Image Classification using AlexNet

Part I: Building a Basic Neural Network

Dataset.csv

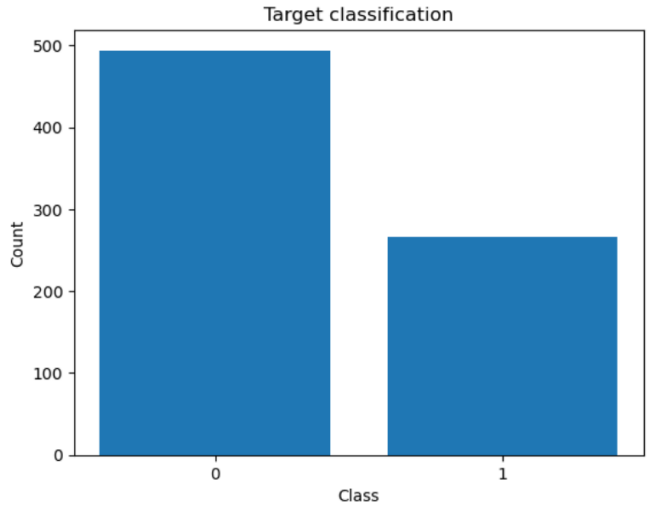
Description

The dataset consists of 766 rows and 8 columns namely – f1, f2, f3, f4, f5, f6, f7, and target. Out of these columns, only f3 and target are numerical columns. We convert f1, f2, f4, f5, f6, f7 into numerical data and replace anything that is not a number with NA. After we drop the NA values from the dataset, its shape is (760,8). Below are the basic statistics of the dataset –

	f1	f2	f3	f4	f5	f6	f7	target
count	760.000000	760.000000	760.000000	760.000000	760.000000	760.000000	760.000000	760.000000
mean	3.834211	120.969737	69.119737	20.507895	80.234211	31.998684	0.473250	0.350000
std	3.364762	32.023301	19.446088	15.958029	115.581444	7.899724	0.332277	0.477284
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	0.000000
25%	1.000000	99.000000	63.500000	0.000000	0.000000	27.300000	0.243750	0.000000
50%	3.000000	117.000000	72.000000	23.000000	36.000000	32.000000	0.375500	0.000000
75%	6.000000	141.000000	80.000000	32.000000	128.250000	36.600000	0.627500	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	1.000000

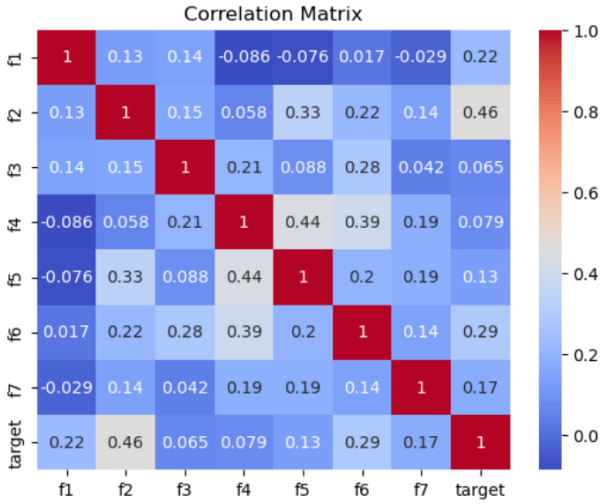
Visualization 1: Target classification vs Count

Below is a bar graph that helps us visualize the number of samples under each classification – 0 and 1. We see that ~500 samples are categorized as 0 and ~270 are categorized as 1 from the below graph.



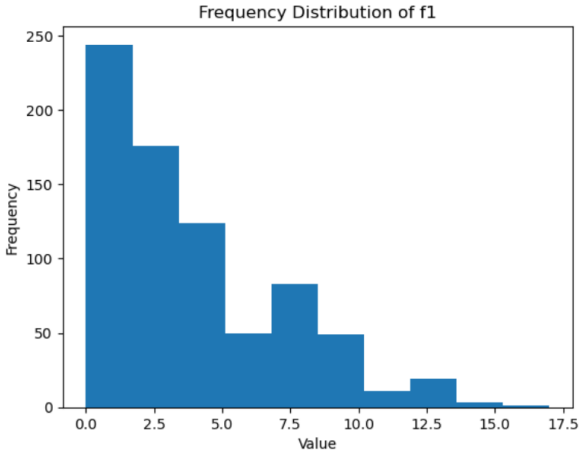
Visualization 2: Correlation Matrix

Correlation Matrix is a square matrix that shows pairwise correlations between all the variables in the dataset. It helps us to identify relationships between the variables, to select variables for analysis, and to identify multicollinearity. Below is the correlation matrix for this dataset. We see that the variable f2 and f6 have higher correlation with the target, compared to other variables in the dataset.



Visualization 3: Frequency Distribution of f1

Using histograms to visualize frequency distribution of one or more variables in a dataset helps us to identify the shape of the distribution, to understand the center and spread of the values, and to identify the gaps and clusters in the data. Below is the histogram of frequency distribution of the f1 variable of the dataset.



Data Pre-Processing

- Convert the columns to numerical: Out of 8 columns, only f3 and target are numerical columns. We convert f1, f2, f4, f5, f6, f7 into numerical data and replace anything that is not a number with NA. After we drop the NA values from the dataset, its shape is (760,8).
- Scale the data: We use the StandardScaler() from scikit-learn library to scale the dataset (all columns except target). Using this function transforms the dataset to have zero mean and standard deviation 1. Below are the basic statistics of the dataset after we apply scaling –

	f1	f2	f3	f4	f5	f6	f7	target
count	7.600000e+02	7.600000e+02	7.600000e+02	7.600000e+02	7.600000e+02	7.600000e+02	7.600000e+02	760.000000
mean	3.739699e-17	9.349247e-18	-2.898266e-16	9.582978e-17	2.337312e-18	-1.986715e-16	1.519253e-16	0.350000
std	1.000659e+00	1.000659e+00	1.000659e+00	1.000659e+00	1.000659e+00	1.000659e+00	1.000659e+00	0.477284
min	-1.140270e+00	-3.780041e+00	-3.556770e+00	-1.285961e+00	-6.946361e-01	-4.053275e+00	-1.190303e+00	0.000000
25%	-8.428760e-01	-6.865065e-01	-2.891809e-01	-1.285961e+00	-6.946361e-01	-5.951826e-01	-6.911439e-01	0.000000
50%	-2.480889e-01	-1.240456e-01	1.482128e-01	1.562691e-01	-3.829623e-01	1.666711e-04	-2.943761e-01	0.000000
75%	6.440919e-01	6.259022e-01	5.598776e-01	7.206199e-01	4.157018e-01	5.828490e-01	4.645270e-01	1.000000
max	3.915421e+00	2.438276e+00	2.721117e+00	4.921898e+00	6.629698e+00	4.446286e+00	5.862677e+00	1.000000

Neural Network Architecture

Below are the details of the NN architecture used for this assignment –

- Number of Input Neurons: 7, as we chose to use f1, f2, f3, f4, f5, f6, and f7 as our features.
- Number of Hidden Layers: 3
- Size of each Hidden Layer: 64 Neurons
- Activation Function: We chose Rectified Linear Unit (ReLU) activation between each layer. Below is the ReLU function –

$$f(x) = x^+ = \max(0, x) = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases} \qquad f'(x) = \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{if } x < 0. \end{cases}$$

- Activation Function between Hidden Layer and Output: We chose Sigmoid Activation function between the hidden layer and output –

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1} = 1 - S(-x).$$

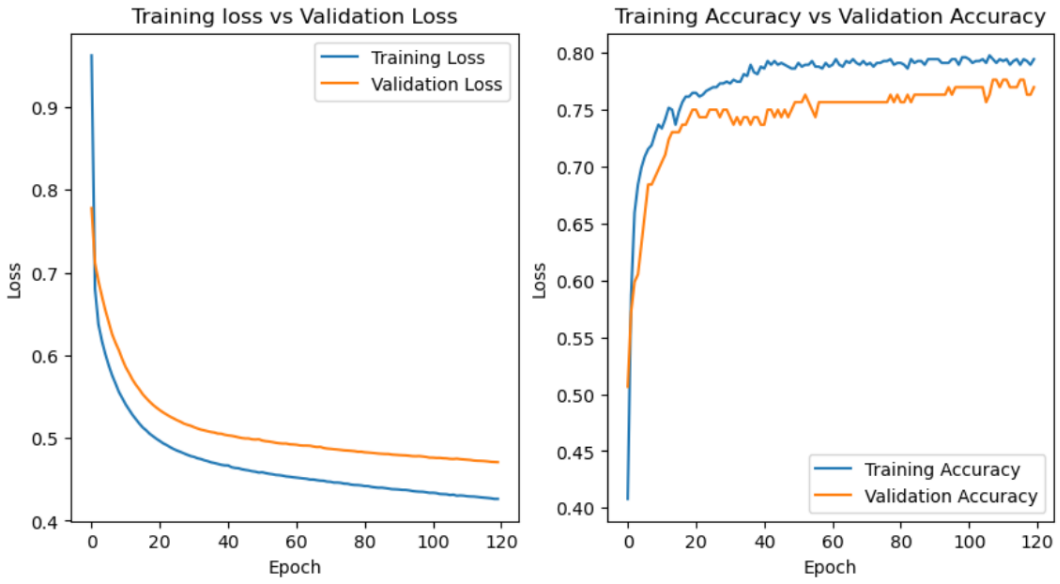
We use the below parameters to train the NN:

- batch_size = 16
- epochs = 120
- learning_rate = 0.002
- loss_func = nn.BCELoss()
- device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
- Dropout Probability = 0
- Initializer = Kaiming Normal

Plots

Observing the training loss vs validation loss plot, we notice that the validation loss decreases along with the training loss as the number of epochs increase. And, looking at the training accuracy vs validation accuracy, we notice that as number of epochs increase, the validation accuracy has increased greatly till epoch 60 and slowly improved after that. This model has achieved a test accuracy of 76.97% and the weights are saved into smahanka_saitejad_assignment2_part1.pt file.

```
Training and Validation Time (in seconds) :  7.315996170043945
Testing loss :  0.4708745981517591
Test Accuracy = 76.97
```



Part II: Optimizing the Neural Network

Dropouts Tuning

Setup and Plots

Setup 1:

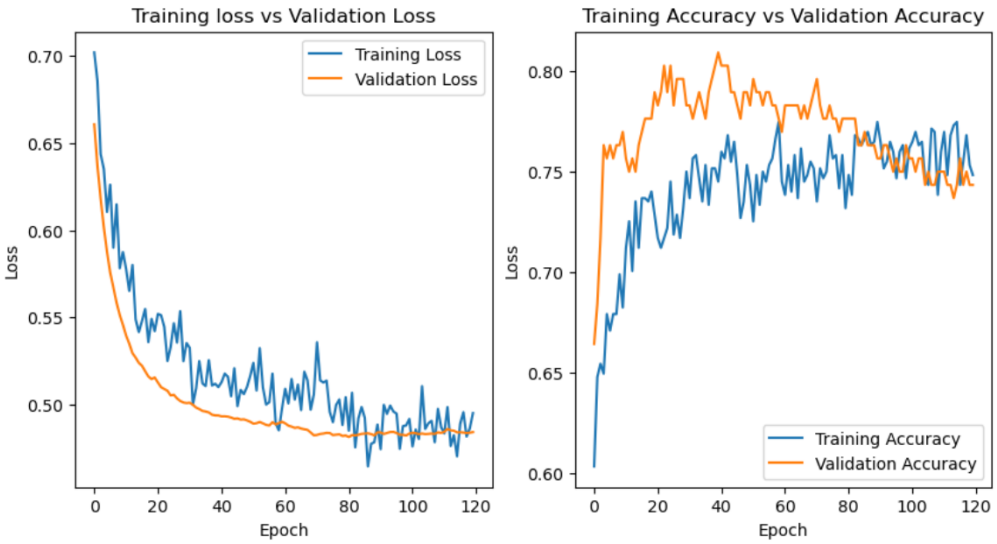
- Dropout Probability: 0.1
- Optimizer: Stochastic Gradient Descent (SGD)
- Activation Function: Rectified Linear Unit (ReLU)
- Initializer: Kaiming Normal

Metrics Observed:

- Training and Validation time (in seconds): 6.05
- Testing Loss: 0.484
- Test Accuracy: 74.34%

We observe that the validation loss is proportionately decreasing with the decrease with training loss, as number of epochs increase. The validation loss at the end of 120 epochs is 0.48. The training accuracy increases greatly till 40th epoch, and then doesn't show much improvement for the rest of the epochs. Coming to the validation accuracy, we see that it reaches its highest at around 40 epochs but then it starts decreasing and settles at 74.34% after 120 epochs. The training and validation time taken by this model is 6.05 seconds.

```
Training and Validation Time (in seconds) :  6.0535643100738525
Testing loss :  0.48428346451960114
Test Accuracy = 74.34
```



Setup 2:

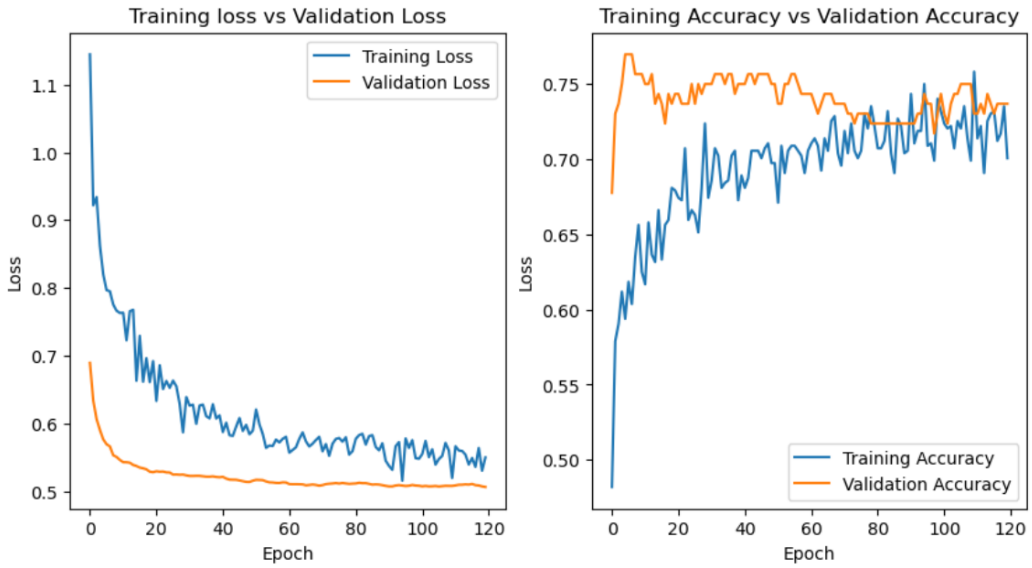
- Dropout Probability: 0.3
- Optimizer: Stochastic Gradient Descent (SGD)
- Activation Function: Rectified Linear Unit (ReLU)
- Initializer: Kaiming Normal

Metrics Observed:

- Training and Validation time (in seconds): 5.79
- Testing Loss: 0.506
- Test Accuracy: 73.68%

We observe that the validation loss starts at 0.7 and is proportionately decreasing with the decrease with training loss, as number of epochs increase. The validation loss at the end of 120 epochs is 0.5. The training accuracy increases gradually till the end of epochs. Coming to the validation accuracy, we see that it reaches its highest at around 10 epochs but then it starts decreasing and settles at 73.68% after 120 epochs. The training and validation time taken by this model is 5.79 seconds.

```
Training and Validation Time (in seconds) :  5.791850328445435
Testing loss :  0.5066452967493158
Test Accuracy = 73.68
```



Setup 3:

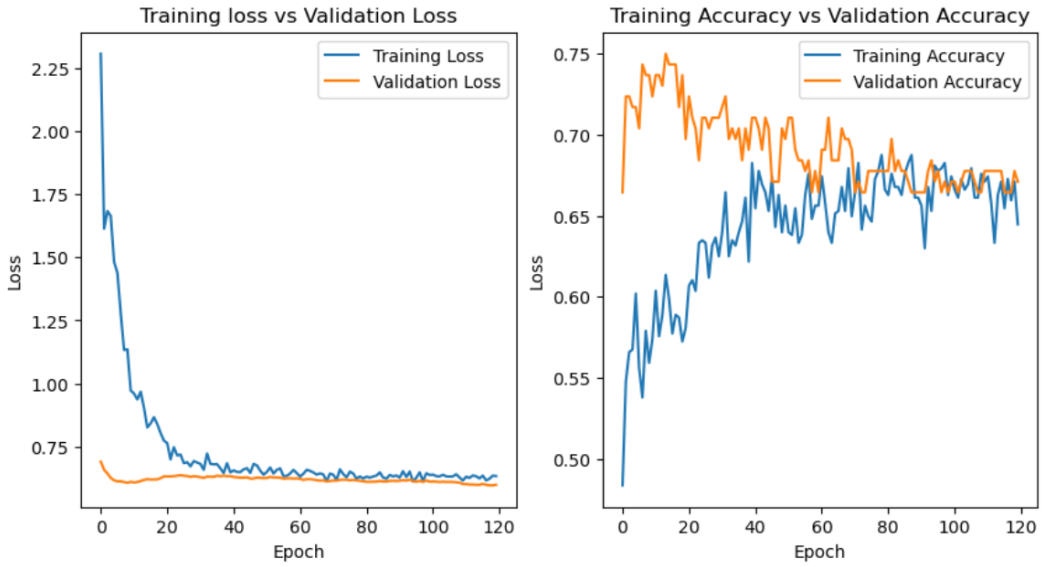
- Dropout Probability: 0.6
- Optimizer: Stochastic Gradient Descent (SGD)
- Activation Function: Rectified Linear Unit (ReLU)
- Initializer: Kaiming Normal

Metrics Observed:

- Training and Validation time (in seconds): 5.92
- Testing Loss: 0.597
- Test Accuracy: 67.11%

We observe that the training loss greatly decreases from 2.26 to 0.75 after 30 epochs and validation loss starts at 0.75 and decreases slowly as number of epochs increase. The validation loss at the end of 120 epochs is 0.59. The training accuracy increases gradually till the end of epochs. Coming to the validation accuracy, we see that it reaches its highest at around 15 epochs but then it starts decreasing and settles at 67.11% after 120 epochs. The training and validation time taken by this model is 5.92 seconds.

```
Training and Validation Time (in seconds) :  5.9274866580963135
Testing loss :  0.5977856391354611
Test Accuracy = 67.11
```



Dropout Tuning						
Hyperparameter	**Setup 1**	**Test Accuracy**	**Setup 2**	**Test Accuracy**	**Setup 3**	**Test Accuracy**
Dropout	0.1	74.34	0.3	73.68	0.6	67.11
Optimizer	SGD		SGD		SGD	
Activation Function	ReLU		ReLU		ReLU	
Initializer	Kaiming Normal		Kaiming Normal		Kaiming Normal	

	Hyperparameter	Setup1	Accuracy1	Setup2	Accuracy2	Setup3	Accuracy3
0	Dropout	0.1	74.34	0.3	73.68	0.6	67.11

Analysis

During training, Dropout means randomly zeroing some of the elements of the input tensor with probability p using samples from a Bernoulli distribution. Each channel will be zeroed out independently on every forward call.
This has proven to be an effective technique for regularization and preventing the co-adaptation of neurons. We have tried three setups with different dropout probabilities (0.1, 0.3, 0.6).
We observe that as dropout probability increases from 0.1 to 0.6 while keeping other parameters constant, the test accuracy has reduced from 74.34% to 67.11%, i.e., 7.2% reduction in the test accuracy. We also observe that the testing loss has increased from 0.48 to 0.597 as we increase the dropout probability from 0.1 to 0.6. Hence, for the given dataset and considered hyperparameters, we can conclude that test loss is directly proportional and test accuracy is indirectly proportional to dropout probability.

Initializer Tuning

Setup and Plots

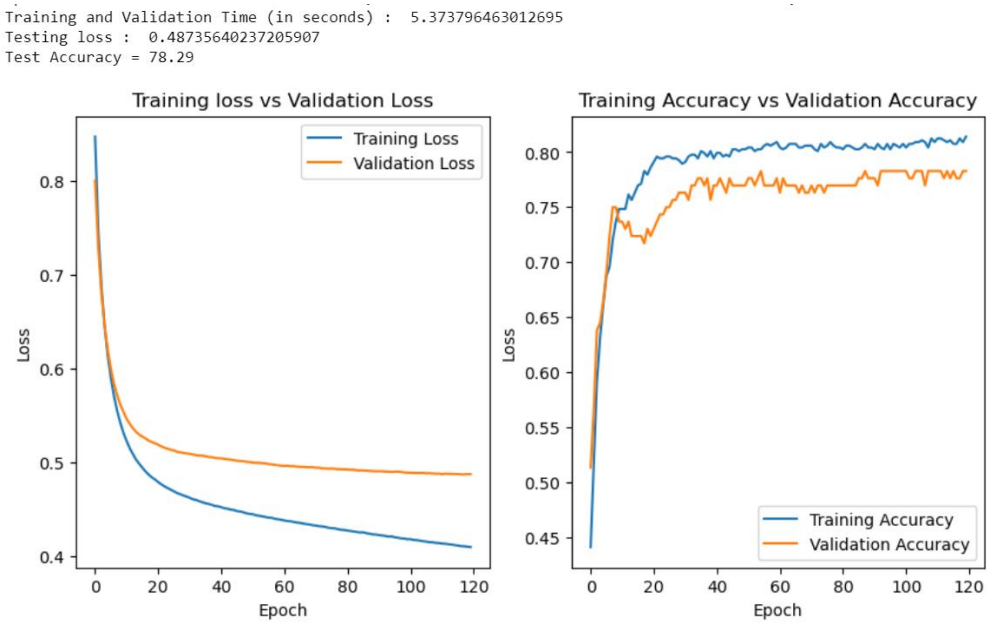
Setup 1:

- Dropout Probability: 0
- Optimizer: Stochastic Gradient Descent (SGD)
- Activation Function: Rectified Linear Unit (ReLU)
- Initializer: Kaiming Uniform

Metrics Observed:

- Training and Validation time (in seconds): 5.37
- Testing Loss: 0.487
- Test Accuracy: 78.29%

We observe that the training loss greatly decreases from 0.85 to 0.5 after 20 epochs and validation loss starts at 0.85 and decreases equally with training loss till 15 epochs and then almost is stable. The validation loss at the end of 120 epochs is 0.48. The training accuracy increases gradually till the end of epochs. Coming to the validation accuracy, we see that it also increases proportionately to the training accuracy and settles at 78.29% after 120 epochs. The training and validation time taken by this model is 5.37 seconds.



Setup 2:

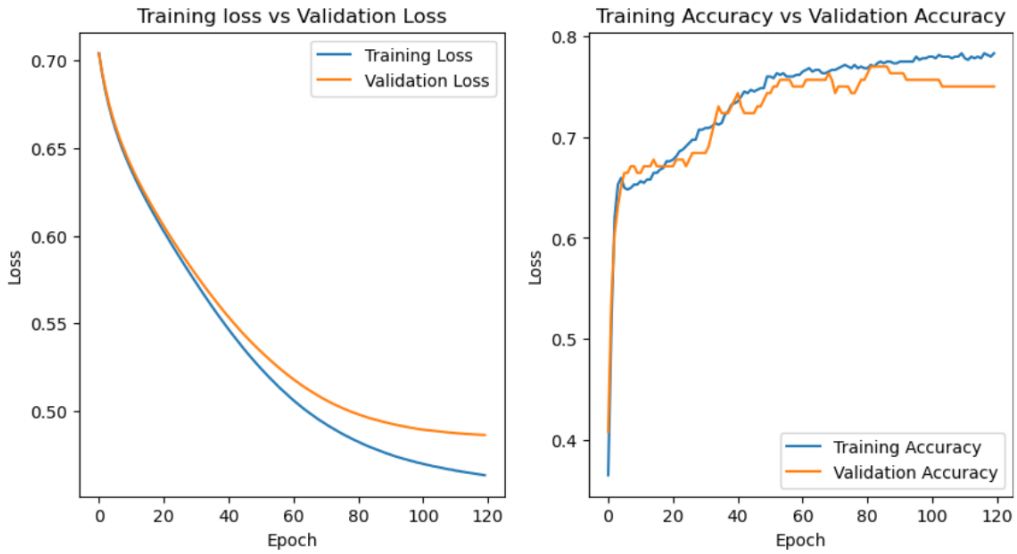
- Dropout Probability: 0
- Optimizer: Stochastic Gradient Descent (SGD)
- Activation Function: Rectified Linear Unit (ReLU)
- Initializer: Xavier Normal

Metrics Observed:

- Training and Validation time (in seconds): 5.08
- Testing Loss: 0.486
- Test Accuracy: 75%

We observe that the training loss and validation loss decrease almost equally as the number of epochs increase. The validation loss at the end of 120 epochs is 0.48. The training accuracy increases gradually till the end of epochs. Coming to the validation accuracy, we see that it also increases proportionately to the training accuracy and settles at 75% after 120 epochs. The training and validation time taken by this model is 5.08 seconds.

```
Training and Validation Time (in seconds) :  5.085077285766602
Testing loss :  0.48640379309654236
Test Accuracy = 75.00
```



Setup 3:

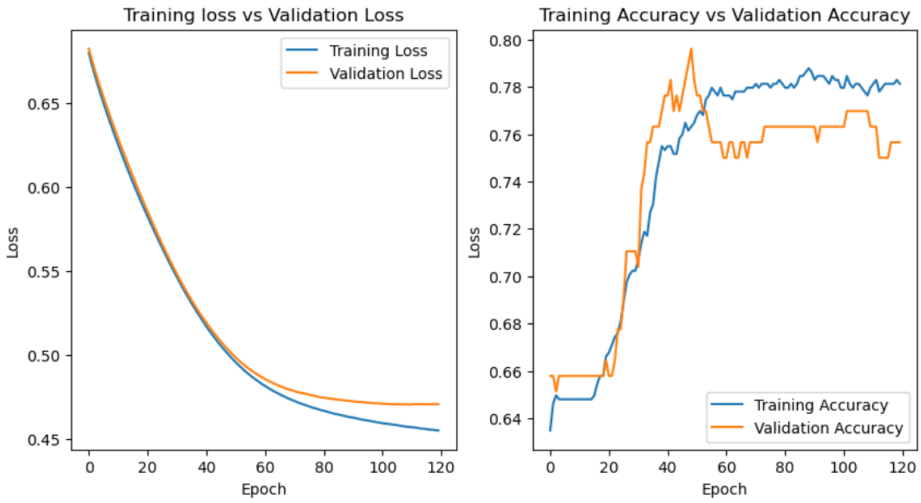
- Dropout Probability: 0
- Optimizer: Stochastic Gradient Descent (SGD)
- Activation Function: Rectified Linear Unit (ReLU)
- Initializer: Xavier Uniform

Metrics Observed:

- Training and Validation time (in seconds): 5.17
- Testing Loss: 0.47
- Test Accuracy: 75.66%

We observe that the training loss and validation loss decrease almost equally as the number of epochs increase. The validation loss at the end of 120 epochs is 0.47. The training accuracy increases gradually till the end of epochs. Coming to the validation accuracy, we see that it reaches its highest after 50 epochs at ~ 80% but then decreases and settles at 75.66% at the end of 120 epochs. The training and validation time taken by this model is 5.17 seconds.

```
Training and Validation Time (in seconds) :  5.174098968505859
Testing loss :  0.47058209306315374
Test Accuracy = 75.66
```



Initializer Tuning						
Hyperparameter	**Setup 1**	**Test Accuracy**	**Setup 2**	**Test Accuracy**	**Setup 3**	**Test Accuracy**
Dropout	0	78.29	0	75.0	0	75.66
Optimizer	SGD		SGD		SGD	
Activation Function	ReLU		ReLU		ReLU	
Initializer	Kaiming Uniform		Xavier Normal		Xavier Uniform	

	Hyperparameter	Setup1	Accuracy1	Setup2	Accuracy2	Setup3	Accuracy3
0	Initializer	Kaiming Uniform	78.29	Xavier Normal	75.0	Xavier Uniform	75.66

Analysis

Initializer specifies the method of setting the initial weights that define the starting point of the Neural Network. We have tried the below three initializers for model setup –

- Kaiming Uniform: This initialization method takes into account the non-linearity of the activation functions, such as ReLU. The weights are initialized such that they belong to a uniform distribution in range - sqrt(6/f_in + f_out) to + sqrt(6/f_in + f_out)
- Xavier Normal: In this method, the weights are initialized such that they belong to a normal distribution with zero mean and standard deviation of sqrt(2/f_in + f_out)
- Xavier Uniform: In this method, the weights are initialized such that they belong to a uniform distribution in range - sqrt(6/f_in + f_out) to + sqrt(6/f_in + f_out). Generally, it is observed that Xavier Uniform method works well with Sigmoid and Tanh activation functions

We observe that all the three methods of initialization give us very similar test loss values in the range 0.47 to 0.485. Xavier Normal method has performed worst of all in terms of test accuracy and test loss even though it has taken the least training time. Kaiming Uniform method has produced the highest accuracy of 78.29% and Xavier Uniform has produced the least testing loss 0.47.

Activation Function Tuning

Setup and Plots

Setup 1:

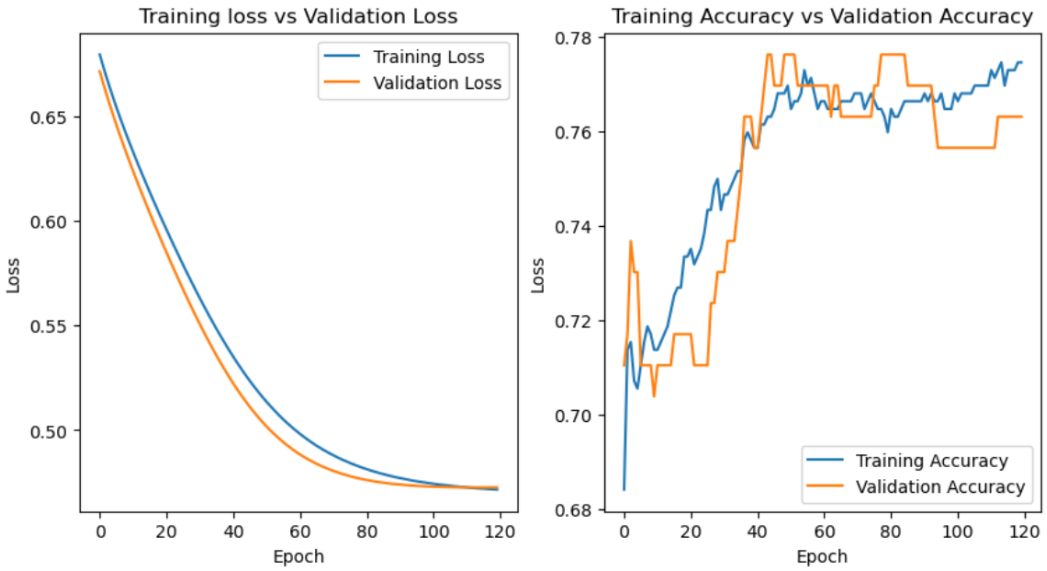
- Dropout Probability: 0
- Optimizer: Stochastic Gradient Descent (SGD)
- Activation Function: Tanh
- Initializer: Kaiming Normal

Metrics Observed:

- Training and Validation time (in seconds): 5.79
- Testing Loss: 0.47
- Test Accuracy: 76.32%

We observe that the training loss and validation loss decrease almost equally as the number of epochs increase. The validation loss at the end of 120 epochs is 0.47. The training accuracy increases gradually till 60 epochs, then dips a bit at 80 epochs and increases again till the end of epochs. Coming to the validation accuracy, we see that it reaches its highest after 50 epochs at ~ 78% but then decreases and settles at 76.32% at the end of 120 epochs. The training and validation time taken by this model is 5.79 seconds.

```
Training and Validation Time (in seconds) :  5.791714906692505
Testing loss :  0.4726148279089677
Test Accuracy = 76.32
```



Setup 2:

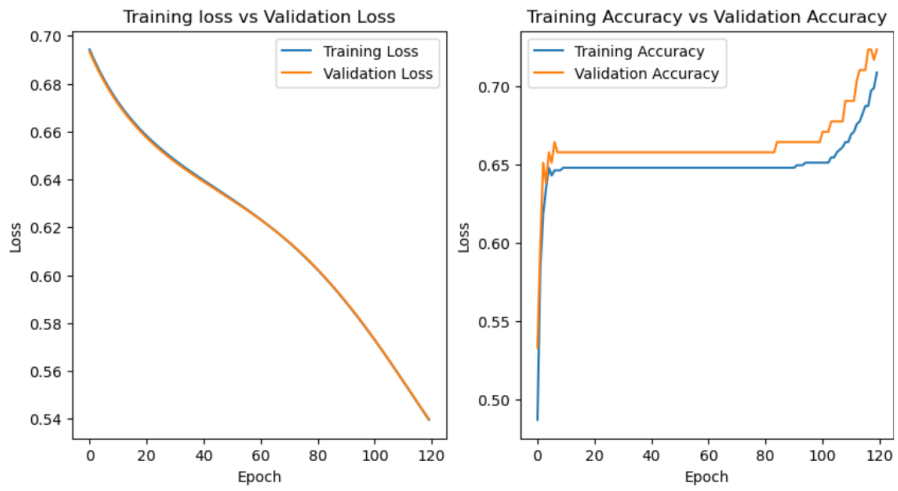
- Dropout Probability: 0
- Optimizer: Stochastic Gradient Descent (SGD)
- Activation Function: Leaky ReLU
- Initializer: Kaiming Normal

Metrics Observed:

- Training and Validation time (in seconds): 5.44
- Testing Loss: 0.53
- Test Accuracy: 72.37%

We observe that the training loss and validation loss decrease almost equally as the number of epochs increase. The validation loss at the end of 120 epochs is 0.539. The training accuracy increases gradually till 10 epochs, then is stable till 100 epochs and then increases till the end of 12 epochs. A similar pattern can be observed for validation accuracy and at 120 epochs it is 72.37%. The training and validation time taken by this model is 5.44 seconds.

```
Training and Validation Time (in seconds) :  5.44028902053833
Testing loss :  0.5398970497281927
Test Accuracy = 72.37
```



Setup 3:

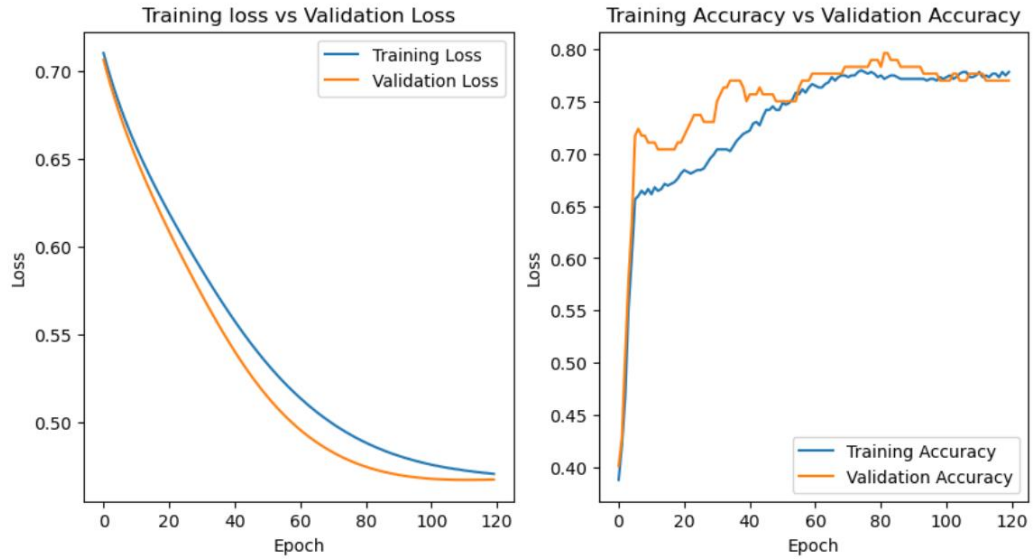
- Dropout Probability: 0
- Optimizer: Stochastic Gradient Descent (SGD)
- Activation Function: Exponential Linear Unit (ELU)
- Initializer: Kaiming Normal

Metrics Observed:

- Training and Validation time (in seconds): 5.77
- Testing Loss: 0.46
- Test Accuracy: 76.97%

We observe that the training loss and validation loss decrease almost equally as the number of epochs increase. The validation loss at the end of 120 epochs is 0.46. The training accuracy shoots up in the first 10 epochs, then gradually increases to around 75% at 80 epochs and maintains the same value till the end of 120 epochs. Coming to the validation accuracy, we see that it follows a very similar pattern as training accuracy and settles at 76.97% at the end of 120 epochs. The training and validation time taken by this model is 5.77 seconds.

```
Training and Validation Time (in seconds) :  5.77984356880188
Testing loss :  0.46753974964744166
Test Accuracy = 76.97
```



Activation Function Tuning						
Hyperparameter	**Setup 1**	**Test Accuracy**	**Setup 2**	**Test Accuracy**	**Setup 3**	**Test Accuracy**
Dropout	0	76.32	0	72.37	0	76.97
Optimizer	SGD		SGD		SGD	
Activation Function	Tanh		Leaky ReLU		ELU	
Initializer	Kaiming Normal		Kaiming Normal		Kaiming Normal	

	Hyperparameter	Setup1	Accuracy1	Setup2	Accuracy2	Setup3	Accuracy3
0	Activation Function	Tanh	76.32	Leaky ReLU	72.37	ELU	76.97

Analysis

Activation function decides whether a neuron in the NN should be activated/considered for predicting output or not using simple mathematical operations. For testing, we have used three different activation functions as below :

- Tanh – Tanh is generally used for solving binary classification problems. Initially, it was preferred over sigmoid function as it gave better performance for multi-layered networks but it didn't overcome the vanishing gradient problem that sigmoid functions had.

$$F(x) = e^{x} – e^{-x}/e^{x} + e^{-x}$$

- Leaky ReLU – The Rectified Linear Unit (ReLU) activation function makes the gradient zero for negative inputs. This creates the dying neurons problem. Leaky ReLU tries to solve that problem by multiplying the negative values with 0.01 instead of making them zero.

$$f(x) = \max(0.01*x , x)$$

- ELU – Exponential Linear Unit is similar to Leaky ReLU as it also tries to solve the dying neurons problem. However, unlike Leaky ReLU, for a negative input x, ELU will compute $e^{x} – 1$.

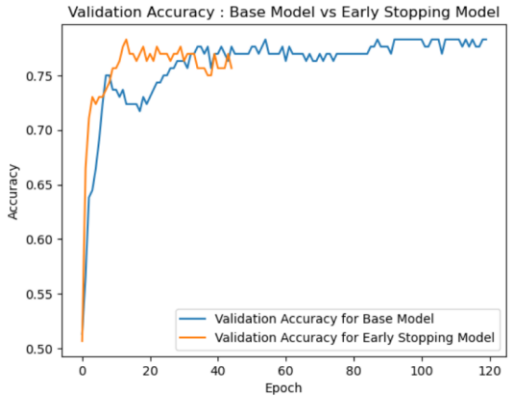
$$F(x) = e^{x} – 1 \text{ if } x<0 \text{ and } x \text{ if } x>0$$

Coming to our test results, we see that Leaky ReLU was the fastest in terms to training and validation time, but performed worst in minimizing test loss and maximizing the test accuracy. As expected, ELU produces better results compared to Leaky ReLU with least test loss among the three functions 0.46 and best test accuracy of 76.97%.

Improvement Methods

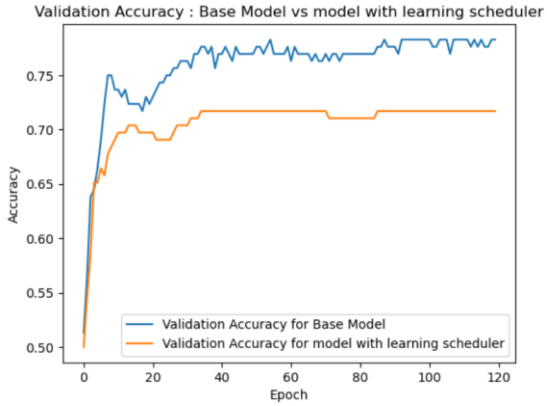
Early Stopping

Early stopping is a method in Neural Networks that stops training the model when there is no improvement in either validation loss or validation accuracy in 3 or more consecutive epochs. It is used to avoid overfitting of the model without compromising on the accuracy. We have used Early stopping method on our base model to improve the training and validation time from 5.37 seconds to 2.09 seconds. We can observe in the graph below where we compare test accuracy from base model without early stopping versus base model with early stopping. We can see that the latter ran only for 44 out of 120 epochs and still achieved a decent 75% accuracy.



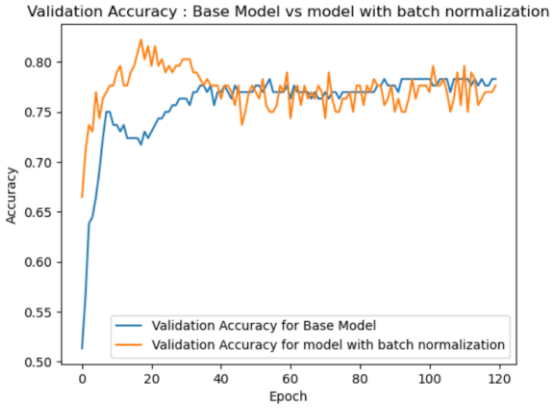
Learning rate Scheduler

Learning rate scheduler is used to adjust the learning rate between epochs or iterations as the training progresses. We have used both StepLR and ExponentialLR to test the learning rate scheduler and chose the latter due to improved accuracy over the former. The training time has improved from 5.37 seconds to 5.1 seconds. The accuracy of this model is 71.71%, which is lesser than the base model.



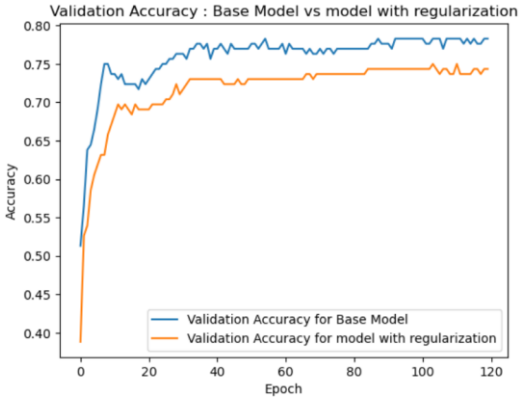
Batch Normalization

Batch Normalization is a method to make the training process faster and more stable by normalizing the layer inputs i.e., re-centering and re-scaling. It reduces the dependence of gradient on the scale of the variables. Using batch normalization can often increase training speed and improve test accuracy. We observed that adding batch normalization to our base model has increased the training and validation time to ~7 seconds and has achieved a test accuracy of 77.63% which is very close to the base model. The testing loss also has been reduced from 0.48 to 0.44. Below is a graph comparing the test accuracy between base model with and without batch normalization.



Regularization

We used L2 regularization on the base model that penalizes the large weight values. In this method, we used lambda value as 0.4 and achieved a very low training and validation time as 4.03 seconds, which is the fastest amongst the 4 improvement methods we've tested. The test accuracy is 74.34%



Part III: Implementing and Improving AlexNet

Description

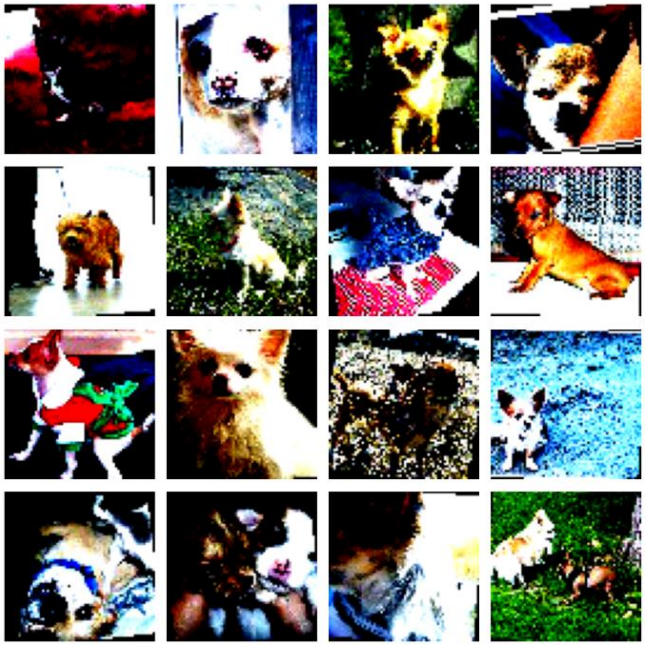
The dataset "cnn_dataset.zip" consists of three folders – dogs, food, and vehicles. Each class consists of 10,000 RGB images of size 64x64 pixels. So, there are a total of 30,000 images. The mean of the dataset is [0.5047, 0.4501, 0.3840] and standard deviation is [0.2386, 0.2384, 0.2404]. We have three values each because we have 3 channels for the images.

Visualizations

As part of data pre-processing, we perform random horizontal flip, rotate the image randomly, and normalize it using mean and standard deviation. Below are some of the visualizations of the pre-processed data.

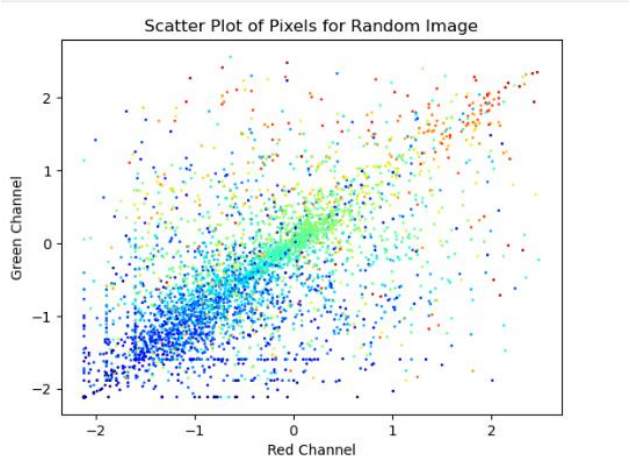
Visualization 1: Print random images from a class

Below is a 4x4 grid of images randomly selected from the class – dogs. As part of data pre-processing, some of these images are flipped horizontally, and some of them are rotated.



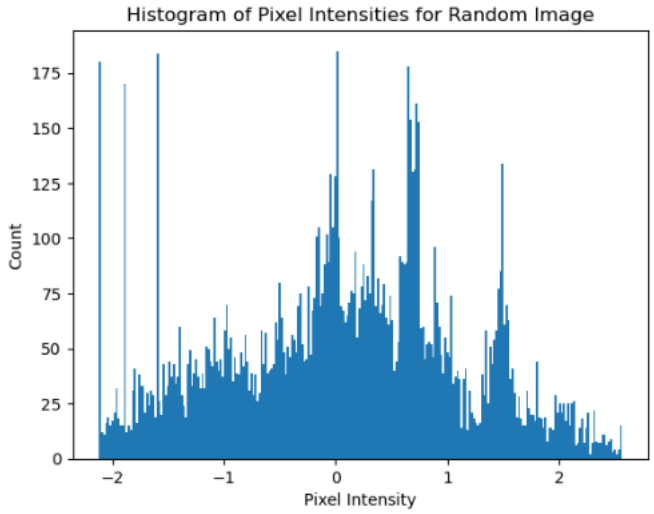
Visualization 2: Scatterplot of pixels for a random image

We randomly select an image from the dataset and try to visualize a scatterplot of the pixels that formed the image. As we can see below, the graph is plotted between the red channel and the green channel.



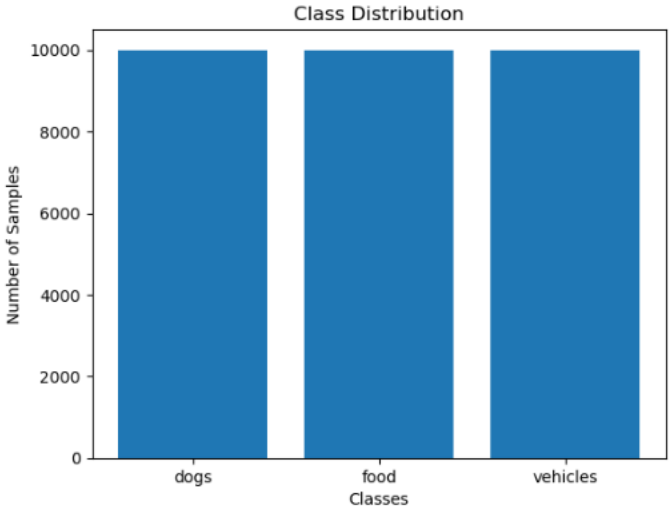
Visualization 3: Histogram of pixel intensities for a random image

As part of this visualization, we randomly select an image from the dataset and plot a histogram of pixel intensity frequencies as below.



Visualization 4: Number of samples per class

We plot a bar graph that represents the count of samples per each class in the dataset. As we see, there are three classes – dogs, food, and vehicles. Each class exactly has 10,000 images.



Build and train AlexNet CNN

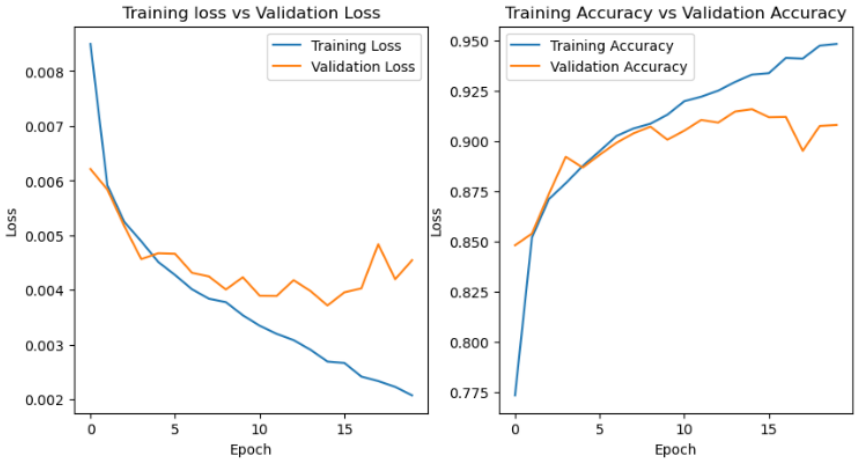
We implement AlexNet CNN in pytorch, modifying only the input and output layers of the model as the original AlexNet accepts 224x224 sized images and outputs 1000 classes. As discussed, our images are of size 64x64 and we need only three output classes. We split the dataset into 80% train set and 20% test set. We have used Cross Entropy Loss as the loss function, Adam optimizer and trained the CNN for 20 epochs.

Hyperparameters –

```
Batch size = 64
```

```
criterion = nn.CrossEntropyLoss()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model1 = AlexNet(num_classes=3).to(device)
optimizer = optim.Adam(model1.parameters(),lr=0.001)
num_epochs = 20
```

The training and validation have taken 1080 seconds and we achieved 90.68% test accuracy. As seen below, the training loss has steadily reduced as epochs increased. The validation loss has reduced from 0.006 to 0.004. As epochs increased, we see that the validation accuracy has increased from 84.8% to ~91% at the end of 20 epochs.



Change the learning rate to 0.001

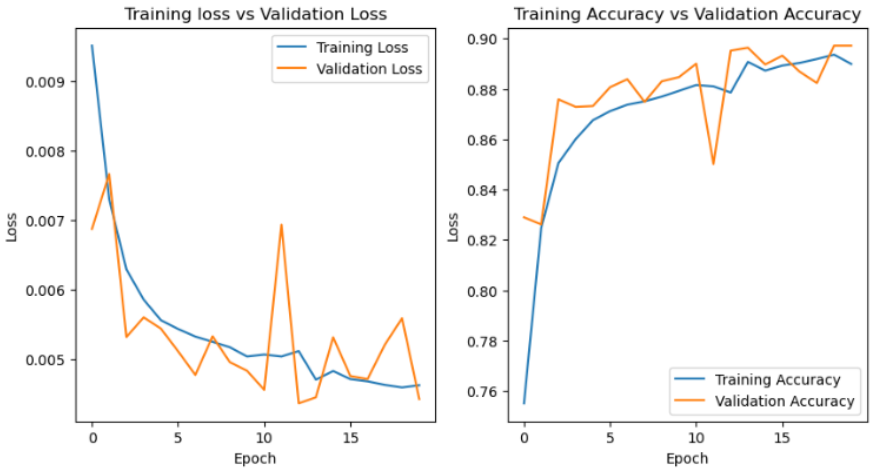
For this model, we have used Cross Entropy Loss as the loss function, Adam optimizer and trained the CNN for 20 epochs. We increased the learning rate from 0.0001 to 0.001 for this model.

Hyperparameters –

```
Batch size = 64
```

```
criterion = nn.CrossEntropyLoss()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model1 = AlexNet(num_classes=3).to(device)
optimizer = optim.Adam(model1.parameters(),lr=0.001)
num_epochs = 20
```

The training and validation have taken 1079 seconds and we achieved 89.91% test accuracy. As seen below, the training loss has steadily reduced as epochs increased. The validation loss has reduced from 0.006 to 0.004. As epochs increased, we see that the validation accuracy has increased from 82.9% to 89.9% at the end of 20 epochs.



Add dropout probability to the model

For this model, we have used Cross Entropy Loss as the loss function, Adam optimizer and trained the CNN for 20 epochs. We modify the original AlexNet to define a new CNN named D_AlexNet with a dropout probability 0.4

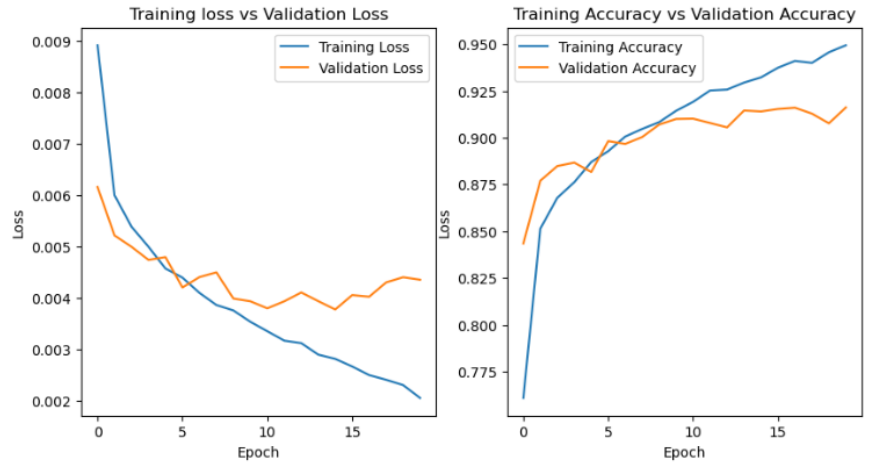
Hyperparameters –

```
Batch size = 64

Dropout Probability = 0.4

criterion = nn.CrossEntropyLoss()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model2 = D_AlexNet(num_classes=3).to(device)
optimizer = optim.Adam(model2.parameters(),lr=0.0001)
num_epochs = 20
```

The training and validation have taken 1055 seconds, which is lesser than previous models. We achieved 91.5% test accuracy. As seen below, the training loss has steadily reduced as epochs increased. The validation loss has reduced from 0.006 to 0.004. As epochs increased, we see that the validation accuracy has increased from 84.3% to 91.6% at the end of 20 epochs.



Add Batch Normalization and L2 Regularization

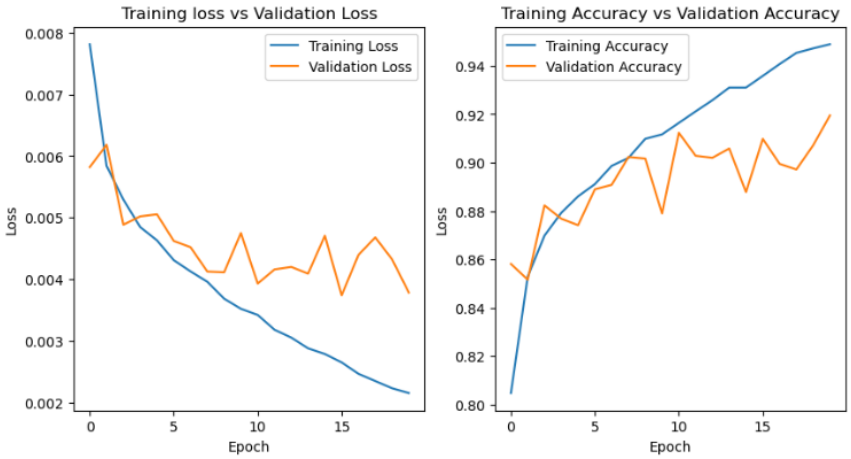
For this model, we have used Cross Entropy Loss as the loss function, Adam optimizer and trained the CNN for 20 epochs. We modify the original model and define BL2AlexNet such that it has a batch normalization layer after each convolution layer and it has L2 regularization penalty

Hyperparameters –

```
Batch size = 64
```

```
criterion = nn.CrossEntropyLoss()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model3 = BL2AlexNet(num_classes=3).to(device)
optimizer = optim.Adam(model3.parameters(),lr=0.0001, weight_decay=0.0001)
num_epochs = 20
```

The training and validation have taken 1115 seconds. We achieved 91.56% test accuracy. As seen below, the training loss has steadily reduced as epochs increased. The validation loss has reduced from 0.005 to 0.003, which is lesser than the previous models. As epochs increased, we see that the validation accuracy has increased from 85.8% to 91.95% at the end of 20 epochs.



Early Stopping Model

To the above model, we add the early stopping condition with a patience count 3. That is, whenever the validation loss increases compared to the previous value, the model patiently waits 3 times and then halts training.

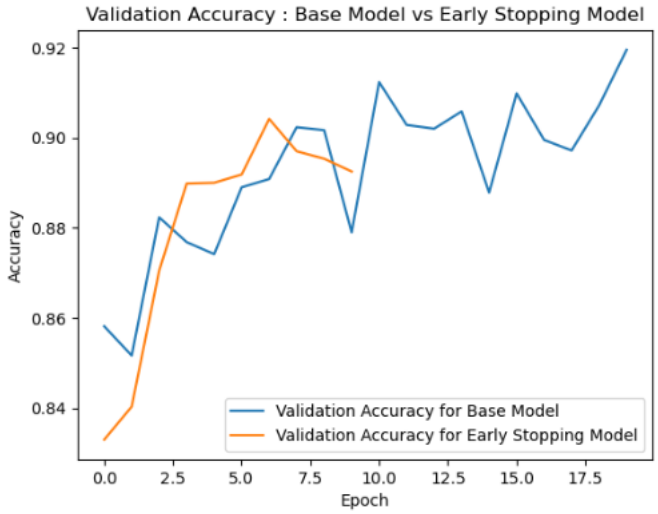
Hyperparameters –

```
Batch size = 64
```

```
Patience = 3
```

```
criterion = nn.CrossEntropyLoss()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model4 = BL2AlexNet(num_classes=3).to(device)
l2_reg = 0.0001
optimizer = optim.Adam(model4.parameters(),lr=0.0001, weight_decay=l2_reg)
num_epochs = 20
```

The training and validation have taken 627 seconds. We achieved 89.68% test accuracy. As epochs increased, we see that the validation accuracy has increased from 83.3% to 89.25% at the end of 9 epochs. Below the line plot comparing the validation accuracy of the previous model to the validation accuracy of the early stopping model.



List of Test Accuracies

Below is the list of test accuracies achieved by the 5 model we have discussed above. We see that Model 3 (AlexNet + Batch normalization + L2 Regularization) has obtained the highest test accuracy and will be considered as the base model.

```
Original AlexNet test acc : 90.683%
Model 1 (0.01 LR) test acc : 89.917%
Model 2 (Dropout 0.4) test acc : 91.517%
Model 3 (Batch Norm + L2 Reg) test acc : 91.567%
Model 4 (Early Stopping) test acc : 89.683%
```

Part IV: Optimizing CNN + Data Augmentation

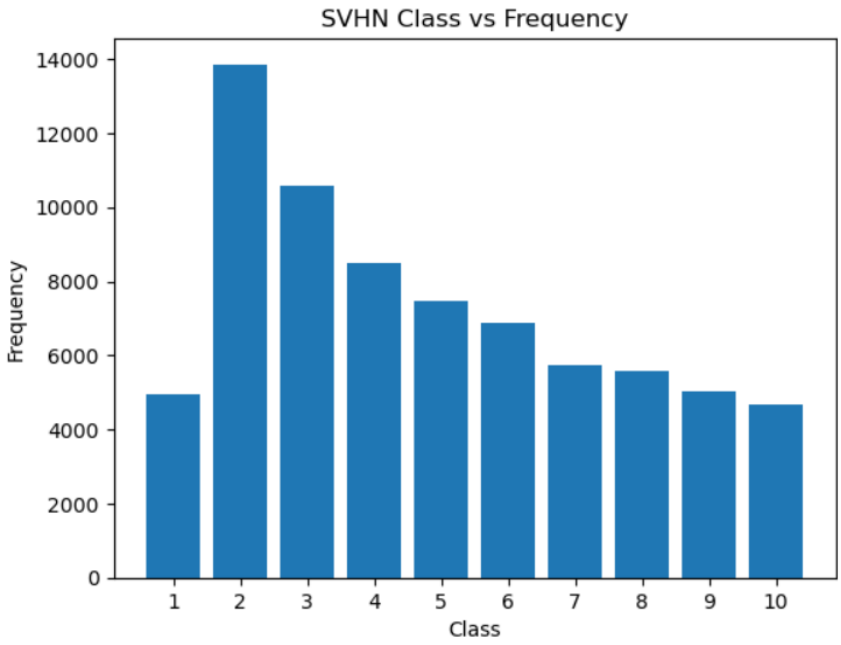
Description

The dataset we use for this part of the assignment is Google Street View House Number (SVHN) data. It consists of two folders – train and test. The train folder consists of 73,257 RGB 32x32 images and the test folder consists of 26,032 images. The mean of the train set is [-0.0776, -0.0729, -0.0646] and the standard deviation is [1.0674, 1.0480, 1.0278]. We have three values each as the images have 3 channels.

Visualizations

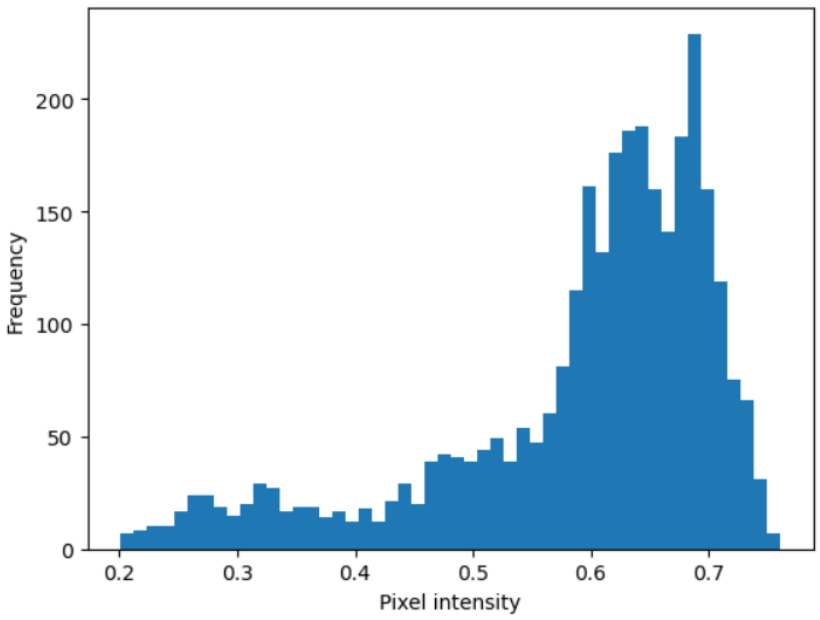
Visualization 1: Number of samples per class

We plot a bar graph that represents the count of samples per each class in the dataset. Since, this is a house numbers dataset, there are 10 classes – one class for each number from 0 to 9.



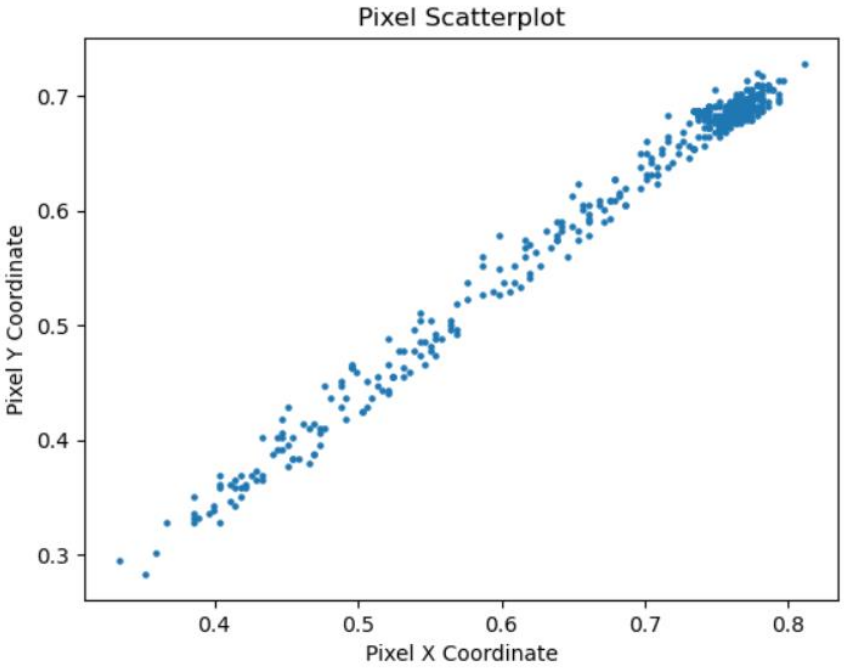
Visualization 2: Histogram of pixel intensities

As part of this visualization, we randomly select an image from the dataset and plot a histogram of pixel intensity frequencies as below.



Visualization 3: Scatter plot of pixels

We randomly select an image from the dataset and try to visualize a scatterplot of the pixels that formed the image. As we can see below, the graph is plotted between the x-coordinate and y-coordinate of the pixel.



Visualization 4: Select a random image and display along with its class

We randomly select 5 images from the dataset and print them along with their class label as part of this visualization.

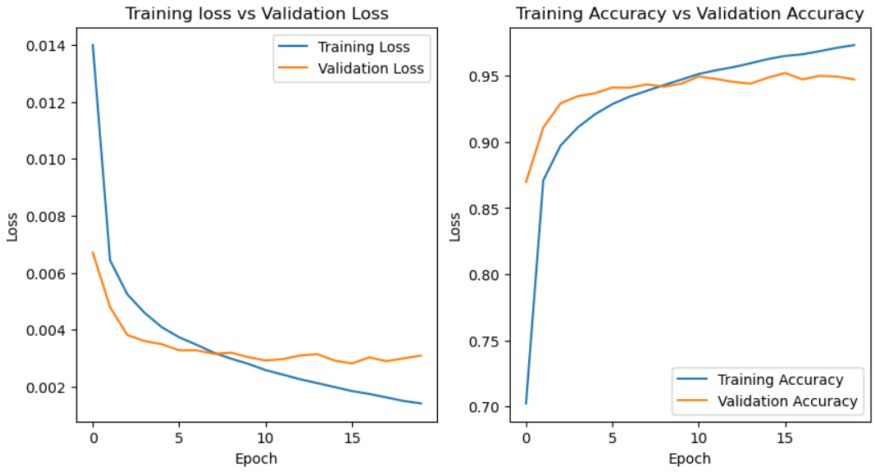
Modify the Base Model

As discussed previously, we chose the model 3 as our base model. However, model 3 is built to accept 3 channel 64x64 images and output 3 classes. However, SVHN dataset consists of 3 channel 32x32 sized images and 10 classes. So, we modify our model SVHN_AlexNet to accept 3 channel 32x32 images and output layer to give out 10 classes.

Hyperparameters –

```
criterion = nn.CrossEntropyLoss()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
svhn_model = SVHN_AlexNet(num_classes=10).to(device)
optimizer = optim.Adam(svhn_model.parameters(),lr=0.0001, weight_decay=0.0001)
num_epochs = 20
```

The training and validation have taken 978 seconds, which is considerably lower than previous training times. We achieved 94.72% test accuracy. As seen below, the training loss has steadily reduced as epochs increased. The validation loss has reduced from 0.006 to 0.003. As epochs increased, we see that the validation accuracy has increased from 86.9% to 94.7% at the end of 20 epochs.



Data Augmentation

Model 1: Random Rotation, Random Crop, and Normalization

We perform data augmentation on the training data and add it to the original training data to double the size of the dataset. In this model, we transform the data by adding a random rotation to the images and cropping them randomly. Then, we normalize the dataset with the mean and standard deviation of the original model.

```
aug_transform1 = transforms.Compose([
    transforms.RandomRotation(10),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize(mean=[-0.0776, -0.0729, -0.0646], std=[1.0674, 1.0480, 1.0278])
```

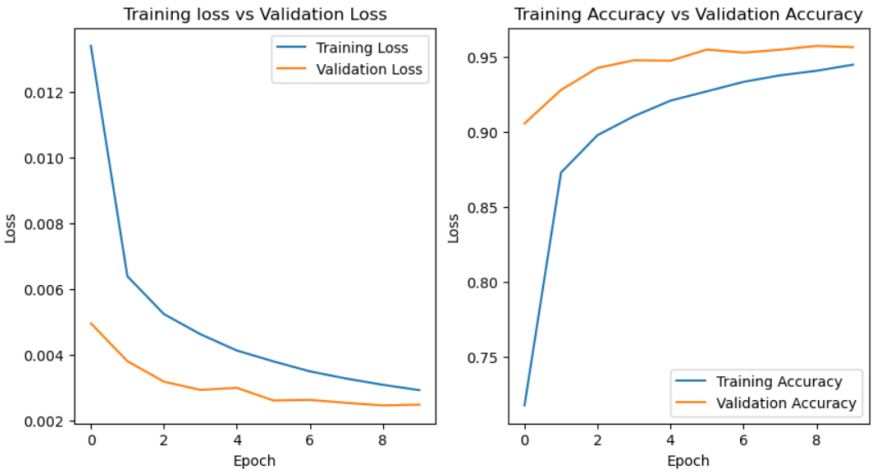
Once we transform the data and add it to the training set, we have below sizes.

```
Trainset length : 146514, Testset length : 26032
```

We then use the below hyperparameters and train the model.

```
criterion = nn.CrossEntropyLoss()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
svhn_model1 = SVHN_AlexNet(num_classes=10).to(device)
optimizer = optim.Adam(svhn_model1.parameters(),lr=0.0001, weight_decay=0.0001)
num_epochs = 10
Batch size = 64
```

The training and validation have taken 873 seconds, as we ran the model through 10 epochs. We achieved 95.65% test accuracy, which is higher than the base model. As seen below, the training loss has steadily reduced as epochs increased. The validation loss has reduced from 0.004 to 0.002, which is also lower compared to previous models. As epochs increased, we see that the validation accuracy has increased from 90.5% to 95.65% at the end of 10 epochs.



Model 2: Color Jitter, Random Horizontal Flip, and Normalization

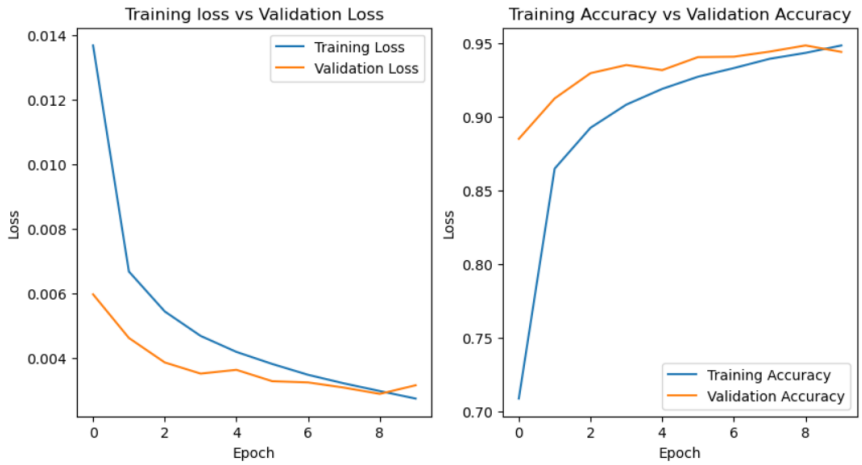
In this model, we transform the data by adding a color jitter by setting the brightness and contrast of the image. Also, we add a random horizontal flip to the images. Then, we normalize the dataset with the mean and standard deviation of the original model.

```
aug_transform2 = transforms.Compose([
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[-0.0776, -0.0729, -0.0646], std=[1.0674, 1.0480, 1.0278])
])
```

We then use the below hyperparameters and train the model.

```
criterion = nn.CrossEntropyLoss()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
svhn_model2 = SVHN_AlexNet(num_classes=10).to(device)
optimizer = optim.Adam(svhn_model2.parameters(),lr=0.0001, weight_decay=0.0001)
num_epochs = 10
Batch size = 64
```

The training and validation have taken 928 seconds for 10 epochs. We achieved 94.37% test accuracy, which is higher than the base model. As seen below, the training loss has steadily reduced as epochs increased. The validation loss has reduced from 0.005 to 0.003. As epochs increased, we see that the validation accuracy has increased from 88.4% to 94.38% at the end of 10 epochs.



Model 3: Random Rotation, Random Crop, Color Jitter, Random Horizontal Flip, Normalization

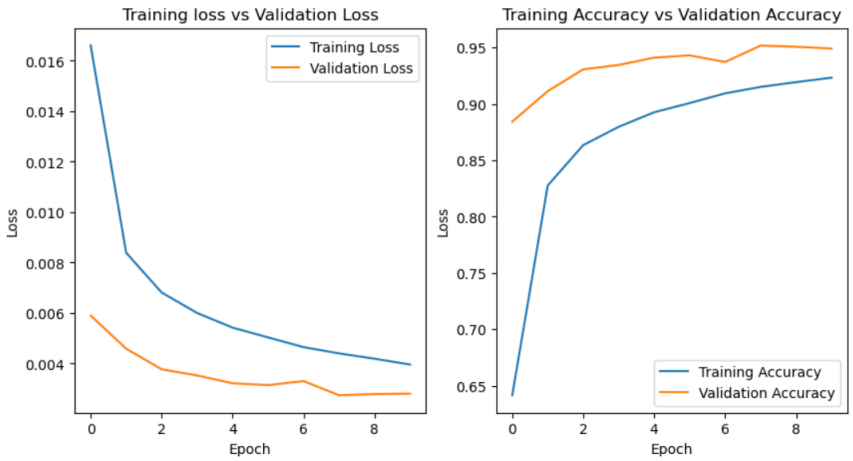
In this model, we try to combine the data augmentation techniques we have used thus far, all at a time. We transform the data by adding a Random rotation and Random Crop. Then, we add a color jitter by setting the brightness and contrast of the image. Also, we add a random horizontal flip to the images. Then, we normalize the dataset with the mean and standard deviation of the original model.

```
aug_transform3 = transforms.Compose([
    transforms.RandomRotation(10),
    transforms.RandomCrop(32, padding=4),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[-0.0776, -0.0729, -0.0646], std=[1.0674, 1.0480, 1.0278])
])
```

We then use the below Hyperparameters and train the model.

```
criterion = nn.CrossEntropyLoss()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
svhn_model3 = SVHN_AlexNet(num_classes=10).to(device)
optimizer = optim.Adam(svhn_model3.parameters(),lr=0.0001, weight_decay=0.0001)
num_epochs = 10
Batch size = 64
```

The training and validation have taken 992 seconds for 10 epochs. We achieved 94.9% test accuracy, which is higher than the base model. As seen below, the training loss has steadily reduced as epochs increased. The validation loss has reduced from 0.005 to 0.002. As epochs increased, we see that the validation accuracy has increased from 88.4% to 94.92% at the end of 10 epochs.



List of Test Accuracies

Below is the list of test accuracies achieved by the 4 models we have discussed above. We see that Model 1 (Random rotation + Random Crop + Normalization) has obtained the highest test accuracy.

```
Test acc without Data Aug : 94.73%
Model 1 Test Acc : 95.659%
Model 2 Test Acc : 94.38%
Model 3 Test Acc : 94.902%
```