Data 640: Predictive Modeling Fall 2023

Assignment # 4– Deep Learning MNIST Fashion

Simon A Hochmuth

Email: shochmuth3@student.umgc.edu

Professor: Ed Herranz

Introduction

Deep Learning is an extension of neural networks that uses multiple hidden layers and learning paradigms to produce learning like how we understand the human brain learns. Deep learning is often used to solve very complex problems that include image recognition, natural language processing, and audio generation (Knode 2023). This analysis is creating image recognition models, meaning successful models could be implemented to add value in things like robotic fashion assembly lines to online image recognition for copyright infringement. The goal is to use this deep learning method to understand the images contained in the MNIST dataset, which will be classified by the model. Five models will be made, and the classification labels created will then be scored for the accuracy of the model and each will be compared to understand the best model in the analysis.

Data Set Description

As mentioned, the Fashion MNIST dataset will be used in this deep learning analysis. It contains 70000 28x28 labeled fashion images, where 60000 examples are in the training set and 10000 examples are in the test. Each of these fashion images relate to one of 10 labels, where 0 = T-Shirt/Top, 1 = Trouser, 2 = Pullover, 3 = Dress, 4 = Coat, 5 = Sandal, 6 = Shirt, 7 = Sneaker , 8 = Bag , and 9 = Ankle Boot. As discussed, each image is 28x28 pixels or 784 total pixels, where the data frame has 1 column for the label and 784 for each pixel. The value for each pixel is 1-255, which dictates the darkness of that individual pixel. It is also important to note that because these are greyscale images, where there is no color, which is why darkness is the only value measured (Zalando Research 2017). The deep learning model will use the pixel darkness to understand each individual image and develop its classification with the goal of being able to

correctly predict the specific clothing types listed in the labels above.

Data Cleansing and Preparation

The MNIST dataset is very well developed and is provided within the tensorflow.keras.datasets library within python, meaning there was no cleaning needed to be conducted. After importing fashion_mnist from this library, the next step was to load it into the necessary variables for the model to be created. For these models a test and train dataset were needed, where there was a variable to save the 784 values for pixel darkness and another to save the category associated for the image. This created 4 total variables called train_x, train_y, test_x, test_y. Where x contained the darkness values and y contained the labels. Now that each MNIST Fashion image was correctly loaded into the associated variable the last step was to categorize each of the label dataset using the tensorflow.keras.utils.to_categorical () command. After all this was completed, the dataset was now ready to have models developed on it.

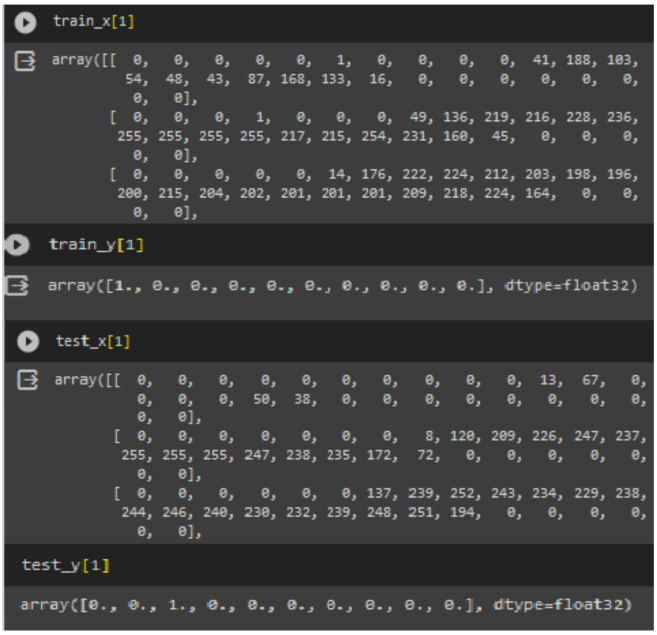

Figure 1: Shows an example of the data in each of the 4 variables, one for train_x, train_y, test_x, and test_y. Note that this is after the labels were categorized.

Deep Learning Models Developed

In total 5 Convolutional Neural Network (CNN) models were developed in this analysis. CNN models are models that are well suited for image recognition and processing. These models use multiple layers like convolutional layers, pooling layers, and fully connected layers to extract features used to create a classification for each image (GeeksforGeeks 2023). Where the main difference between each model was the parameters chosen for each of these CNN models. Model 1 utilized the same parameters given by the professor for the MNIST Fashion template file. Model 2 modified the dropout parameter to be 0.3, pool size = 3, and added a softmax and sigmoid activation layer. Model 3 combined a relu and sigmoid activation layer. Model 4 utilized only a tanh activation layer. Lastly, Model 5 utilized sigmoid, relu and softmax activation layers. Table 1 below summarizes each of the differences described, and figures 2 to 6 can be reviewed to see the complete parameters utilized. It was chosen to focus on changing the activation layers to see if adding more than one would increase the performance. The only difference from this is that Model 2 added an extra layer but also changed the values for a few of the parameters to see how performance was affected. The goal was to review how each of these changes changed model performance, with the hope that the accuracy would increase, and the loss would decrease.

Model Number	Change from Template	Figure Number
1	This is the original	2
2	Dropout = 0.3 , with a softmax and sigmoid activation layer, and pool size = 3	3
3	relu and sigmoid activation layers	4
4	tanh activation layer	5
5	utilized sigmoid , relu and softmax activation layers	6

Table 1: Shows the main differences in each model from the original model 1.

Results

Now that the 5 models were created, the last step was to analyze and compare the results to see which of the models performed the best in classifying the images. To compare the models, the accuracy and loss were compared across each to measure overall performance. Accuracy is simply the percentage of the classifications that are correct, for example if the image is a T-Shirt, one would expect the predicted classification to be correctly labeled 1. Loss is a summation of the differences between the predictions and true values, where an ideal model would be 0 loss because the values would be identical. The larger the differences between predictions and true values, the larger the sum of the loss. The goal is to have 0 loss and 100% accuracy. The first model had a loss of 0.888 and 69.5% when predicting using the test data, with very similar values for predictions on the training dataset. This value was set as the base to compare against, where other models slightly changed model 1 parameters to see if performance could be increased. The results showed that each of the other models did significantly worse than the model. Where the next closest model 2 had 33% test accuracy and 2.11 loss, which is 36% lower than the accuracy of model 1. For model two activation layer was added on Next, the 3rd model had 23.5% accuracy and 1.72 loss in the test predictions, which is significantly worse than the previous two models. The last models, 4 and 5, had 5.6% and 10.8% accuracy overall, which shows than the parameters chosen for these significantly degraded the overall model performance. The total loss and accuracy values for each of the models can be seen in table 2 below. Figure 7 below shows plot of the score and accuracy of the test predictions by model. Where the orange line, which depicts accuracy, decreases from model 1 to the others. This shows that the accuracy clearly lowered when model 1 was changed, showing that the parameters chosen did not increase model performance. The blue line in figure 7, which depicts the loss score, increases from model 1 to the others. This shows that the values predicted become

increasingly incorrect leading to higher differences between predictions and true values. The loss score paired with the accuracy proves that the parameters chosen for model 2 to 5 lowered the overall performance of the model. Figure 8 in the appendix shows a similar loss/accuracy graph but for the training dataset, and figures 9 and 10 show the plot of just model accuracy for test and training predictions. Figures 7 to 10 all confirm that the model accuracy decreased from model 1 and loss increased when parameters were changed for both test and training data.

Model Number	Train Loss	Train Accuracy	Test Loss	Test Accuracy
1	0.86	70.9%	0.888	69.5%
2	2.11	34.0%	2.11	33.0%
3	1.71	23.0%	1.72	23.5%
4	5.52	5.8%	5.43	5.6%
5	7.49	11.1%	7.49	10.8%

Table 2: shows the loss and accuracy for each model on the test and train datasets.

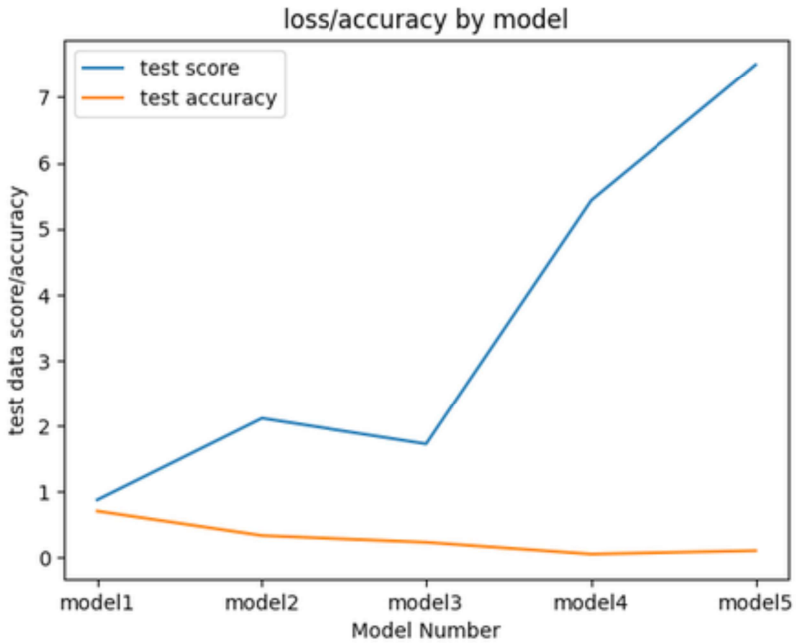

Figure 7: plot of the score and accuracy of the test predictions by model

As previously discussed, table 1 above shows the parameters that were changed from model 1. Model 2 added a sigmoid activation layer and changed the pool size and dropout values, which led to a 36% decrease in accuracy and a worse overall model. Model 3 changed the activation layer to relu and sigmoid, which decreased the accuracy by 46%. Model 4 replaced the softmax layer with a single tanh layer, which led to a 64% decrease in accuracy. Lastly, model 5 had a 58.7% decrease in accuracy after adding a sigmoid, relu, and softmax activation layer. Overall, these changes did not add to the performance of the models which lead to the decision to choose model 1 as the best performing model.

Conclusions and Takeaways

The results of the 8 models showed that model 1 with parameters shown in figure 2 below provided the best performance out of the other 4. Model 1 had an accuracy of 69.5% and loss score of 0.88 on test predictions and an accuracy of 70.9% and a loss score of 0.86 on the training dataset predictions. Model 1 served as the base for the other models, and parameters were changed slightly from model 1 to create the other 4, with a focus on changing the activation layers. It was found that the changes made to create the other 4 models significantly degraded the performance from model 1. Meaning model 1 did significantly better, which can be seen in figure 7 above and figures 8 to 10 below. Since the main takeaway was that added activation layers lowered the overall performance, future work would focus on changing the parameter values instead of adding layers. With the goal of raising model performance through altering parameters. One weakness with these deep learning models is that there is very little insight into the algorithm behind each model, making it very hard to interpret how the model performance was impacted. The goal of this future work would be to continue to analyze the model accuracy to find parameters that increase the accuracy and lower the score in Model 1.

References:

Knode, S. (2023). *Deep Learning Overview*. UMGC DATA 640. https://learn.umgc.edu/d2l/le/content/920742/viewContent/31268373/View

Zalando Research. (2017). *Fashion MNIST*. Kaggle.com. https://www.kaggle.com/datasets/zalando-research/fashionmnist

GeeksforGeeks. (2023, February 3). *Convolutional neural network (CNN) in machine learning*. https://www.geeksforgeeks.org/convolutional-neural-network-cnn-in-machine-learning/#

Appendix

```
#MODEL ONE

[5]  model = Sequential()
     model.add(Conv1D(filters = 236, kernel_size = 2,  input_shape = (28, 28)))
     model.add(MaxPooling1D(pool_size = 2))
     model.add(Dropout(0.2))
     model.add(BatchNormalization())
     model.add(Conv1D(filters = 128, kernel_size = 2))
     model.add(MaxPooling1D(pool_size = 2))
     model.add(Dropout(0.2))
     model.add(BatchNormalization())

     model.add(Flatten())
     model.add(Dense(units = 10, activation = 'softmax'))
```

Figure 2: the parameters added to model 1.

```
# MODEL TWO

model = Sequential()
model.add(Conv1D(filters = 236, kernel_size = 2,  input_shape = (28, 28)))
model.add(MaxPooling1D(pool_size = 3))
model.add(Dropout(0.3))
model.add(BatchNormalization())
model.add(Conv1D(filters = 128, kernel_size = 2))
model.add(MaxPooling1D(pool_size = 3))
model.add(Dropout(0.2))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(units = 10, activation = 'softmax'))
model.add(Dense(units = 10, activation='sigmoid'))
```

Figure 3: the parameters added to model 2.

```
[14] # MODEL THREE

     model = Sequential(

                        )
     model.add(Conv1D(filters = 236, kernel_size = 2,  input_shape = (28, 28)))
     model.add(MaxPooling1D(pool_size = 2))
     model.add(Dropout(0.2))
     model.add(BatchNormalization())
     model.add(Conv1D(filters = 128, kernel_size = 2))
     model.add(MaxPooling1D(pool_size = 2))
     model.add(Dropout(0.2))
     model.add(BatchNormalization())

     model.add(Flatten())
     model.add(Dense(10, activation="relu", name="layer1"))
     model.add(Dense(10, activation="sigmoid", name="layer2"))
```

Figure 4: the parameters added to model 3.

```
# MODEL FOUR

[20] model = Sequential()
     model.add(Conv1D(filters = 236, kernel_size = 2,  input_shape = (28, 28)))
     model.add(MaxPooling1D(pool_size = 2))
     model.add(Dropout(0.2))
     model.add(BatchNormalization())
     model.add(Conv1D(filters = 128, kernel_size = 2))
     model.add(MaxPooling1D(pool_size = 2))
     model.add(Dropout(0.2))
     model.add(BatchNormalization())

     model.add(Flatten())
     model.add(Dense(units = 10, activation='tanh'))
```

Figure 5: the parameters added to model 4.

```
# MODEL FIVE

model = Sequential()
model.add(Conv1D(filters = 236, kernel_size = 2,  input_shape = (28, 28)))
model.add(MaxPooling1D(pool_size = 2))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Conv1D(filters = 128, kernel_size = 2))
model.add(MaxPooling1D(pool_size = 2))
model.add(Dropout(0.2))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(units = 10, activation='sigmoid'))
model.add(Dense(units = 10, activation='softmax'))
model.add(Dense(units = 10, activation='relu'))
```

Figure 6: the parameters added to model 5.

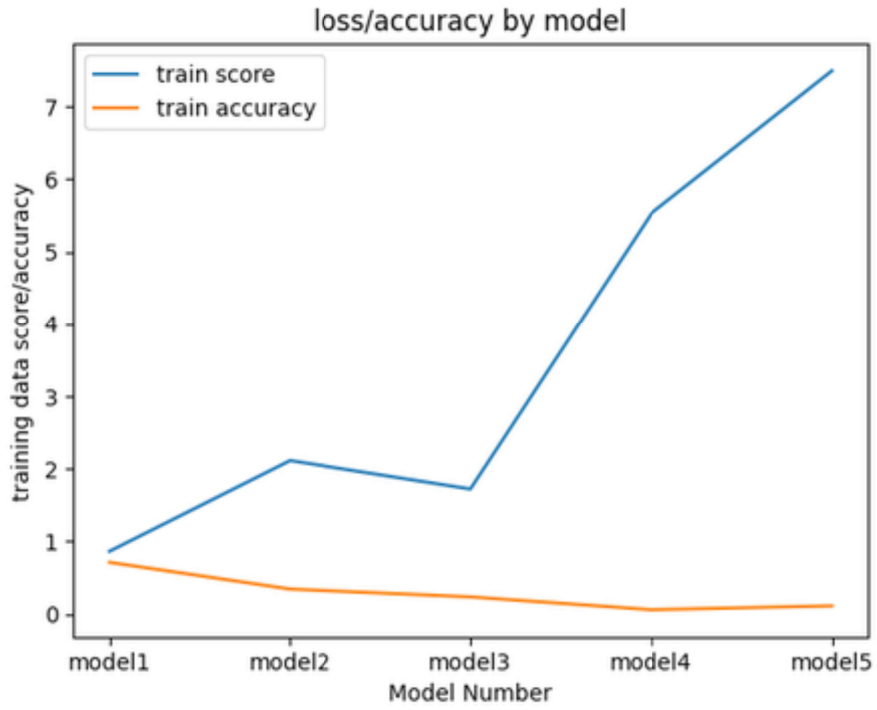

Figure 8: The training prediction score and accuracy by model

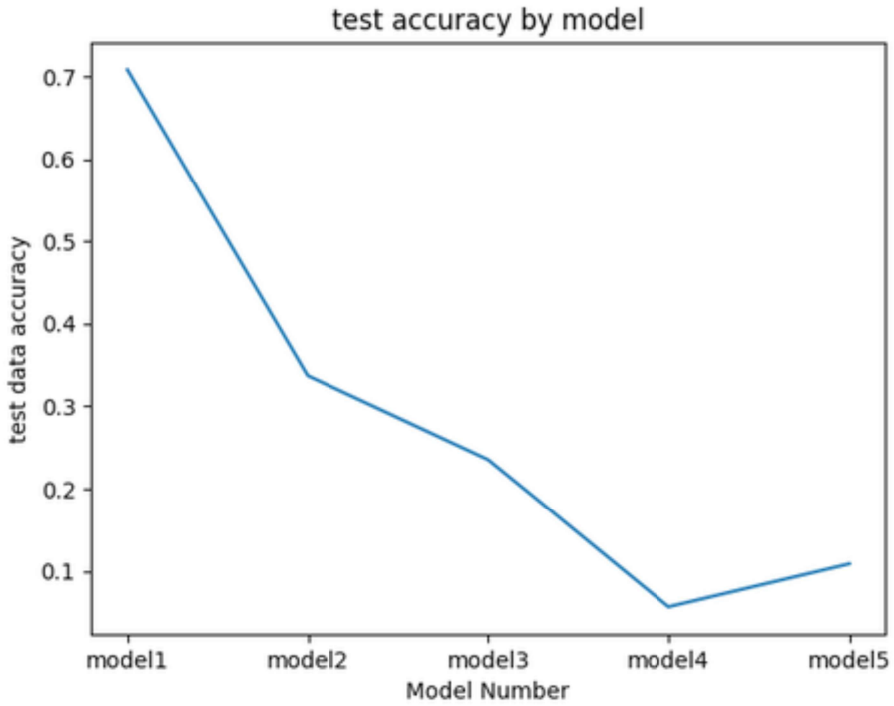

Figure 9: The test prediction accuracy by model

Figure 10: The training accuracy by model