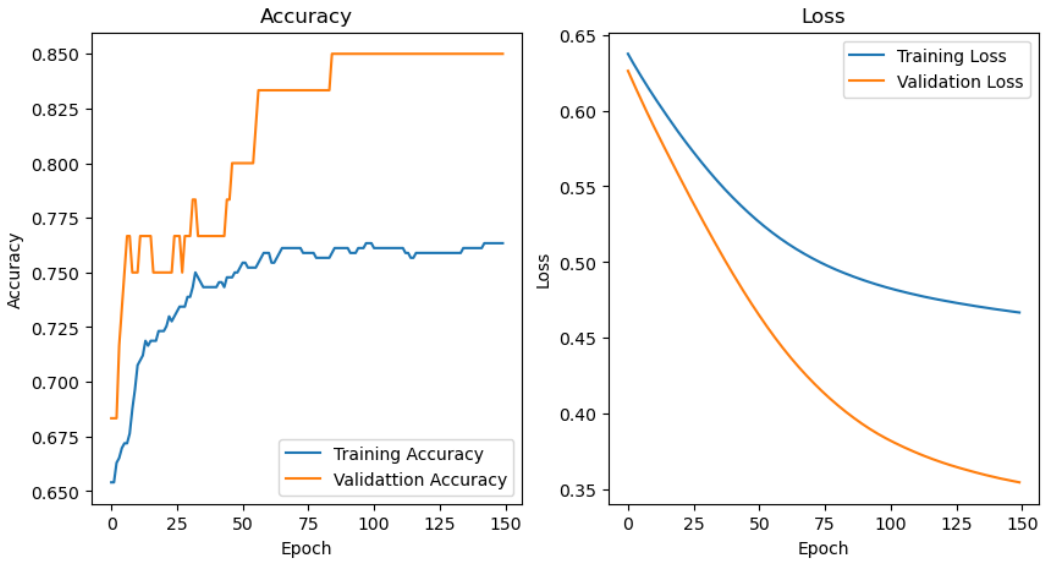


Figure 8: Activation Function Tuning- Setup 1

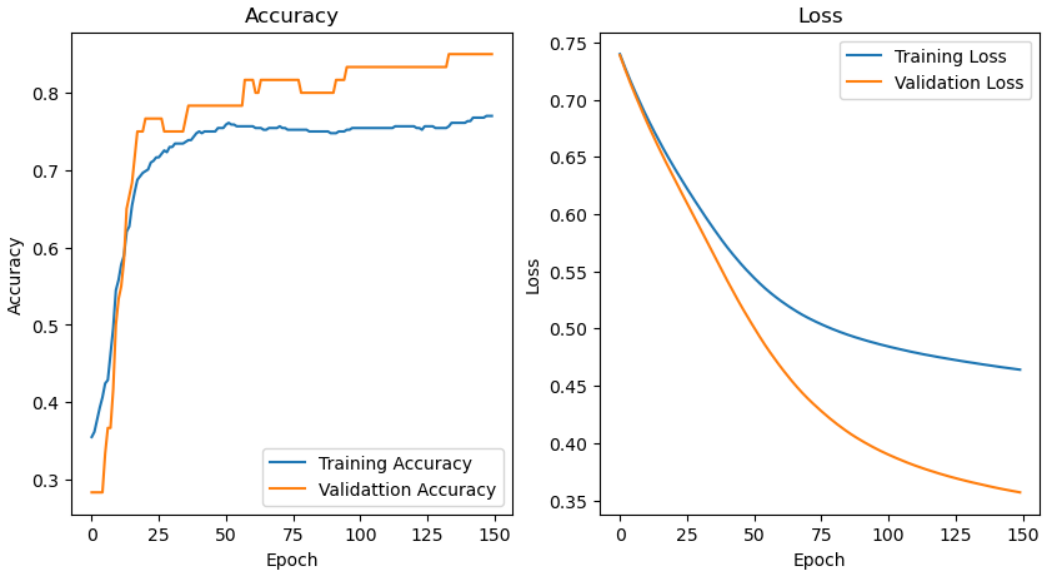


Figure 9: Activation Function Tuning- Setup 2

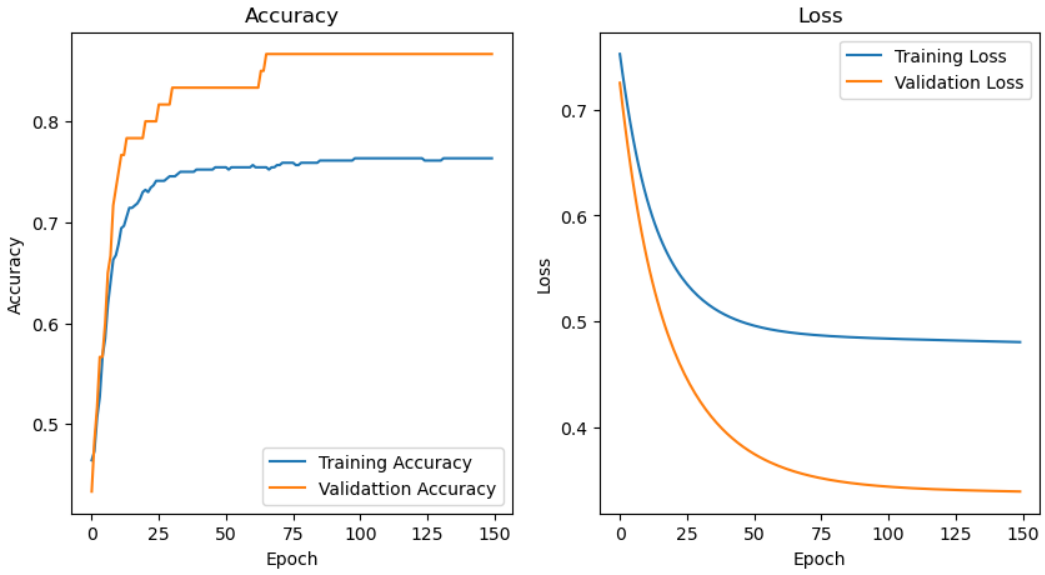


Figure 10: Activation Function Tuning- Setup 3

2.2 Optimization Function Tuning

Figures 12, 13 and 14 are obtained when we varying the optimization function by keeping all the other parameters same as basic model NN of part-I.

The test accuracy results for optimization function tuning are tabulated in Table 11.

	Setup 1	Test Accuracy	Setup 2	Test Accuracy	Setup 3	Test Accuracy
Dropout	0	0.6974	0	0.7763	0	0.7697
Optimizer	Adagrad		Adamax		RMSprop	
Activation Function	Tanh()		Tanh()		Tanh()	
Initializer	xavier_normal_		xavier_normal_		xavier_normal_	
Learning rate	0.0001		0.0001		0.0001	

Figure 11: Test accuracy for different Setups when Tuning the Optimization Function

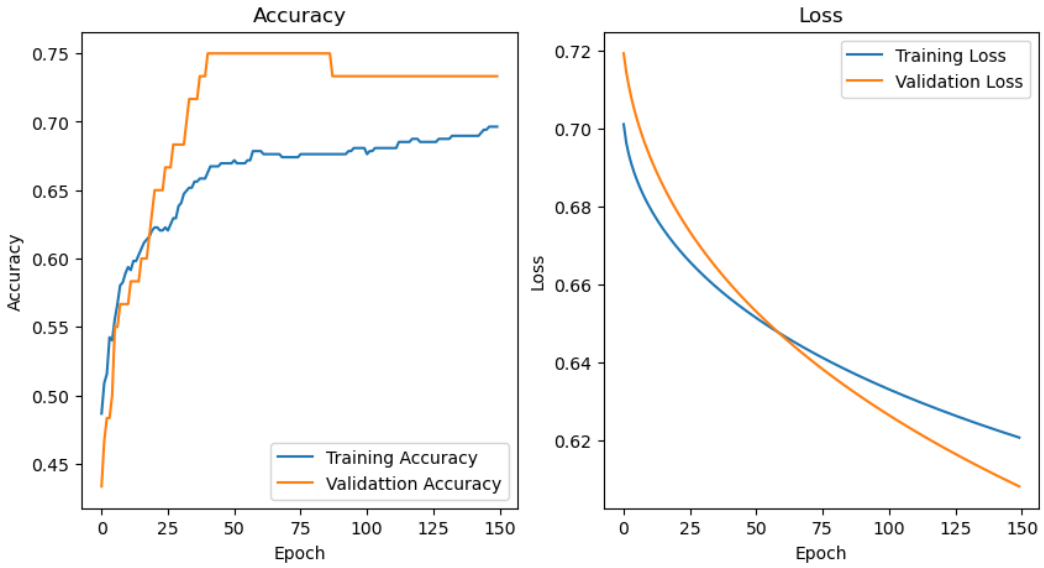


Figure 12: Optimization Function Tuning- Setup 1

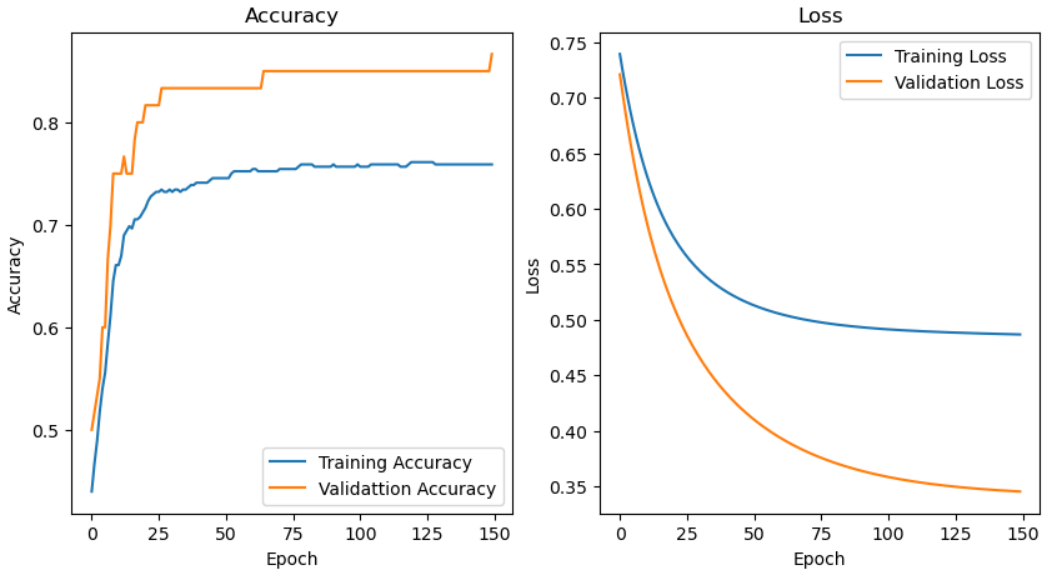


Figure 13: Optimization Function Tuning- Setup 2

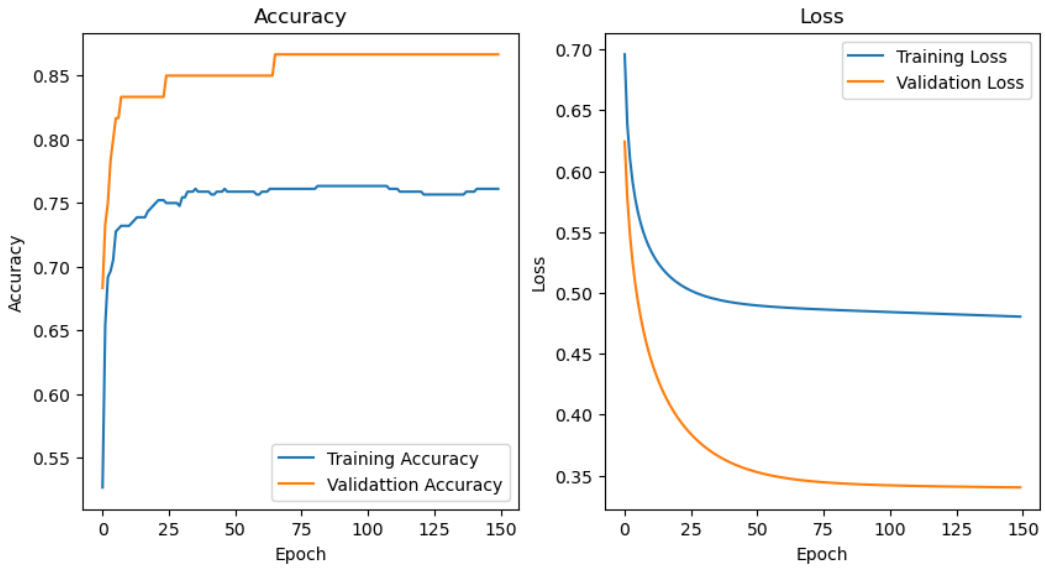


Figure 14: Optimization Function Tuning- Setup 3

2.3 Dropout Rate Tuning

Figures 16, 17 and 18 are obtained when we varying the dropout rates by keeping all the other parameters same as basic model NN of part-I.

The test accuracy results are tabulated in Table 15.

	Setup 1	Test Accuracy	Setup 2	Test Accuracy	Setup 3	Test Accuracy
Dropout	0.05	0.7763	0.15	0.75	0.25	0.7632
Optimizer	AdamW		AdamW		AdamW	
Activation Function	Tanh()		Tanh()		Tanh()	
Initializer	xavier_normal_		xavier_normal_		xavier_normal_	
Learning rate	0.0001		0.0001		0.0001	

Figure 15: Test accuracy for different Setups when Tuning the Dropout rate

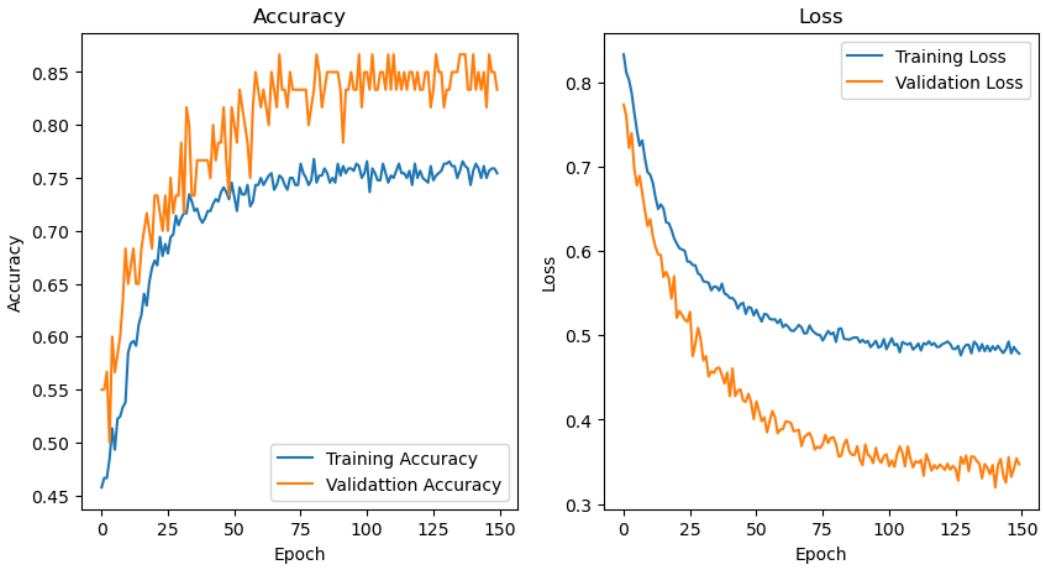


Figure 16: Dropout Rate Tuning- Setup 1

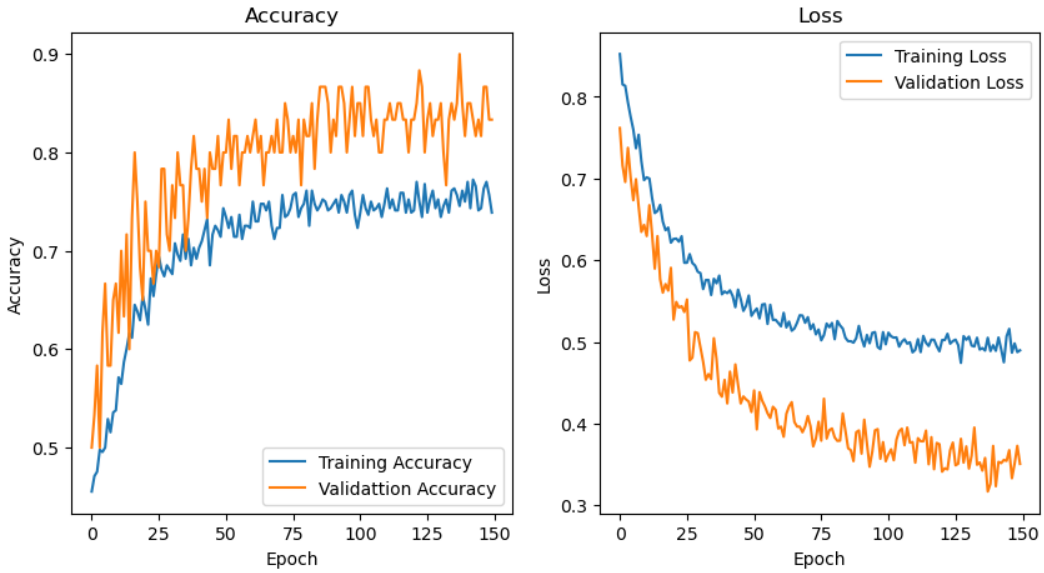


Figure 17: Dropout Rate Tuning- Setup 2

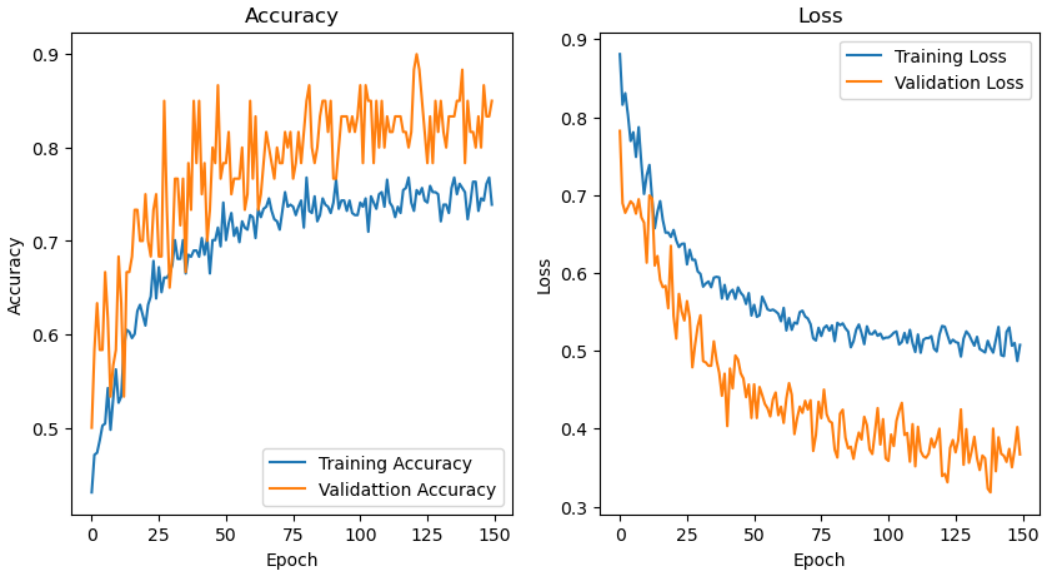


Figure 18: Dropout Rate Tuning- Setup 3

2.4 Initializer Tuning

Figures 20, 21 and 22 are obtained when we varying the weights initializers by keeping all the other parameters same as basic model NN of part-I.

The test accuracy results for initializer tuning are tabulated in Table 19.

	Setup 1	Test Accuracy	Setup 2	Test Accuracy	Setup 3	Test Accuracy
Dropout	0	0.7566	0	0.75	0	0.7632
Optimizer	AdamW		AdamW		AdamW	
Activation Function	Tanh()		Tanh()		Tanh()	
Initializer	kaiming_normal_		kaiming_uniform_		xavier_uniform_	
Learning rate	0.0001		0.0001		0.0001	

Figure 19: Test accuracy for different Setups when Tuning the Initializer

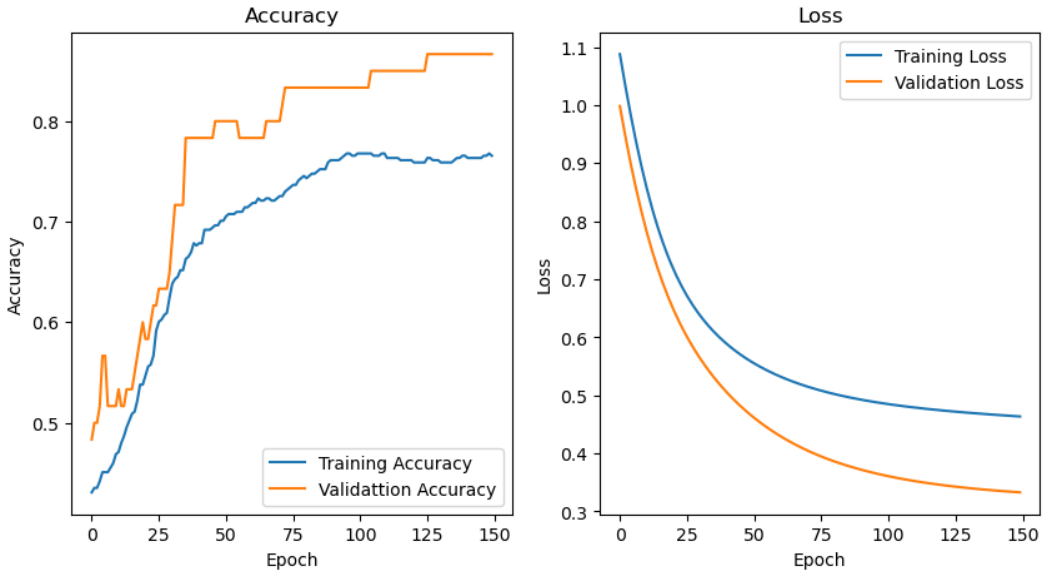


Figure 20: Initializer Tuning- Setup 1

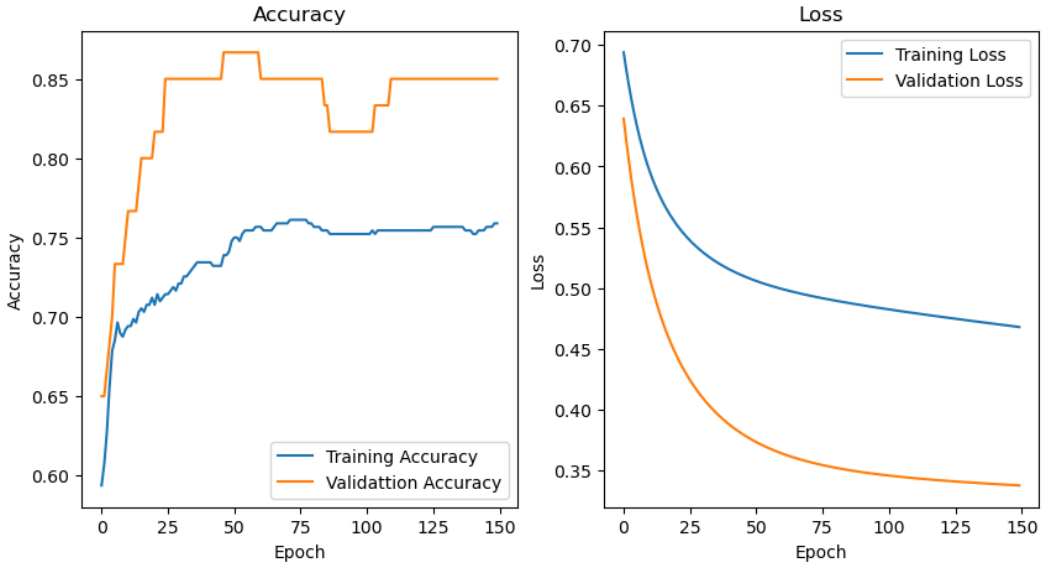


Figure 21: Initializer Tuning- Setup 2

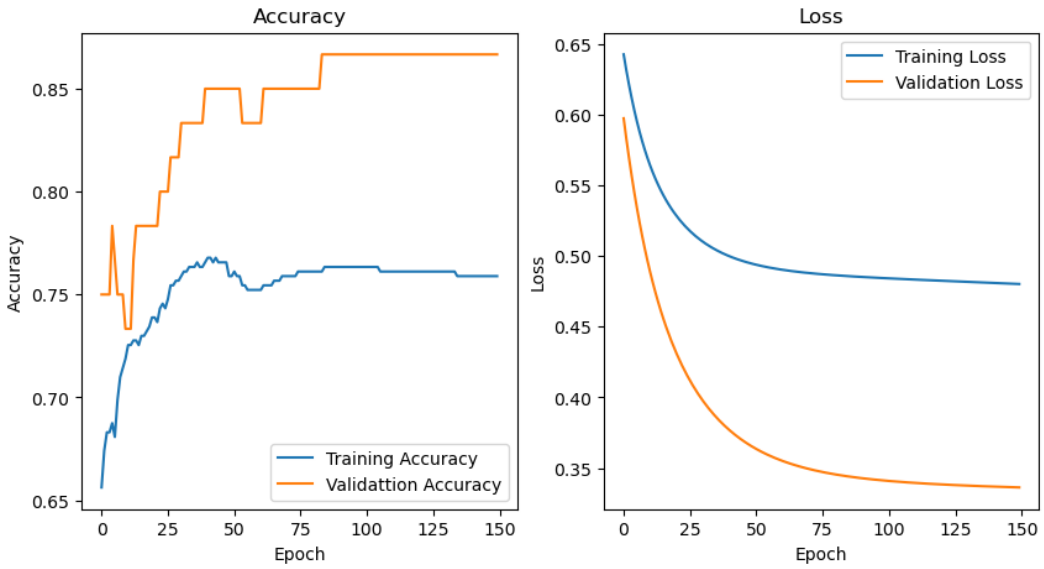


Figure 22: Initializer Tuning- Setup 3

2.5 Analysis of Different NN Setups

The base model was created with a dropout rate of 0, an AdamW optimizer, a Tanh activation function, a xavier_normal_ initializer, and a learning rate of 0.0001, achieving an accuracy of 75%. We performed a grid search, testing a range of values for each hyperparameter while keeping the others constant, and recorded the test accuracy for each combination.

Table 7 shows the results of tuning the activation function, where ReLU, LeakyReLU, and ELU were tested, achieving test accuracies of 75.66%, 76.32%, and 76.32%, respectively. The best accuracy was achieved using LeakyReLU with a negative slope of 0.01. Table 11 shows the results of tuning the optimizer, where Adagrad, Adamax, and RMSprop were tested, achieving test accuracies of 69.74%, 77.63%, and 76.93%, respectively. The best accuracy was achieved using Adamax. Table 15 shows the results of tuning the dropout rate, where rates of 0.05, 0.15, and 0.25 were tested, achieving test accuracies of 77.63%, 75%, and 76.32%, respectively. The best accuracy was achieved using a dropout rate of 0.05. Table 19 shows the results of tuning the initializer, where kaiming_norma_, kaiming_uniform_, and xavier_uniform_ were tested, achieving test accuracies of 75.66%, 75%, and 76.32%, respectively. The best accuracy was achieved using xavier_uniform_.

Overall, the results suggest that using a Tanh activation function and an AdamW optimizer provide good performance. However, tuning other hyperparameters can further improve accuracy. It's important to note that the optimal hyperparameters can vary depending on the specific dataset and task, and further experimentation and tuning may be required to find the best hyperparameters for a particular problem.

Hyperparameter tuning is essential in machine learning to improve the performance of the model. By varying hyperparameters and testing their impact on the accuracy of the model, it is possible to find the optimal combination of hyperparameters. The grid search approach used in this study allowed us to evaluate the impact of individual hyperparameters on the model's performance. The results suggest that using a Tanh activation function and an AdamW optimizer can provide good performance, and tuning other hyperparameters can further improve accuracy. However, through testing various hyperparameters, the optimal combination of hyperparameters for the neural network model was determined, which included using the ELU activation function, AdamW optimizer, kaiming_uniform_ initializer, 0.05 dropout rate, and a learning rate of 0.0001 which is 81.57%, howevevr this would consider to change all the hyperparamaters. But this combination of hyperparameters is not chosen because the combination of these hyperparameters with other values is not giving the

good performance. However, we have included this set here just to mention that we can fine tune the hyperparameters to get optimum accuracy.

2.6 Optimization Methods

In order to further improve the performance of the NN, we have implemented four optimization methods. The performance of the base model (which has given best performance out of the 4 tables is chosen here) is given in Figure 23, because this would be used for comparison with performance of the optimization methods. In Figure 23, we have represented test loss of the base model along with the training loss and validation losses. Similarly, the test accuracy is plotted along with training and validation accuracies. But we get a single value for test loss and accuracy because we test the performance of the NN model on the entire test set once. So, we plotted the test loss and accuracy as a constant line along with the training, validation losses and accuracies. This just gives us a comparison of how the training, validation loss and accuracies are evolving as the training of the NN model progresses.

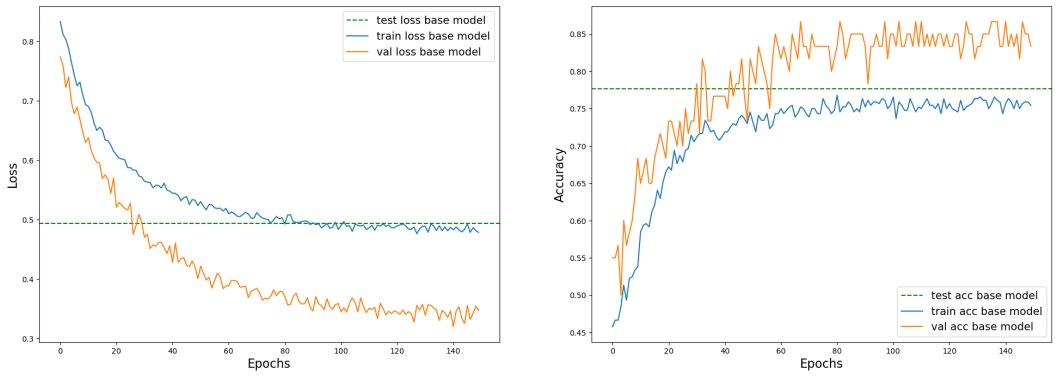


Figure 23: Performance of Base Model- Loss and Accuracy of Training and Validation sets

2.6.1 Optimization Method 1: Early Stopping

We create an instance of a neural network model, training it on the training data, evaluating its performance on the validation data, and testing it on the test data. It also plots the training curves for the loss and accuracy of the model. The model uses binary cross entropy as the loss function, AdamW as the optimizer function, and a learning rate of 0.0001 with early stopping. The training time is 1.76 seconds, with a test loss of 0.520 and a test accuracy of 0.7829. Using early stopping is beneficial as it helps prevent overfitting and reduces training time, while achieving comparable or better results than training the model until convergence as shown in Figure 24. This approach may be useful when training large models or working with limited computational resources.

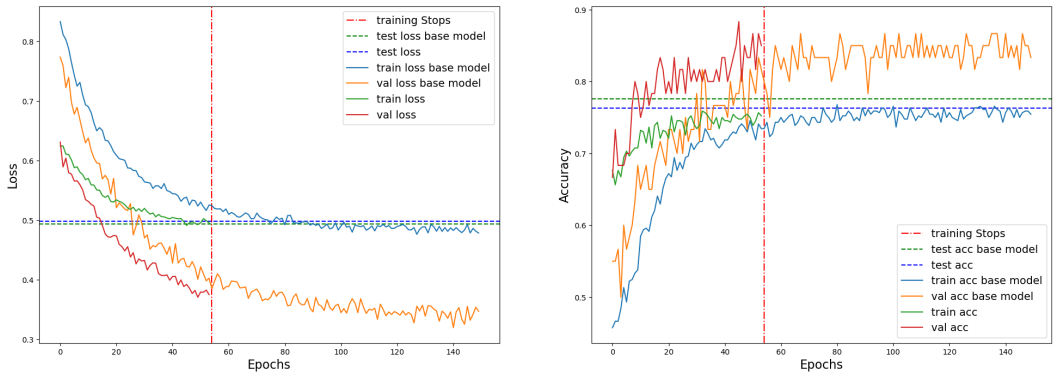


Figure 24: Loss and Accuracies for Early-Stopping

2.6.2 Optimization Method 2: Learning Rate Scheduler

The learning rate scheduler is a technique used to automatically adjust the learning rate of the optimizer during training. It can help improve the performance of the model by gradually reducing the learning rate as the training progresses. This allows the model to converge more smoothly and avoid getting stuck in local minima. A lower learning rate towards the end of training also helps fine-tune the model and prevent overfitting. In the code above, the learning rate scheduler is used in combination with the AdamW optimizer to train the neural network model. Overall, using a learning rate scheduler can help improve the performance and stability of a machine learning model by adjusting the learning rate automatically during training. However, the choice of learning rate schedule and parameters can have a significant impact on the performance of the model. Figure 25 shows the implementation of learning rate schedule for our base model.

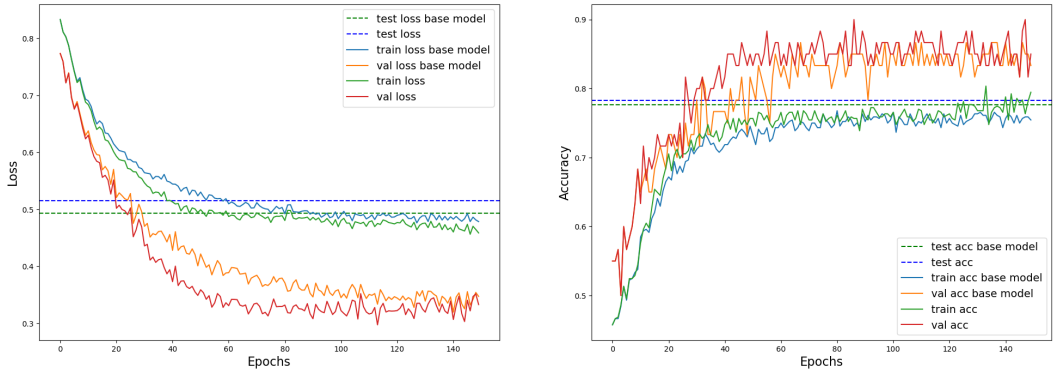


Figure 25: Loss and Accuracies for Learning Rate Scheduler

2.6.3 Optimization Method 3: K-Fold Cross Validation

K-fold cross-validation, a technique used to evaluate machine learning models by splitting the data into k equal-sized subsets and training the model on k-1 folds while validating it on the remaining fold. This process is repeated k times, and the results are averaged to reduce the risk of overfitting or underfitting. The neural network model in the code uses binary cross-entropy loss and the AdamW optimizer. The code computes the average training and validation loss and accuracy and test loss and accuracy over all the folds. The model achieves an average test accuracy of 0.7393 and an average test loss of 0.5573. The average training time is 1.6459 seconds. K-fold cross-validation is useful for evaluating and optimizing machine learning models and can provide more reliable estimates of performance compared to a single train-test split. Figure 26 shows implementation of k-fold cross validation for our base model.

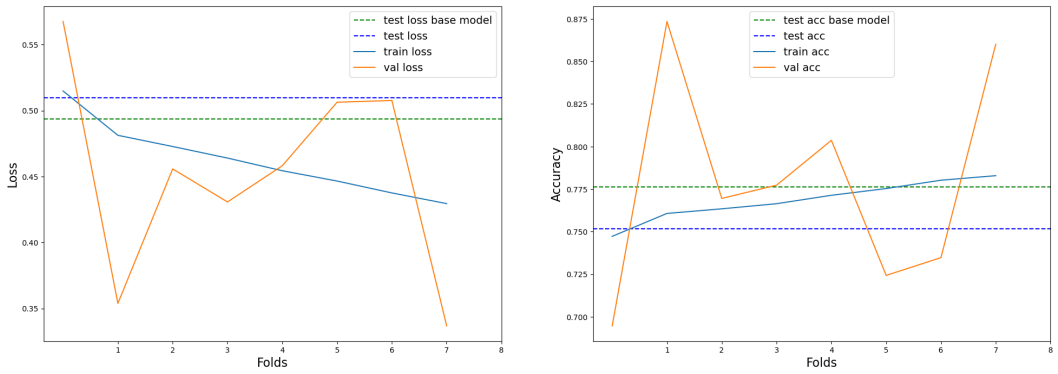


Figure 26: Loss and Accuracies for k-Fold Cross Validation

2.6.4 Optimization Method 4: Batch Normalization

Batch Normalization is a technique used to improve the training of neural networks by normalizing the inputs of each layer. This technique can help prevent the vanishing gradient problem and improve the model's generalization performance.

The Batch Normalization layers are inserted between the linear layers and the activation functions to normalize the outputs of each layer of base NN model. Overall, this network architecture with Batch Normalization can help improve the training and generalization performance of the model, as shown in Figure 27.

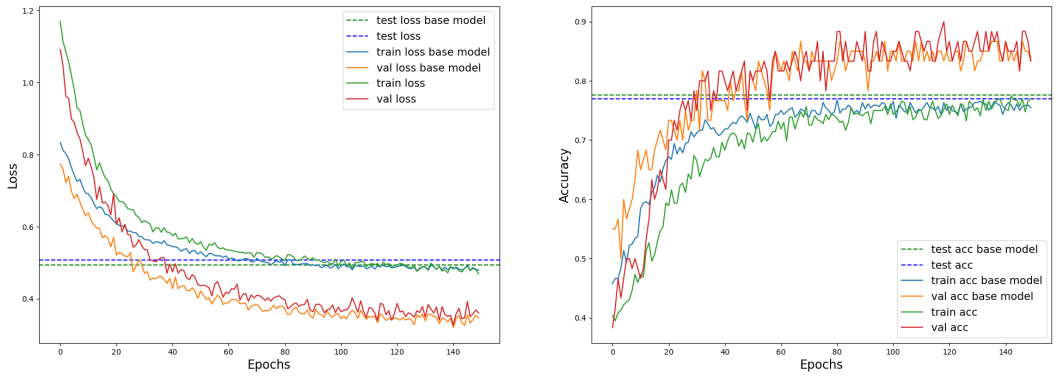


Figure 27: Loss and Accuracies for Batch Normalization

2.6.5 Analysis of Training Times of Different Optimization Methods

The comparison four optimization methods w.r.t. the training times is given in Table 4. It shows the training times for different machine learning models and techniques, with the Base Model (which doesn't incorporate any additional techniques) taking the least amount of time and Batch Normalization taking the most. The table shows that the training times vary depending on the model and technique used. For example, Early-Stopping and K-fold Cross validation are faster than the Base Model, while Batch Normalization takes the longest time to train. However, it's important to note that training time is not the only factor to consider when choosing a machine learning model or technique. Performance, accuracy, and other factors are also important considerations.

Model	Training Time (in Secs)
Base Model	1.81
Early-Stopping	0.65
Learning Rate Scheduler	1.9
K-fold Cross validation	1.71
Batch Normalization	2.51

Table 4: Training Times for Different Models

3 Part-III: Implementing & Improving AlexNet

In this section, we discuss details about the dataset, AlexNet architecture and its implementation for image classification and finally hyper-parameter tuning for improving the accuracy.

3.1 Details about the Dataset

The CNN dataset is a collection of 30,000 color images, where each image is of size 64x64 pixels. The images are divided into three categories: dogs, food, and vehicles. Each category contains 10,000 images, which makes the dataset balanced in terms of the number of examples per class.

These images are at different scales, colors, angles, etc., it is a very good data to train any CNN model, because the images are real-world images and hence the model can work for any test image of these categories.

Table 5: Details of CNN Dataset

Category	Number of Images	Remarks
Dogs	10,000	Different breeds of dogs, in various poses, scales, sizes, orientations, etc.
Food	10,000	Varieties of Food like burger, pizza, etc.
Vehicles	10,000	Cars of different colors, sizes, angles, etc.

3.2 Train, Validation and Test sets

For model training, validation after each epoch and final testing of our model, we have splitted the original CNN dataset into three parts namely Train, Validaiton and Test sets in 70%, 20% and 10% ratio respectively.

Table 6: CNN Dataset split for model training, validation and testing

Dataset	Split Ratio	Total Images
Train	70%	21000
Validation	20%	6000
Test	10%	3000
Total Images		30000

3.3 Sample Images Visualization

Two images from each class of CNN dataset are taken and calculated their histogram, scatter plot and heatmapt and is plotted in Figure 28.

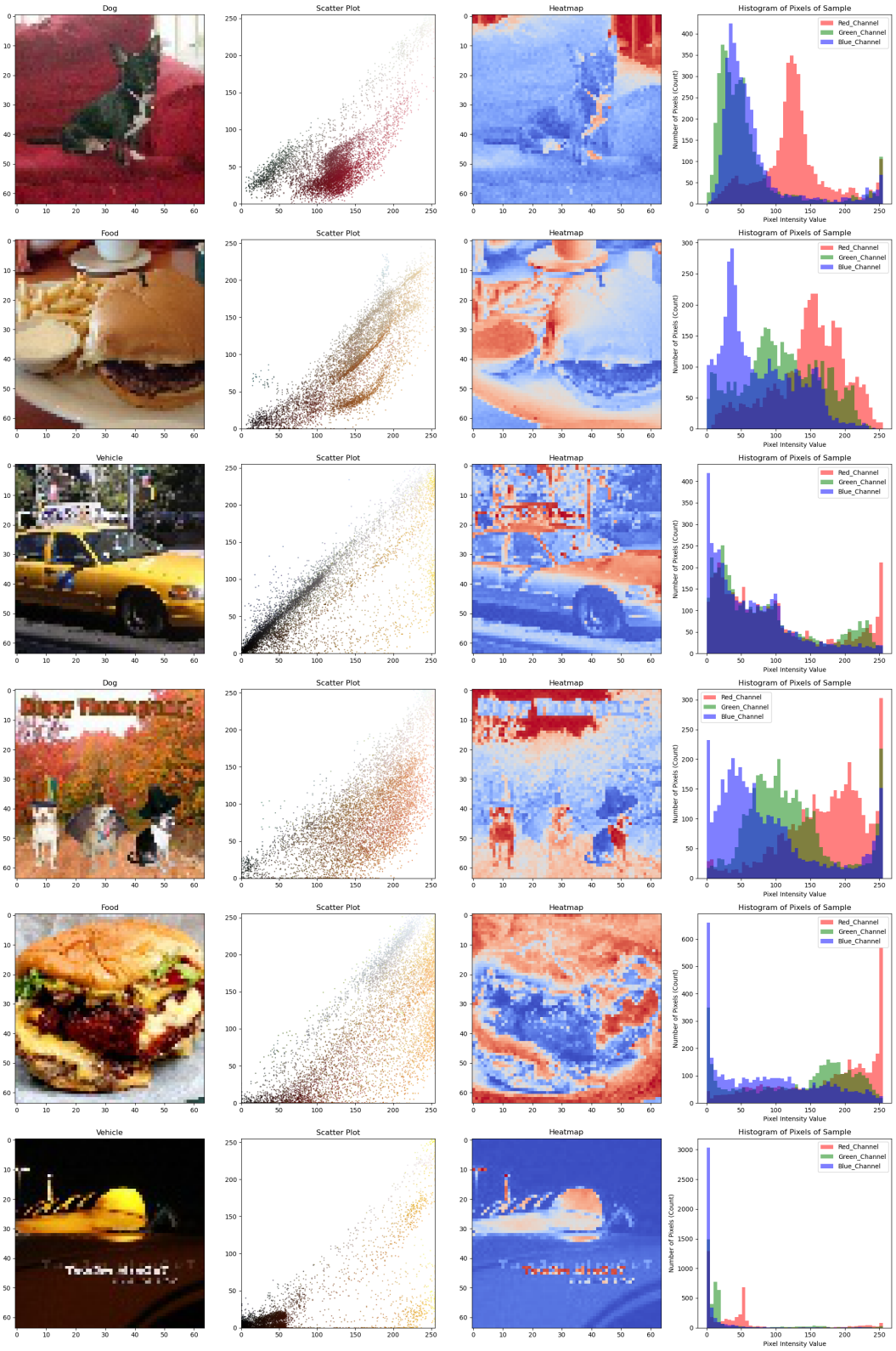


Figure 28: Sample Images of CNN dataset and their Histogram, Scatter plot and Heatmap

A scatter plot shows the relationship between the intensity values of two different color channels. By plotting the intensities of two color channels against each other, we can see if there are any patterns or relationships between them. For example, if the scatter plot shows a linear relationship between the red and green channels, this could indicate that the image contains a lot of yellow or orange colors. For food images of burger, pizza, etc., the yellow color spikes up and hence we can see that there are yellow color points in the scatter plot. If we see histogram of cars, different channels will have pixels in the same range, because usually cars color is uniform. A heatmap shows the spatial distribution of pixel intensities in an image. It can help you identify areas of high or low contrast and see how different parts of the image relate to each other. By comparing the heatmaps of different images, we can see how they differ in terms of contrast and spatial patterns.

In general, the following scatter plot trend can be observed from the CNN dataset, unless the focus is more on things other than those objects. The scatter plot for the dogs dataset shows a fairly uniform distribution of points across the plot. There doesn't appear to be any discernible pattern or clustering in the data. The scatter plot for the food dataset shows a slight clustering of points towards the center of the plot. This could be indicative of a common color or texture among the images in the dataset. The scatter plot for the cars dataset shows two distinct clusters of points. This suggests that there may be two subgroups of images in the dataset that have different features, such as color or shape.

The CNN dataset typically exhibits the following trend in the heatmap, unless the emphasis is on something other than those specific objects. The heatmap shows that the head region of the dog is the most important region for classification. This makes sense, as dogs are often identified by their facial features. The heatmap shows that the central region of the food image is the most important for classification, likely because it contains the main ingredients and visually represents the dish. The heatmap shows that the front of the car, including the headlights and grille, is the most important region for classification. This is likely because the front of a car is one of the most distinctive and recognizable parts of the vehicle.

Unless the focus is mainly on objects other than those included in the cnn dataset, the histogram trend can generally be observed. For the dogs images, we can observe a significant peak in the range of pixel intensities between 100-150, which indicates that the images have a lot of medium to high intensity pixels. This could be due to the fact that dogs have fur that tends to be darker in color, which increases the pixel intensities in those areas. In contrast, the food images have a more uniform distribution of pixel intensities, with no clear peaks or troughs. This could be because food is often photographed in evenly lit environments to bring out the colors and textures of the food, resulting in a more evenly distributed pixel intensity range. The cars images also show a clear peak in pixel intensities around the 100-150 range, similar to the dogs images. This could be due to the reflective surfaces of cars, which tend to have bright spots that increase the pixel intensities in those areas.

3.4 Building AlexNet Model

Alex et.al. [7] proposed a CNN architecture namely AlexNet for Imagenet classificaiton challenge. We have adopted the same architecture here for performing image classification, with two major changes in the output layer of the original architecture. It is discussed below: (i) the output layer of original AlexNet has 1000 neurons because it was originally designed for Imagenet challenge where they have to classify images of 1000 categories. But, in our dataset, we have only three types of images and hence we have changed the number of neurons in the output layer to 3.

Image upscaling to match the input layer size of AlexNet architecture: The input layer of original AlexNet expects images of size 224x224, while the images in our cnn dataset are of size 64x64. So, we have upscaled the image size from 64x64 to 224x224, to make them compatible for model training.

These are the two major changes we made to the original AlexNet architecture for performing our image classification task. The summary of AlexNet architecture that we have implemented is shown in Figure 29, which lists out the name of the layer, shape of the out feature map and number of parameters of each layer.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 96, 54, 54]          34,944
              ReLU-2          [-1, 96, 54, 54]               0
         MaxPool2d-3          [-1, 96, 26, 26]               0
 LocalResponseNorm-4          [-1, 96, 26, 26]               0
            Conv2d-5         [-1, 256, 26, 26]         614,656
              ReLU-6         [-1, 256, 26, 26]               0
         MaxPool2d-7         [-1, 256, 12, 12]               0
 LocalResponseNorm-8         [-1, 256, 12, 12]               0
            Conv2d-9         [-1, 384, 12, 12]         885,120
             ReLU-10         [-1, 384, 12, 12]               0
           Conv2d-11         [-1, 384, 12, 12]       1,327,488
             ReLU-12         [-1, 384, 12, 12]               0
           Conv2d-13         [-1, 256, 12, 12]         884,992
             ReLU-14         [-1, 256, 12, 12]               0
        MaxPool2d-15           [-1, 256, 5, 5]               0
AdaptiveAvgPool2d-16           [-1, 256, 6, 6]               0
          Dropout-17                [-1, 9216]               0
           Linear-18                [-1, 4096]      37,752,832
             ReLU-19                [-1, 4096]               0
          Dropout-20                [-1, 4096]               0
           Linear-21                [-1, 4096]      16,781,312
             ReLU-22                [-1, 4096]               0
           Linear-23                   [-1, 3]          12,291
================================================================
Total params: 58,293,635
Trainable params: 58,293,635
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 11.06
Params size (MB): 222.37
Estimated Total Size (MB): 234.01
----------------------------------------------------------------
```

Figure 29: AlexNet Model Architecture

3.5 Results of AlexNet Model

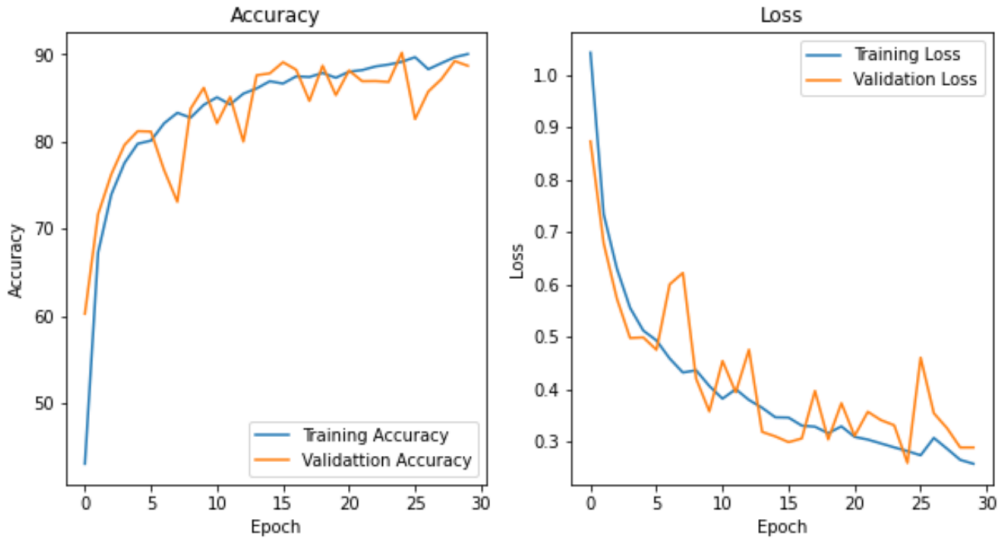
As mentioned earlier, 70% of the data is used for training the AlexNet model. After each epoch, we validate the performance of the model on the Validation set (which is 20% of cnn dataset). After training the model for predefined number of epochs, we finally test the performance of the mode on the Test set (10% of the original cnn dataset).

The parameters set for AlexNet model are given in Table 7.

Table 7: Hyperparameters for AlexNet Model (Step3 of Part-III)

Parameter	Value
Number of Epochs	30
Batch size	128
optimizer	SGD
Learning rate	0.01
Momentum	0.9
Activation function	ReLU

For each epoch, we recorded the average training loss and accuracy, also, the validation set loss and accuracy after every epoch is also recorded. After training phase, we plotted the training, validation loss against number of epochs as Shown in Figure 30. From this plot, we can observe that as the number of epochs increases the training loss and validation loss decreases (though there are some fluctuations in the validation loss, the overall trend is downwards only). Similarly, we have also plotted the training accuracy and validation accuracy against the number of epochs as shown in Figure 30. As the number of epochs increases, the training and validation accuracy also increases.




```
Test Loss: 0.25 -Test Accuracy: 90.73%
```

Figure 30: Loss and Accuracy vs. Epochs for Train and Validation Sets for AlexNet Model

Finally, after the training phase, we evaluated the trained (original) AlexNet model on the test set and got a train loss of 0.25, while the Train Accuracy of 90.73%.

3.6 Improved AlexNet Model

As discussed in previous subsection, the original AlexNet has given a Test accuracy of 90.73%. To further improve its performance, we have implemented three techniques namely: (i) Batch Normalization, (ii) Learning Rate Scheduler and (iii) Early Stopping.

All these three techniques are discussed in detain in Section 2.6 Optimization Methods.

Batch normalization is implemented to the original AlexNet model discussed in section 3.4. After each convolution layer in the original AlexNet architecture, a batch normalization layer is added. The batch normalization layer operates on the output of a layer, normalizing it by subtracting the mean and dividing by the standard deviation of the batch. This helps to stabilize the network by reducing internal covariate shift, which can improve training speed and performance. In other words, the batch

normalization layer can be added to any neural network layer with any number of neurons. This has helped in improving the accuracy of our model.

Also, we have implemented the learning rate scheduler to decrease the learning rate of the optimizer. There are many ways of implementing learning rate scheduler, we have reduced the learning rate for each epoch as a function of the current epoch, as given in Table 8. This type of learning rate scheduler is called Exponential decay. This can help to improve the performance of the model by allowing it to make larger updates to the parameters early in training when the gradients are larger, and smaller updates later in training when the gradients are smaller.

For early stopping we set a patience value of 8, i.e., if the validation loss of the current epoch increases than that of the previous epoch 8 number of times, then the training will be stopped. Since we ran the model for 32 epochs and there is no significant over-fitting of the model, early-stopping did not stop the training phase. During our training phase, we have observed the nature of the dataset and the model and did not see the validation loss in the current epoch increasing than the previous epoch. So, early stopping didn't help us in stopping the training early and hence there is no improvement in the accuracy because of early-stopping.

The parameters of Modified AlexNet are given in Table 8.

Table 8: Hyperparameters for Modified AlexNet Model (Step4 of Part-III)

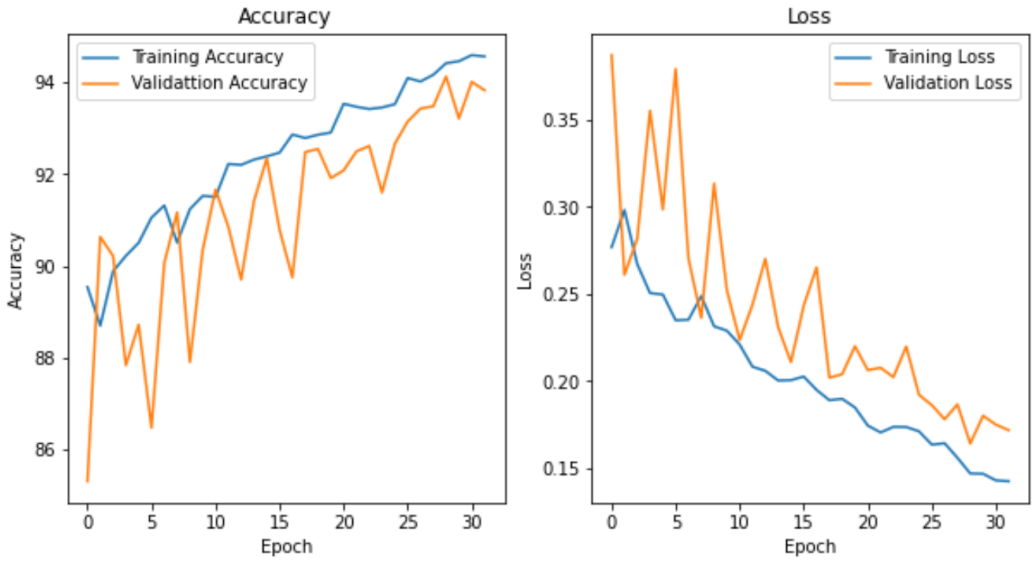
Parameter	**Value**
Number of Epochs	32
Batch size	128
optimizer	SGD
Learning rate	0.01
Momentum	0.9
Activation function	ReLU
BatchNormalization	appropriate to the previous Conv Layer
Early stopping	patience of 8
Learning Rate Scheduler	initial_lr * (1-(current_epoch/total_epochs))

3.7 Results of Train, Validation and Test for Improved AlexNet Model

During each epoch, we monitored and recorded the average training loss and accuracy, as well as the validation set loss and accuracy. Subsequently, we plotted the training and validation loss against the number of epochs, as depicted in Figure 32. The plot reveals a decline in both training and validation loss as the number of epochs increases. Although the validation loss exhibits some fluctuations, the overall trend is downward. In addition, we also plotted the training accuracy and validation accuracy against the number of epochs, as presented in Figure 32. As the number of epochs increases, the training and validation accuracy also increase.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 96, 54, 54]          34,944
       BatchNorm2d-2           [-1, 96, 54, 54]             192
              ReLU-3           [-1, 96, 54, 54]               0
         MaxPool2d-4           [-1, 96, 26, 26]               0
 LocalResponseNorm-5           [-1, 96, 26, 26]               0
            Conv2d-6          [-1, 256, 26, 26]         614,656
       BatchNorm2d-7          [-1, 256, 26, 26]             512
              ReLU-8          [-1, 256, 26, 26]               0
         MaxPool2d-9          [-1, 256, 12, 12]               0
LocalResponseNorm-10          [-1, 256, 12, 12]               0
           Conv2d-11          [-1, 384, 12, 12]         885,120
      BatchNorm2d-12          [-1, 384, 12, 12]             768
             ReLU-13          [-1, 384, 12, 12]               0
           Conv2d-14          [-1, 384, 12, 12]       1,327,488
      BatchNorm2d-15          [-1, 384, 12, 12]             768
             ReLU-16          [-1, 384, 12, 12]               0
           Conv2d-17          [-1, 256, 12, 12]         884,992
      BatchNorm2d-18          [-1, 256, 12, 12]             512
             ReLU-19          [-1, 256, 12, 12]               0
        MaxPool2d-20            [-1, 256, 5, 5]               0
AdaptiveAvgPool2d-21            [-1, 256, 6, 6]               0
          Dropout-22                 [-1, 9216]               0
           Linear-23                 [-1, 4096]      37,752,832
             ReLU-24                 [-1, 4096]               0
          Dropout-25                 [-1, 4096]               0
           Linear-26                 [-1, 4096]      16,781,312
             ReLU-27                 [-1, 4096]               0
           Linear-28                    [-1, 3]          12,291
================================================================
Total params: 58,296,387
Trainable params: 58,296,387
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 15.64
Params size (MB): 222.38
Estimated Total Size (MB): 238.60
----------------------------------------------------------------
```

Figure 31: Improved AlexNet Model Architecture




```
Test Loss: 0.17 -Test Accuracy: 93.63%
```

Figure 32: Loss and Accuracy vs. Epochs for Train and Validation Sets for Modified AlexNet Model

Finally, after the training phase, we evaluated the trained modified and improved AlexNet model on the test set and got a train loss of 0.17, while the Train Accuracy of 93.63%, which is better than the original AlexNet model.

4 Part-IV: Optimizing CNN + Data Argumentation of Google SVHN Dataset

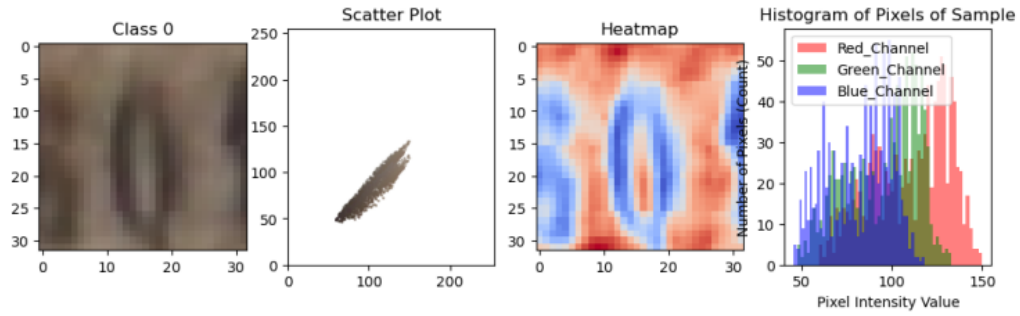
4.1 Details about the Dataset

In this part, we do image classification of Street View House Numbers (SVHN) Dataset [8] using the optimized AlexNet architecture from part-III with few modifications (discussed in next section).

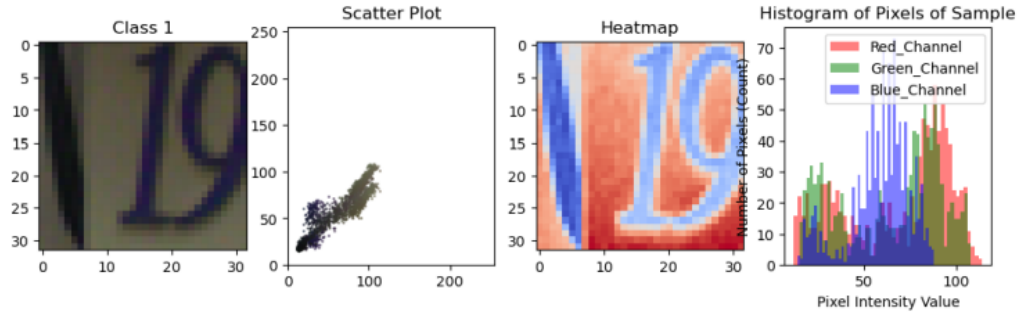
The Google Street View House Numbers (SVHN) dataset is a large dataset of digit images that was collected from house numbers visible in Google Street View images. The dataset was released in 2011 by Google and has since become a popular benchmark dataset for image classification tasks, particularly for digit recognition. We have downloaded the data from torchvision.datasets import SVHN, where the training set contains 73,257 images, the validation set (which is loaded from the 'extra' split) contains 26,032 images, and the test set (also loaded from the 'test' split) contains 26,032 images. The images are labeled with the corresponding digit, and there are 10 classes, one for each digit from 0 to 9. One of the challenges of the SVHN dataset is that the digits in the images may be overlapping or occluded, making it more difficult to accurately classify them. Additionally, the dataset contains some images that are blurry or distorted, which can further complicate the task of digit recognition.

The analysis of a sample image from each class in the google SVHN dataset is given in the Figures 33, 34 and 35.

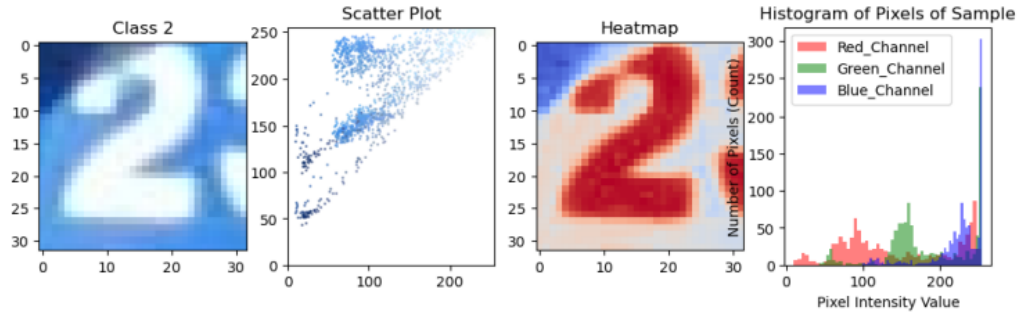
All the images are of 32x32 and hence the number of pixels is less. The scatter plot of these images



<Figure size 640x480 with 0 Axes>



<Figure size 640x480 with 0 Axes>



<Figure size 640x480 with 0 Axes>

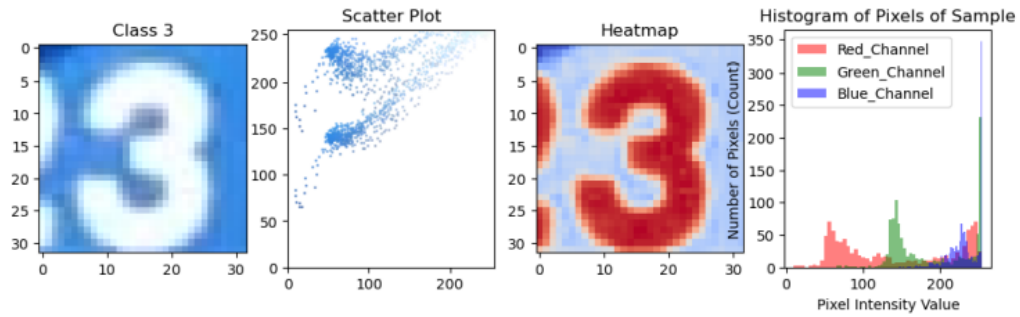
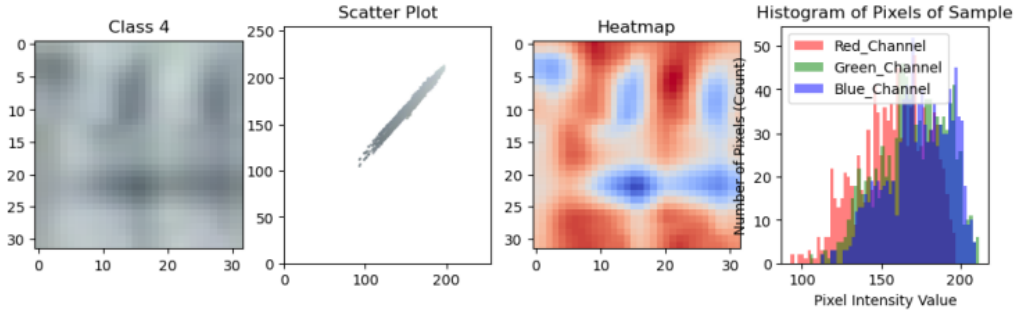
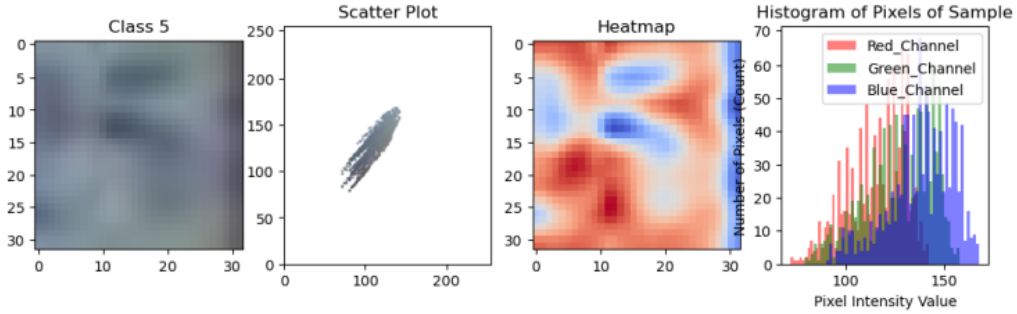


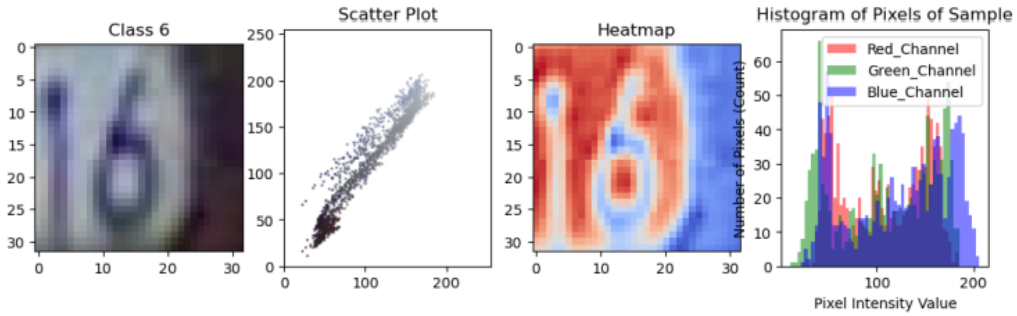
Figure 33: Google SVHN dataset digits 0, 1, 2, 3



<Figure size 640x480 with 0 Axes>



<Figure size 640x480 with 0 Axes>



<Figure size 640x480 with 0 Axes>

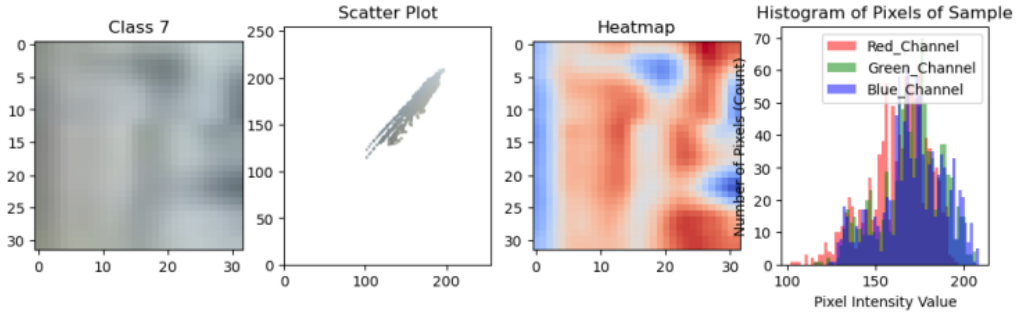


Figure 34: Google SVHN dataset digits 4,5,6,7

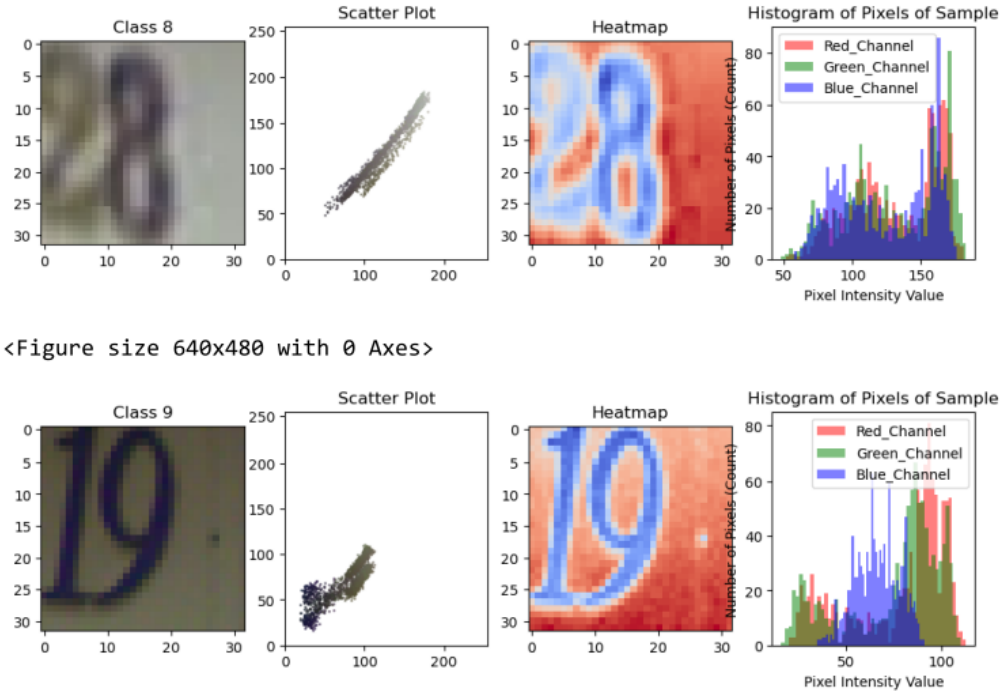


Figure 35: Google SVHN dataset digits 8, 9

We compare the histograms of the SVHN images, we might notice that the histograms of images containing house numbers with multiple digits have a wider range of pixel intensities than those containing only single digits. This could be due to the fact that images with multiple digits contain more complex backgrounds, colors, and textures.

Similarly, we notice that the histograms of images containing blurry or low-quality house numbers have less distinct peaks, indicating that the pixel intensities are more evenly distributed across the image. This could be due to the fact that blurry or low-quality images have less contrast between the foreground (house number) and background, making it harder to distinguish the individual pixels.

Analyzing histograms can provide valuable insights into the properties of images, and can be used to compare and contrast different types of images.

In the case of SVHN dataset, which contains images of house numbers, the heatmaps can help in identifying the regions of the image that contain the number and its location within the image.

For example, if we compare the heatmaps of different images of the same house number, we can observe that the regions with higher intensity values correspond to the areas where the number is present, while the background has lower intensity values. Additionally, we can also observe variations in the heatmaps for the same house number, which can be attributed to variations in the lighting conditions, image quality, and other factors.

Overall, comparing heatmaps can help in understanding the variations and patterns within a dataset, which can be useful for developing better models and improving the accuracy of image classification tasks.

The scatter plots of the SVHN dataset show the distribution of the RGB color values for each image. By looking at the plots, we can observe that the distribution of color values varies across different images. Some images have a uniform distribution, while others have more variation in their color values. Additionally, some images have a dominant color that stands out in the scatter plot.

These differences in the scatter plots can provide insights into the characteristics of the images. For example, images with a dominant color could be associated with certain types of objects, such as

traffic signs or buildings. By examining the scatter plots, we can identify these patterns and potentially use them to improve our image recognition models.

4.2 Improved AlexNet Model from Part-III and Modifications

In this part, to solve the Google SVHN challenge, We have adopted the same AlexNet architecture that we have optimized in part-III, but with one major change in the output layer of the original architecture. It is discussed below: (i) the output layer of original AlexNet has 1000 neurons because it was originally designed for Imagenet challenge where they have to classify images of 1000 categories. But, in our Google SVHN dataset, we have only 10 types of images and hence we have changed the number of neurons in the output layer to 10.

Image upscaling to match the input layer size of AlexNet architecture: The input layer of original AlexNet expects images of size 224x224, while the images in Google SVHN are of 32x32 size. So, we have upscaled the image size from 32x32 to 224x224, to make them compatible for model training.

4.3 Data Augmentation Methods

Data augmentation is a technique used in machine learning to increase the amount of training data available for a model by creating modified versions of the original data. The goal of data augmentation is to improve the generalization of the model by exposing it to more diverse examples and reducing overfitting.

In this work, to generate a new images, from the original dataset, we have used the following techniques: rotation, horizontal flip, vertical flip, color jitter. Since we have increased the number of training images, the training time also increased. But the model has got exposed to a variety of images, i.e., the diversity of the images is increased. Also, the size of the dataset increased as well, which helped us in reducing the overfitting and hence increase in the accuracy on the test.

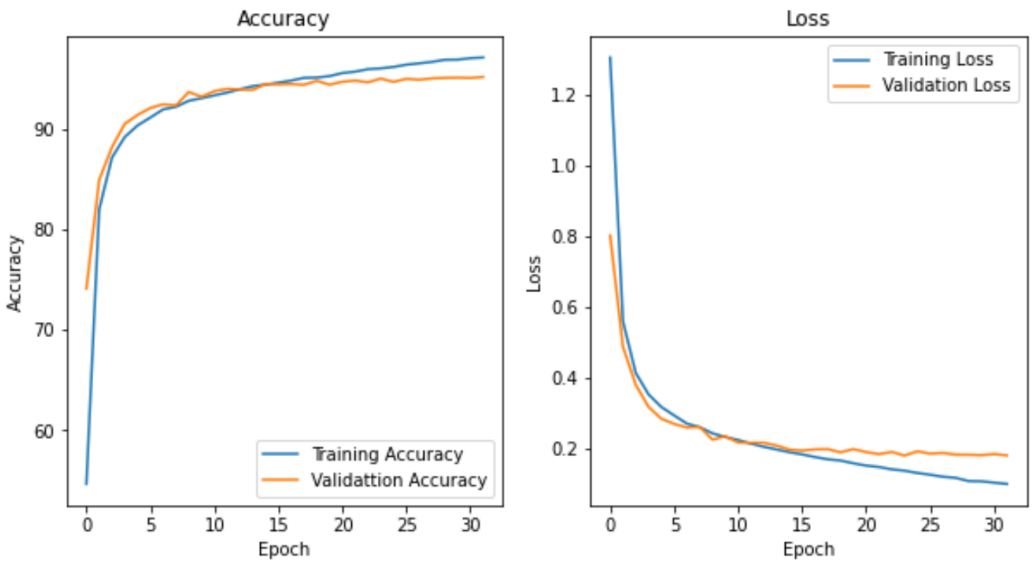
4.4 Results of Train, Validation and Test of SVHN Dataset using Improved AlexNet Model

We train the improved AlexNet architecture with output layer changes and input image upscaling. The parameters of the improved AlexNet model are kept same as part-III improved model which are given in Table 8.

As discussed earlier, we have noted the training, validation loss and accuracy for every epoch and plotted the results in Figure 37. This plot shows that as the number of epochs increases, both training and validation loss decrease. In addition, we can see that the training and validation accuracy also increase as the number of epochs increases. This suggests that the model is learning and improving over time. The results are visualized in Figure 37.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 96, 54, 54]          34,944
       BatchNorm2d-2           [-1, 96, 54, 54]             192
              ReLU-3           [-1, 96, 54, 54]               0
         MaxPool2d-4           [-1, 96, 26, 26]               0
 LocalResponseNorm-5           [-1, 96, 26, 26]               0
            Conv2d-6          [-1, 256, 26, 26]         614,656
       BatchNorm2d-7          [-1, 256, 26, 26]             512
              ReLU-8          [-1, 256, 26, 26]               0
         MaxPool2d-9          [-1, 256, 12, 12]               0
LocalResponseNorm-10          [-1, 256, 12, 12]               0
           Conv2d-11          [-1, 384, 12, 12]         885,120
      BatchNorm2d-12          [-1, 384, 12, 12]             768
             ReLU-13          [-1, 384, 12, 12]               0
           Conv2d-14          [-1, 384, 12, 12]       1,327,488
      BatchNorm2d-15          [-1, 384, 12, 12]             768
             ReLU-16          [-1, 384, 12, 12]               0
           Conv2d-17          [-1, 256, 12, 12]         884,992
      BatchNorm2d-18          [-1, 256, 12, 12]             512
             ReLU-19          [-1, 256, 12, 12]               0
        MaxPool2d-20            [-1, 256, 5, 5]               0
AdaptiveAvgPool2d-21            [-1, 256, 6, 6]               0
          Dropout-22                 [-1, 9216]               0
           Linear-23                 [-1, 4096]      37,752,832
             ReLU-24                 [-1, 4096]               0
          Dropout-25                 [-1, 4096]               0
           Linear-26                 [-1, 4096]      16,781,312
             ReLU-27                 [-1, 4096]               0
           Linear-28                   [-1, 10]          40,970
================================================================
Total params: 58,325,066
Trainable params: 58,325,066
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 15.64
Params size (MB): 222.49
Estimated Total Size (MB): 238.71
----------------------------------------------------------------
```

Figure 36: Improved AlexNet Model Architecture for Google SVHN dataset




```
Test Loss: 0.18 -Test Accuracy: 95.31%
```

Figure 37: Loss and Accuracy vs. Epochs for Train and Validation Sets for Google SVHN Dataset

Finally, after the training phase, we evaluated the trained modified and improved AlexNet model on the test set of SVHN dataset and got a train loss of 0.18, while the Train Accuracy of 95.31%.

4.5 Different Setups for Part-IV

The optimized AlexNet model from part-III has given us an accuracy of 95.31% as mentioned above. Though we have tried two more variants of AlexNet, we did not see much improvement in the accuracy.

The first variant is a deeper version of the network with two additional convolutional layers, 384 and 256 filters respectively, and reduced filters in the first layer to 64. We also removed local response normalization layers and did not add any additional regularization techniques.

The second variant is an improved version of AlexNet, where we increased the number of filters, added batch normalization after every convolutional layer, and dropout after every fully connected layer. We replaced nn.BatchNorm2d with nn.BatchNorm1d after the fully connected layers, omitted local response normalization layers, and did not add data augmentation techniques.

5 Conclusion

In this assignment, we have explored and implemented fully connected neural networks (NN) and convolutional neural networks (CNN) in four different parts.

In the first part, we performed a comprehensive data analysis and built a basic NN to for a binay classification task. We observed the impact of different activation functions and network architectures on the model's accuracy.

In the second part, we learned about optimization techniques such as learning rate scheduler, early stopping, batch normalization and K-Fold cross validation to improve the accuracy and training time.

In the third part, we implemented AlexNet architecture to classify images in CNN dataset of 3 classes, namely dogs, food and cars. We observed the impact of varying the number of convolutional layers, filters and other hyperparameters on the accuracy of the model. We also applied optimization techniques on the AlexNet to improve its performance further.

In the fourth and final part, we applied optimization techniques such as data augmentation on Google SVHN dataset and classified them using the optimized AlexNet from third part. We observed the effect of different data augmentation techniques on the accuracy of the model.

Overall, this assignment has allowed us to gain hands-on experience in building and optimizing different types of neural networks for image classification tasks. We have also learned about important techniques such as early stopping, learning rate scheduler, data augmentation, and batch normalization, which can help improve the performance of neural networks.

References

[1] Pytorch documentation. https://pytorch.org/docs/stable/index.html. Last accessed on 03-30-2023.

[2] Numpy documentation. https://numpy.org/doc/stable/. Last accessed on 03-31-2023.

[3] Pandas documentation. https://pandas.pydata.org/docs/index.html. Last accessed on 03-31-2023.

[4] Lecture slides of Intro to Machine Learning course by Dr. Alina Vereshchaka, Dept. of CSE, UB.

[5] Machine Learning Mastery. https://machinelearningmastery.com/develop-your-first-neural-network-with-pytorch-step-by-step/. Last accessed on 03-31-2023.

[6] Stack Overflow. https://stackoverflow.com/. Last accessed on 03-31-2023.

[7] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Communications of the ACM 60.6 (2017): 84-90.

[8] The Street View House Numbers (SVHN) Dataset. Last accessed on 04-13-2023.