Deep Learning PA3 Report

Sahith Bhamidipati

March 2022

1 Problem Statement

The purpose of this assignment is to build a model that classifies images of various objects as the correct object. For example, an image of an airplane should be correctly classified as an airplane, an image of a bird should be classified as a bird, and so on. The possible objects are an airplane, an automobile, a bird, a cat, a deer, a dog, a frog, a horse, a ship, and a truck. The final model that provides a solution to this problem is expected to classify these images with a reasonably high accuracy, such as in the 90%'s.

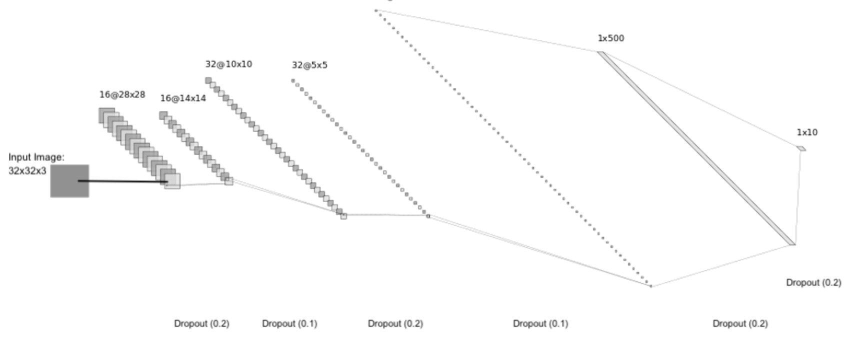
2 Model Description

The model built is a deep Convolutional Neural Network (CNN) model that contains three convolution layers each padded with ReLU activation and max pooling layers, as well as a flatten layer after the convolution layers, a fully connected layer, and the final output layer with a softmax output function. In addition, in between each of these layers is a dropout layer used only during training to help combat overfitting. The definitions for the ReLU and softmax functions are shown below:

$$ReLU(x) = max(0, x)$$

$$softmax = \sigma(z_i) = \frac{z_i}{\sum_{j=0}^{len(z)} z_j}$$

The first convolution layer contains 16 filters of $5 \times 5 \times 3$ size each. The convolution occurs with a stride of 1 and no zero padding is used, and the output runs through the ReLU activation function in the end. After an intermediate dropout layer with a drop rate of 0.2, max pooling is done in 2×2 slots of the data with a stride of 2. Another dropout layer is placed after the pooling, this time with a drop rate of 0.1 to avoid losing too much data. The next convolution layer contains 32 filters with the rest of the parameters being the same as the first layer. After this layer and another dropout layer of 0.2 drop rate comes another max pooling layer with a pooling area of 2×2 and a stride of 2, as well as another dropout layer with a drop rate of 0.1. The last convolution layer contains 64 filters of $3 \times 3 \times 3$ size each. The stride is still 1, along with the ReLU activation function and lack of zero padding. Another 0.2 drop rate dropout layer is added after this, then the output is flattened to begin official classification. After the flatten layer and another dropout layer of 0.1 drop rate, a fully connected layer of 500 nodes and ReLU activation is used. Then, after another dropout layer of 0.2 drop rate comes the final layer, which contains 10 nodes for the number of possible classes and a softmax output function. Some of the dropout layers were removed in the final version of the model due to experiments shown in the results section. The illustration below depicts the architecture described above of the final model:



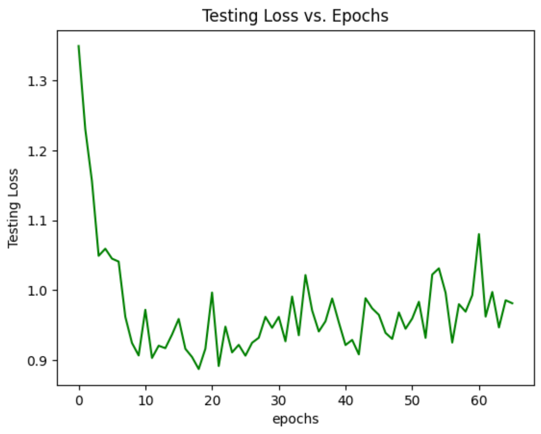
The weights associated with each of the convolution layers, the fully connected layer, and the output layer are all trained as part of the training process. This training is done using Tensorflow's built-in `fit` function, which is closely related to the `tf.keras.Model` prototype.

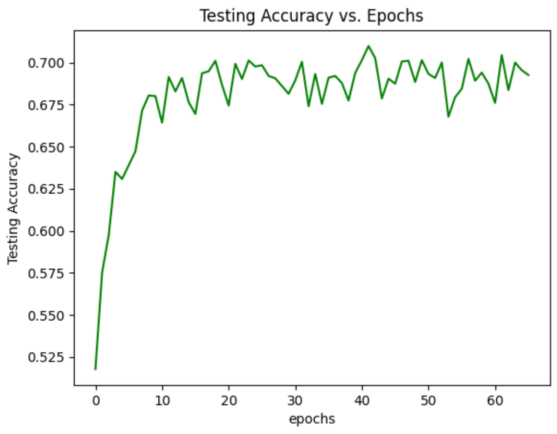
3 Results

To ensure that the model used for this problem was trained to its best potential, the model was trained with different settings for hyperparameters. The parameters tested were the type of optimizer used, the learning rate, and the usage of dropout layers. The score of each of the parameters were measured in terms of testing accuracy. Measuring in terms of training accuracy could lead to favoring parameters that increase the chance of overfitting during training.

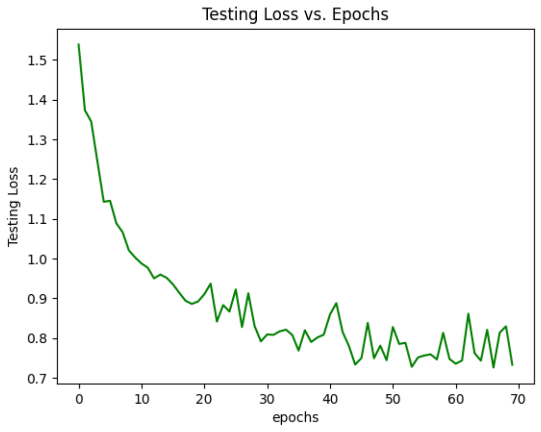
Firstly, the different optimizers tested were the Adam, Adamax, Adagrad, and Adadelta optimizers. This parameter was the first to be varied due to having a higher impact on the creation of the model. Below are the testing loss and accuracy plots for each of the optimizers:

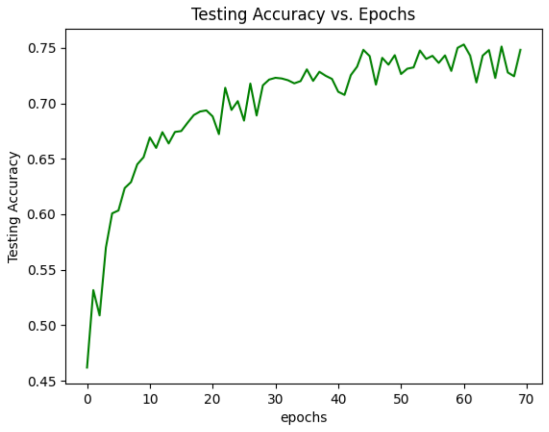
Adam optimizer:



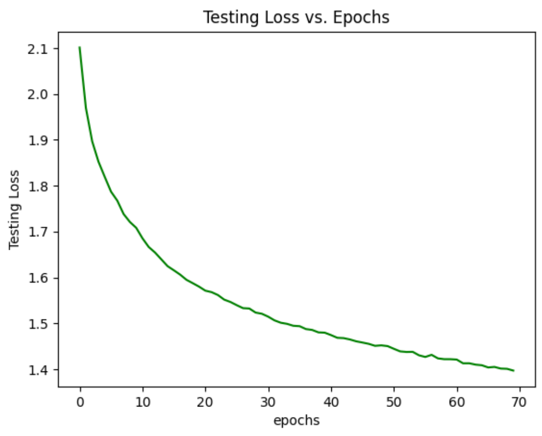


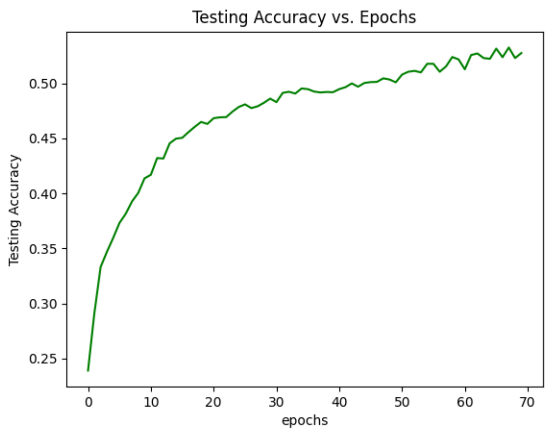
Adamax optimizer:

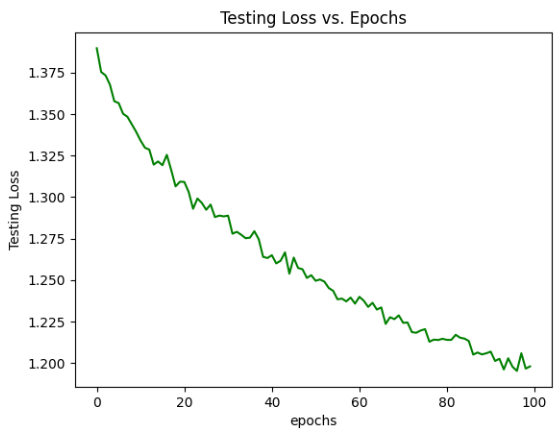


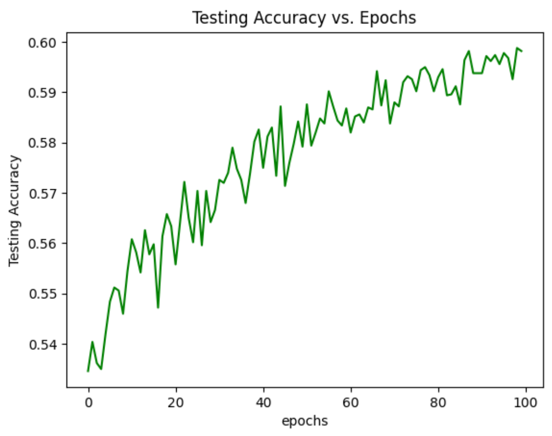


Adagrad optimizer (tested multiple times due to taking more iterations to reach convergence):

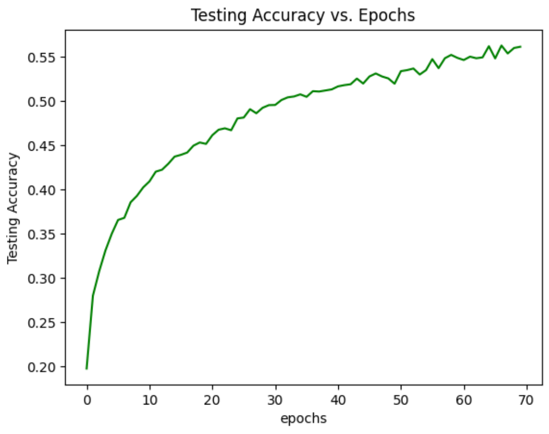


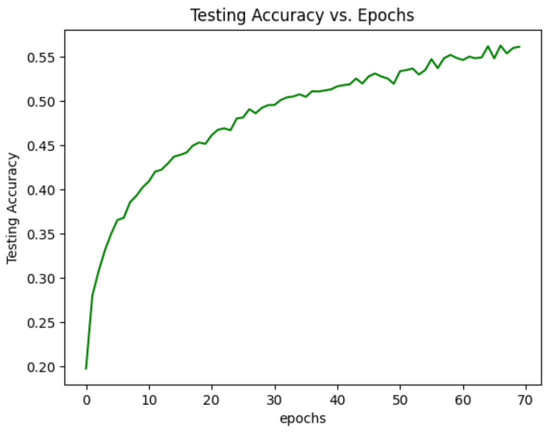


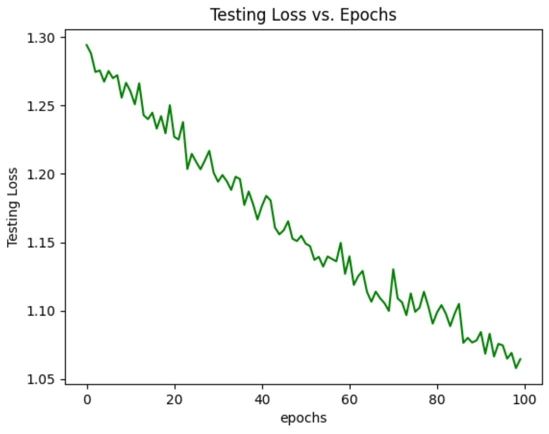


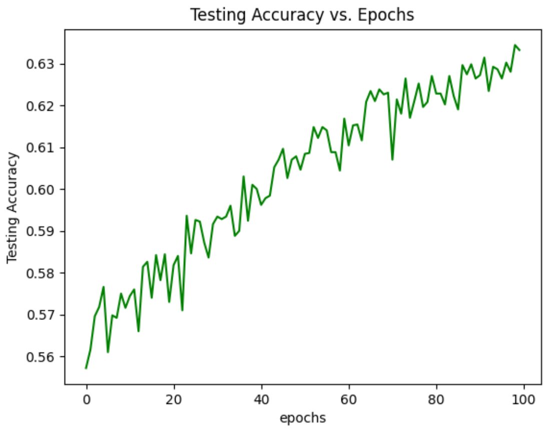


Adadelta optimizer (tested multiple times due to taking more iterations to reach convergence):





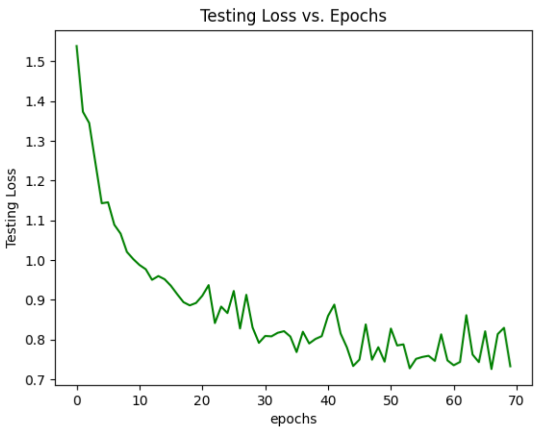


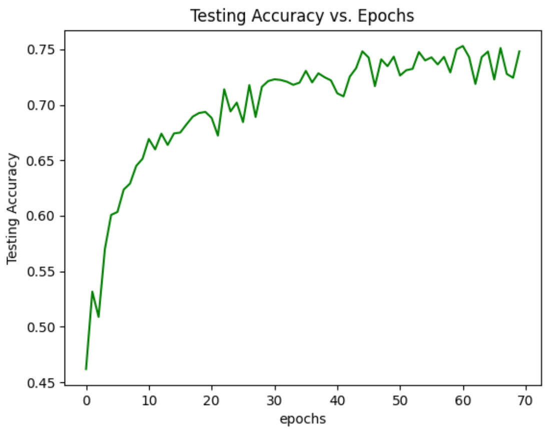


Based on these plots, the best optimizer to use was the Adamax optimizer.

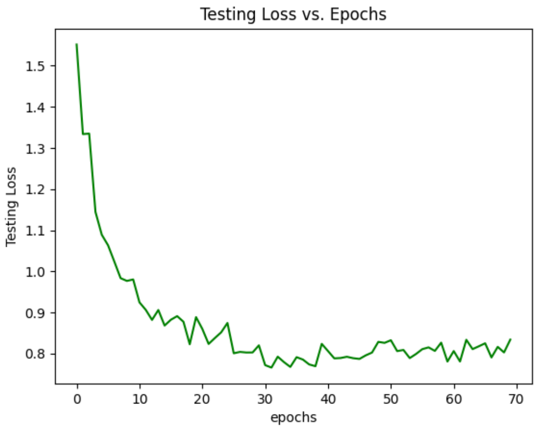
With the optimizer finalized, the model was tested with or without certain dropout layers. The first test contained all dropout layers, the second test removed dropout layers after the pooling and flatten layers since they have no associated weights, then the final test removed all dropout layers. Below are the testing loss and accuracy plots for each of these tests:

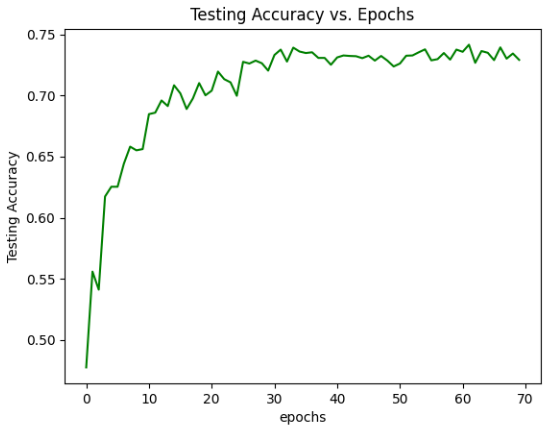
All dropout layers:



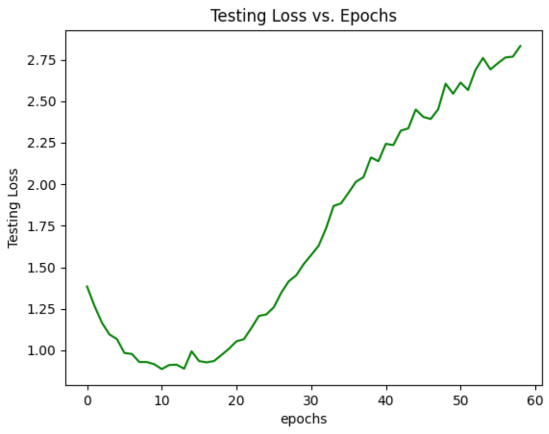


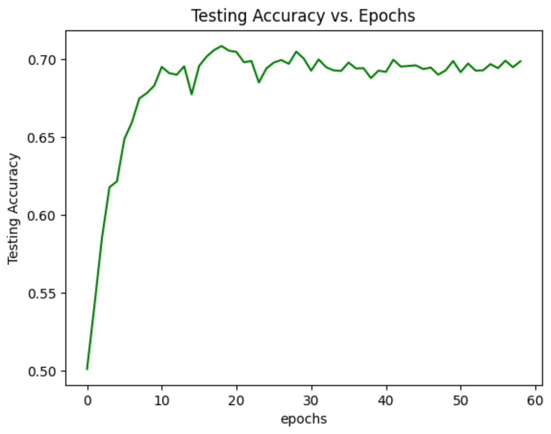
Some dropout layers:





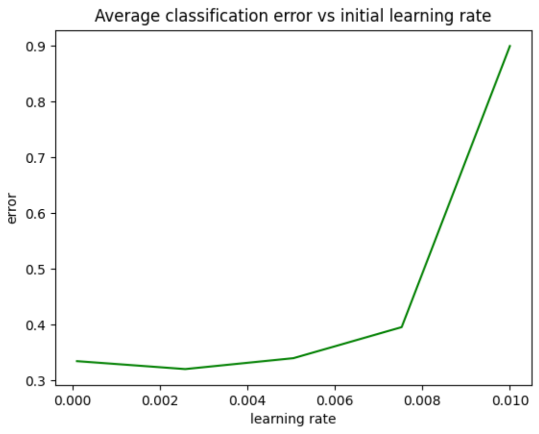
No dropout layers:





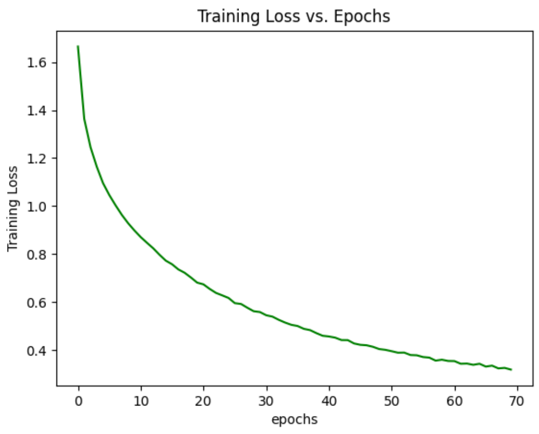
Without dropout layers, the model suffers from severe overfitting, which is apparent in the increase in testing loss over time. The testing accuracy for the test with some dropout layers is more steady than the one with all dropout layers, but doesn't achieve as high of a maximum accuracy. However, despite this, the steadiness ensures that the accuracy of the model won't change drastically between iterations, so the final architecture of the model only included the some of the dropout layers.

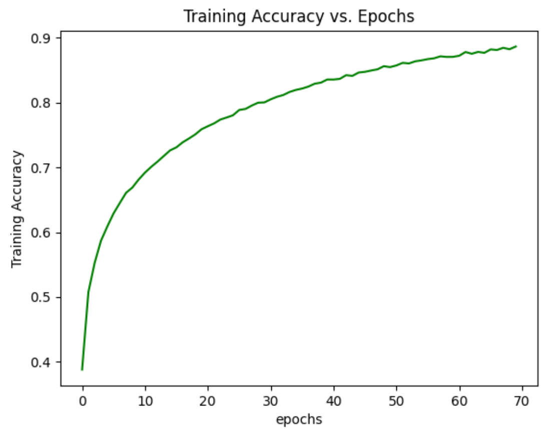
Lastly, with the correct optimizer and architecture, the initial learning rate was modified over a series of tests. The plots below show final classification errors after training until convergence as a function of the initial learning rate:

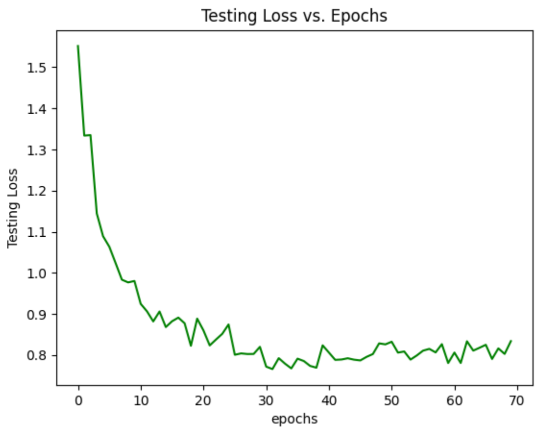


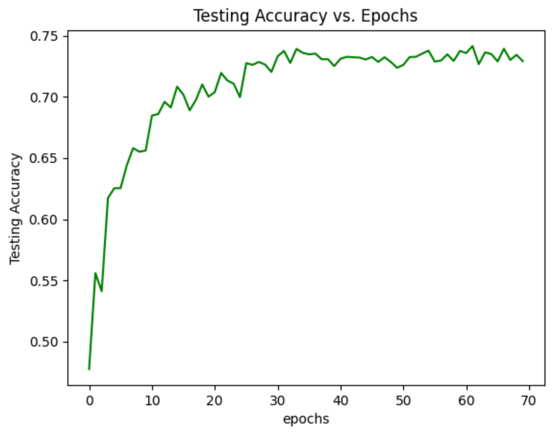
(The wrong upper limit for learning rate was used accidently, meant to be 0.001, ran out of time to conduct second test)

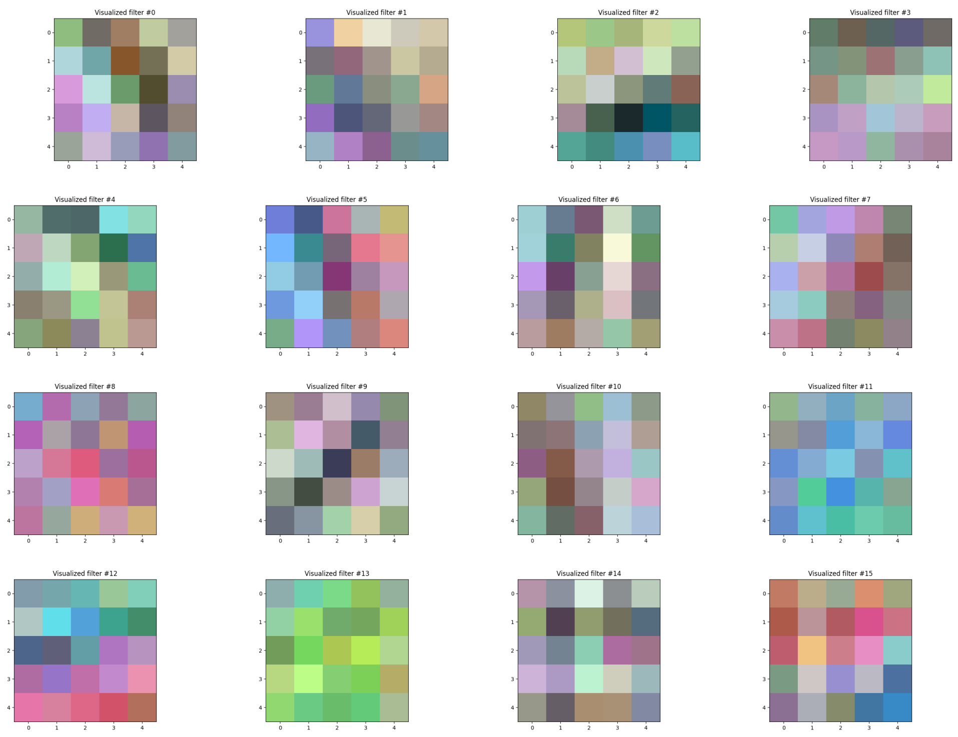
From the tests above, the best optimizer, learning rate, and dropout architecture combination was decided to be the Adamax optimizer with an initial learning rate of 0.001 and dropout layers with dropout rates of 0.2 only after layers with trainable weights. The plots and visualizations along with classification error for each class and average overall classification error below are taken from the final model built with the finalized hyperparameters:











Classification error of class 0 : 31%
Classification error of class 1 : 23%
Classification error of class 2 : 38%
Classification error of class 3 : 42%
Classification error of class 4 : 19%
Classification error of class 5 : 42%
Classification error of class 6 : 18%
Classification error of class 7 : 23%
Classification error of class 8 : 13%
Classification error of class 9 : 22%
Average classification error: 27%

4 Discussion

This model is built entirely using Tensorflow 2.8.0 and Numpy 1.22.2 sparingly. It utilizes the `tf.keras.Model` parent class and `tf.keras.layers` for building the layers of the model. The model inherits this class to have access to its training and testing functions, along with other essential computations, such as optimization, loss computation, and other metrics. The built-in layer methods provide built-in trainable weights for each layer, as well as activation functions and padding in between the layers. With direct access to training and evaluation methods, each process is optimized in speed and efficiency in the underlying framework and produces the best possible results given the defined layers and hyperparameters.

The trained hyperparameters were various optimizers, the learning rate, and the usage of dropout layers. The final model utilizes the Adamax optimizer with an initial learning rate of 0.001 and default values for all other parameters and contains a dropout layer of 0.2 drop out rate between each layer with trainable weights. More details regarding hyperparameters can be found in the results section.

Each epoch took about 15 seconds to train. Usually, in later epochs, the training loss would start low then relatively increase while fitting the model on the rest of the samples during that epoch. However, with the right model architecture and settings, the loss would continue to decrease over each epoch until convergence.