Parin Patel

ID and Classify Clothing Items Based on Image

Introduction:

The goal of this lab is to identify if we can use algorithms to compute and identify clothing items. More specifically, we determine which algorithms and compute methodology results in the most efficient way for classifying simply fashion images.

Our Approach:

For our project, we will use the Zalando Research dataset from https://github.com/zalandoresearch/fashion-mnist. The dataset consists of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28 x 28 grayscale image, associated with 10 label-types of clothing. We then loaded our data using Tensorflow keras library. The training images are contained within the X_train file. The labels for each of these images are in the Y_train file. The X_test file contains the testing images, and the Y_test file contains their labels. The labels for both are listed below:

Labels:

- 0 T-shirt/top
- 1 Trouser
- 2 Pullover
- 3 Dress
- 4 Coat
- 5 Sandal
- 6 Shirt
- 7 Sneaker
- 8 Bag
- 9 Ankle boot

We will then normalize and visualize our data before running our models. We chose to run a Naive Bayes model, then a Decision Tree, and finally a Convolutional Neural Network to compute and classify fashion images.

Analysis:

Data Pre-Processing: Importing Datasets and Cleaning

We started by first loading our tensorflow datasets and then updating the label names from numeric integers to the labels stated above. Since each number corresponded with an associated

clothing type, it was an easy find and replace update. We then chose to review the shape of the data and could see that the dataset was split evenly, with both datasets having 6,000 cells. The X_train dataset had 28 columns. Finally we normalized our data and re-visualized the first 6 fashion images. We can see the difference these cleaning steps made by comparing Figure 1 below to Figure 2.

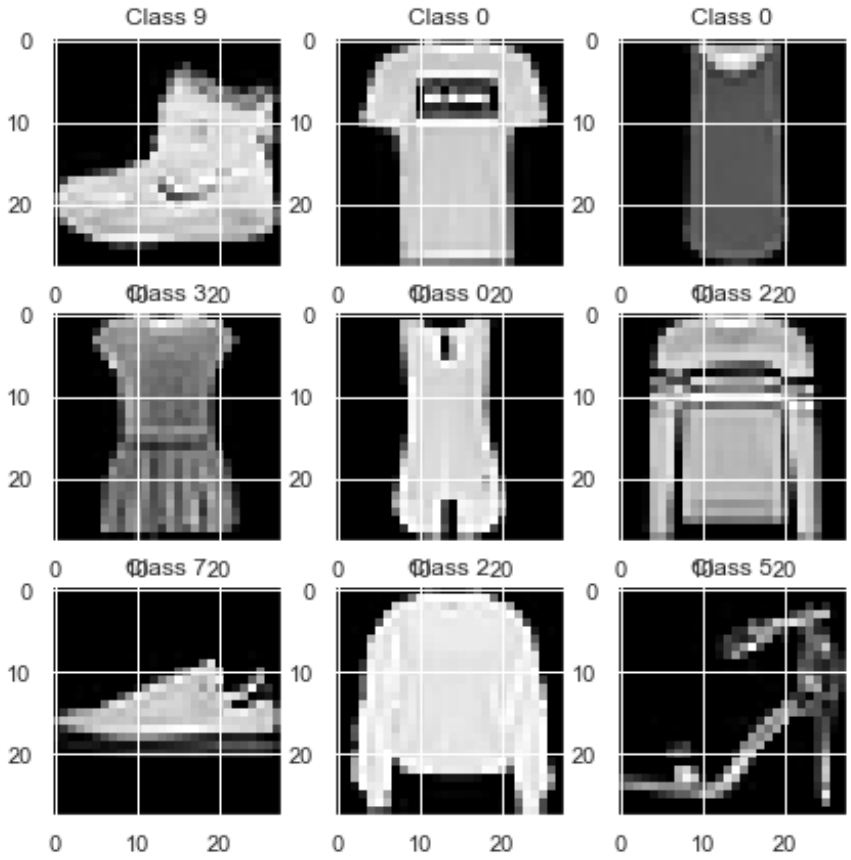

Figure 1

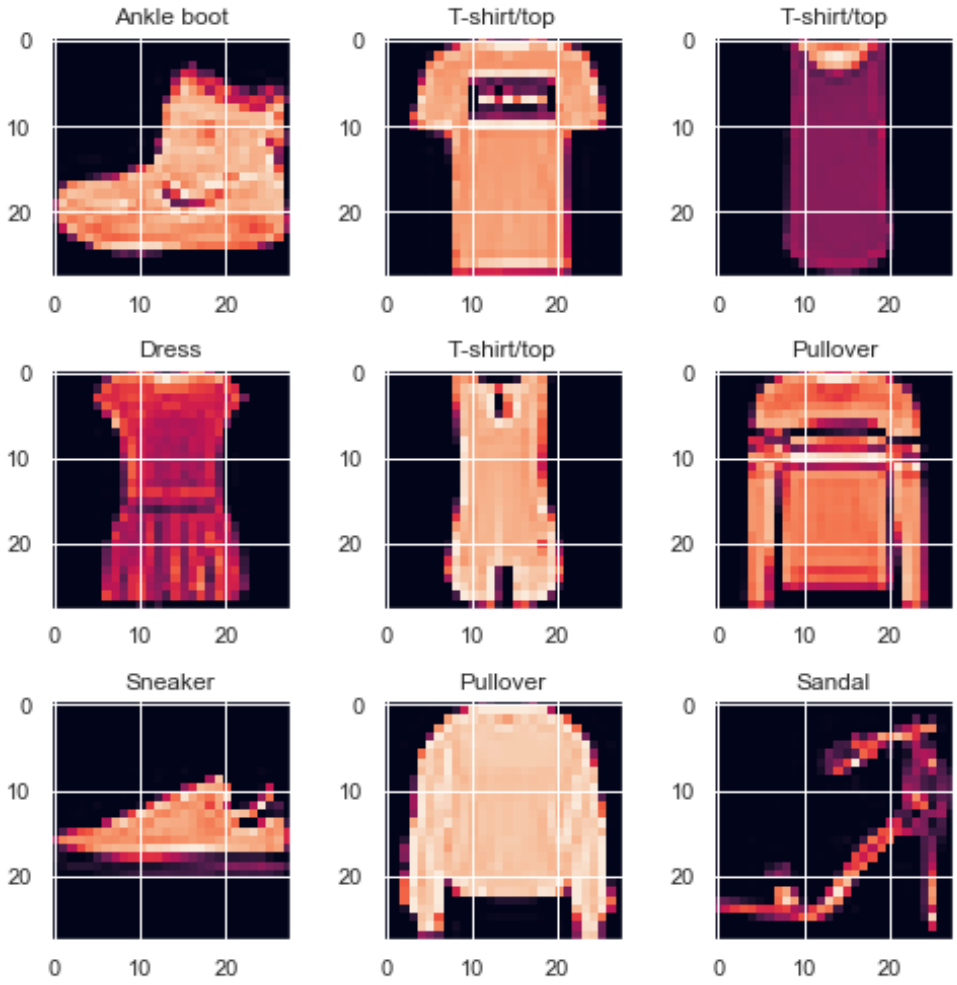

Figure 2

From our initial visualizations we can see that there is some some discrepancy between their image and their classification. For example, I am not sure what the middle image is, due to the poor quality of the image.

Model 1 : Gaussian Naive Bayes

Our first model will be a Gaussian Naive Bayes model. Unlike Classical Naïve Bayes, Gaussian supports a continuous valued features and models each as conforming to a Gaussian (normal) distribution. Therefore, from our data, we assume that each feature comes from a class-conditional univariate Gaussian distribution. Below, Figure 3 and 4, shows the results of our Naïve Bayes prediction model. Overall, while our model had a 59% accuracy, we can see that the highest precision categories are Bag, Ankle Boot, and T-Shirt/Top. We can also see a high degree of mis-categorized products. For example, there is a high degree of correlation between sandals and sneakers. In addition, we see a correlation between shirt and coat. This is likely due to issues regarding image quality.

.

	precision	recall	f1-score	support
T-shirt/top	0.81	0.59	0.68	1000
Trouser	0.64	0.94	0.76	1000
Pullover	0.59	0.32	0.42	1000
Dress	0.44	0.55	0.49	1000
Coat	0.38	0.78	0.51	1000
Sandal	0.93	0.28	0.43	1000
Shirt	0.32	0.04	0.07	1000
Sneaker	0.51	0.99	0.67	1000
Bag	0.83	0.71	0.77	1000
Ankle boot	0.91	0.67	0.77	1000
accuracy			0.59	10000
macro avg	0.64	0.59	0.56	10000
weighted avg	0.64	0.59	0.56	10000

Figure 3

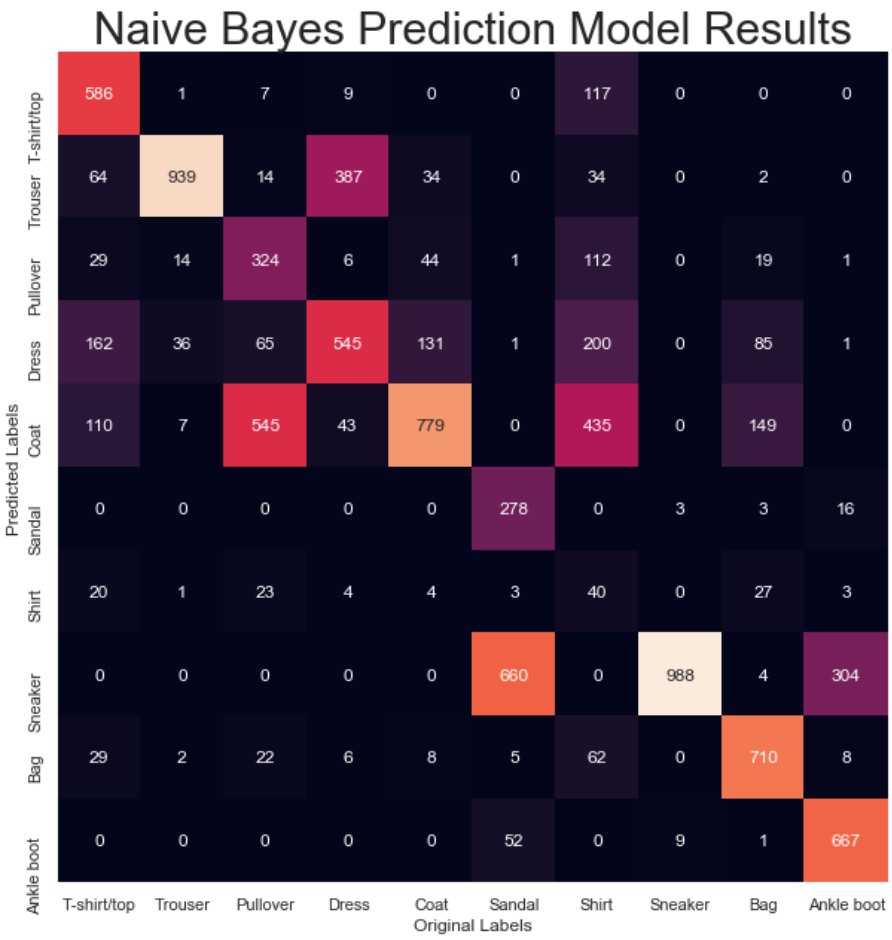

Figure 4

Model 2 : Decision Tree

We decided to make a decision tree for our second model using the scikit-learn-DecisionTreeClassifier. We chose this model because decision trees have several hyperparameters, which affect the accuracy of the learned model. We can tune these parameters to increase our accuracy. We start by preparing our data. Our rows and columns are set to parameters of 28. The classifier criterion used is the 'Gini' ratio and we have a minimum sample split of 2. The model results can be seen below in Figure 5 and 6. We overall see a higher accuracy than the first Naïve Bayes model. We are also able to see less mis-matched/wrongly matched categories. This is a good sign as we work to increase our model's effectiveness.

	precision	recall	f1-score	support
T-shirt/top	0.75	0.73	0.74	1000
Trouser	0.95	0.94	0.95	1000
Pullover	0.64	0.65	0.65	1000
Dress	0.81	0.78	0.79	1000
Coat	0.65	0.64	0.65	1000
Sandal	0.92	0.89	0.90	1000
Shirt	0.52	0.55	0.53	1000
Sneaker	0.88	0.89	0.88	1000
Bag	0.91	0.91	0.91	1000
Ankle boot	0.90	0.91	0.91	1000
accuracy			0.79	10000
macro avg	0.79	0.79	0.79	10000
weighted avg	0.79	0.79	0.79	10000

Figure 5

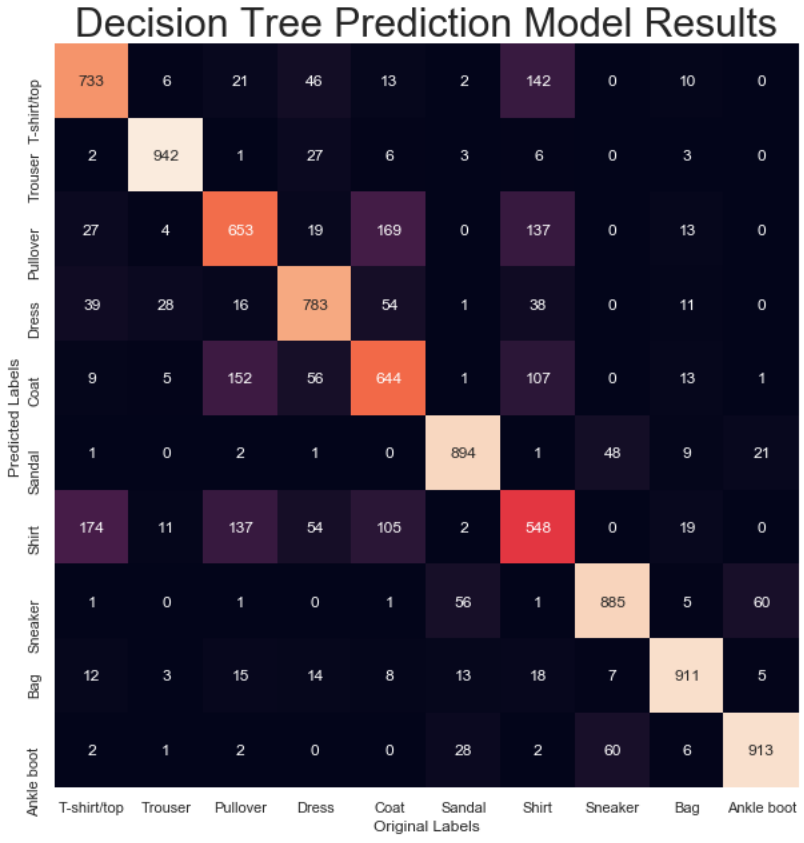

Figure 6

Model 3 : Convolutional Neural Networks

The third model we used was Neural Networks. Specifically, after testing out the accuracy of a normal neural network, we compared it also to the results of the Convolutional Neural Network (CNN). For this model, we initially had a lot of trouble running the CNN through the same TensorFlow package we ran the previous two models through. Specifically, the "Conv2D" function was not running, despite multiple updates and trying Google Co-Lab, in addition to Jupyter Notebook. We will explore this issue at a later time. Instead we decided to use the Sequential model within the keras package. This way takes more time to create but it works. We will run the Adams optimizer, like planned. We will run normal neural network first that does not have local connectivity and shared weights. We will then run our CNN to see if having weights and if having input from multiple units from the previous layer increases accuracy.

Because of the new package, we decided to reimport our data and re-scale it. We decided to format the input data layer by flattening the matrix into a 784-vecotr, so the network can accept it. So we reshaped our X_train data by dividing by 255 so the 60,000 28 x 28 matrices are now 60,000 784-length vectors. We did the same for the X_test data but the reshape resulted in the 10,000 28 x 28 matrices to now be 10,000 784-length vectors. We also then changed the integers to 32-bit floating point numbers, and then normalized each value for each pixel for the entire vector. Normalizing the input helps to place all additional dimensions on the same scale. We set our unique number of classes to 10 and then worked to define our model's architecture. For our sequential model, we set our activation as 'relu' which means we set the function that is applied to the output of the layer above as being with a "rectified linear unit" or 'relu' for short. We basically, by doing this, set all values below 0 to 0. We then protect our data from overfitting by setting our "Dropout" function to 0.2. We then decided to activate "SoftMax" to make sure the output is a valid probability distribution; therefore, the sum of all the values are all non-negative and will then sum to 1. We will again use the Adams optimizer when compiling our model. After training our model we can see that our normal neural network has only has an accuracy of 34.7%. We now run our CNN model.

To do this, we decided to reload our data and look at the image more closely. In figure 7 below we can see how each individual pixel a number has associated with the darkness of that pixel. These numbers are fed into the computer and then compared for image recognition. After reviewing the image more closely, we modified our classes to be in the one-hot format. So, 0 -> [1, 0, 0, 0, 0, 0, 0, 0, 0, 0], 1 -> [0, 1, 0, 0, 0, 0, 0, 0, 0, 0], 2 -> [0, 0, 1, 0, 0, 0, 0, 0, 0, 0], etc. Therefore, if the final output of our network is very close to one of these classes, then it is most likely that class. For example, if the final output is: [0, 0.94, 0, 0, 0, 0, 0.06, 0, 0] , then it is most likely that the image is the digit of 1.

Again, we now compile the keras model using the loss function called categorical cross-entropy, which is a loss function that compares two probability distributions. The optimizer helps to determine how quickly the model learns through gradient descent (which is the rate at which descents a gradient is called the learning rate).

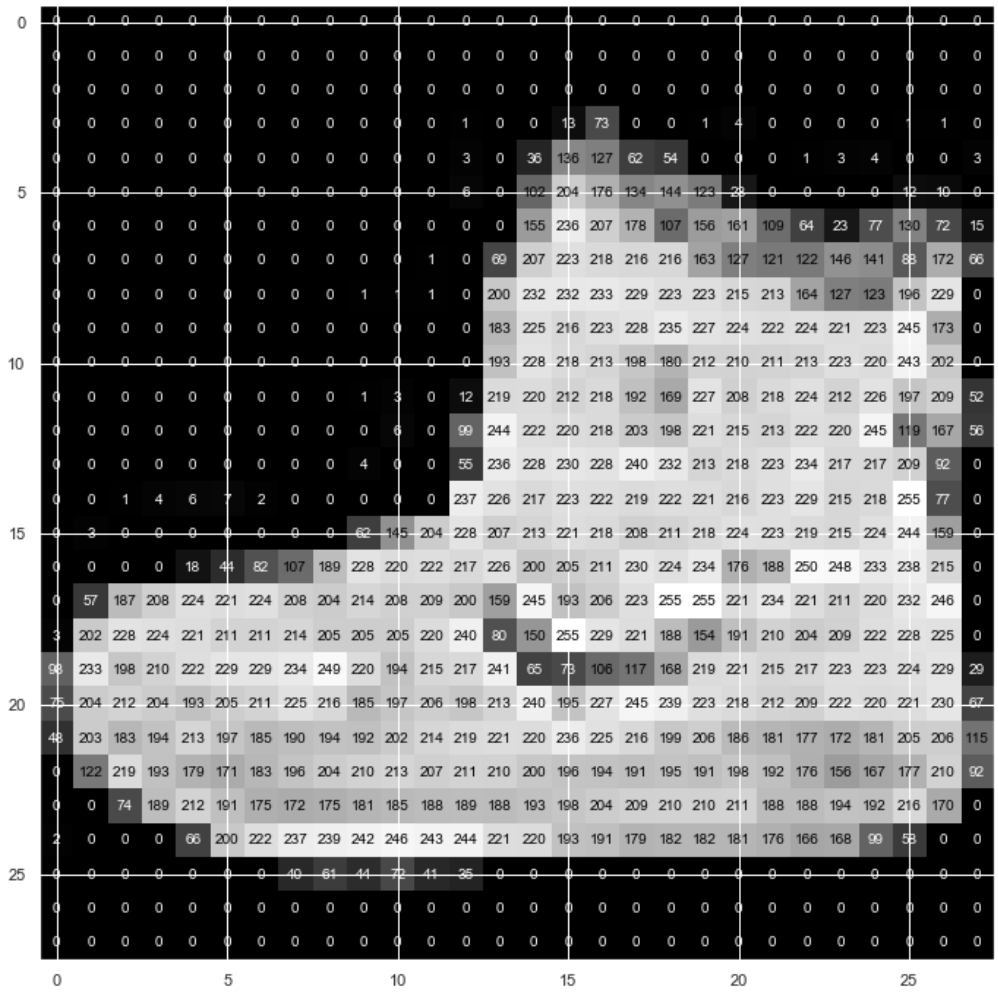

Figure 7

Specifically, we

```
# summarize the model
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 32)        4128
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 32)          4128
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 3, 3, 32)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 3, 3, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 288)               0
_________________________________________________________________
dense_4 (Dense)              (None, 500)               144500
_________________________________________________________________
dropout_4 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                5010
=================================================================
Total params: 158,086
Trainable params: 158,086
Non-trainable params: 0
_________________________________________________________________
```

Figure 8

From our above test model summary, we see that our test model accuracy is 1.67%. We then train our model which is fed data by the batch load. Finally, from our results, we see that our CNN has the highest accuracy rate with an accuracy of 89.89%

Results and Conclusion:

After analyzing three different models on the MINST fashion dataset, our best performing model is the final CNN with an accuracy of 89.89%. This is likely due to the use of the keras package instead of TensorFlow. In addition, we took additional pre-processing steps like re-sizing the pixels and modifying our classes to be in the one-hot format. There are many tradeoff of using one approach vs the other. Naïve Bayes is one of the simplest techniques and fastest technique for a wide variety of domains. However, this approach assumes conditional independence and makes the Naïve Bayes approach unable to using two or more pieces of evidence together. Decision Tree classifiers can cope with a combination of terms that can produce a strong accuracy of results. However, training a DT classifier can take more work, and can be

computationally exhaustive. Neural Networks are good for representing some of the more complex relationships between inputs and outputs. They can be used for data with a large number of domains. However, it's important to remember that neural networks need a larger amount of training data. In addition , while both a neural network and decision tree find non-linear solutions, and have interaction between independent variables, decision trees are better than neural networks when there is a large set of categorical values in the training data or when the scenario demands an explanation over a decision.

References:

https://medium.com/@tifa2up/image-classification-using-deep-neural-networks-a-beginner-friendly-approach-using-tensorflow-94b0a090ccd4

https://www.dtreg.com/methodology/view/decision-trees-compared-to-regression-and-neural-networks

https://arxiv.org/pdf/1008.3282.pdf
https://en.wikipedia.org/wiki/MNIST_database