Laboratory Exercise – 3

IST 718 Big Data Analytics

Dan Caley

Table of Contents

Obtain

There was only 1 datasets used in performing this analysis:

1. The data can be found here https://github.com/zalandoresearch/fashion-mnist/tree/master/data/fashion
2. For this data set the keras.datasets can be used to access the fashion dataset.
3. Specifically the fashion dataset can be accessed by importing fashion_mnist from keras.atasets.
4. The data was then seperated into 4 arrays called x_train, y_train, x_test, and y_test.
5. This will explained further in model section.

MNIST Fashion Data

To obtain the MNIST Fashion Dataset did the following:

```
[8] # Bringing in the Fashion MNIST dataset
    from keras.datasets import fashion_mnist

[9] # Breaking the data into training and testing
    (x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

    Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
    32768/29515 [==============================] - 0s 0us/step
    40960/29515 [==============================] - 0s 0us/step
    Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
    26427392/26421880 [==============================] - 0s 0us/step
    26435584/26421880 [==============================] - 0s 0us/step
    Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
    16384/5148 [==============================================================================================] - 0s 0us/step
    Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
    4423680/4422102 [==============================] - 0s 0us/step
    4431872/4422102 [==============================] - 0s 0us/step
```

The data was then broken up into a train test set. This will be explained further in the modeling section.

About MNIST Data

- Fashion MIST is a dataset from Zalano's article image. In total 70,000 images of a wide variety of styles.
- The training data consists of 60,000 images each at 28 x 28 pixels.
- The training data consists of 10,000 images each at 28 x 28 pixels.
- The data set has 10 different styles.
    - T-shirt/top
    - Trouser
    - Pullover
    - Dress
    - Coat
    - Sandal

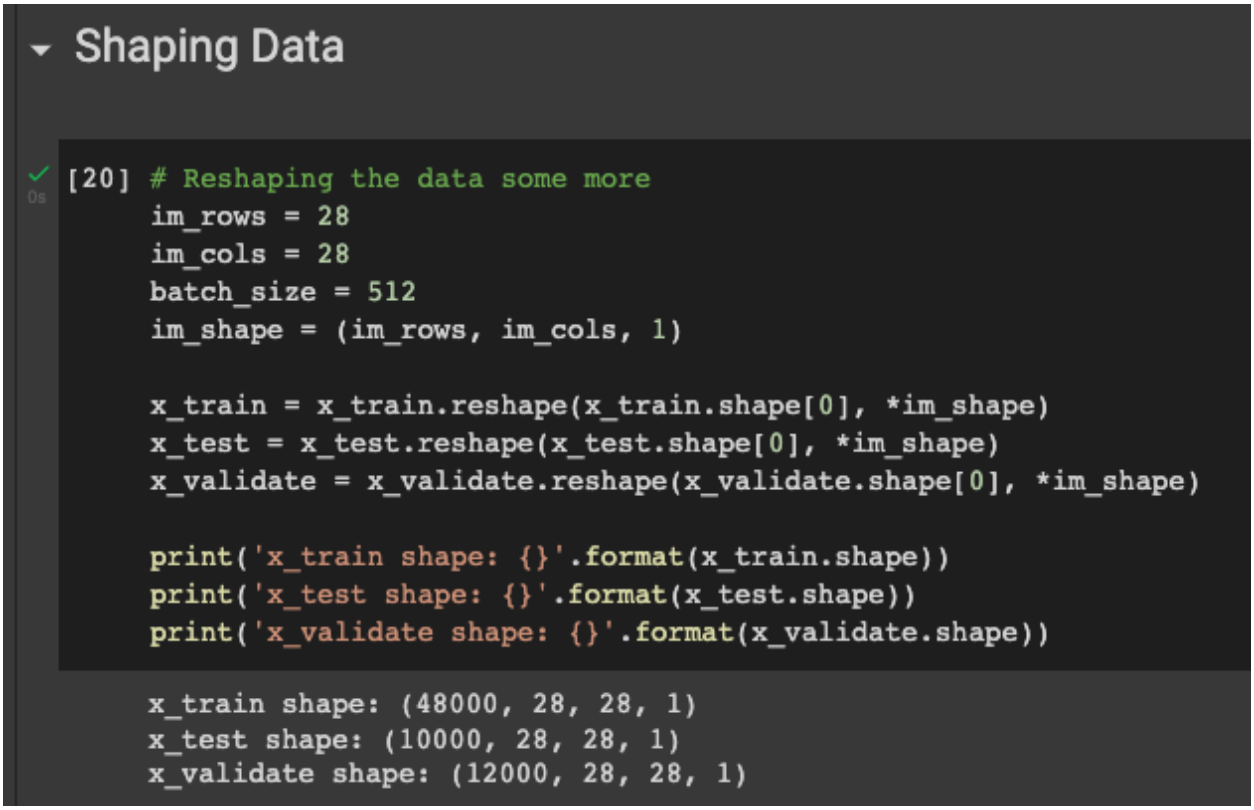
- Shirt
- Sneaker
- Bag
- Ankle boot

Scrub

Scrubbing and Shapping - cnn

In order to run the MNIST data set into a cnn model the following scrubbing and shaping methods were applied:

1. Reshaping the data was important in order to isolate for the rows and columns needed for each image.
2. A train and test set along with a validation set was created in order to build and identify accuracy of the cnn model.
3. The image below shows that the model will be trained on 48,000 data points, tested and 10,000, and validated on 12,000 for accuracy.



Scrubbing and Shapping - svm

In order to run the MNIST data set into a SVM model the following scrubbing and shaping methods were applied:

1. Reshaping the data was important in order to isolate for the rows and columns needed for each image.
2. A train and test set was created in order to build and identify accuracy of the SVM model.
3. For this data the train set has 60,000 data points where the test has 10,000 data points.
4. A 2 x 2 array needs to be created rather than an array greater than 2 values for svm to work.

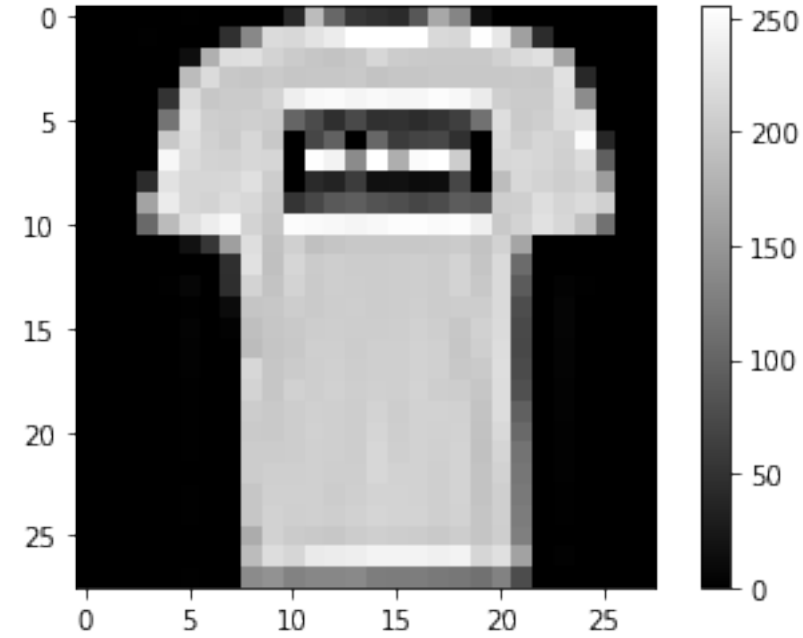
```
[21] # Breaking the data into training and testing
     (x_train_svm, y_train_svm), (x_test_svm, y_test_svm) = fashion_mnist.load_data()

     x_train_svm = np.array(x_train_svm, dtype=np.float32) / 255.0
     x_test_svm = np.array(x_test_svm, dtype=np.float32) / 255.0

     # Creating the seed
     random_state = check_random_state(3)

     # Training Data Need Shuffling
     permutation = random_state.permutation(len(x_train_svm))

     x_train_svm = x_train_svm[permutation]
     y_train_svm = y_train_svm[permutation]

     # For the svm models shaping the image data.
     x_train_svm = x_train_svm.reshape(x_train_svm.shape[0], -1)
     X_test_svm = x_test_svm.reshape(x_test_svm.shape[0], -1)

     # For the svm models shaping the label data.
     y_train_svm = np.array(y_train_svm, dtype=np.int32)
     y_test_svm = np.array(y_test_svm, dtype=np.int32)
```

Explore

MNIS Data Set

To get an idea at what type of fashion images and the quality of the image the following code can show us a visualization.

```
# Plotting a figure
plt.figure()
plt.imshow(x_train[1], cmap='gray')
plt.colorbar()
plt.grid(False)
plt.show()
```



With the following cleaning techniques a 9x9 image can be given with the labels associated to them.

```
[15] # Creating Label names to apply to graphics
     label_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                    'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

```
     # Creating a plot for the first 9 images in the data set with their associated labels
     plt.figure(figsize=(12,12))

     # Looping through the first 9
     for i in range(9):
       plt.subplot(3,3,i+1)

       # Removing the axis ticks
       plt.xticks([])
       plt.yticks([])

       # Removing gridlines
       plt.grid(False)

       # Showing the images
       plt.imshow(x_train[i], cmap=plt.cm.binary)

       # Adding associated label names
       plt.xlabel(label_names[y_train[i]])

     plt.show()
```

From the naked eye these images can be identified as the labels given. For an individual to do this on the entire dataset would take too long and other objects might be introduced later.

Modeling

K-Nearest Neighbors

The steps to create the K-Nearest Neighbors model is the following:

- Define the Model
- Create the Model
- Fit the Model

Explaining pieces of the code are as followed:

- Passing a kernel of 3 allows for an image to pass over each image to then identify the best fit.
- Kernal's can be adjusted to identify better fit.
- Here we will start with 3
- MaxPooling allows you to down sample.
- What down sampling is performing is that it's shrinks the height and width down to 2.
- Adding a dense layer creates a posterior probability.

```
[ ] cnn_model = Sequential([
                          Conv2D(filters=32, kernel_size=3, activation='relu', input_shape=im_shape),
                          MaxPooling2D(pool_size=2),
                          Dropout(0.2),
                          Flatten(),
                          Dense(32, activation='relu'),
                          Dense(10, activation='softmax')
    ])
```

For the creating steps consist compiling the model:

```
cnn_model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=Adam(lr=0.001),
    metrics=['accuracy']
)
```

Then fitting the model are as followed:

```
cnn_model.fit(
    x_train, y_train, batch_size=batch_size,
    epochs=10, verbose=1,
    validation_data=(x_validate, y_validate)
    #,callbacks=[tensorboard]
)
```

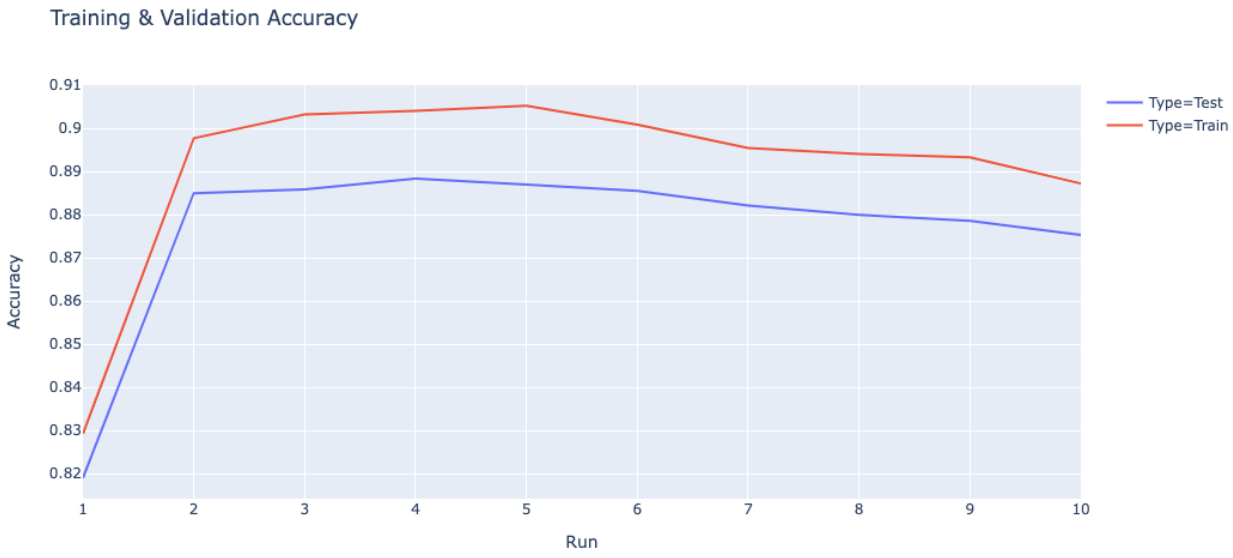
The results based on a kernel size of 3.

```
Test
-------------------
test loss: 0.3125
 test acc: 0.8894
-------------------
Train
train loss: 0.2662
 train acc: 0.9042
-------------------
```

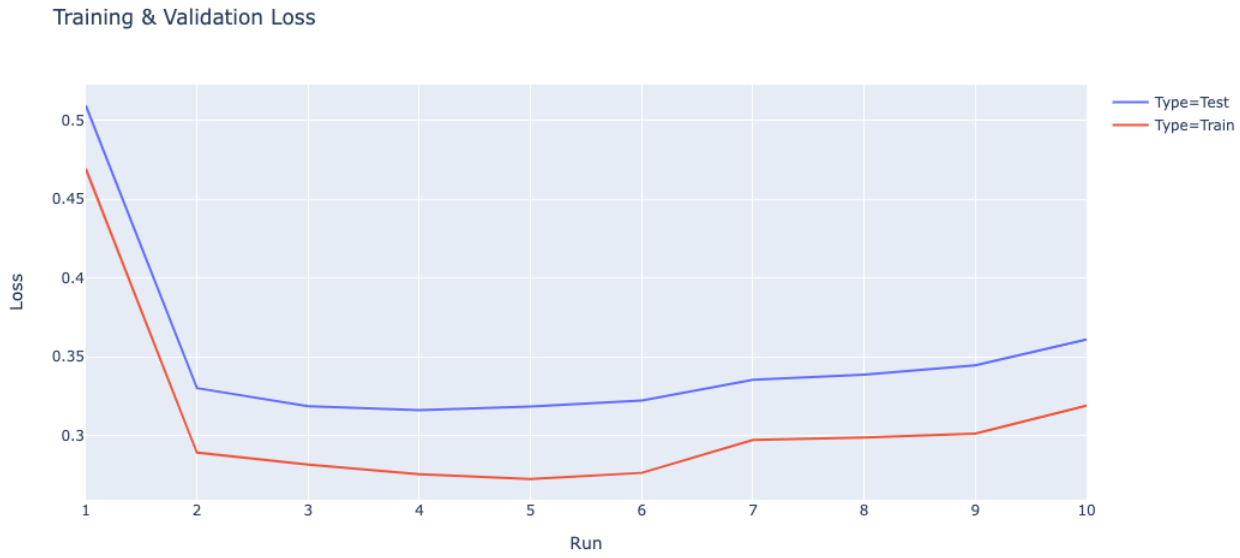
The test set was able to achieve an accuracy score of 88.9% but with a loss of 31.25% a 5% difference from the train set.

K-Nearest Neighbors – Compute Performance

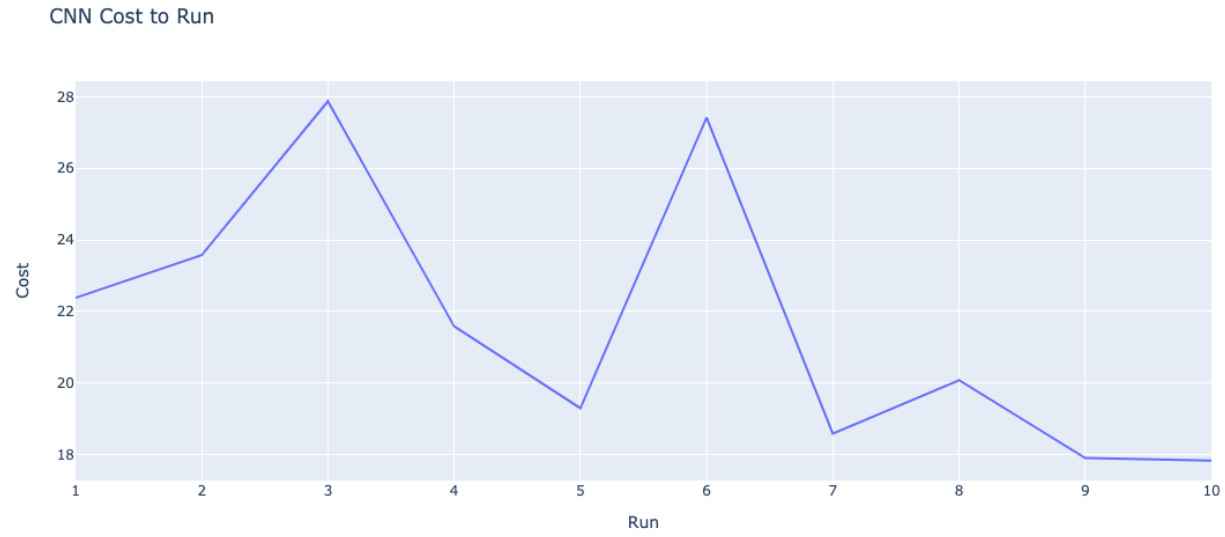
The graph below shows that the jump from 1 kernel to 2 is the greatest boost to the model. Each additional run after any additional kernel greater than 5 decreases the models accuracy.



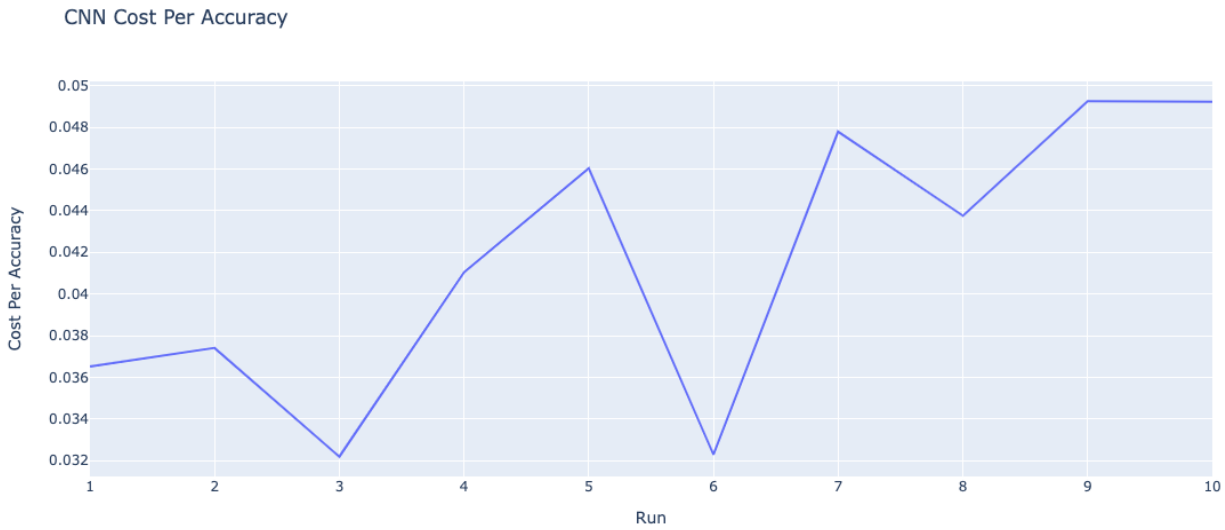
The Loss looks suspiciously similar to the graph above but just reversed. Meaning the greatest decrease is from kernel 1 to 2 and then marginally improve until after 5.



The cost of the model is not that much with coming in with only taking 28 seconds at most to run. Regardless many industry might need to the model to run quickly. Therefor having 6 kernals is the second best in accuracy and the second best in the cost to running the model. Although running 5 kernals has a higher accuracy the cost is the second highest in the model.



Said differently looking at the Cost per accuracy having 3 kernals is the best but 6 comes in second with a marginally better accuracy. Doing a Cost per accuracy helps normalize the data and evaluate how much juice a model builder is getting. Some instance there are minimum thresholds for cost but here an extra second might not be an issue.



SVM

The steps to create the SVM model is the following:

- Define the Model
- Fit the Model
- Create the Model

In any model building the following has to be performed. The difference here though is that we need to Fit the model before creating the model. For SVM we define the parameters by creating a function. Then look at 5000 values in both the x_train and y_train.

Another difference is that the K-Nearest neighbor had 3 values for the array. For SVM we just need to. The scrubbing piece was updated to take on SVM models along with the potential for other models.

Why we need to Fit the model first is that we identify what the parameters that need to be inputted in. To do this we first build a parameter function do then insert into the model fit.

```
[52] def param_query(x,y):
       # Creating a grid of parameters
       param_grid = {'C': [0.1,1, 10, 100, 1000], 'gamma': [1,0.1,0.01,0.001,0.0001], 'kernel': ['rbf']}

       # Instantiation of GridSearchCV: estimater (SVM), grid, and a verbose (text output)
       grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=3,n_jobs=-1)

       # Fitting the model
       grid.fit(x,y)

       # Printing the best output
       print(grid.best_params_)
```

Then we create the parameter by running this code:

```
# Subsetting the first 5000 points data to feed through GridSearch to find the best parameters
x = x_train_svm[:5000]
y = y_train_svm[:5000]

param_query(x,y)
```

This returns C=10, gamma=0.01, verbose=3, kernel='rbf'

This is inputted into the SVC to create the model then used to fit the model.

```
# Define the model
model = SVC(C=10,gamma=0.01,verbose=3,kernel='rbf')

# Fit the model
model.fit(X_train,y_train)
```

The final part is to create the accuracy for the model. I created a df_accuracy data frame to input more models like Neural Network. This will have to be explored in future projects.

The accuracy of the model:

	Accuracy	Model
0	0.8999	SVM

SVM – Compute Performance

The compute performance is quite costly to perform this model. In the K-Nearest Neighbor max the cost was 20 seconds to run the model. In total to see 10 kernels that would take less than 3 minutes.

SVM takes 23 minutes to run, a 6,800% increase over K Nearest neighbor. Due to how computationally intensive creating the parameters take, along with fitting the model and then outputting the model SVM is not the best model to go with for this instance. The accuracy is marginally lower therefore the accuracy over time to run isn't the same value as cnn.