Lampros Floudas

f3352126

Deep Learning First Assignment

Fashion MNIST

Introduction

The purpose of this assignment is to create a classifier using Deep Learning techniques and architecture in order to solve the problem of a multiclass image classification. The data that is used in the assignment is the MNIST – Fashion dataset. This dataset consists of 60000 images as a training set along with a test set of 10000 images that all have dimensions of 28x28. There are 10 classes that these images are classified to. The classes are Top, Trousers, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot.

Data Preprocessing

At first, the data set is loaded using the fashion_mnist.load_data() function and it is split into X train, Y train and Y test, X test. Since the images are greyscale, which means each pixel value represents a number between 1 and 255, we can divide each pixel by 255 to normalize the values between 0 and 1. This will make training converge faster. The library used for this assignment and building of the model is Tensorflow / Keras.

MLP Baseline Model

MLP is a simple neural network for simple classification tasks.

To build an MLP model, first a baseline model is created. The model is trained in 100 epochs and the batch size is 128. The model is fed by a training set of 28x28 and is flattened using the Flatten() function by Keras into a 1-hot array of [None,784]. The baseline model consists of an input layer, a flatten layer and a dense layer for the output layer. The Adam optimizer is used in order for the categorical crossentropy loss to be minimized. As a callback, earlystopping is being used to end the training of the model. A patience of 10 is used for the early stopping so that if the validation accuracy is not improved for ten consecutive epochs, the training process will be automatically terminated. Restore best weights must also be used here for the early stopping. The hidden layer uses relu as an activation function while the dense (output) layer uses softmax to produce the probabilities for the ten classes. For this baseline model, all hyperparameters were hardcoded.

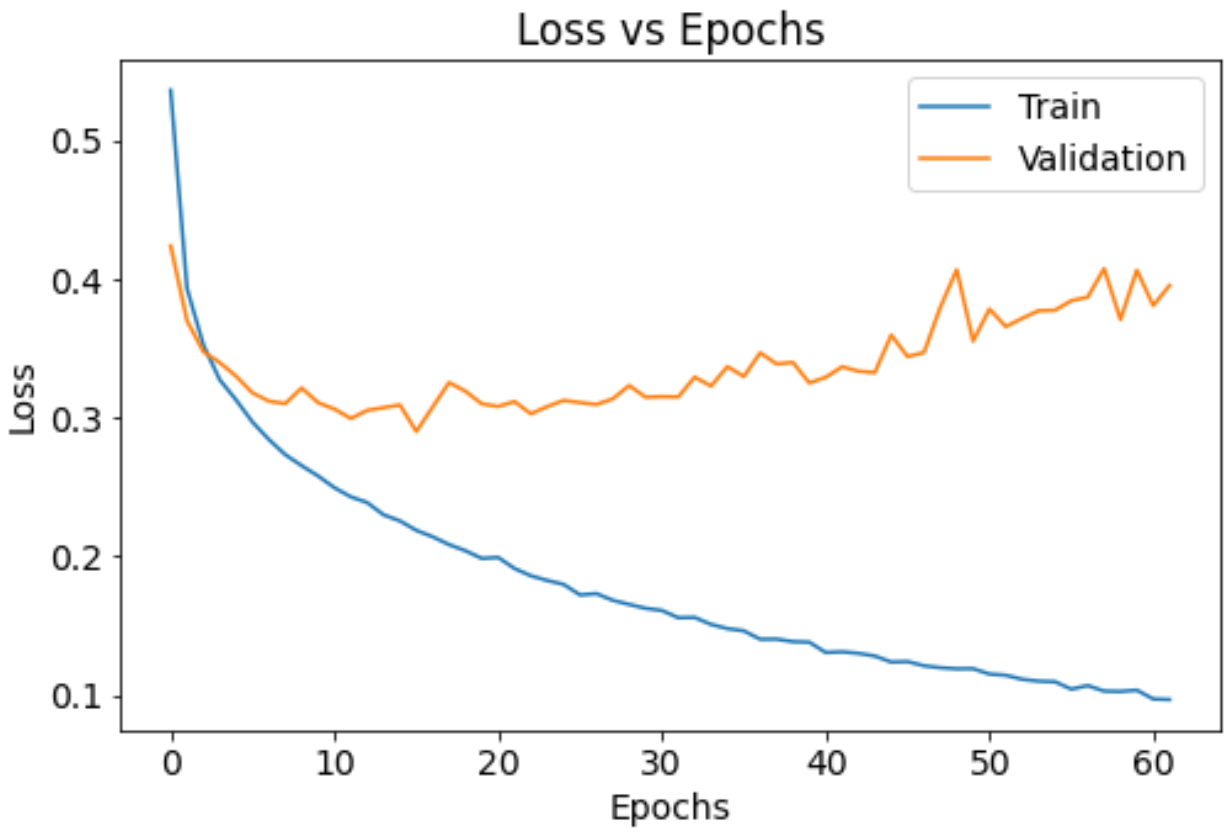
Baseline results

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense_1 (Dense)             (None, 512)               401920

 dropout (Dropout)           (None, 512)               0

 dense (Dense)               (None, 10)                5130

=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

Figure 1: Baseline Results

As it can be seen by the curve, the model does not work well despite of its 90% Validation accuracy. Also, it takes 407050 total parameters. It is clear that this model can take a lot of learning optimization.

```
Train accuracy: 0.9624
Test accuracy: 0.8956
```



Plot 1: Baseline curve.

MLP Hyperparameter Tuning

After the base line model is build, hyperparameter tuning takes place in order to find the best hyperparameters and build the best possible model. Kerastuner was used here in order to tune the hyperparameters. This module saves time and resources by automatically searching for the optimal parameters rather than the user manually searching for them.

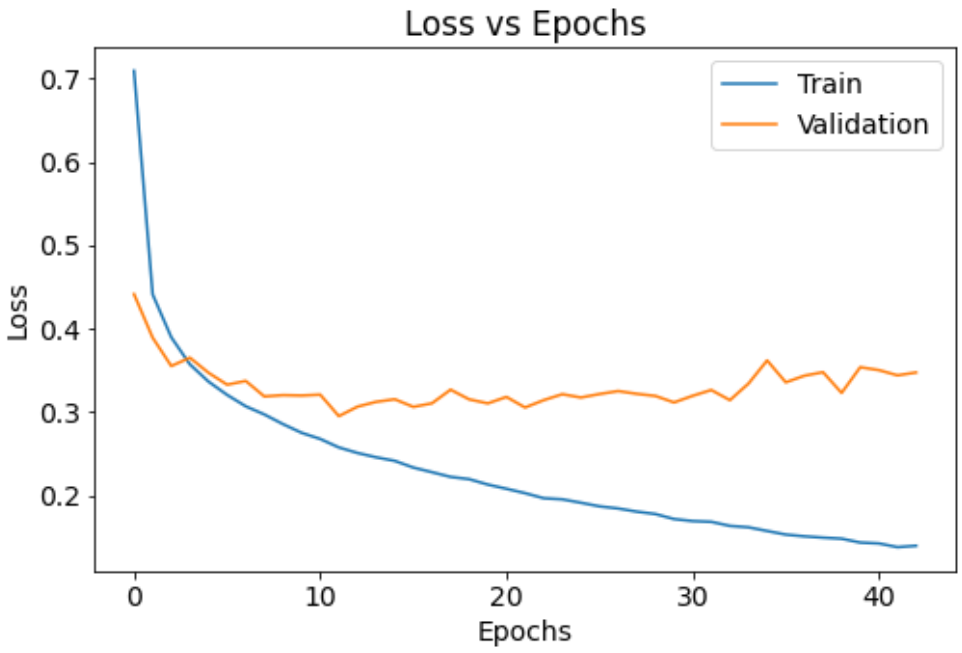
As a first step, a function called build_model(hp) is defined, where the model is built. In there, by using hp.Int, Float and Choice, the model is given several values for its hyperparameters, from which the tuner can choose which one is the optimal. For the MLP in this assignment, the hyperparameters that will be tuned are the number of hidden layers, which will be between 2 and 5 (inclusive), the number of units that each hidden layer will have, which takes values between 256 and 512 with a step of 128, the hidden layers activation functions, between relu and tanh, their dropout between 0 and 0.3 with a step of 0.1 and the Adam optimizer's learning rates which are either 0.01 and 0.0001.

```
[ ]  tuner.search_space_summary()

     Search space summary
     Default search space size: 7
     layers (Int)
     {'default': None, 'conditions': [], 'min_value': 2, 'max_value': 5, 'step': 1, 'sampling': None}
     units_0 (Int)
     {'default': None, 'conditions': [], 'min_value': 256, 'max_value': 512, 'step': 128, 'sampling': None}
     activation (Choice)
     {'default': 'relu', 'conditions': [], 'values': ['relu', 'tanh'], 'ordered': False}
     dropout_0 (Float)
     {'default': 0.0, 'conditions': [], 'min_value': 0.0, 'max_value': 0.3, 'step': 0.1, 'sampling': None}
     units_1 (Int)
     {'default': None, 'conditions': [], 'min_value': 256, 'max_value': 512, 'step': 128, 'sampling': None}
     dropout_1 (Float)
     {'default': 0.0, 'conditions': [], 'min_value': 0.0, 'max_value': 0.3, 'step': 0.1, 'sampling': None}
     learning_rate (Choice)
     {'default': 0.01, 'conditions': [], 'values': [0.01, 0.0001], 'ordered': True}
```

Figure 2: MLP Tuner Summary.

In this step, two tuners from kerastuner were used. Hyperband and BayesianOptimization. The first was a bit quicker with giving results but they had the same results. The tuners were fed with the model that was built in the previous step. As an objective, validation accuracy was used. The tuner was given a maximum of 10 trials for time and resource management, although more should be given. When the tuner is built, using tuner.search(), the search for the optimal hyperparameters begins.

As far as the hyperband tuner is concerned it yielded the below curve:



Plot 2: MLP Tuner Curve using Adam optimizer with Hyperband tuner.

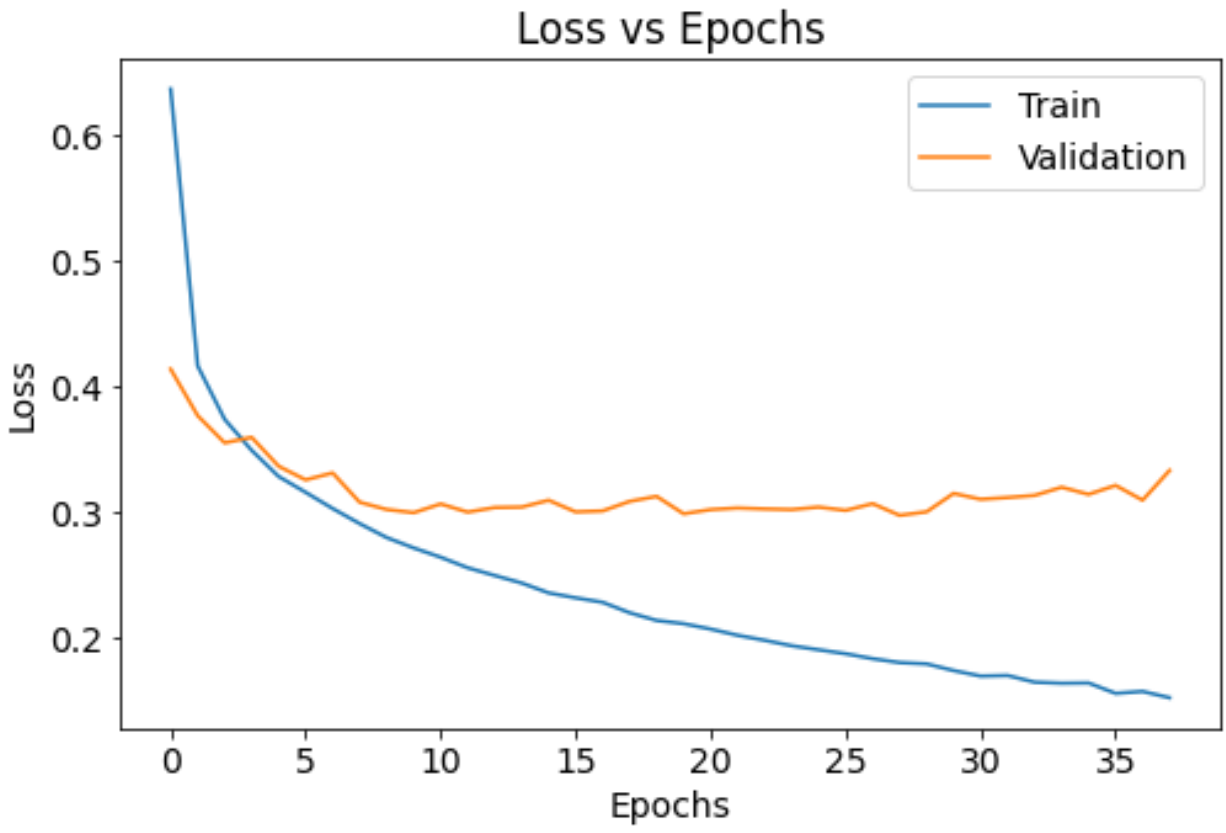
After 30 minutes, 10 trials and 20 epochs per trial (with patience of 5 this time), the best hyperparameters were found with a validation accuracy of 0.8913.

```
Trial 10 Complete [00h 03m 22s]
val_accuracy: 0.875

Best val_accuracy So Far: 0.8913333415985107
Total elapsed time: 01h 00m 44s
INFO:tensorflow:Oracle triggered exit
```

Figure 3: MLP Tuner results.

Now that the optimal hyperparameters are known, they are extracted and then they are used to build the better MLP model. It is seen that the model has a better accuracy than the baseline model. The Hyperband Tuner and the BayesianOptimizer Tuner yield similar results.

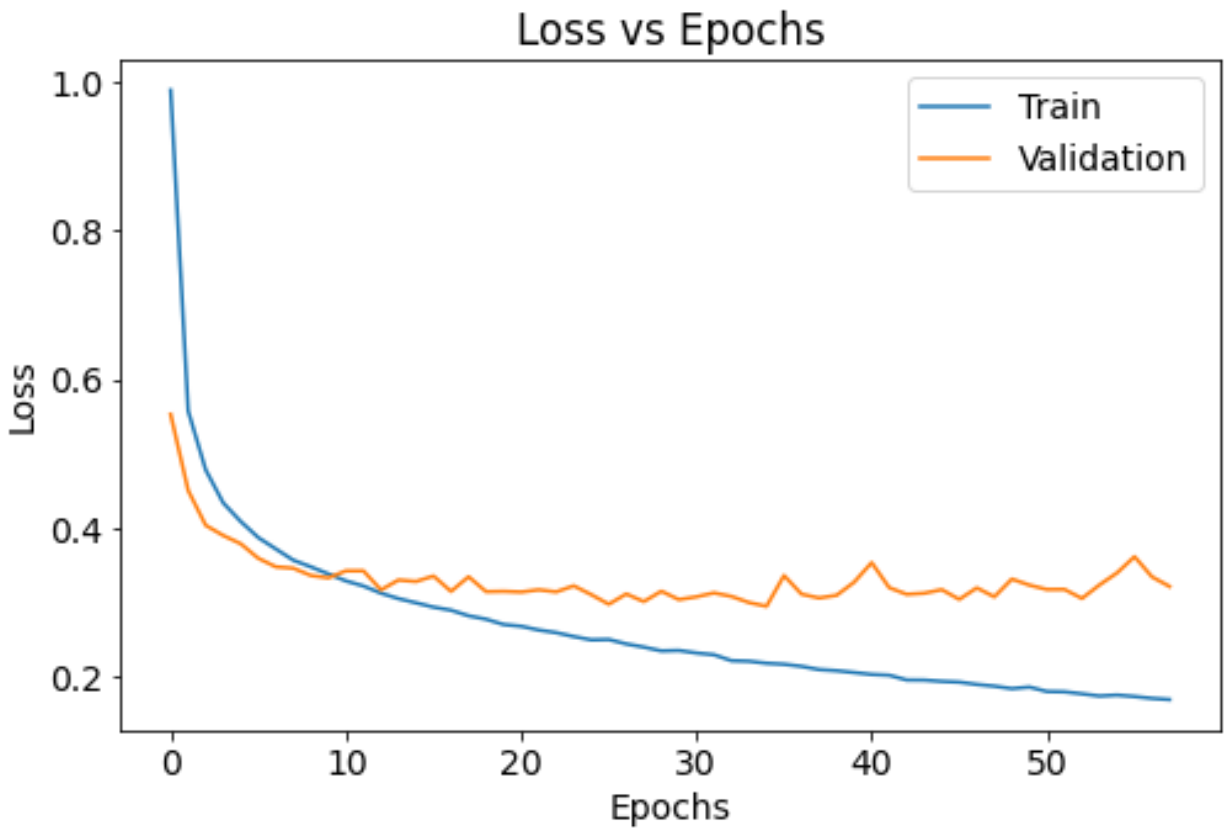
```
Train accuracy: 0.9275
Test accuracy: 0.893
```



Plot 3: MLP Tuner Curve using Adam optimizer.

It can be observed that in the end around epoch 15 the model may be starting to overfit. However it is not taken into account since it is at the very end of the training process.

After that, another model was created. The same hyperparameter choices were fed to the model, however this time an SGD optimizer was used. It yielded the same accuracy approximately, however the learning curve was so much better.



Plot 4: MLP Tuner Curve using SGD optimizer.

CNN Hyperparameter Tuning

A new build_model(hp) function is built here depicting a CNN neural network this time. Again, by using hp.Int, hp.Choice and Float, the CNN model is given a few values that the tuner can chose for its best hyperparameter values. For the CNN neural network, three convolutional layers are used with each being added Max Pooling. The hyper parameters that are tuned are the number of filters for the convolutional layers with numbers between 32 and 128 for the first one and 32 and 64 for the two remaining, each with a step of 32. In addition, the kernel size is given the choice between 3 and 5, the dropout rates are fixed and scaling from 0.3 to 0.5 for each layer in order to try and avoid overfitting. The dense layer is also given the choice for its units with a

choice between 32 and 128 with a step of 32. Lastly, same as in the MLP model, the learning rate can be either 0.01 or 0.0001.

```
[43] tuner_CNN.search_space_summary()

     Search space summary
     Default search space size: 8
     conv_1_filter (Int)
     {'default': None, 'conditions': [], 'min_value': 32, 'max_value': 128, 'step': 32, 'sampling': None}
     conv_1_kernel (Choice)
     {'default': 3, 'conditions': [], 'values': [3, 5], 'ordered': True}
     conv_2_filter (Int)
     {'default': None, 'conditions': [], 'min_value': 32, 'max_value': 64, 'step': 32, 'sampling': None}
     conv_2_kernel (Choice)
     {'default': 3, 'conditions': [], 'values': [3, 5], 'ordered': True}
     conv_3_filter (Int)
     {'default': None, 'conditions': [], 'min_value': 32, 'max_value': 64, 'step': 32, 'sampling': None}
     conv_3_kernel (Choice)
     {'default': 3, 'conditions': [], 'values': [3, 5], 'ordered': True}
     dense_1_units (Int)
     {'default': None, 'conditions': [], 'min_value': 32, 'max_value': 128, 'step': 32, 'sampling': None}
     learning_rate (Choice)
     {'default': 0.01, 'conditions': [], 'values': [0.01, 0.0001], 'ordered': True}
```

Figure 4: The choices given to the CNN tuner as well as the hyperparameters that will be tuned.

Same as before, the Hyperparameter Tuner that is used is the BayesianOptimizer() and the objective fed into it along with the model is the validation accuracy. By once again using the tuner.search(), the tuner looks for the best possible hyperparameters from the once that were given into the model.

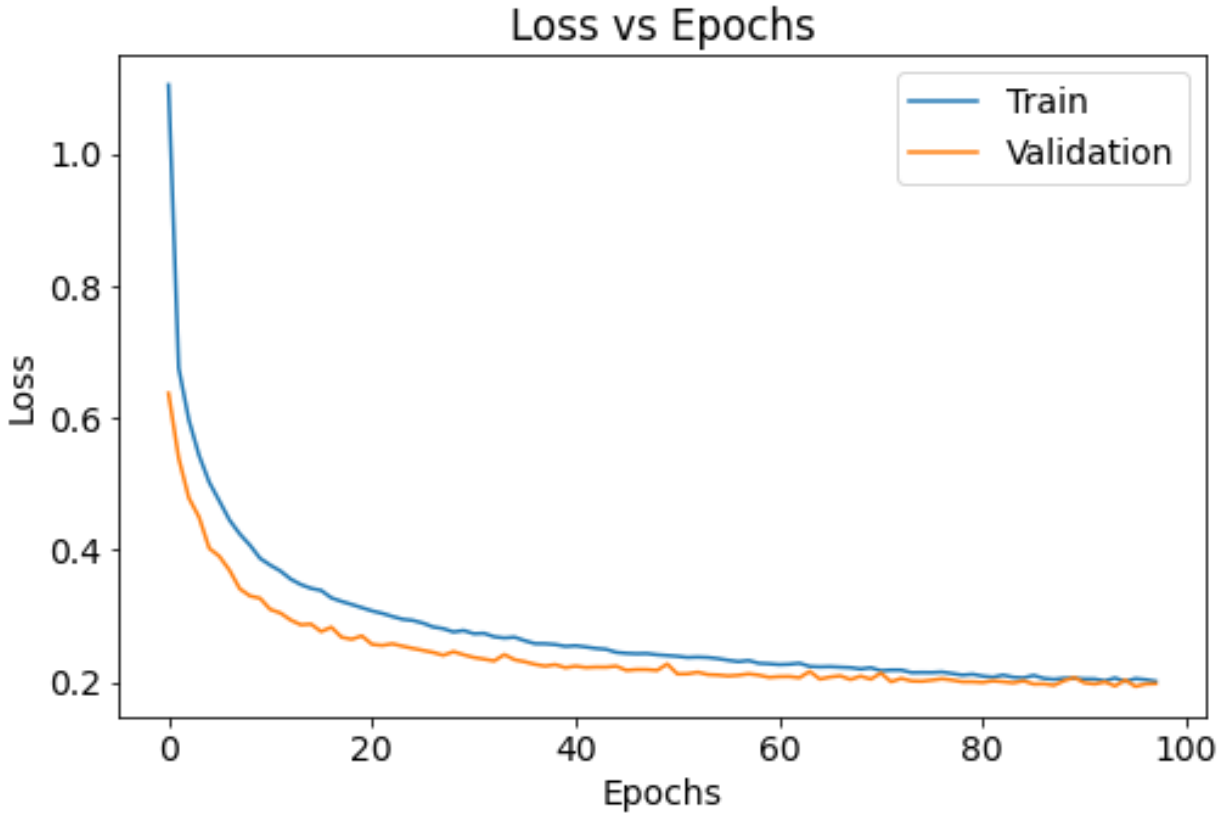
```
cnnmodel.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 26, 26, 96)        960

 MaxPool2D1 (MaxPooling2D)   (None, 13, 13, 96)        0

 Dropout1 (Dropout)          (None, 13, 13, 96)        0

 conv2d_4 (Conv2D)           (None, 9, 9, 64)          153664

 MaxPool2D2 (MaxPooling2D)   (None, 5, 5, 64)          0

 Dropout2 (Dropout)          (None, 5, 5, 64)          0

 conv2d_5 (Conv2D)           (None, 3, 3, 64)          36928

 MaxPool2D3 (MaxPooling2D)   (None, 2, 2, 64)          0

 Dropout3 (Dropout)          (None, 2, 2, 64)          0

 flatten_1 (Flatten)         (None, 256)               0

 dense_2 (Dense)             (None, 64)                16448

 dense_3 (Dense)             (None, 10)                650

=================================================================
Total params: 208,650
Trainable params: 208,650
Non-trainable params: 0
```

This time (After using the GPU capabilities) 30 minutes later and 10 trials of 20 epochs each, the best accuracy scored was 0.9 and the optimal hyperparameters were extracted. These hyperparameters were used in order to build a good CNN model. The model took 98 epochs to be build with a patience of 5, which means that epoch 39 yielded the best result. The total number of parameters used on CNN model is 208650. At first, a patience of 5 was used for the model and it took 44 epochs, yielding worse results.

As it is depicted, the train accuracy of the model is 0.9507 and the test accuracy is 0.9263.

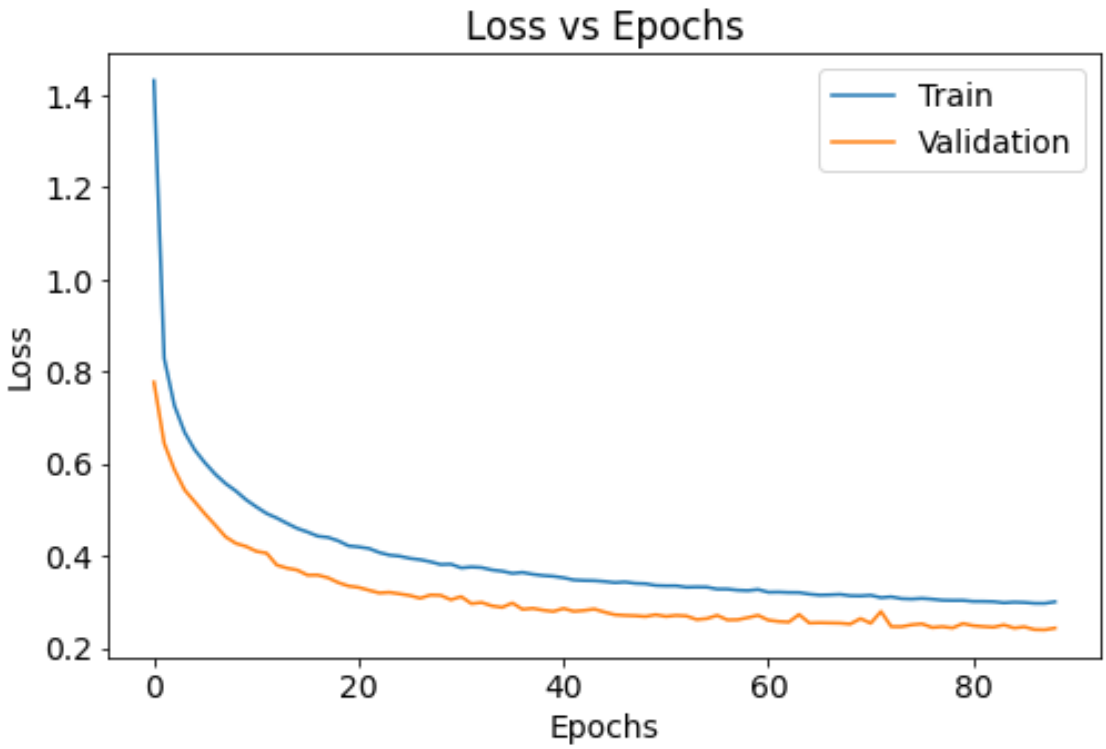
```
Train accuracy: 0.9507
Test accuracy: 0.9263
```



Plot 5: CNN Tuner Curve using Adam optimizer.

As it is seen in the plot, there is no overfitting in the model and the training of it was successful.

After that, another model was built using SGD Optimizer and the same hyperparameter choices. The results were a bit similar but a bit worse.



Conclusion

In conclusion, the learning capability of the model was improved significantly with both the MLP and the CNN hyperparameter tuning. There are a lot more ways to tune the model, for example by feeding it with more choices in the layers, using more trials, making the tuner choose the dropout rate for the CNN model in each layer. Also, more optimizers could be used and tuned accordingly and see which one gives better results. Between the MLP and the CNN tuned models, it seems like the CNN performs significantly better than the MLP trained model. While the SGD optimizer seems to have better results for the MLP model, the Adam optimizer yields better results for the CNN model.

https://colab.research.google.com/drive/13Pg4msAzsfqBrjfjRIG3vZAWvtWriNMX?usp=sharing code