Introduction

For this project, I used an open-source neural network library called *Keras* to help me create a 2-Layer Convolutional Neural Network that is capable to tell the difference between different articles of clothing as provided by the Fashion-MNIST data set (https://github.com/zalandoresearch/fashion-mnist). In the end, the model itself could achieve consistent accuracy scores on the test data set of higher than 90%, a higher score can easily be achieved with a larger epoch and batch size values.

Inspiration

I was heavily inspired by both the Lecture 8 slides/videos as well as this tutorial found online by Aditya Sharma (https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python). Aditya's tutorial allowed me to have an understand of what *Keras* was doing behind the scenes and how I would implement my own CNN.

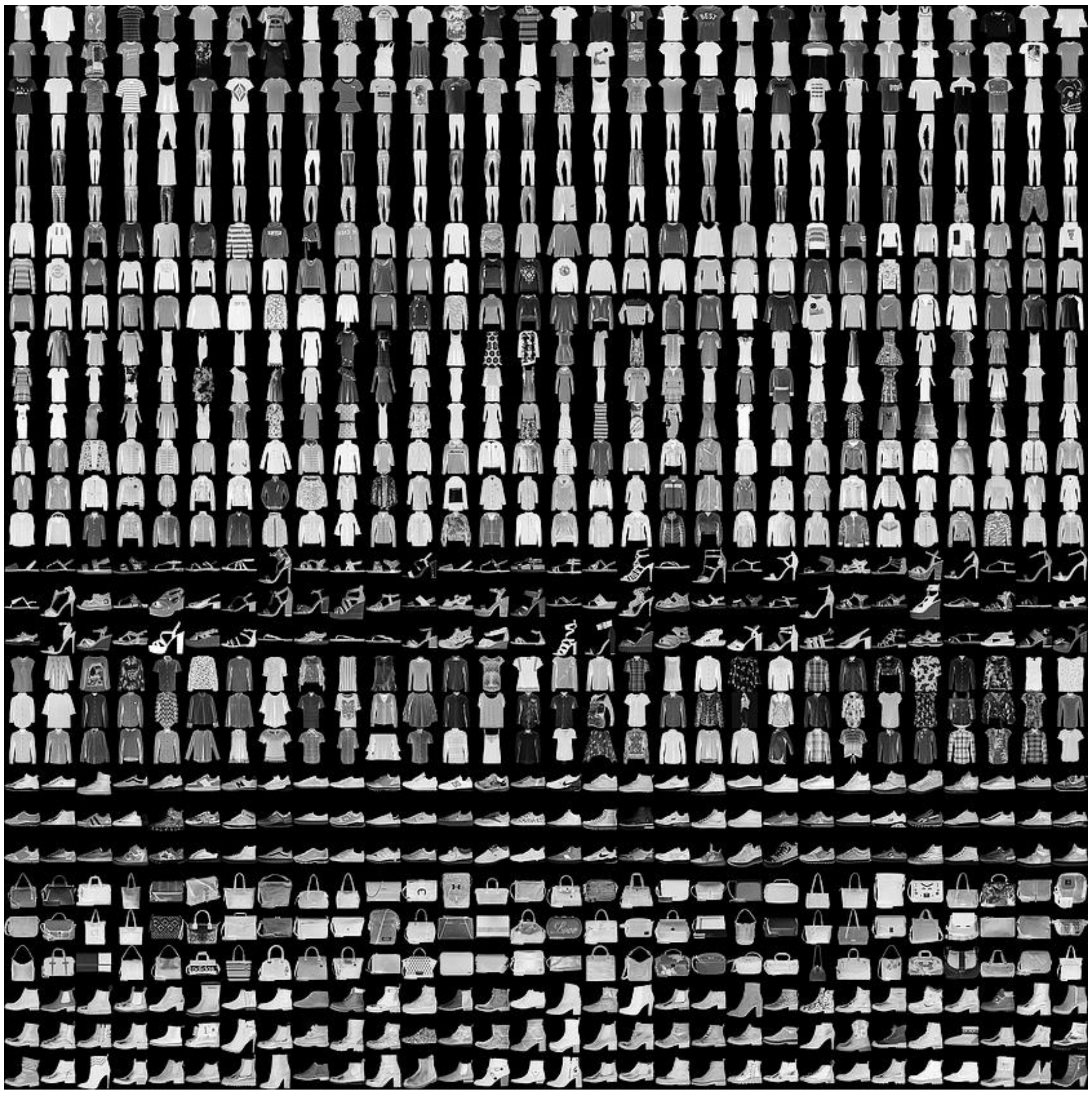
Data Set

The data set that I used can be found with the following links: https://arxiv.org/abs/1708.07747 and https://github.com/zalandoresearch/fashion-mnist .

The data set is comprised of 70,000 28x28 grayscale pictures of 10 different kinds of articles of clothing. The classes are numbered 0-9, and this is what they mean:

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

The CNN I have created aims at looking at these pictures and correctly identifying its class, or say, what kind of article of clothing the computer is looking at.



Source: https://github.com/zalandoresearch/fashion-mnist/raw/master/doc/img/fashion-mnist-sprite.png

The python script takes in these 28x28 images and does some processing on these images before hitting the CNN. The image's pixels' intensity is converted down from 0 to 255 to a 0 to 1 range and given and extra 3rd dimension of 1 in order to properly be fed into the network. The input labels are then converted to one-hot lists where instead of being a single digit representing what article of clothing, it is actually a list of length of the number of classes for each label, where the index of the only one in the list indicates the class of that data input image. The training data is the split between training and validation data, and once that is done, it is ready to be placed into the CNN.

Algorithm

The network itself has the following layers:

Input (28x28x1)

Layer 1: Convolution 32-(3x3) with linear activation → Max Pooling (2x2) → Leaky ReLu

Layer 2: Convolution 128-(3x3) with softmax activation → Max Pooling (2x2) → Leaky ReLu → Flattening Layer → Dense Layer (128)

Output (10)

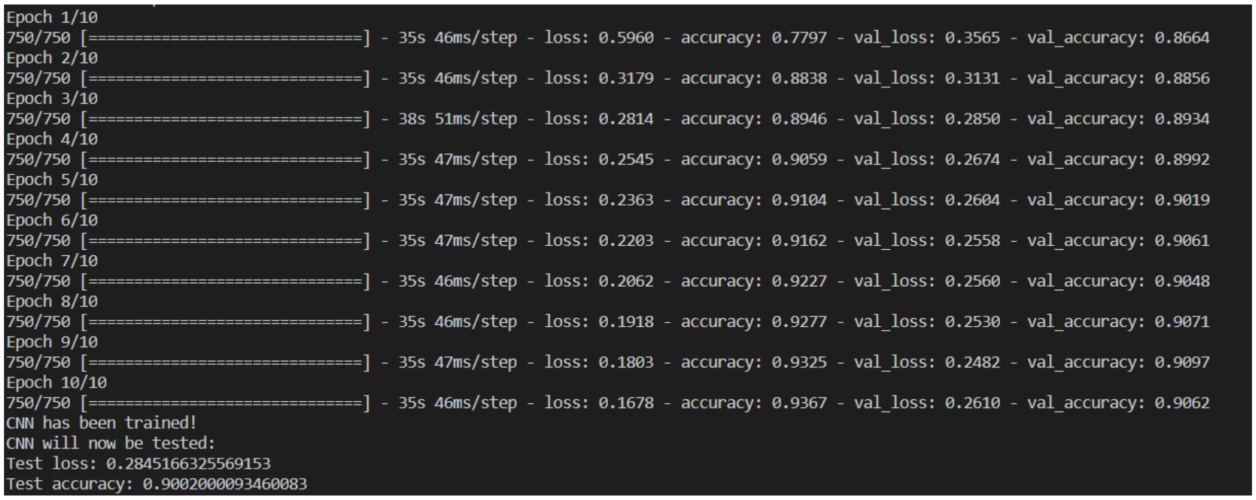
The input data of the image goes through its first convolution layer that has 32 (3x3) filters, and a linear activation function. This then goes through (2x2) max pooling and then a leaky ReLu activation function, which is different from a normal ReLu activation function as it avoids "dying" (aka staying 0 forever). The second layer is like the first, but the convolution layer has 128 (3x3) filters with a softmax activation function. It then also goes through another year of max pooling (2x2) and a Leaky ReLu activation function. The flattening layer get's all the data outputs from the last layers into one dimension to be easily hooked up the last dense layer that has 128 units with a linear activation function. Those 128 units are then connected to the output layer that has 10 units with a softmax activation function. The unit with the highest value would then be the predicted class for the input image.

Output from model.summary

```
Model: "sequential"

_________________________________________________________________

Layer (type)              Output Shape           Param #

=================================================================

conv2d (Conv2D)            (None, 28, 28, 32)      320

_________________________________________________________________

leaky_re_lu (LeakyReLU)    (None, 28, 28, 32)      0

_________________________________________________________________

max_pooling2d (MaxPooling2D) (None, 14, 14, 32)     0

_________________________________________________________________

conv2d_1 (Conv2D)          (None, 14, 14, 128)     36992

_________________________________________________________________

leaky_re_lu_1 (LeakyReLU)   (None, 14, 14, 128)     0

_________________________________________________________________
```

```
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 128)        0

_________________________________________________________________

flatten (Flatten)          (None, 6272)           0

_________________________________________________________________

dense (Dense)              (None, 128)            802944

_________________________________________________________________

leaky_re_lu_2 (LeakyReLU)    (None, 128)            0

_________________________________________________________________

dense_1 (Dense)            (None, 10)             1290

=================================================================

Total params: 841,546

Trainable params: 841,546

Non-trainable params: 0
```

Example console output when training model



```
Epoch 1/10
750/750 [==============================] - 35s 46ms/step - loss: 0.5960 - accuracy: 0.7797 - val_loss: 0.3565 - val_accuracy: 0.8664
Epoch 2/10
750/750 [==============================] - 35s 46ms/step - loss: 0.3179 - accuracy: 0.8838 - val_loss: 0.3131 - val_accuracy: 0.8856
Epoch 3/10
750/750 [==============================] - 38s 51ms/step - loss: 0.2814 - accuracy: 0.8946 - val_loss: 0.2850 - val_accuracy: 0.8934
Epoch 4/10
750/750 [==============================] - 35s 47ms/step - loss: 0.2545 - accuracy: 0.9059 - val_loss: 0.2674 - val_accuracy: 0.8992
Epoch 5/10
750/750 [==============================] - 35s 47ms/step - loss: 0.2363 - accuracy: 0.9104 - val_loss: 0.2604 - val_accuracy: 0.9019
Epoch 6/10
750/750 [==============================] - 35s 47ms/step - loss: 0.2203 - accuracy: 0.9162 - val_loss: 0.2558 - val_accuracy: 0.9061
Epoch 7/10
750/750 [==============================] - 35s 46ms/step - loss: 0.2062 - accuracy: 0.9227 - val_loss: 0.2560 - val_accuracy: 0.9048
Epoch 8/10
750/750 [==============================] - 35s 46ms/step - loss: 0.1918 - accuracy: 0.9277 - val_loss: 0.2530 - val_accuracy: 0.9071
Epoch 9/10
750/750 [==============================] - 35s 47ms/step - loss: 0.1803 - accuracy: 0.9325 - val_loss: 0.2482 - val_accuracy: 0.9097
Epoch 10/10
750/750 [==============================] - 35s 46ms/step - loss: 0.1678 - accuracy: 0.9367 - val_loss: 0.2610 - val_accuracy: 0.9062
CNN has been trained!
CNN will now be tested:
Test loss: 0.2845166325569153
Test accuracy: 0.9002000093460083
```