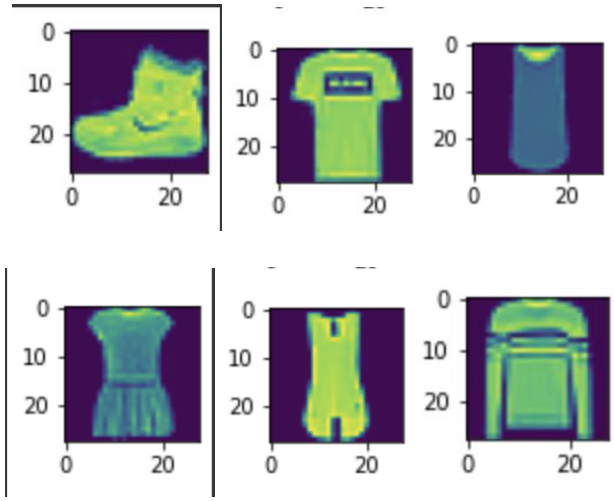
Part 1 - Building CNN

The fashion mnist dataset contains images of clothing items, bags and foot wear with 10 different labels. Each image is a 28 X 28 grayscale image which means it has 28 pixels in height and 28 pixels in width. Each image also has a single pixel value linked to it, this indicates the lightness or darkness of the pixel. In our case, the single pixel value is equal to 1 (28 X 28 X 1). The dataset consists of 60,000 training data, 10,000 test data and 785 columns.

Data Visualization

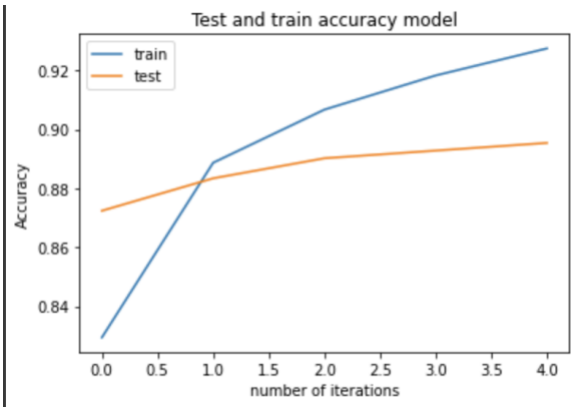


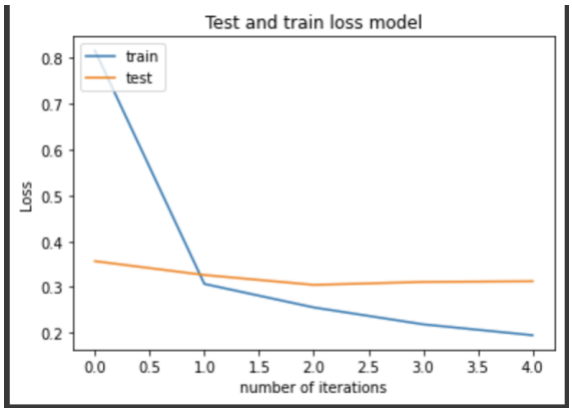
As shown in the images, the dataset consists of different clothing items like dresses, shoes, bags, shirts etc. We can also see that the images are about 28 pixels in height and 28 pixels in width.

Structure of CNN

Our CNN model contains two convolutional layers and an output layer with some hyperparameters and dense layers added in between . The model takes in an input shape of 28 X 28 X 1. The first layer is a single convolutional layer with 64 filters, a kernel size of 3 and uses "relu" activtaion. The second convolution layer has 32 filters, also a kernel size of 3 and uses "relu" activation function. This is followed by a max pooling layer with a pool size of (2,2), a dropout layer of 0.02 and a flatten layer. After the flatten layer, we added a dense layer of 64 nodes which also uses the "relu activation function" and another dropout layer of 0.02. This dense layer helps interpret the features extracted from the flatten layer before passing it on to the output layer.

Our output layer has 10 nodes and uses "softmax" activation. It requires 10 nodes because our dataset has 10 classes and our model has to predict the probability of an image belonging to each of the 10 classes. We use "Nadam" optimizer for this model. This model generated a test accuracy of **89.54%** and a test loss of **31.28%.** The test and train accuracy graph and loss graph is shown below.



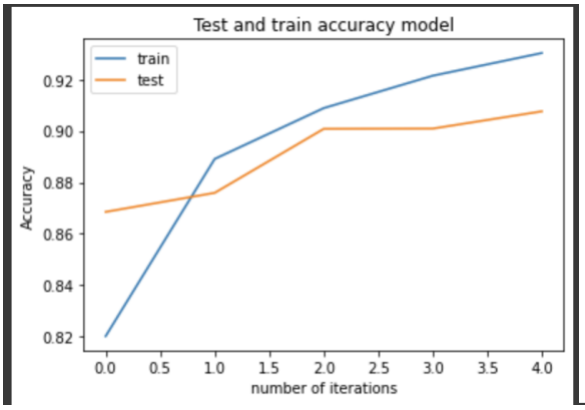


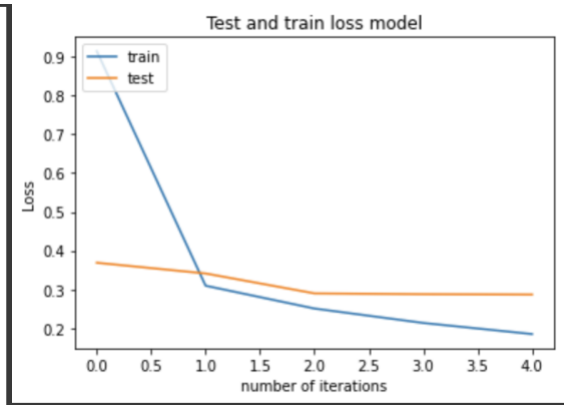
Methods to Improve Model

To improve our model, we added the same methods we used to improve our NN model which we decribed in the NN section above. These methods are early stopping, regularization, batch normalization and K-fold cross validation.

Early stopping

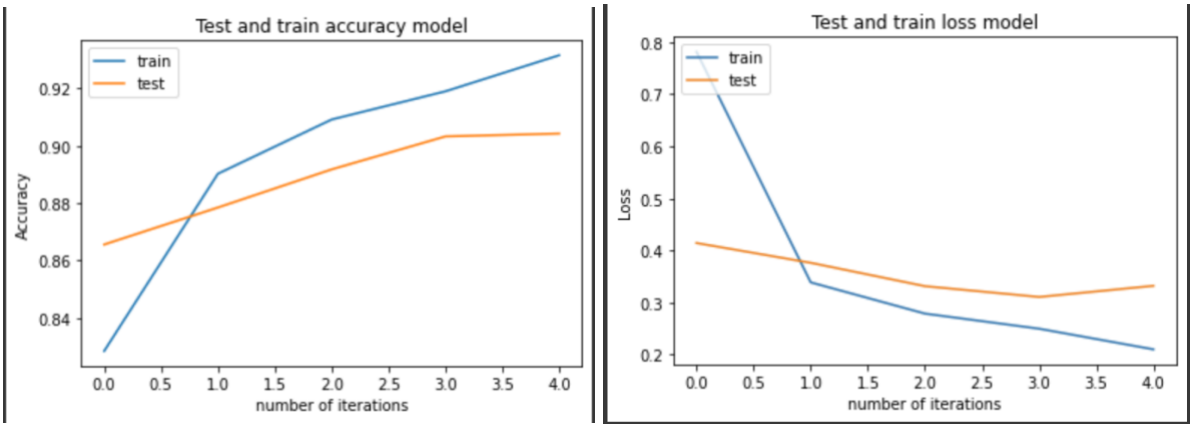
Adding early stopping call back to our CNN model increased the test accuracy to **90.76%** and a reduced the test loss to **28.65%**.





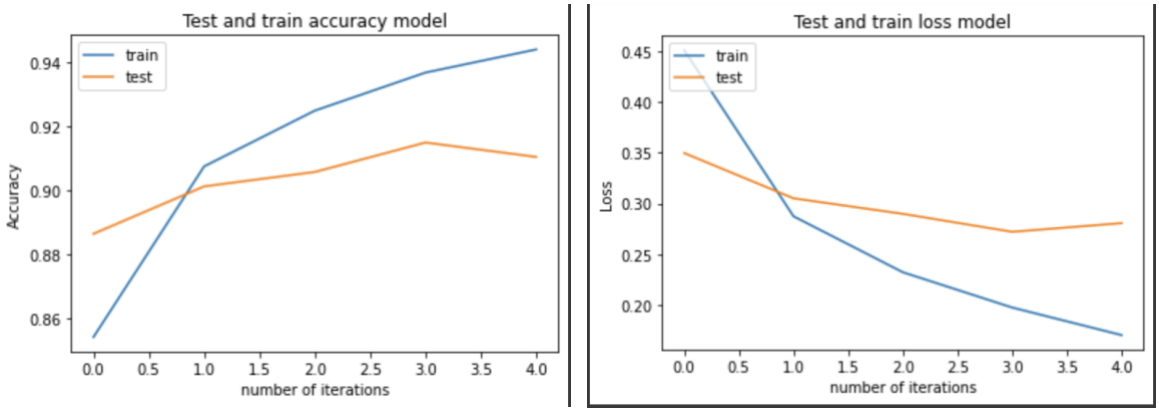
Adding Regularizer

Then we added L2 regularizer of 0.001 to the first and second convolution layer. This slightly reduced our test accuarcy to **90.43%** and increased our test loss to **33.22%.**



Batch Normalization

We added our batch normalization layer after the first convolution layer and it generated a test accuracy of **91.06%** and a test loss of **28.06%.**

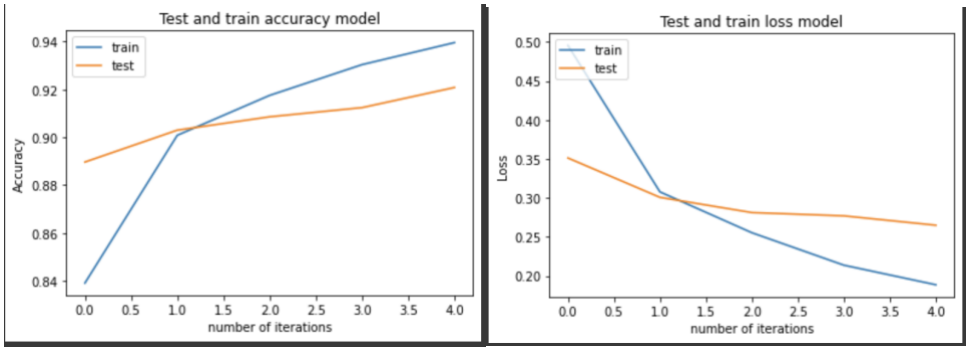


K-fold cross validation

Lastly, we added a K-fold cross validation of 5 folds. For each fold we got a different accuracy which is listed in the table below.

K	Test Accuracy (%)
1	90.79
2	91.26
3	92.08
4	91.54
5	91.08

The highest test accuracy of **92.08%** was generated after the 3rd fold. The graphs below is the test and train accuracy and loss for tor the 3rd fold.



Part 2 - Optimizing CNN + Data Argumentation

Modifying Hyperparameters

The next step was to optimize our CNN model by tuning the hyper parameters. The next following tables and graphs are the parameter details for every hyper parameter that's being tuned and the corresponding accuracy and loss graphs for the test and train data.

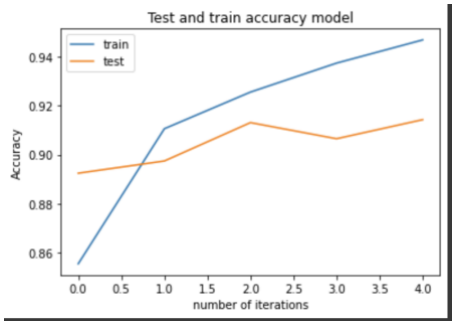
Step 1

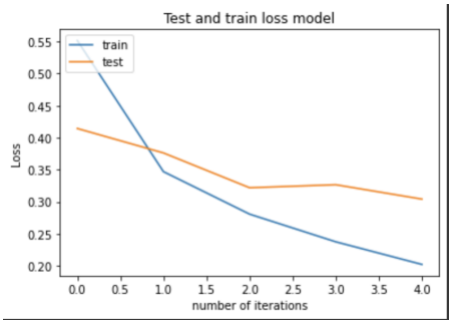
We started by modifying the kernel initializer, we tried "he_uniform", ramdom normal and truncated normal. "he _uniform" generated the highest test accuracy as shown in the table below. "he_uniform" initializer draws samples from a uniform distribution within the number of inputs in the weights tensor.

Kernel Initializer Tuning

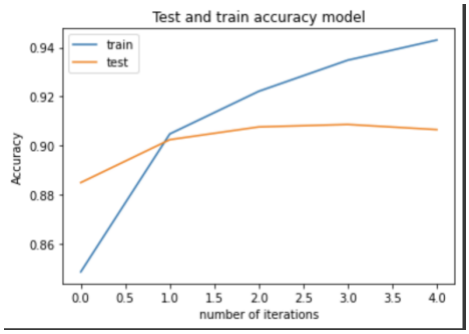
Kernel Initializer	Activation Function	Dropout	Test Accuracy (%)
he_uniform	softmax	0.02	91.43
Random Normal	softmax	0.02	90.65
Truncated Normal	softmax	0.02	91.03

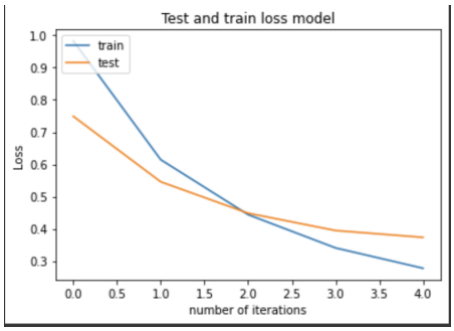
he_uniform Initializer



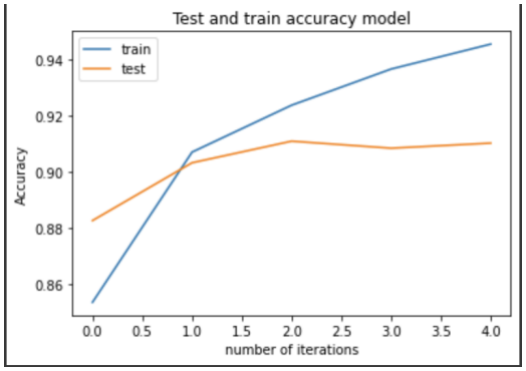


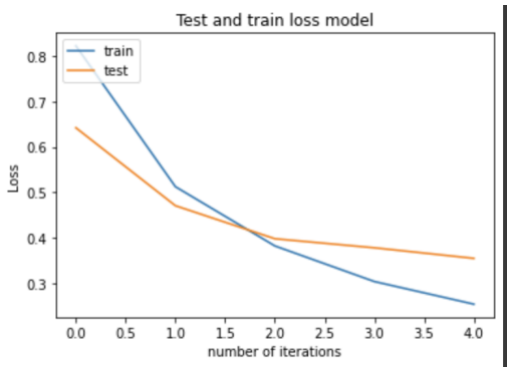
Random Normal Initializer





Truncated Normal Initializer





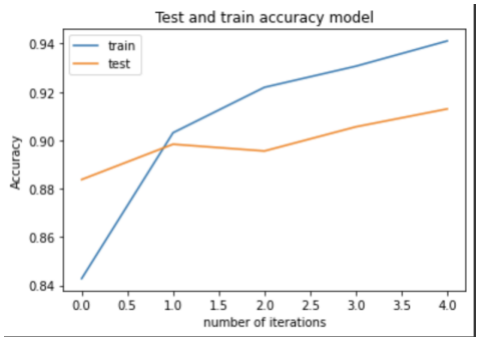
Step 2

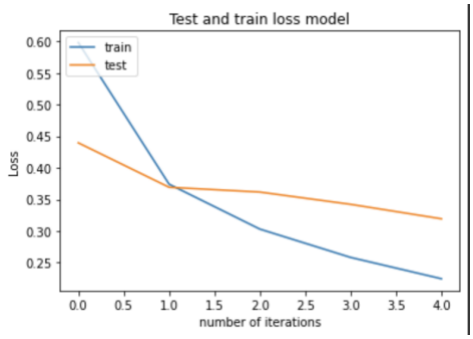
We modified the activation function using sigmoid, linear and softmax. Sigmoid gave the highest test accuracy of **91.30%** .

Activation Function Tuning

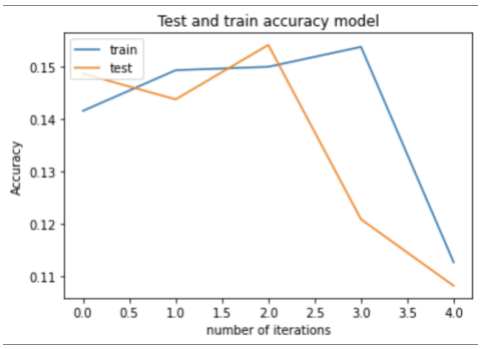
Kernel Initializer	Activation Function	Dropout	Test Accuracy (%)
he_uniform	Sigmoid	0.02	91.30
he_uniform	Linear	0.02	10.82
he_uniform	Softmaxs	0.02	91.25

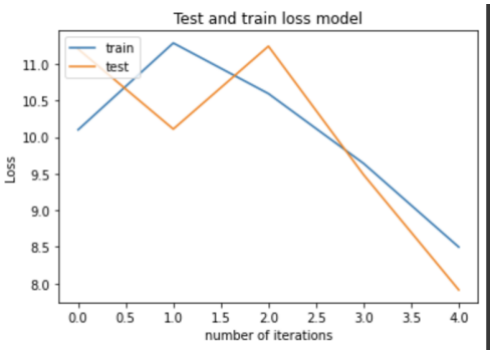
Sigmoid Activation Function



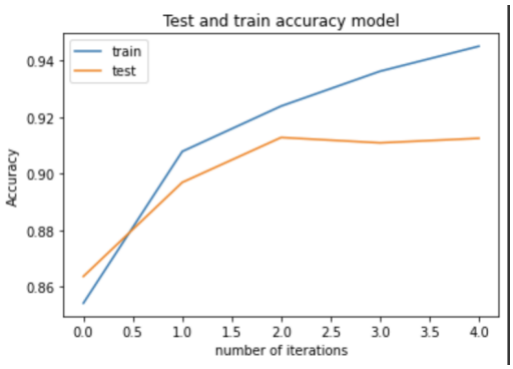


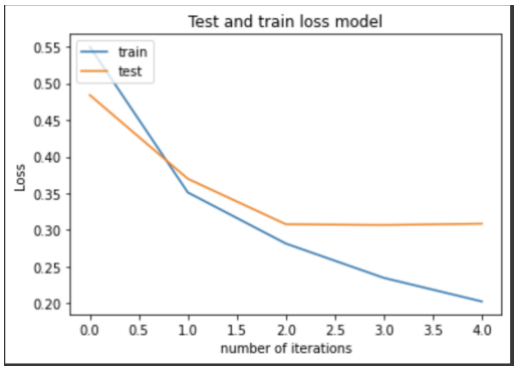
Linear Activation Function





Softmax Activation Function





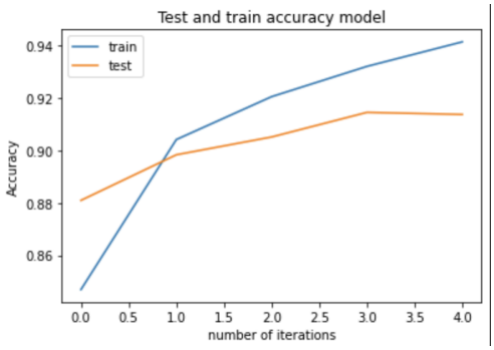
Step 3

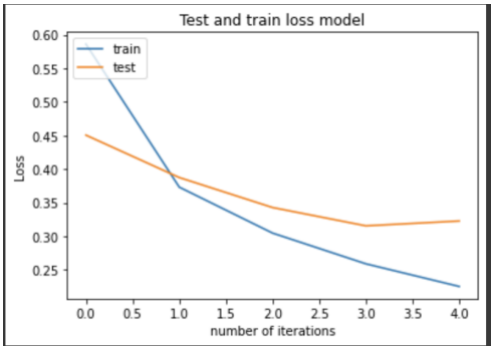
We started varying the Dropout tuning to see if it had any impact to our test accuracy. We received the best accuracy with a Drop out value of 0.01 which tells us that the accuracy increased as we reduced the Drop out values.

Dropout Tuning

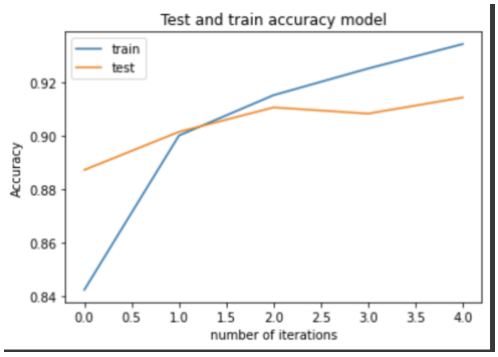
Kernel Initializer	Activation Function	Dropout	Test Accuracy (%)
he_uniform	Sigmoid	0.04	91.38
he_uniform	Linear	0.08	91.42
he_uniform	Softmaxs	0.01	91.61

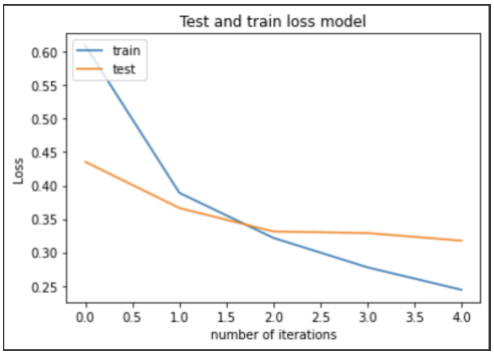
0.04 Dropout



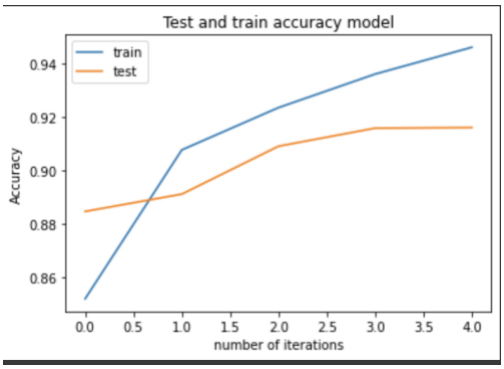


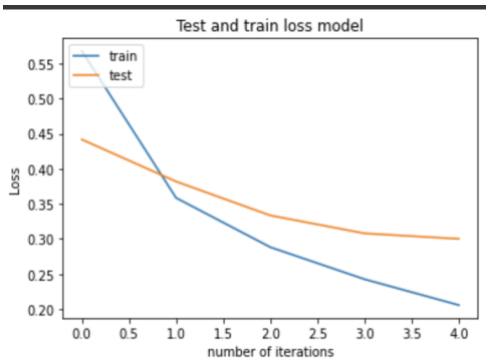
0.08 Dropout





0.01 Dropout





Data Augmentation

To increase the dataset by x4 we used the rotation technique. Rotation augmentation randomly rotates the image clockwise by the number of degrees specified. Before adding rotation, we one hot encoded the y variable for both the training and test set. Then we applied a 45-degree rotation to the dataset. This increased our dataset from 60,000 examples to 24,000 examples. The pixel height and width remained the same.

We applied the rotated dataset to our updated CNN model and obtained a test accuracy of **88.55%** and a test loss of **37.89%.**