Assignment 2 - Building Neural Networks and Convolutional Neural Networks

Ajith Kumar Ethirajulu aethiraj@buffalo.edu
Deepak Raj Mohan Raj dmohanra@buffalo.edu

Team Member	Assignment Part	Contribution (%)
Ajith Kumar Ethirajulu	Part I, Part II, Part III, Part IV	50
Deepak Raj Mohan Raj	Part I, Part II, Part III, Part IV	50

Abstract

This assignment focuses on building fully connected neural networks (NN) and convolutional neural networks (CNN). It consists of four parts where we practice dealing with various datasets and implement, train, and adjust neural networks. The first part consists of performing data analysis and building a basic NN. In the second part, we learn how to optimize and improve our NN using various techniques. In the third part, we will implement a basic CNN and apply optimization and data augmentation techniques in the fourth part.

1 Part I: Building a Basic NN

In this assignment, we have implemented a neural network using the Keras library. We will train the network on the dataset provided, which contains of seven features and a target. Our goal is to predict a target, that has a binary representation.

1.1 Dataset

The provided dataset has 766 entries (rows) and 8 variables (columns), including 7 predictor variables (f1-f7) and 1 binary target variable (target).

The dataset consists of numerical values, with no missing values. The main statistics for each variable are as follows:

Table 1: Main Statistics of the dataset

Variable	Count	Mean	Std	Min	Max
f1	760.0	3.834211	3.364762	0.000	17.00
f2	760.0	120.969737	32.023301	0.000	199.00
f3	760.0	69.119737	19.446088	0.000	122.00
f4	760.0	20.507895	15.958029	0.000	99.00
f5	760.0	80.234211	115.581444	0.000	846.00
f6	760.0	31.998684	7.899724	0.000	67.10
f7	760.0	0.473250	0.332277	0.078	2.42
target	760.0	0.350000	0.477284	0.000	1.00

1.2 Visualization

The boxplot below gives us an idea of number of outliers in each feature of the dataset

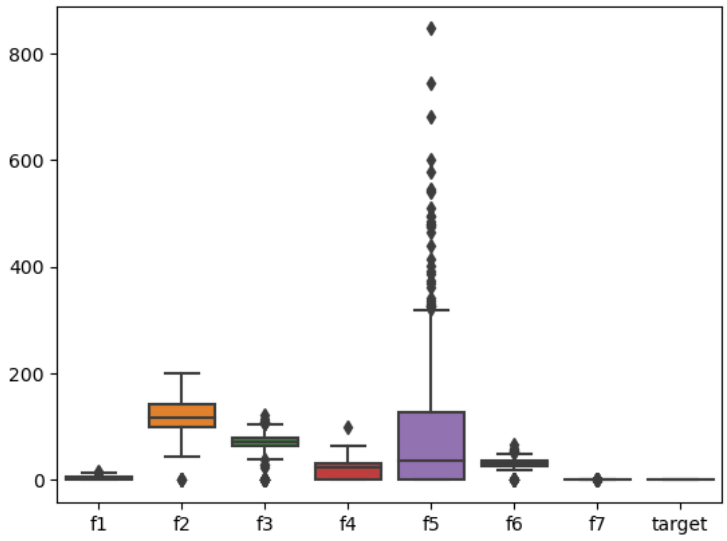

Figure 1: Distribution Boxplot of Dataset

By removing the outliers, we are loosing 185 records. As the dataset is small, we are proceeding without removing the outliers.

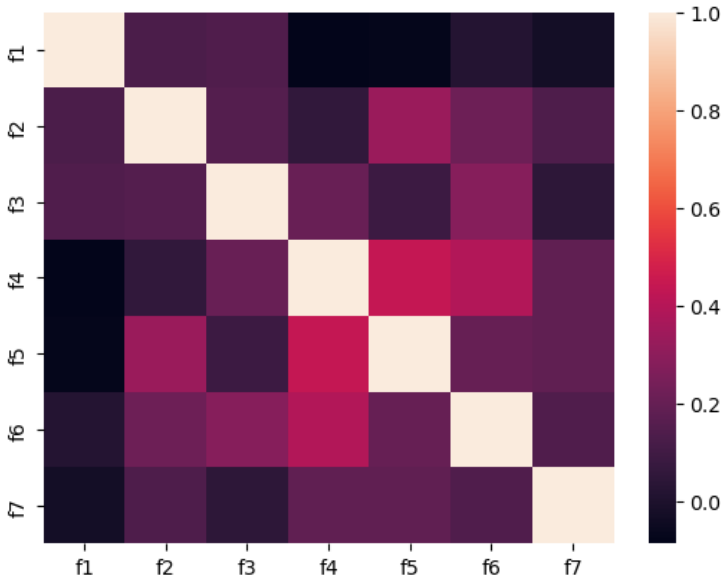

Figure 2: Distribution Heatmap of Dataset

Correlation heat map does not imply causation, and additional analysis is needed to establish causal relationships between variables.

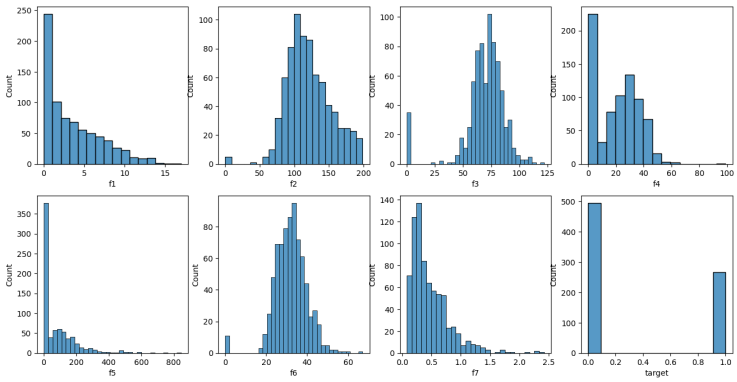

Figure 3: Distribution Histogram of Dataset

From the histogram we can note that the f1, f4, f5 has more 0 and the f2, f3, f6 are grouped over the center, so the imply good normal distribution.

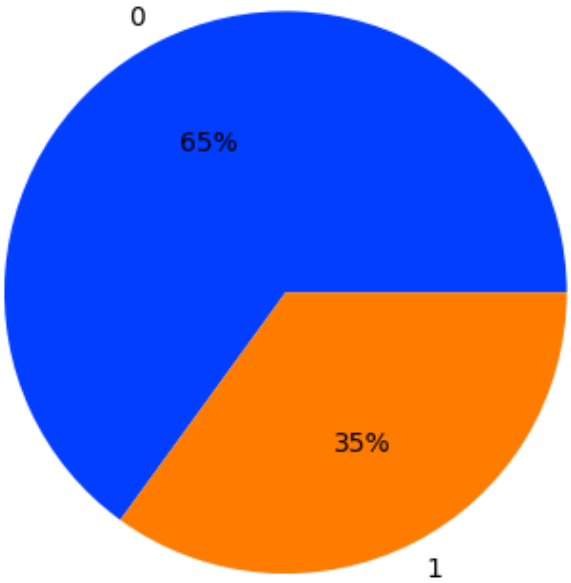

Figure 4: Distribution piechart of target

From the above graph, it is evident that there is a target imbalance in the dataset.

1.3 Preprocessing

The following is the statistics of the dataset before pre-processing. This states that there are non-numeric values present in all the columns of the dataset except f3 and target. This also implies that there are no missing values in these two columns.

Table 2: Variable Statistics

Variable	Count	Mean	Std	Min	Max
f3	766.0	69.118799	19.376901	0.0	122.0
target	766.0	0.349869	0.477240	0.0	1.0

Now on finding the unique values in all the columns we found that there are character value present in them. The following is the values in each column with characters:

Table 3: Character values in each column

Variable	f1	f2	f4	f5	f6	f7
Value	c	f	a	b	d	e

We have removed the record with these characters from the dataset. And have converted the dataset into numeric. On filtering the na values, we can see that there are no missing values.

1.4 Model Architecture Structure

The model is a sequential neural network that takes input of shape (7) and outputs a binary classification (0 or 1) using a sigmoid activation function. The model consists of three dense layers and two dropout layers to prevent overfitting.

The first dense layer has 128 neurons and uses the specified activation function and kernel initializer to transform the input data. The dropout layer randomly drops out a fraction of the neurons to prevent overfitting.

The second dense layer has 64 neurons and uses the same activation function and kernel initializer as the first layer. The second dropout layer also randomly drops out a fraction of the neurons.

The final dense layer has a single neuron and uses a sigmoid activation function to output a probability score between 0 and 1 for binary classification.

Overall, the model consists of two fully connected layers with dropout regularization to prevent overfitting and a final output layer with a sigmoid activation function to predict binary classification.

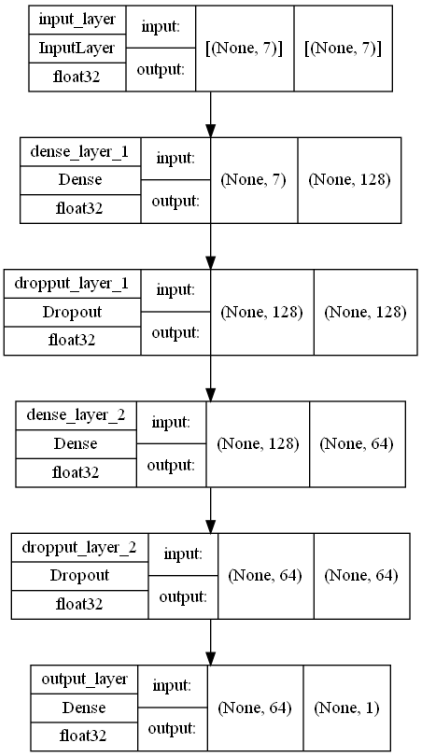

Figure 5: Model Architecture Diagram

1.5 Accuracy and Validation

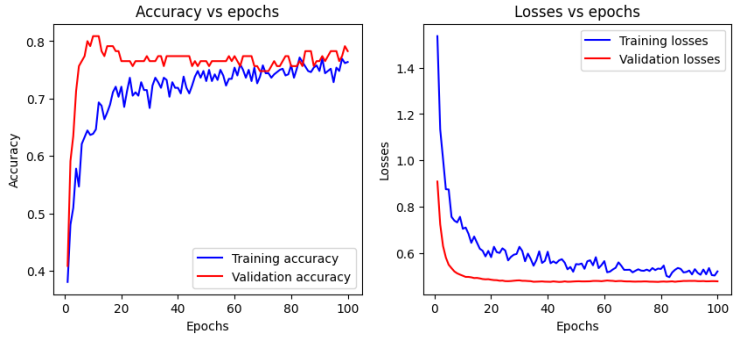

Figure 6: Test and Training accuracy, Test and Training loss

Table 4: Hyperparameters Setup and Test Accuracy

Hyperparameters	**Setup**
Dropout	0.4
Activation	LeakyReLU
Learning Rate	0.01
Initializer	HeUniform
Optimizer	SGD
Test Accuracy	0.798

We could see the validation accuracy higher than the training accuracy for all the epochs. We can say that the model is not over fitting with good accuracy.

2 Part II: Optimizing NN

2.1 Hyper Parameter Tuning

2.1.1 Dropout Tuning

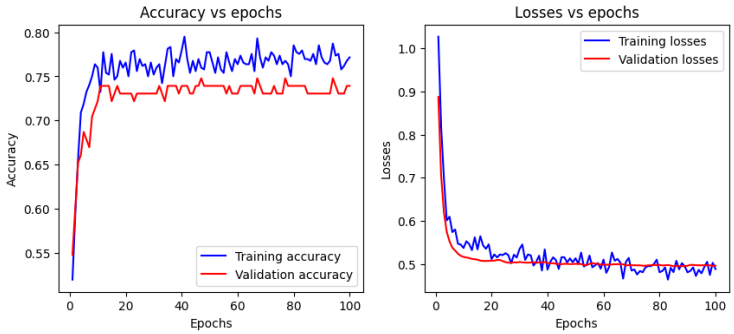

Figure 7: Dropout Setup 1 (0.1)

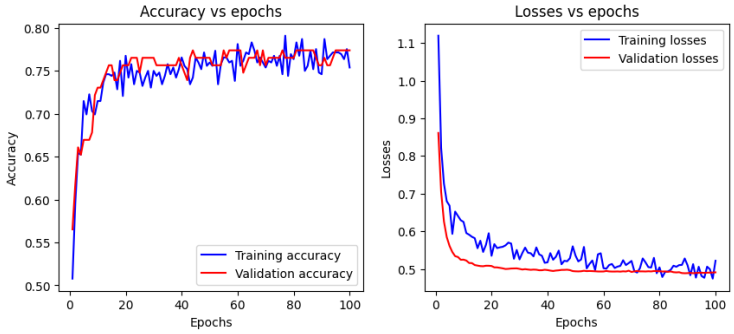

Figure 8: Dropout Setup 2 (0.2)

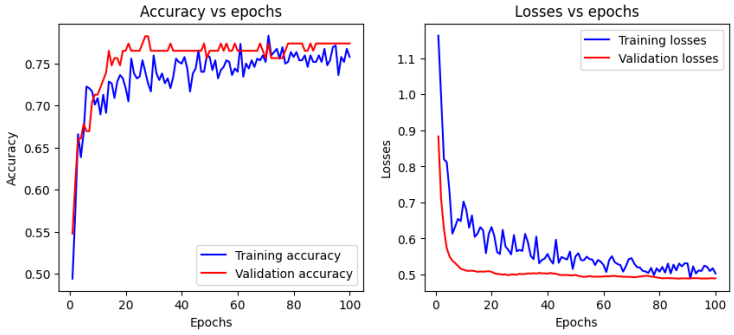

Figure 9: Dropout Setup 3 (0.3)

Hyperparameters	Setup1	Setup2	Setup3
Dropout	0.1	0.2	0.3
Activation	LeakyReLU	LeakyReLU	LeakyReLU
Learning Rate	0.01	0.01	0.01
Initializer	HeUniform	HeUniform	HeUniform
Optimizer	SGD	SGD	SGD
Test Accuracy	0.807	0.815	0.833

Table 5: Dropout Tuning

For all the three setups of dropouts, the model has given more than 80 percentage test accuracy. However, from the plots we could see model is getting over fitted for dropout values 0.1 and 0.2. Hence we are choosing dropout 0.3 as best setup. Since it gives highest test accuracy.

2.1.2 Activation Tuning

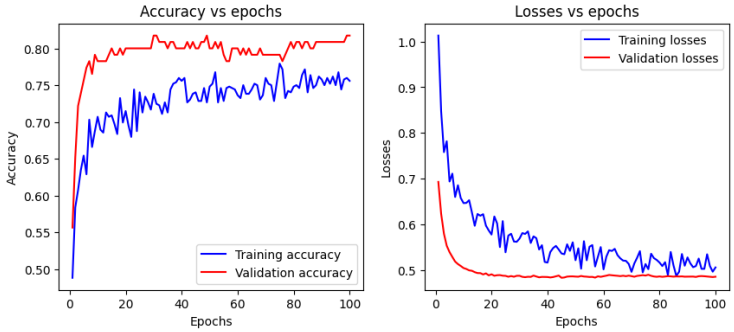

Figure 10: Activation Setup 1 (relu)

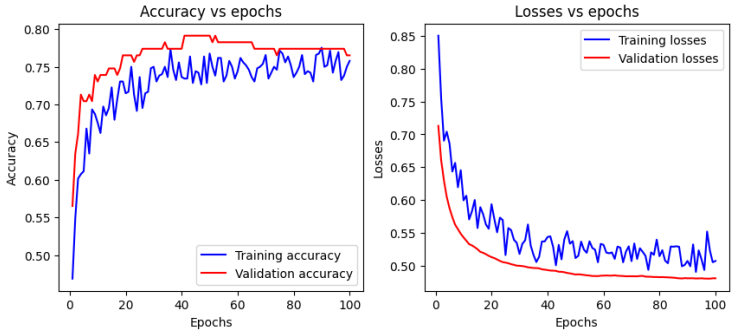

Figure 11: Activation Setup 2 (tanh)

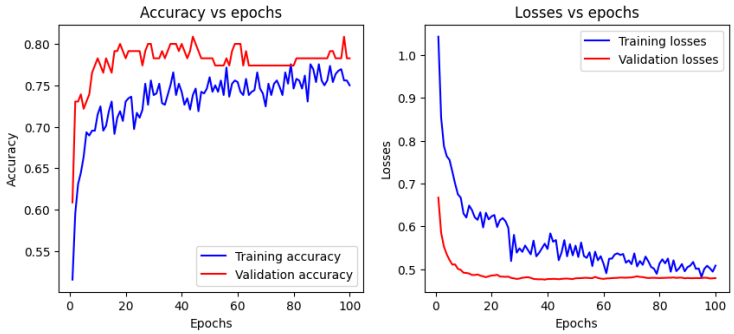

Figure 12: Activation Setup 3 (elu)

Table 6: Activation tuning

Hyperparameters	**Setup1**	**Setup2**	**Setup3**
Dropout	0.3	0.3	0.3
Activation	relu	tanh	elu
Learning Rate	0.01	0.01	0.01
Initializer	HeUniform	HeUniform	HeUniform
Optimizer	SGD	SGD	SGD
Test Accuracy	0.824	0.771	0.807

For all three setups of activation functions, the accuracy remains lesser than 0.83 which was highest in previous setup. Hence, we are proceeding with the LeakyRelu as activation function.

2.1.3 Optimizer Tuning

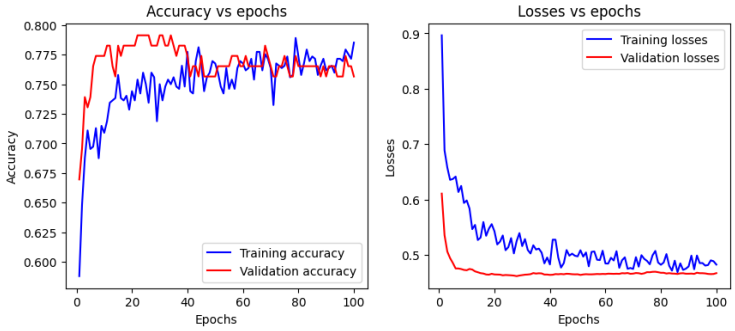

Figure 13: Optimizer Setup 1 (Adagrad)

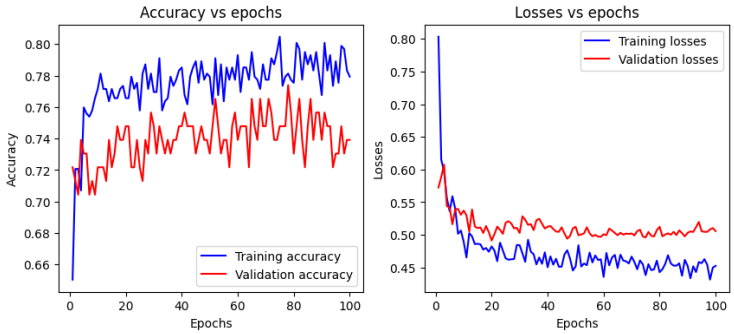

Figure 14: Optimizer Setup 2 (Adam)

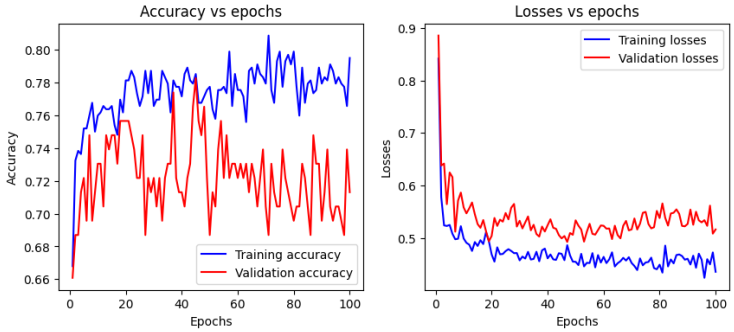

Figure 15: Optimizer Setup 3 (RMSprop)

Table 7: Optimizer tuning

Hyperparameters	**Setup1**	**Setup2**	**Setup3**
Dropout	0.3	0.3	0.3
activation	LeakyReLU	LeakyReLU	LeakyReLU
Learning Rate	0.01	0.01	0.01
Initializer	HeUniform	HeUniform	HeUniform
Optimizer	Adagrad	Adam	RMSprop
Test Accuracy	0.807	0.798	0.780

For all three setups of optimizer setups, models are gettig overfitted as per the plots. Hence, we are proceeding with the SGD as optimizer function.

2.1.4 Learning Rate Tuning

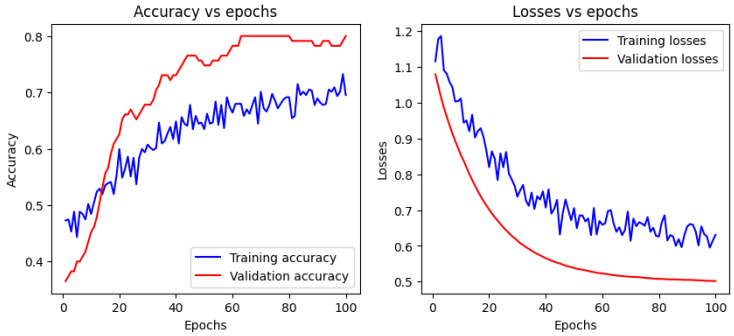

Figure 16: Learning Rate Setup 1 (0.001)

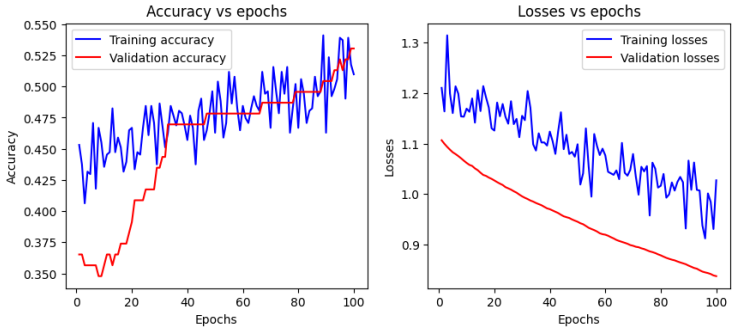

Figure 17: Learning Rate Setup 2 (0.0001)

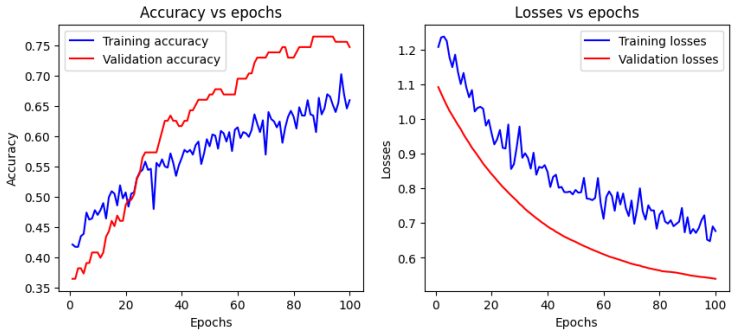

Figure 18: Learning Rate Setup 3 (0.0005)

Table 8: Learning Rate tuning

Hyperparameters	**Setup1**	**Setup2**	**Setup3**
Dropout	0.3	0.3	0.3
activation	LeakyReLU	LeakyReLU	LeakyReLU
Learning Rate	0.001	0.0001	0.0005
Initializer	HeUniform	HeUniform	HeUniform
Optimizer	SGD	SGD	SGD
Test Accuracy	0.798	0.464	0.728

For all the three setups of learning rates, the test loss is significantly higher than the previous best model. This means SGD optimizer takes more time to converge with lower learning rates. Hence, we are choosing 0.01 as learnning rate for SGD optimizer.

2.1.5 Initializer Tuning

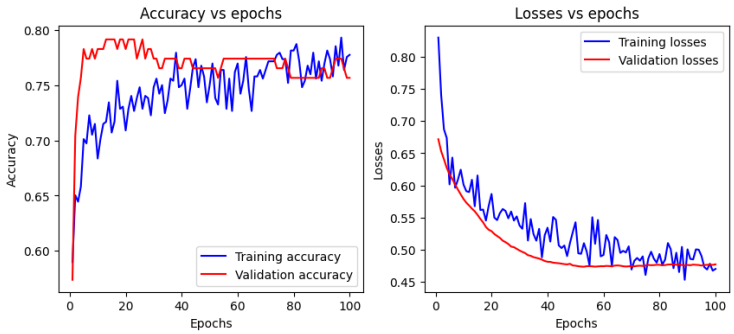

Figure 19: Initializer Tuning Setup 1 (GlorotUniform)

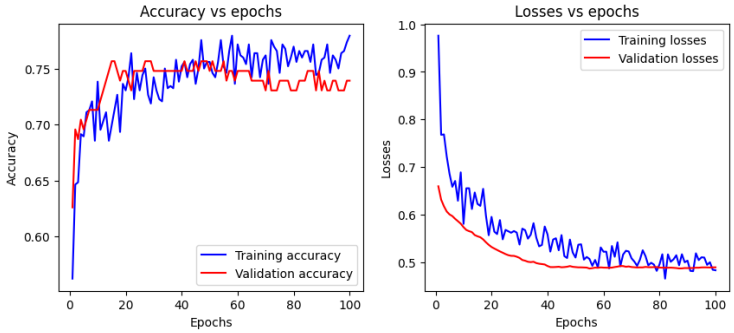

Figure 20: Initializer Tuning Setup 2 (GlorotNormal)

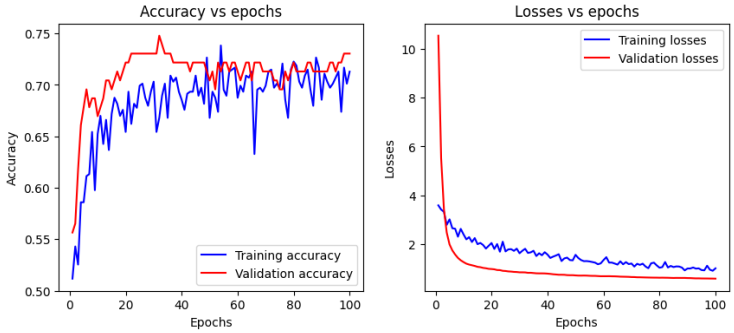

Figure 21: Initializer Tuning Setup 3 (RandomNormal)

Table 9: Initializer tuning

Hyperparameters	**Setup1**	**Setup2**	**Setup3**
Dropout	0.3	0.3	0.3
activation	LeakyReLU	LeakyReLU	LeakyReLU
Learning Rate	0.01	0.01	0.01
Initializer	GlorotUniform	GlorotNormal	RandomNormal
Optimizer	SGD	SGD	SGD
Test Accuracy	0.780	0.798	0.815

For all the three initializers, model tends to overfit or the loses are significantly higher, which is aparent from the plots. Hence, we are going ahead with HeUniform initializer as the best initilaizer.

2.2 Optimization Methods

2.2.1 Base Model

Using the best hyper parameters from earlier tuning setups, we have finalized the base model with the following setup. We can use the below graph to compare the training and test results with optimization methods.

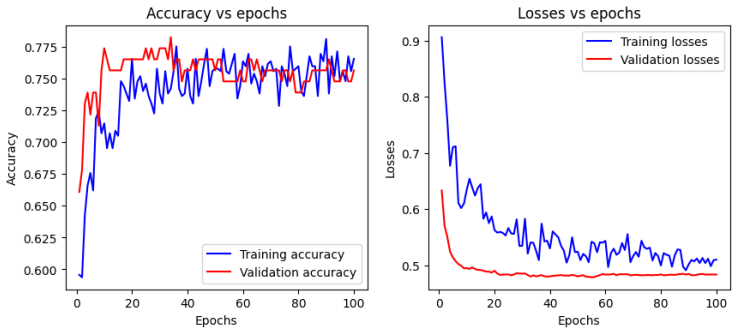

Figure 22: Base Model

Table 10: Hyperparameters Setup and Test Accuracy

Hyperparameters	**Setup**
Dropout	0.3
Activation	LeakyReLU
Learning Rate	0.01
Initializer	HeUniform
Optimizer	SGD
Training Time	0.413

2.2.2 Early Stopping

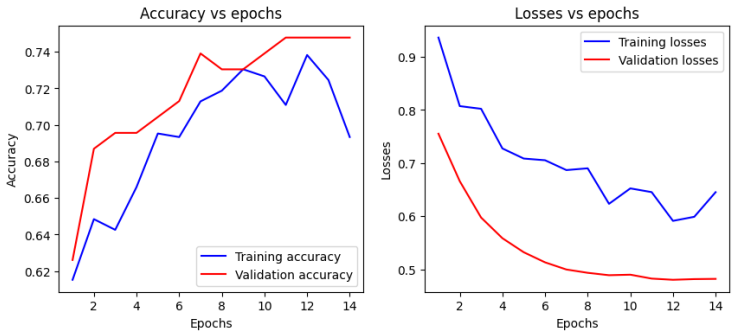

Figure 23: Early stopping

Table 11: Early Stopping

Hyperparameters	Setup
Patience	3
Test Accuracy	0.815
Training Time	0.413

Early stopping is one of the techniques to prevent over fitting, improve model performance and save time and performance. We can monitor either validation losses or validation accuracy's, if does not improve for fixed number of consecutive epochs(patience) we will stop the training loop. Implementing early stopping reduced the training time while not impacting test accuracy significantly.

2.2.3 K-fold

Table 12: k-fold

Hyperparameters	Setup
Test Accuracy	0.735
Training Time	1.885

K-fold cross-validation can be used to assess the generalization performance of the model. It can help to estimate the variability of the model's performance

We have partitioned the dataset into folds, trained the model for each fold, monitored the losses and accuracies for each fold, and estimated the average test accuracy and test loss. This has not improved our model performance significantly, but it has increased the train time and resources compared to the base model as shown in the table.

2.2.4 Learning Rate Scheduler

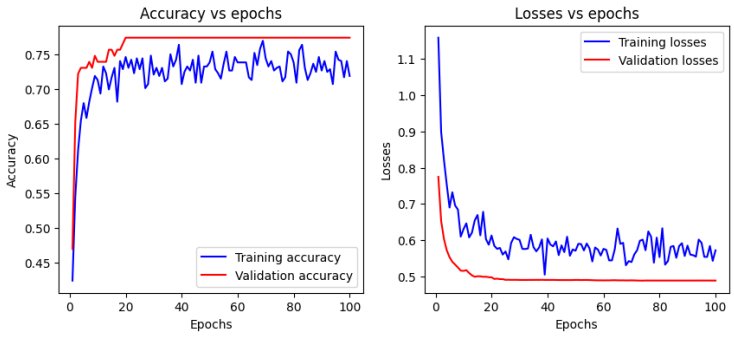

Figure 24: Learning Rate Scheduler

Table 13: Learning Rate Scheduler

Hyperparameters	Setup
Test Accuracy	0.807
Training Time	0.393

A learning rate scheduler can improve model performance in optimizing by adjusting the learning rate during the training process to help the model converge to a better solution faster and more accurately.

A learning rate scheduler can adjust the learning rate dynamically based on predefined rules or metrics, such as the number of epochs, validation loss, or accuracy. For example, a learning rate scheduler can start with a high learning rate to make large steps in the beginning of training, then gradually decrease the learning rate to fine-tune the model as it approaches convergence.

In our model, we have modified the learning rate at 25 and 75 respectively.

2.2.5 Batch Normalization

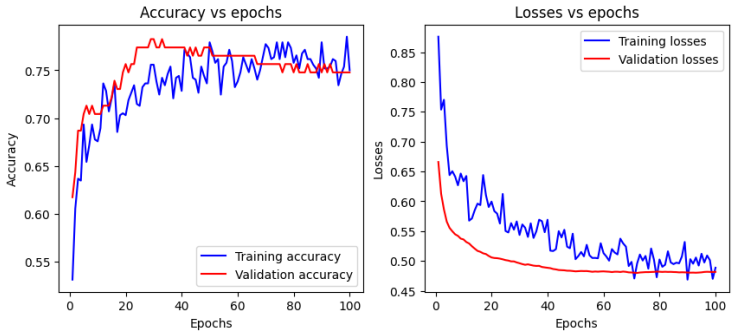

Figure 25: Batch Normalization

Table 14: Batch Normalization

Hyperparameters	Setup
Test Accuracy	0.815
Training Time	0.629

Batch Normalization works by normalizing the input values of each layer to have zero mean and unit variance. This normalization helps to reduce the internal covariate shift, which occurs when

the distribution of the inputs to each layer changes as the parameters of the preceding layers change during training.

3 Part III: Implementing & Improving AlexNet

3.1 Dataset

The dataset is an image classification dataset consisting of 30,000 JPG images of dogs, food, and vehicles. The dataset is organized into three folders, with each folder containing 10,000 images of a specific class. Since this is an image classification dataset, there are no traditional columns, and each image file represents a record. The features in the dataset are represented by the pixels in each image, which can be used to train a machine learning model to classify new images into one of the three categories.

Table 15: Main Statistics of the dataset

Mean	Std
0.4458519	0.2668456

3.2 Preprocessing

Normalizing the pixel values of the image dataset between 0 and 1 by dividing by 255.0 is a common preprocessing step used in deep learning models, as it scales the pixel values to a standardized range and makes the training process more efficient.

Dividing by 255.0 ensures that the pixel values are in the range of 0 to 1, which is important for certain types of models such as those that use sigmoid or softmax activation functions.

3.3 Visualization

The histogram graph shows the number of images for each class in the dataset:

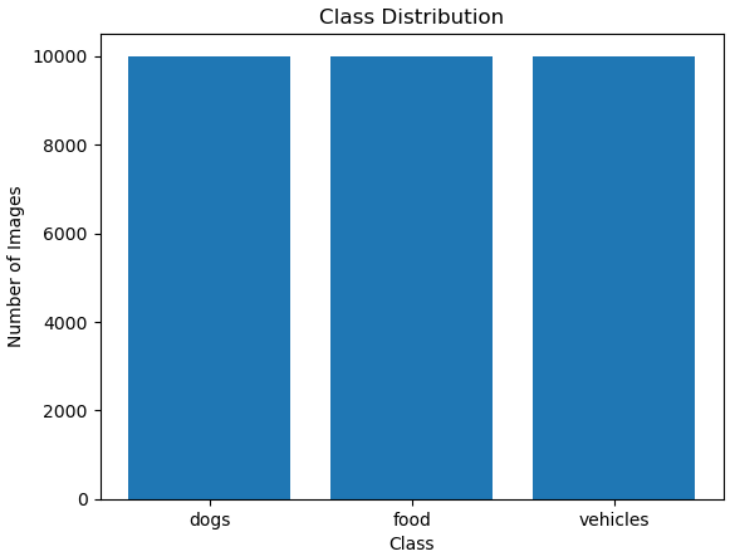

Figure 26: Distribution histogram of Dataset

We can see that the distribution is balanced, with each class containing around 10,000 images.

By visualizing the class distribution using a histogram, we can observe any class imbalances that may exist in the dataset. In this case, the histogram shows that the dataset is evenly distributed across the three classes, which is important for ensuring that the machine learning model is not biased towards any particular class.

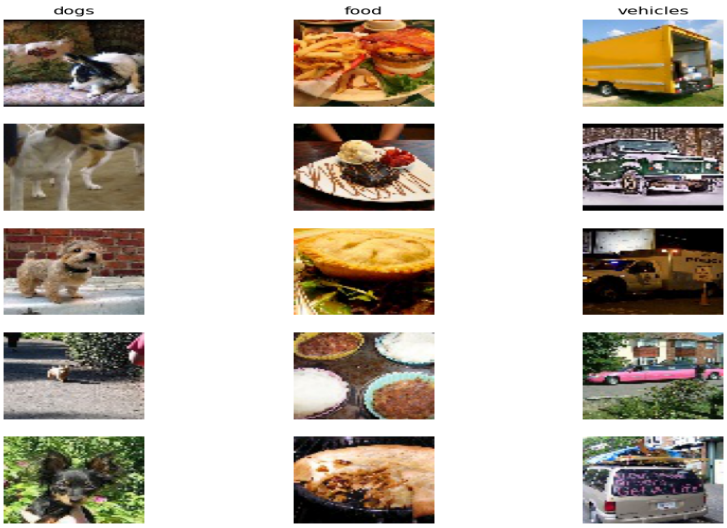

Figure 27: Image Grid

An image grid is a way of displaying multiple images in a grid-like format, where each image occupies a cell in the grid. It's a useful tool for visualizing large sets of images and comparing them side-by-side.

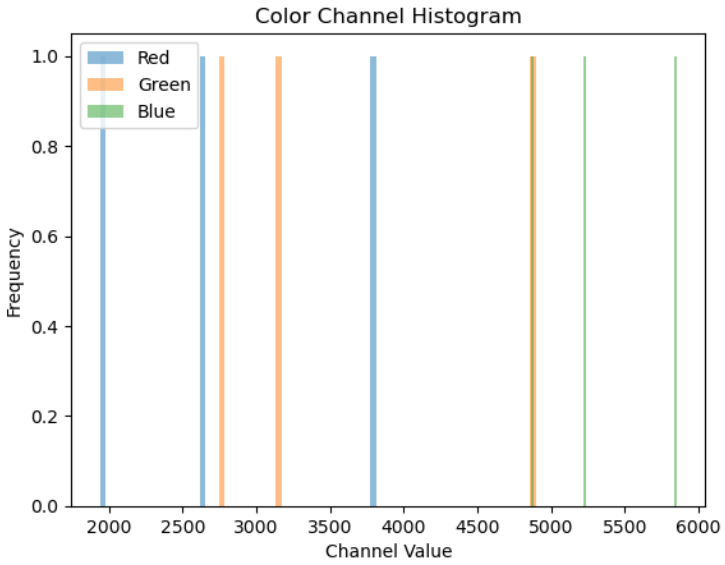

Figure 28: Colour Channel distribution of images

The histogram of the color channels (red, green, blue) is used to see the color distribution of the images in the dataset. This can help us understand whether the images are predominantly one color or whether there is a balanced distribution of colors.

3.4 Model Architecture Structure

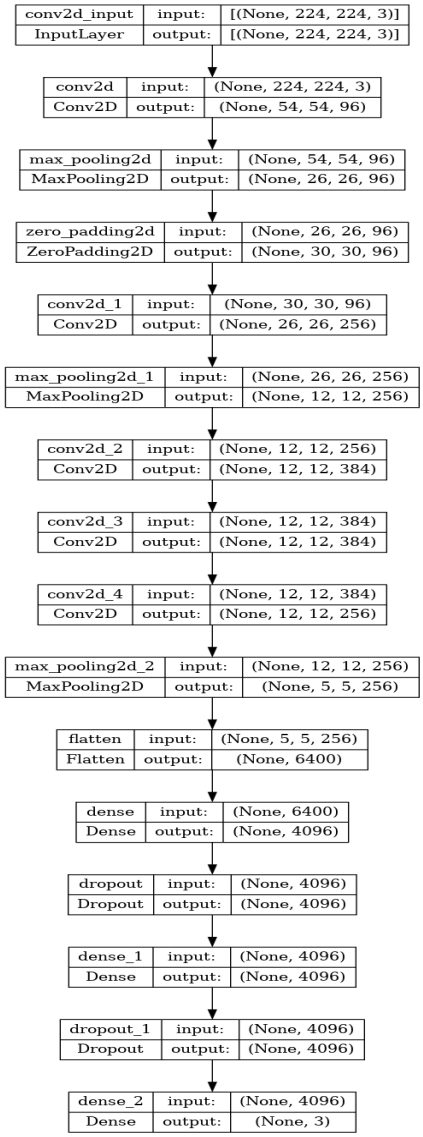

Figure 29: Model Architecture Diagram

The model is a Convolutional Neural Network (CNN) architecture consisting of multiple convolutional layers with different filter sizes and pooling layers for down-sampling the output. It has three fully connected layers with ReLU activation and a final output layer with softmax activation for multi-class classification. The input shape of the image is (224, 224, 3) and the output layer has 3 nodes for classifying the input images into one of the three classes. Dropout regularization is applied after the fully connected layers to avoid overfitting.

3.5 Accuracy and Validation

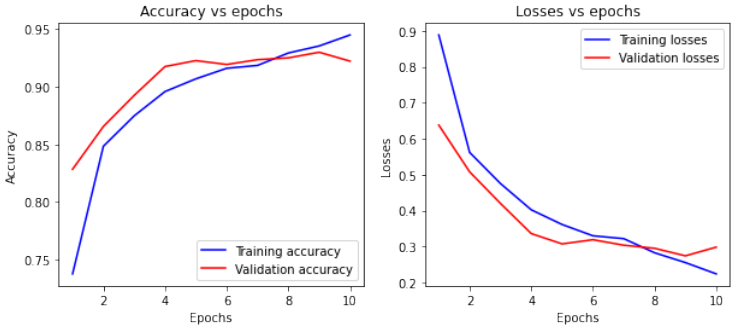

Figure 30: Test and Training accuracy, Test and Training loss

Table 16: Hyperparameters Setup

Hyperparameters	**Setup**
Activation	relu
Optimizer	SGD(momentum=0.9)
Learning Rate	0.001
Loss function	CategoricalCrossentropy
Initializer	glorot_uniform
dropout	0.5
Test Accuracy	0.922

We could see the validation accuracy higher than the training accuracy for initial epochs but the model seems to overfit towards the final epochs.

3.6 Modifying the Architecute and fine tuning parameters

3.6.1 Adding one Conv2D layer with 256 filters to the original AlexNet structure

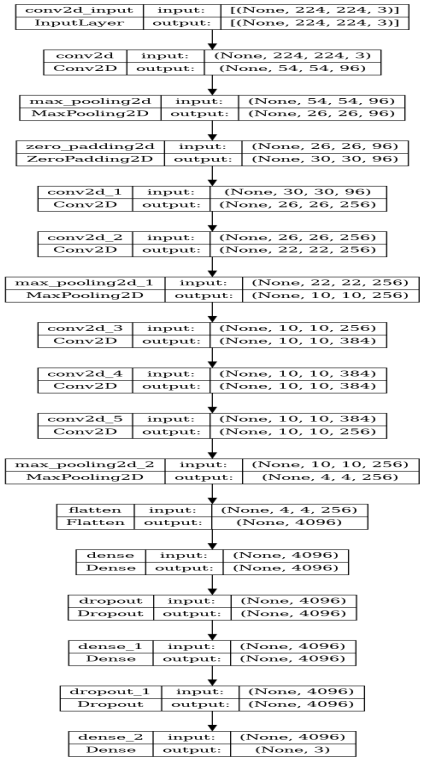

Figure 31: Model Architecture

Table 17: Hyperparameters Setup

Hyperparameters	**Setup**
Activation	relu
Optimizer	SGD(momentum=0.9)
Learning Rate	0.001
Loss function	CategoricalCrossentropy
Initializer	glorot_uniform
dropout	0.5
Test Accuracy	0.906

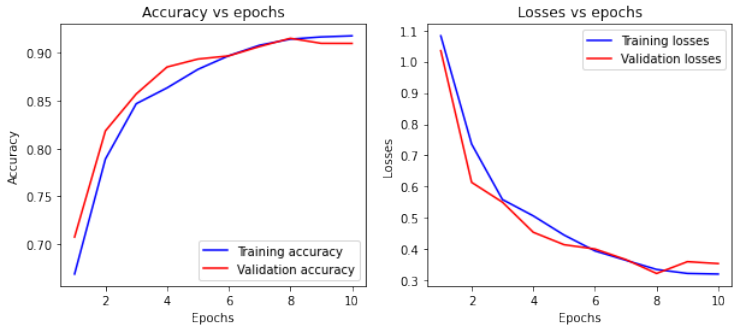

Figure 32: Test and Training accuracy, Test and Training loss

We could see a similar trend as earlier architecture, the validation accuracy is dropping as the number of epoch increases and the validation loss seems to be increasing. This could also denote of the model.

3.6.2 Adding one Conv2D layer with 384 filters to the original AlexNet structure

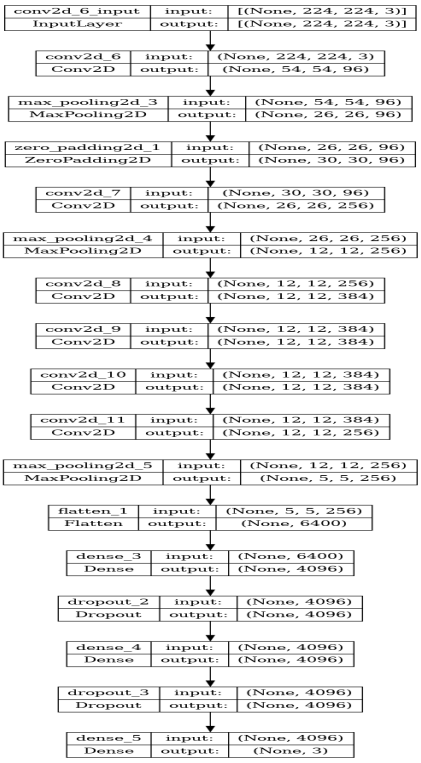

Figure 33: Model Architecture

Table 18: Hyperparameters Setup

Hyperparameters	**Setup**
Activation	relu
Optimizer	SGD(momentum=0.9)
Learning Rate	0.001
Loss function	CategoricalCrossentropy
Initializer	glorot_uniform
dropout	0.5
Test Accuracy	0.928

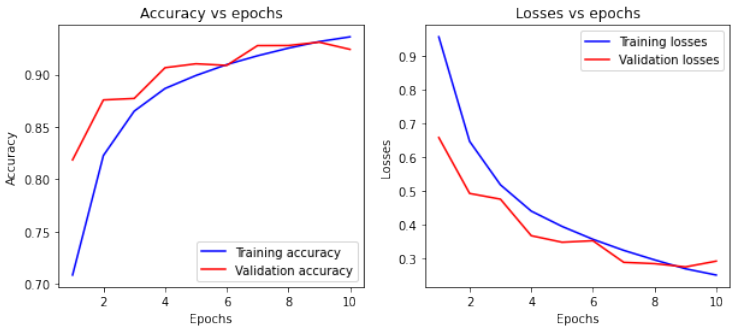

Figure 34: Test and Training accuracy, Test and Training loss

Even though the validation accuracy is significantly higher than the training accuracy initially, it seems to decrease towards the final epoch. Same is the case with training loss and validation loss.

3.6.3 Removing a dropout layer before the final dense layer

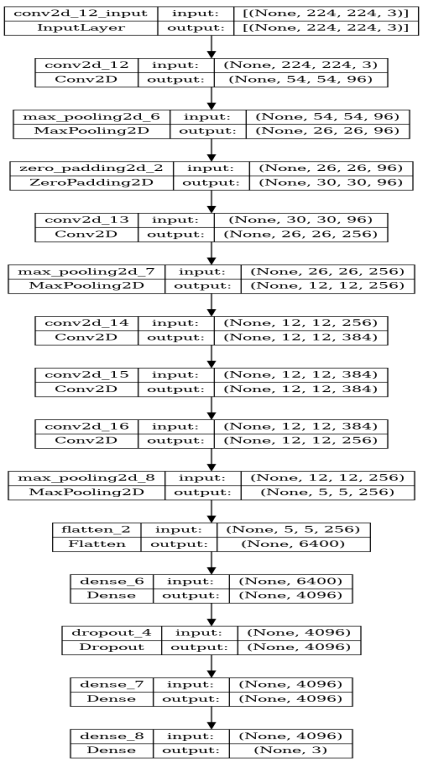

Figure 35: Model Architecture

Table 19: Hyperparameters Setup

Hyperparameters	**Setup**
Activation	relu
Optimizer	SGD(momentum=0.9)
Learning Rate	0.001
Loss function	CategoricalCrossentropy
Initializer	glorot_uniform
dropout	0.5
Test Accuracy	0.917

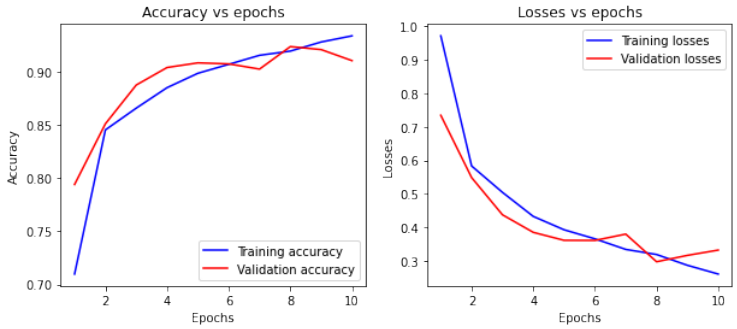

Figure 36: Test and Training accuracy, Test and Training loss

Removing the dropout layer does not help much with decreasing overfitting which is evident from the above graph as the trend is similar to earlier architecture.

3.6.4 Modifying the optimizer from SGD to Adam to the original AlexNet structure

Table 20: Hyperparameters Setup

Hyperparameters	**Setup**
Activation	relu
Optimizer	Adam
Learning Rate	0.001
Loss function	CategoricalCrossentropy
Initializer	glorot_uniform
dropout	0.5
Test Accuracy	0.911

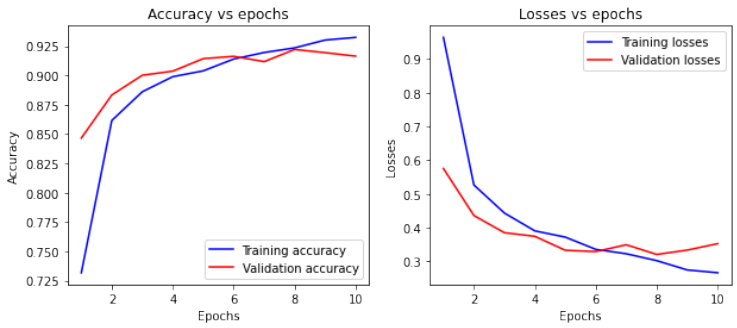

Figure 37: Test and Training accuracy, Test and Training loss

Here, changing the optimizer to adam resulted in lower validation loss at initial epochs. However, the model ended up over fitting similar to earlier cases.

3.6.5 Modifying the activation function from relu to tanh to the original AlexNet structure

Table 21: Hyperparameters Setup

Hyperparameters	**Setup**
Activation	tanh
Optimizer	SGD(momentum=0.9)
Learning Rate	0.001
Loss function	CategoricalCrossentropy
Initializer	glorot_uniform
dropout	0.5
Test Accuracy	0.554

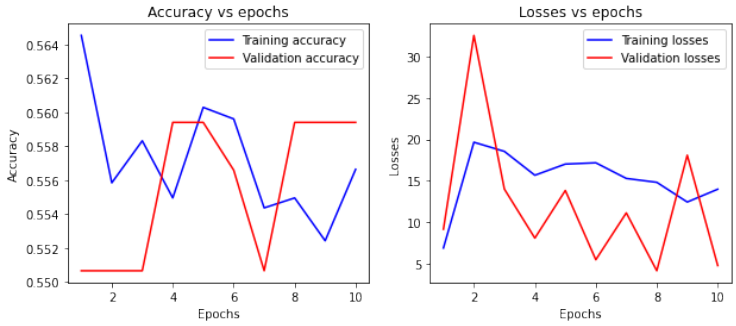

Figure 38: Test and Training accuracy, Test and Training loss

From the above graph, tanh activation function resulted in very poor performance as the accuracy and losses are highly fluctuating for each epoch.

3.6.6 Modifying the learning rate of SGD from default(0.001) to 0.01 to the original AlexNet structure

Table 22: Hyperparameters Setup

Hyperparameters	**Setup**
Activation	relu
Optimizer	SGD(momentum=0.9)
Learning Rate	0.01
Loss function	CategoricalCrossentropy
Initializer	glorot_uniform
dropout	0.5
Test Accuracy	0.921

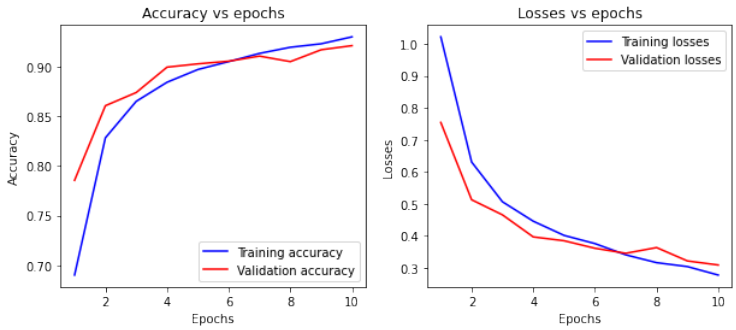

Figure 39: Test and Training accuracy, Test and Training loss

Increasing the learning rate from 0.001 to 0.01 did not help our model, as the accuracy and loss graph still indicates over-fitting towards the final epoch.

3.6.7 Removing zero padding layer from the original AlexNet structure

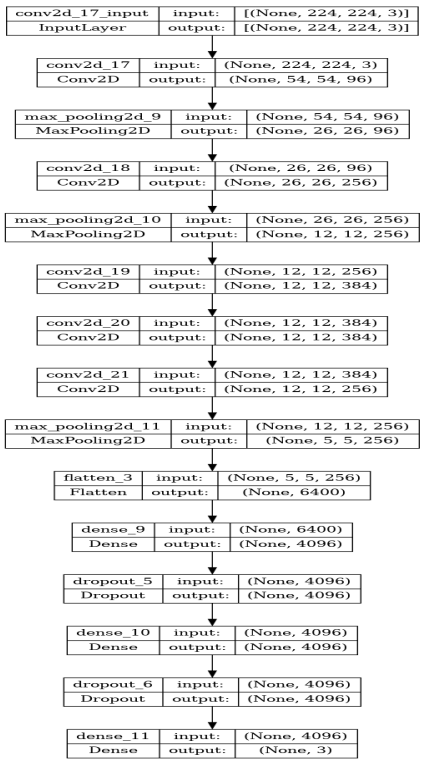

Figure 40: Model Architecture

Table 23: Hyperparameters Setup

Hyperparameters	**Setup**
Activation	relu
Optimizer	SGD(momentum=0.9)
Learning Rate	0.001
Loss function	CategoricalCrossentropy
Initializer	glorot_uniform
dropout	0.5
Test Accuracy	0.926

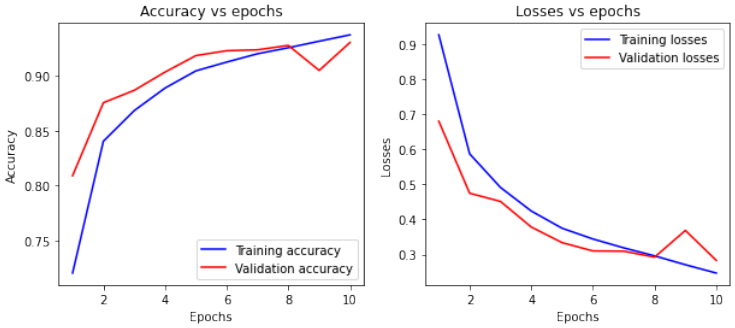

Figure 41: Test and Training accuracy, Test and Training loss

After removing zero padding layer, the model seems to have not improved in any way in both accuracy and loss.

3.6.8 Implementing Early Stopping with adding a Conv2D(384) to the original AlexNet structure

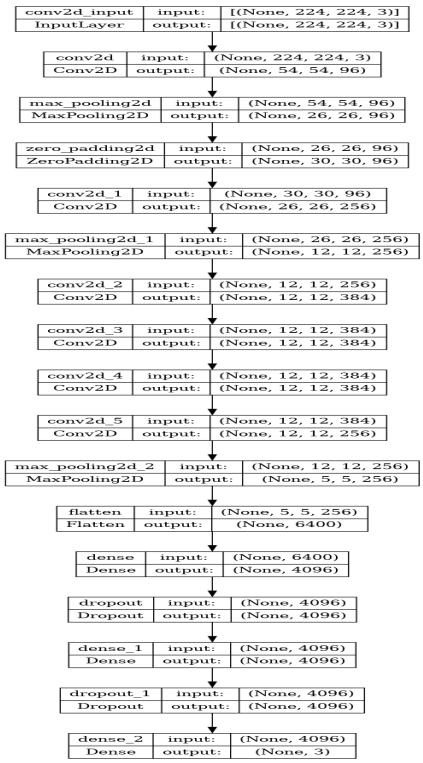

Figure 42: Model Architecture

Table 24: Hyperparameters Setup

Hyperparameters	**Setup**
Activation	relu
Optimizer	SGD(momentum=0.9)
Learning Rate	0.001
Loss function	CategoricalCrossentropy
Initializer	glorot_uniform
dropout	0.5
Test Accuracy	0.936

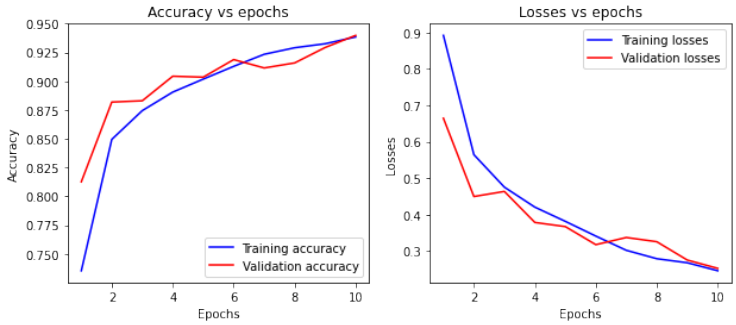

Figure 43: Test and Training accuracy, Test and Training loss

Implementing early stopping and adding a layer of Conv2D(384) has increased the accuracy and the over fitting has decreased.

3.6.9 Adding multiple Conv2D layers with filters 384 and 256 to the original AlexNet structure and changing the dropout from 0.5 to 0.4

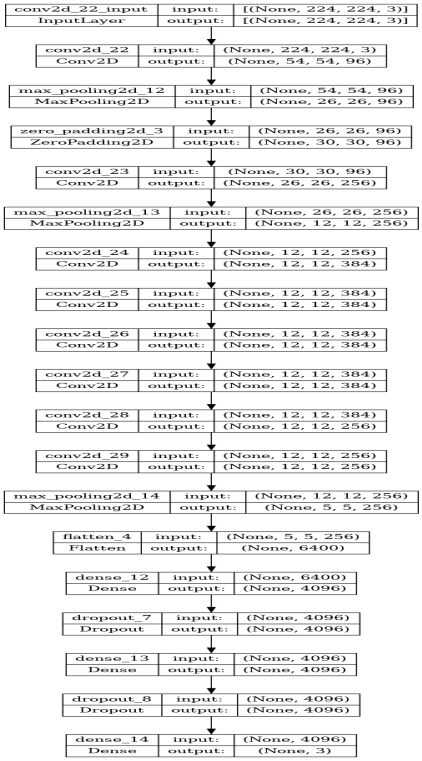

Figure 44: Model Architecture

Table 25: Hyperparameters Setup

Hyperparameters	**Setup**
Activation	relu
Optimizer	SGD(momentum=0.9)
Learning Rate	0.001
Loss function	CategoricalCrossentropy
Initializer	glorot_uniform
dropout	0.4
Test Accuracy	0.914

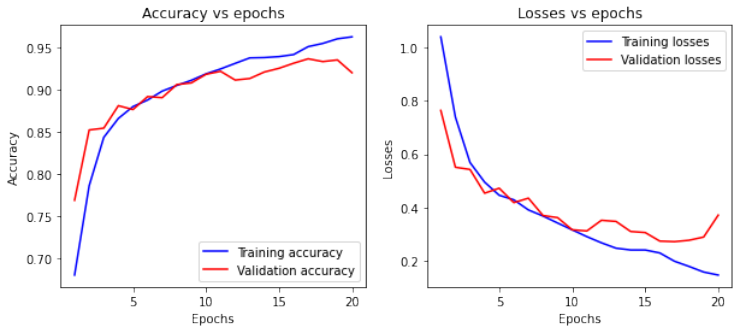

Figure 45: Test and Training accuracy, Test and Training loss

Adding multiple convolution layers and changing the dropout to 0.4 has decreased the accuracy significantly and the overfitting has also increased evidently.

3.6.10 Implementing Batch Normalization along with Early Stopping while training the model along with adding an additional dropout layer and couple of Conv2D layers

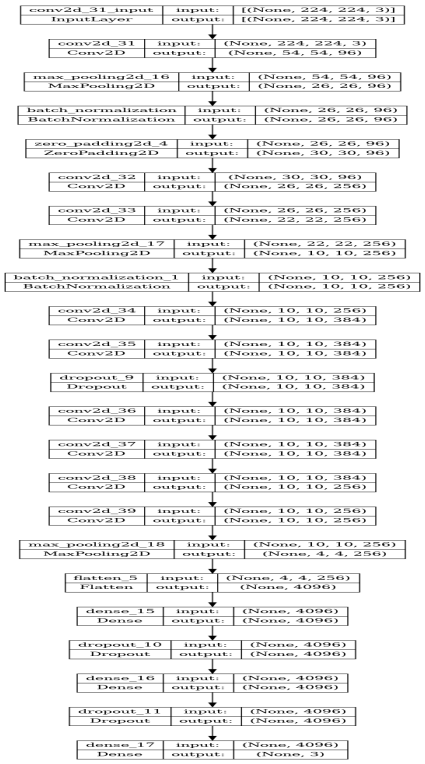

Figure 46: Model Architecture

Table 26: Hyperparameters Setup

Hyperparameters	**Setup**
Activation	relu
Optimizer	SGD(momentum=0.9)
Learning Rate	0.001
Loss function	CategoricalCrossentropy
Initializer	glorot_uniform
dropout	0.4
Test Accuracy	0.943

This is the best accuracy we have got by modifying the architecture and implementing optimizing methods and fine tuning parameters.

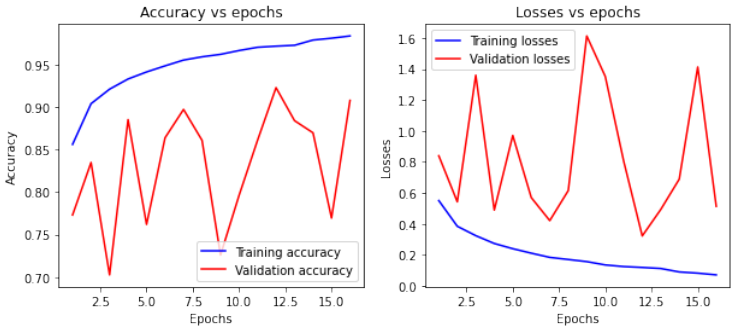

Figure 47: Test and Training accuracy, Test and Training loss

3.7 Part IV: Optimizing CNN + Data Argumentation

3.8 Dataset

The dataset provided contains images of house numbers in the range of 0-9. The dataset is part of the SVHN (Street View House Numbers) dataset, which is a large-scale dataset obtained from Google Street View images.

The files train_32x32.mat and test_32x32.mat contain images of house numbers, each of size 32x32 pixels. The train file contains 73,257 images, while the test file contains 26,032 images. Each image in the dataset is labeled with the corresponding digit it represents. The image is in th form of pixel values.

The SVHN dataset is commonly used for training and testing machine learning algorithms, particularly those related to image recognition and classification. The dataset is challenging because the images are taken under real-world conditions, with varying lighting conditions, backgrounds, and perspectives.

Table 27: Descriptive Statistics

Number of images in training set	73257
Number of images in test set	26032
Image size	(32, 32)
Number of channels	3
Number of classes	10

Table 28: Label distribution

Label	Training	Test
1	13861	5099
2	10585	4149
3	8497	2882
4	7458	2523
5	6882	2384
6	5727	1977
7	5595	2019
8	5045	1660
9	4659	1595
10	4948	1744

3.9 Visualization

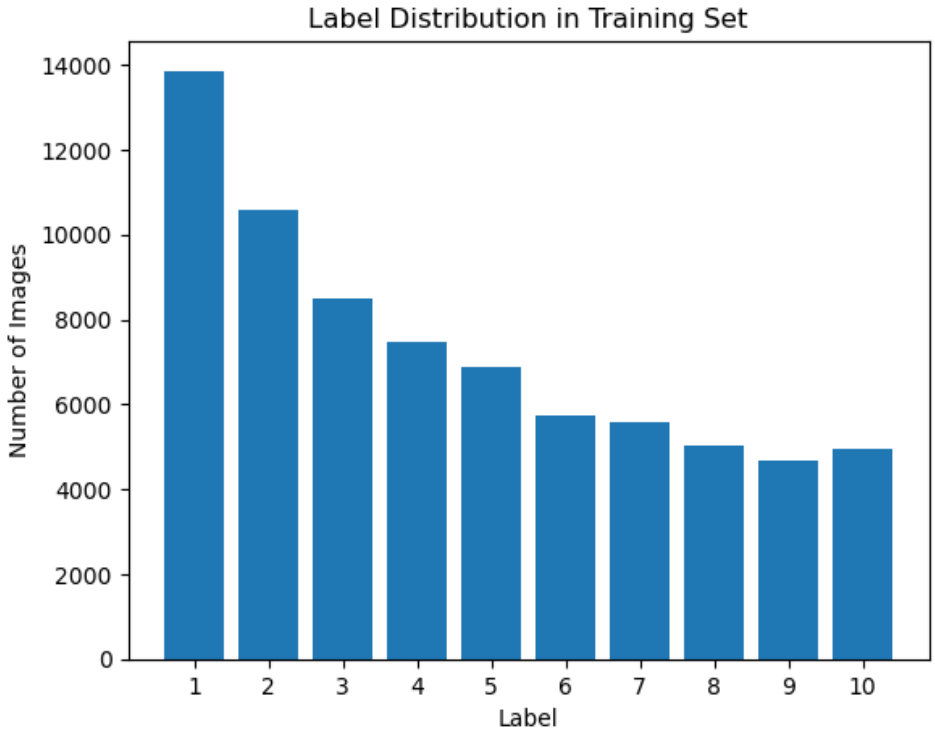

Figure 48: Histogram of label distribution

The histogram of label distribution in the training dataset shows the number of images in each of the 10 possible classes, ranging from 0 to 9. The class with the largest number of images is class 1, followed by class 2, and the class with the smallest number of images is class 9. The distribution is somewhat imbalanced, with some classes having significantly more images than others.

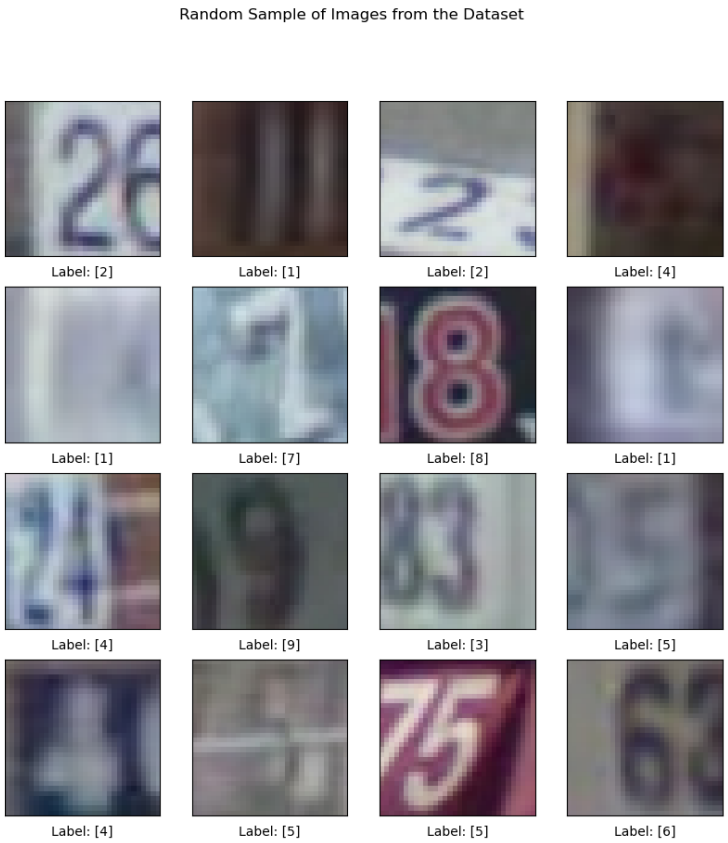

Figure 49: Random Sample of Images from the Dataset

The above graph shows 16 randomly selected images from the Street View House Numbers dataset, which consists of 73257 32x32 color images of house numbers. The graph displays the selected images in a 4x4 grid, along with their corresponding labels. The images have varying levels of brightness, contrast, and digit orientation, reflecting the real-world variability of house numbers in different contexts.

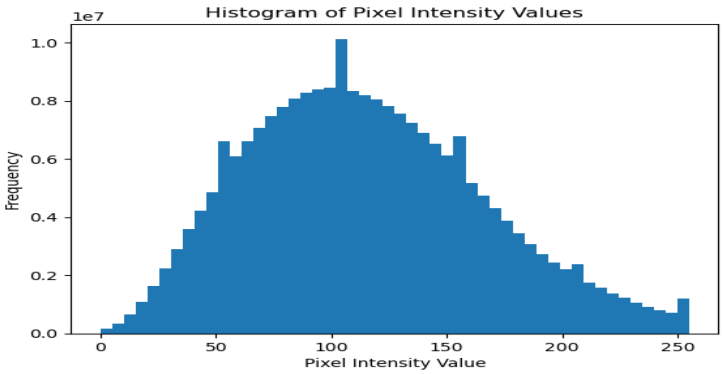

Figure 50: Histogram of pixel intensity values

The histogram of pixel intensity values of the train_32x32 dataset shows that the majority of pixel intensity values are concentrated around the middle range of 0-150, with a long tail of higher intensity

values. This suggests that the images in the dataset have a wide range of pixel intensities, with some containing very bright or very dark regions.

3.10 Model Architecture

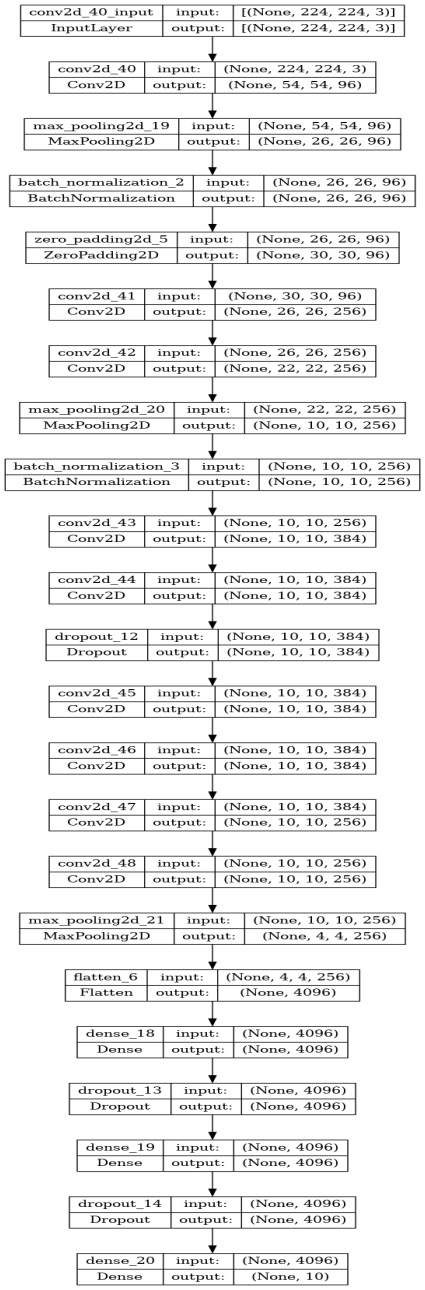

Figure 51: Best CNN Architecture from Part III as base

This is a modified AlexNet architecture which gave the best test accuracy for part III. This includes multiple layers of convolutions of filter size 384 and 256. It also includes batch normalization and dropout layers of 0.4. This model also includes early stopping implementation.

3.11 Adjusting Model from Part III

The best acuuracy model from part III has 3 output filters in the final dense layer because it was predicting only three classes. For SVHN, the target classes are 10 in number. Inorder to use that model, we need to adjust the final dense layer from 3 to 10. This is the only change that was required to be made for the base model.

3.12 Data Preprocessing

We have resized(224,224) the dataset to fit the dataset to our model. Then we are rescaling the pixel value by divinding them by 255.0. The pixels are now in the range 0 to 1. This can help in normalization and ensure that the pixel values have a consistent range across all images. Normalization can help the model converge faster and improve its performance.

3.13 Data Augmentation

In real-world scenarios, images may be captured under different lighting conditions, angles, and scales. Data augmentation can help make the model more robust to such variations by teaching it to recognize the same object regardless of the transformation applied to it.

Data augmentation techniques can be used to create new examples from the existing data by applying various transformations to the images, such as rotating(0.2), flipping(horizontal/vertical), brightness(-0.95 to +0.95) and contrast(0.1 to 0.9). By doing so, we can effectively increase the size of the dataset and provide the model with more examples to learn from, which can improve its accuracy.

After augumentation the training dataset in now at 146,514. This is twice the number of images in the initial dataset.

3.14 Trying different Setups

3.14.1 Base Model from Part III

Table 29: Hyperparameters Setup

Hyperparameters	**Setup**
Activation	relu
Optimizer	SGD(momentum=0.9)
Learning Rate	0.001
Loss function	CategoricalCrossentropy
Initializer	glorot_uniform
dropout	0.4
Test Accuracy	0.945

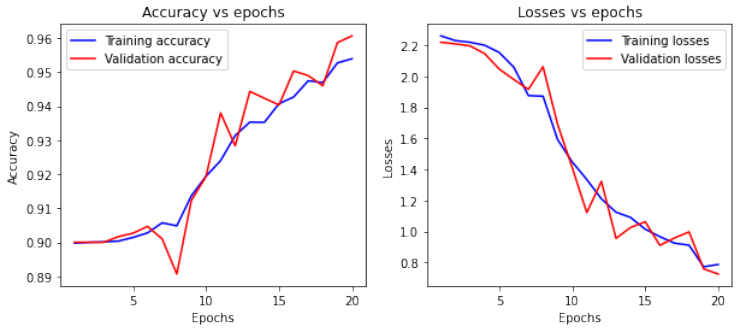

Figure 52: Test and Training accuracy, Test and Training loss

Based on the above plot, the model is not overfitting and the validation accuracy is almost reaching 96% and validation loss is as low as 0.8.

3.14.2 Changing learning rate from default 0.001 to 0.01

Table 30: Hyperparameters Setup

Hyperparameters	**Setup**
Activation	relu
Optimizer	SGD(momentum=0.9)
Learning Rate	0.01
Loss function	CategoricalCrossentropy
Initializer	glorot_uniform
dropout	0.4
Test Accuracy	0.899

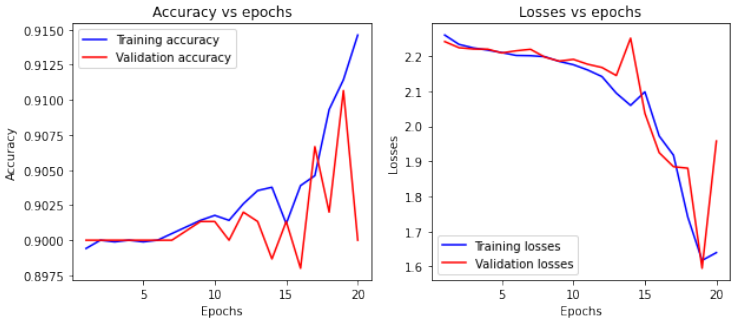

Figure 53: Test and Training accuracy, Test and Training loss

3.14.3 Changing optimizer to Adam

Table 31: Hyperparameters Setup

Hyperparameters	**Setup**
Activation	relu
Optimizer	Adam
Learning Rate	0.001
Loss function	CategoricalCrossentropy
Initializer	glorot_uniform
dropout	0.4
Test Accuracy	0.899

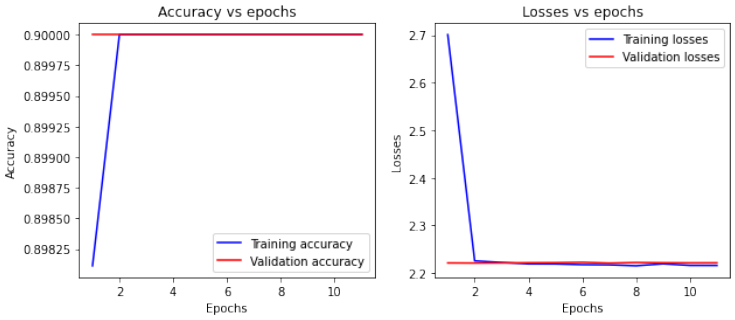

Figure 54: Test and Training accuracy, Test and Training loss

3.14.4 Changing Adam learning rate from default 0.001 to 0.01

Table 32: Hyperparameters Setup

Hyperparameters	**Setup**
Activation	relu
Optimizer	Adam
Learning Rate	0.01
Loss function	CategoricalCrossentropy
Initializer	glorot_uniform
dropout	0.4
Test Accuracy	0.899

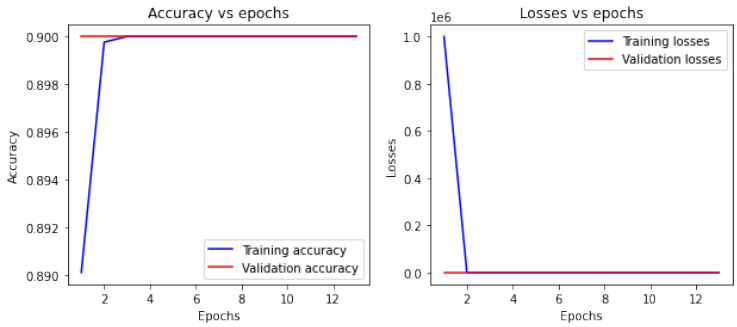

Figure 55: Test and Training accuracy, Test and Training loss

3.14.5 Using SGD and adding multiple Dropout layers

Table 33: Hyperparameters Setup

Hyperparameters	**Setup**
Activation	relu
Optimizer	SGD(momentum=0.9)
Learning Rate	0.001
Loss function	CategoricalCrossentropy
Initializer	glorot_uniform
dropout	0.4
Test Accuracy	0.899

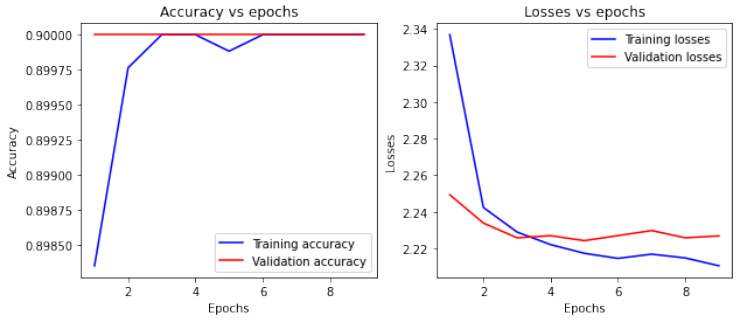

Figure 56: Test and Training accuracy, Test and Training loss

3.14.6 Using LeakyReLU activation function instead of ReLU along with SGD and multiple dropout layers

Table 34: Hyperparameters Setup

Hyperparameters	**Setup**
Activation	LeakyReLU
Optimizer	SGD(momentum=0.9)
Learning Rate	0.001
Loss function	CategoricalCrossentropy
Initializer	glorot_uniform
dropout	0.4
Test Accuracy	0.899

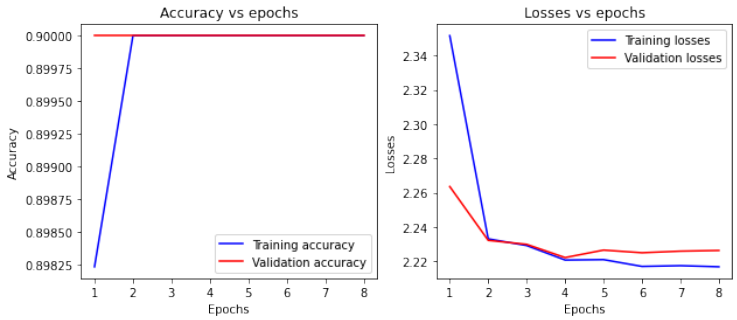

Figure 57: Test and Training accuracy, Test and Training loss

3.14.7 Using Adam optimizer with Leaky ReLU activation and multiple dropout layers

Table 35: Hyperparameters Setup

Hyperparameters	**Setup**
Activation	LeakyReLU
Optimizer	Adam
Learning Rate	0.001
Loss function	CategoricalCrossentropy
Initializer	glorot_uniform
dropout	0.4
Test Accuracy	0.825

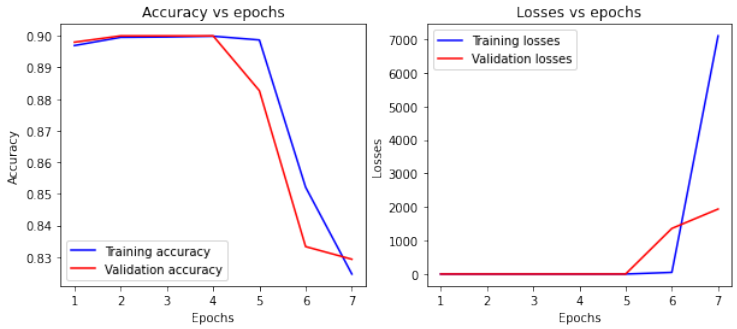

Figure 58: Test and Training accuracy, Test and Training loss

For all of the above different setups, the model is highly overfitting and test accuracy does not exceed 0.89. The base model from part III has performed well with SVHN dataset, as the accuracy is 94.5.

References

[1] Part I: Building a Basic NN for PART III & IV.

[2] Part II: Optimizing NN for PART III & IV.

[3] https://www.tensorflow.org/api_docs/python/tf/keras/.

[4] https://stackoverflow.com/.

[5] https://towardsdatascience.com/.

[6] https://keras.io/api/.

[7] https://machinelearningmastery.com/.