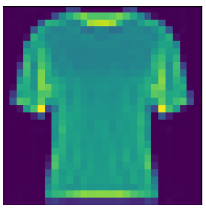
Jose Reyes
8.29.2020
IST 718



Lab 9: Fashion MNIST

Objective

Fashion-MNIST is a classic practice dataset for many data scientists to test several supervised and unsupervised learning techniques for their effectiveness. While deep learning has become the more popular option in recent years, many more-classical algorithms can still hold a strong position field.

In this report, four different approaches have been used to demonstrate how different algorithms can compute and identify clothing items in the Fashion-MNIST dataset. Each model is tested with the same dataset, and their varying accuracies and computation times are discussed and compared with each other.

Figure 1: Sample Images from Fashion MNIST dataset

Jose Reyes
8.29.2020
IST 718

Dataset

The Fashion-MNIST dataset consists of 60,000 training and 10,000 test images. Each image can be loaded as a 28 x 28 NumPy array with pixel values ranging from 0 to 255. All these images can be classified as one of the following 10 categories: T-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and boots.

Models

- Random Forest
- K Nearest Neighbors
- Convolutional Neural Network (CNN) from two different libraries: Keras and Pytorch

Each of the four models, except for K Nearest Neighbors, was trained on the 60,000 images. They were then tested for accuracy and computation time on the 10,000-image set.

The training and test datasets were loaded using built in functions from Keras and Pytorch.

```
#Keras Fashion MNIST Dataset
fashion_mnist = keras.datasets.fashion_mnist
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()

#Pytorch Fashion MNIST Dataset
train_set = torchvision.datasets.FashionMNIST("./data", download=True, transform=
                                              transforms.Compose([transforms.ToTensor
()]))
test_set = torchvision.datasets.FashionMNIST("./data", download=True, train=False, tran
sform=
                                             transforms.Compose([transforms.ToTensor(
)]))
```

Modelling

The first two models are Random Forest and K Nearest Neighbors (KNN) which are established sklearn machine learning classifiers. Random Forest is essentially a collection of decision trees and the average/majority vote of this ensemble approach is used as the predicted output for classification.

Three different Random Forest models with varying criteria and maximum depths were created and tested to compare for accuracy. The model with the default criterion (gini) and default maximum depth (None) produced the highest accuracy.

Jose Reyes
8.29.2020
IST 718

```
start_time = timeit.default_timer()

rf = RandomForestClassifier(criterion='entropy', max_depth=None)
rf.fit(x_train, y_train)

elapsed = timeit.default_timer() - start_time

rf_pred = rf.predict(x_test)
rf_accuracy = metrics.accuracy_score(y_test, rf_pred)

print('\n', 'elapsed time:', elapsed)
print("Accuracy score: {}".format(rf_accuracy))

elapsed time: 75.77741579999997
Accuracy score: 0.8793
```

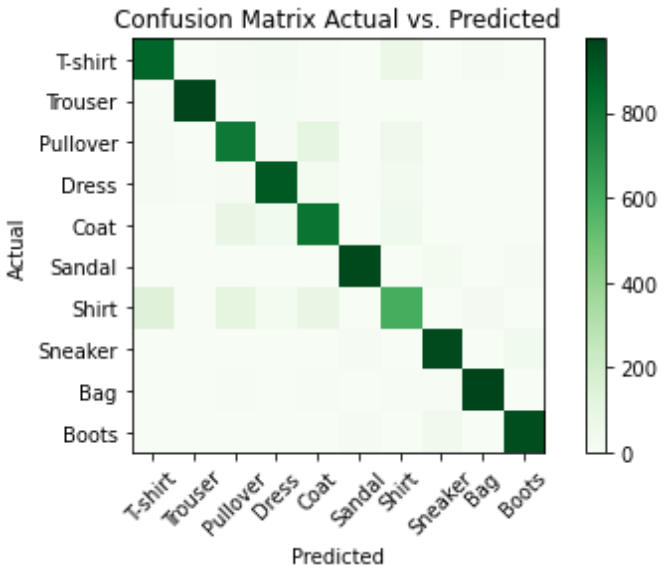


Figure 2: Results and Confusion Matrix for Random Forest Results

KNN identifies a number of each datapoint's closest neighbors (defined by k) and uses a majority vote to decide its predicted classification. Two different KNN models were created with k values of 3 and 5. Both models produced the same accuracy and similar computation times.

```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train, y_train)

start_time = timeit.default_timer()

knn_pred = knn.predict(x_test)
knn_accuracy = metrics.accuracy_score(y_test, rf_pred)

elapsed = timeit.default_timer() - start_time

print('\n', 'elapsed time:', elapsed)
print("Accuracy score: {}".format(knn_accuracy))
```

Jose Reyes
8.29.2020
IST 718

```
13.
14. elapsed time: 511.3606656000002
15. Accuracy score: 0.8793
```

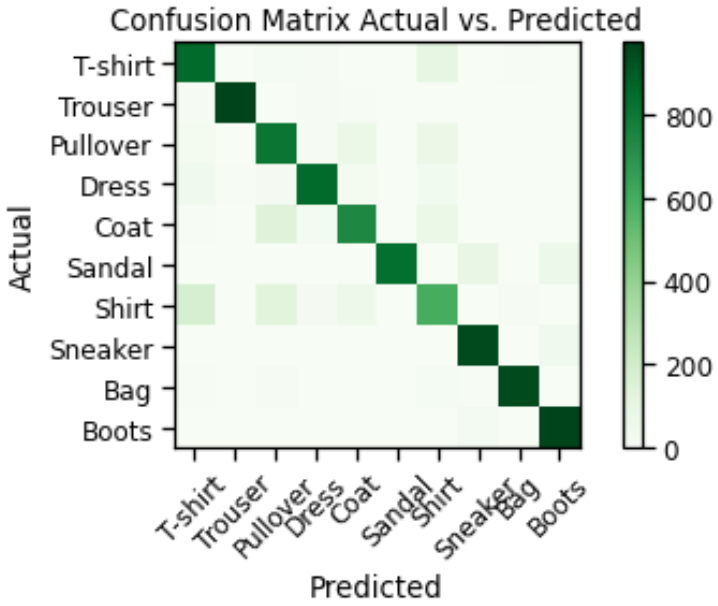


Figure 3: Results and Confusion Matrix for K Nearest Neighbors Results

Convolutional Neural Network (CNN)

CNN's are a class of deep neural network that is commonly used for visual imagery. Two CNN models were created from different open source libraries: Keras and Pytorch.

The Keras model layers used in this report were leveraged from the Medium article *Fashion-MNIST with tf.Keras* written by Margaret Maynard-Reid. Source: https://medium.com/tensorflow/hello-deep-learning-fashion-mnist-with-keras-50fcff8cd74a.

The model layers used are defined below:

```
1.  #Keras Model Layers
2.  Model: "sequential"
3.  _________________________________________________________________
4.  Layer (type)                 Output Shape              Param #
5.  =================================================================
6.  conv2d (Conv2D)               (None, 28, 28, 64)        320
7.  _________________________________________________________________
8.  max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0
9.  _________________________________________________________________
10. dropout (Dropout)             (None, 14, 14, 64)        0
```

Jose Reyes
8.29.2020
IST 718

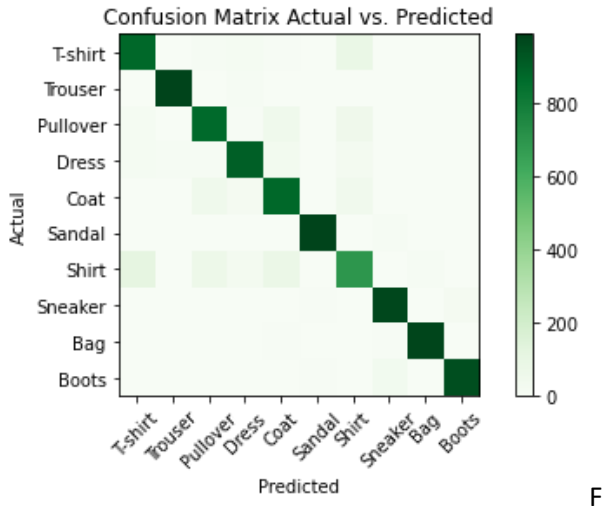
```
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 32)        8224
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 1568)              0
_________________________________________________________________
dense (Dense)                (None, 256)               401664
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                2570
=================================================================
Total params: 412,778
Trainable params: 412,778
Non-trainable params: 0
```

The model was trained on 60,000 images. After 10 cycles, the model was used on the test images.

```
score_cnn = model.evaluate(x_test, y_test, verbose=0)

print('\n', 'elapsed time:', elapsed)
print('\n', 'accuracy:', score_cnn[1])

elapsed time: 264.69924900000115
accuracy: 0.9075000286102295
```



F

Figure 4: Results and Confusion Matrix for CNN Model (Keras) Results

Lastly, a CNN model from the Pytorch library was created. The Pytorch model layers used in this report were leveraged from a Kaggle post *Fashion MNIST with Pytorch* written by Pankaj Joshi. Source: https://www.kaggle.com/pankajj/fashion-mnist-with-pytorch-93-accuracy.

The model layers used are defined below:

Jose Reyes
8.29.2020
IST 718

```
FashionCNN(
  (layer1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc1): Linear(in_features=2304, out_features=600, bias=True)
  (drop): Dropout2d(p=0.25, inplace=False)
  (fc2): Linear(in_features=600, out_features=120, bias=True)
  (fc3): Linear(in_features=120, out_features=10, bias=True)
)
```

Again, the model was trained on 60,000 images. After 6 cycles, the model was used on the test images.

Results and Accuracy per Category for CNN Model (Pytorch) Results

```
Iteration: 500, Loss: 0.48215094208717346, Accuracy: 87%
Iteration: 1000, Loss: 0.33429375290870667, Accuracy: 89%
Iteration: 1500, Loss: 0.3432767391204834, Accuracy: 88%
Iteration: 2000, Loss: 0.20704509317874908, Accuracy: 88%
Iteration: 2500, Loss: 0.13984020054340363, Accuracy: 89%
Iteration: 3000, Loss: 0.2067953497171402, Accuracy: 91%
elapsed time: 405.3803110000008

Accuracy of T-shirt: 89.50%
Accuracy of Trouser: 98.50%
Accuracy of Pullover: 80.40%
Accuracy of Dress: 94.10%
Accuracy of Coat: 87.80%
Accuracy of Sandal: 97.80%
Accuracy of Shirt: 69.20%
Accuracy of Sneaker: 97.40%
Accuracy of Bag: 99.10%
Accuracy of Boots: 96.80%
```

Jose Reyes
8.29.2020
IST 718

Overall Results:

Model Name	Accuracy	Computation Time (sec)
Random Forest	0.88	75.78 (train)
K Nearest Neighbors	0.88	511.36 (test)
CNN Model: Keras	**0.91**	**264.70 (train)**
CNN Model: Pytorch	**0.91**	**405.38 (train)**

Interestingly, the two classical machine learning methods produced the same accuracy. These accuracy results appear to be on-par with several other established sklearn classifiers and were significantly faster than many of them. It is possible that some of the hyperparameters can be adjusted to produce higher accuracies.

The CNN models from both the Keras and Pytorch libraries produced an accuracy of 91% with the Keras model having a faster computation time. These models were retrained several times on the same datasets with the same parameters and produced variable accuracies ranging from 90% to nearly 92%. It is highly likely that some of the model layers can be adjusted to produce even higher accuracies.

Across all four models, distinguishing between shirt vs. t-shirt and pullover vs. coat were the most difficult. Overall, the CNN models were clearly superior with the higher accuracies and their computation times were not significantly longer even without the aid of GPU processing (I need to switch my AMD to NVIDIA!)

Conclusion

Four models including Random Forest, K Nearest Neighbors, and CNN models from the Keras and Pytorch libraries were created and tested on the Fashion-MNIST datasets. This is a classic data set which can be used to test for effectiveness in both classical machine learning techniques and neural network architectures.

Overall, none of the models consumed a significant amount of computation time and power. The CNN models produced the highest accuracies of 91% while the more classical machine learning techniques produced an accuracy of 88%.