ABSTARCT for the Project Fashion MNIST Data Classifier

In this project, we have built a fashion apparel recognition using the Convolutional Neural Network (CNN) model. To train the CNN model, we have used the Fashion MNIST dataset. After successful training, the CNN model can predict the name of the class given apparel item belongs to. This is a multiclass classification problem in which there are 10 apparel classes the items will be classified.

The fashion training set consists of 70,000 images divided into 60,000 training and 10,000 testing samples. Dataset sample consists of 28x28 grayscale images, associated with a label from 10 classes.

So the end goal is to train and test the model using Convolution neural network.

OBJECTIVE

This work is a part of my experiment with fashion –MNIST dataset using various machine learning algorithm/models.

The objective is to identify different fashion product from the given images using various best possible machine learning models(algorithm) and compare their results (performance measures) to arrive at the best ML model.

The prime objective of this article is to implement a CNN to perform image classification on the famous fashion MNIST dataset. In this, we will be implementing our own CNN architecture. The process will be divided into three steps: data analysis, model training, and prediction.

Understanding And Analysing The Dataset

Fashion MNIST Training dataset consists of 60,000 images and each image has 784 features (i.e. 28×28 pixels). Each pixel is a value from 0 to 255, describing the pixel intensity. 0 for white and 255 for black.

The class labels for Fashion MNIST are:

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

Introduction To The `Project Fashion MNIST Data Classifier`

Deep learning is a subfield of machine learning related to artificial neural networks. The word deep means bigger neural networks with a lot of hidden units. Deep learning’s CNN’s have proved to be the state-of-the-art technique for image recognition tasks. Keras is a deep learning library in Python which provides an interface for creating an artificial neural network. It is an open-sourced program. It is built on top of Tensorflow.
Classification is one of the main problems in machine learning. In this project, a fashion-MNIST dataset, which includes figures of ten different fashion classes, is classified through various machine learning methods using TensorFlow, Keras and scikit-learn.
The input to our algorithm is a greyscale fashion image, for example, shoes, t-shirt, dress, etc.. We then use different machine learning models, such as CNN, ResNet, VGG, random forest, KNN, SVM, and Adaboost to output a predicted class for it.

Related Work

Resnet is proposed to solve the common issues in training deep neural networks, Gradient Vanishing and curse of dimensionality problems. The model has achieved great success on image classification problems. This model makes training deep models much easier than before. It works especially well when the input data is not so complicated while the network is big. Resnet model has many different implementations from 18 layers to 152 layers. The number of layers is extremely large when Resnet was proposed. It successfully trained the 152-layer model with its residual block design.

Dataset and Features

The dataset we used for our project is the Fashion-MNIST dataset provided by Kaggle. Within our dataset, we have 10 different categories and 70,000 unique images, including a training set of 60,000 images and a test set of 10,000 images. Each of the image is a 28 by 28 greyscale image. We also tried standarding the data to make it between a minimum value of 0 and a maximum value of 1. But this pre-processing step did not improve the result. We think the reason is that we used batch normalization in our models, which already made use of standardizing data.

Methods

1. Convolutinal Neural Network (CNN)

CNN [Simard et al., 2003] has been proved to be the most successful image classifier. Its capability to extract features from input regardless of space information makes it immune to the position of a key object in the image. The convolutional neural network uses a series of filter layers to constantly extract useful features and reduce dimensions on raw data, the resulting data will then be evaluated for final classification to certain node (category).

2. ResNet

Our next model is Resnet. Resnet has achieved huge success in image classification tasks. The design of this model is driven by a simple intuition. The bigger (more layers and more parameters) the model is, the better it should be. However, in practice we often witness the fact that complicated networks generate poor results than simple networks. The reason lies in that weights in early layers of the model is not well updated by the back-propagated gradients. Resnet addresses this issue by adding a shortcut at between certain layers. Because our task is to classify 28x28x1 images which are relatively small and we found VGG works worse than a simple and shallow CNN, we want to see if Resnet can solve the problem. In the Section 4, we introduced our result and gave our analysis on Resnet.

3. VGG

In this experiment, we used VGG16[1] as a method for classifying our data. VGG16 is a deep and simple network, that uses 16 convolution filters with the kernel size of 3 for all, and 5 maxpooling for increasing the effective receptive field. When the convolutions and maxpooling is done, it appends 3 fully connected layers as classifiers, to detect the class of the input data. VGG16 gives better results compared to some networks like ZFNet. Since it is using more convolution layers, its effective receptive field is bigger, which means it can extract more feature from the data. This also means that it uses more activation functions,that causes VGG16 to be more discriminant.

4.Conventional Machine Learning Methods

Apart from CNN, ResNet and VGG, some conventional machine learning methods are also applied in this classification problem, including Random Forest, k-Nearest Neighbor (kNN), Support Vector Machine (SVM) and Adaboost.

Methodology

A comprehensive description of all algorithms used in this project follows:

1. Preprocessing

Before the data can be classified it is often desirable to undergo two steps of preprocessing: normalization and dimensionality reduction. Normalization often converts the data to a scale between -1 and 1. In the case of pixel values between 0 and 255, a simple division by 255 is enough to normalize the data. Second, the data can undergo dimensionality reduction. In many instances of pattern recognition, one may be provided many features (dimensions) for each data point. This is often referred to as the curse of dimensionality. With the addition of each dimension, we 3 need exponentially more training data to obtain a truer understanding of the data. Also, with more dimensions, we risk overfitting and using features that in actuality are not as important. In this paper, we investigate two methods of reduction, including Fisher's Linear Discriminant and principal component analysis.

Two methods of dimensionality reduction were implemented in this project:

- Fisher's Linear Discriminant: FLD is a supervised approach which uses the training data set to establish a projection matrix W that will best discriminate the testing data. Mathematically, given c classes with d dimensions, we reduce the number of dimensions to c − 1 dimensions. The equation in (1)

$$\begin{aligned} S_W &= \sum_{k=1}^{K}(x-m_k)(x-m_k)^T \\ S_B &= \sum_{k=1}^{K} N_k(m_k-m)(m_k-m)^T \qquad (1) \\ W &= \max_D(eig(S_W^{-1}S_B)) \end{aligned}$$

  was used to calculate W.

- Principal Component Analysis: PCA is an unsupervised approach that aims to minimize information loss. The projection matrix P can be calculated as seen in (2). The number of dimensions m is chosen so that it does not exceed a given maximum error.

$$\begin{aligned} \Sigma_{x,dxd} &= \mathrm{cov}(X) \\ E_{dxd} &= \mathrm{eig}(\Sigma_{x,dxd}) \qquad (2) \\ P_{dxm} &= \begin{bmatrix} e_1 & e_2 & e_3 & \ldots & e_m \end{bmatrix} \end{aligned}$$

In both FLD and PCA, the projection vector W and basis vectors P are found using the training set and are then applied to the testing set.

2.Bayesian Discriminant Functions

After the data is preprocessed, classification can be performed. The first classification method explored in this paper are the Baysian discriminant functions gi(x), in which a given test sample's feature vector x are passed through functions for each class and the outputs are compared. Namely, the feature vector x is assigned to class i over class j if gi(x) > gj (x). In this project, we use the discriminant function seen in .

gi(x) = p(x|ωi) + ln P(ωi) … (3)

Three discriminant functions were developed, each with different assumptions. This report will refer to them as Case I, Case II, and Case III. Case I represents the most simplified case, in which all of the following three assumptions were made.

3.Non-Parametric Learning

K-nearest neighbors (kNN) performs classification without assuming a model. With kNN, a test sample's Euclidean distance from each training sample is measured. The k nearest training samples' class labels are found and the majority is assigned to the test sample. The posterior probability of a tests sample is justified as in (10), where ki are the k closest training samples of class i. ni is the total number of training samples of class i and n is the total number of training samples, and thus ni n represents the prior probability. Thus, kNN assumes a prior probability based on the distribution of the training data.

$$P(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{p(\mathbf{x})} = \frac{\frac{k_i/n_i}{V}\frac{n_i}{n}}{\frac{k/n}{V}} = \frac{k_i}{k}$$

…(4)

4.Clustering

K-means clustering is one of the more popular unsupervised clustering techniques. In this iterative technique, a certain number of classes (or clusters) k is assumed in a dataset. k cluster means (centroids) are arbitrarily chosen to begin with. For each sample, the nearest cluster mean is found. For each cluster mean, a new cluster mean is recalculated using the nearest data samples.

5.Decision Trees

Decision trees are a non-statistical approach to pattern classification. All samples start at the root node and are divided up and classified as one progresses through the tree. At each node N, a property query is made to distinguish each sample into two groups. The objective of decision trees is to maximize the change in impurity from each node to the next layer. This change in impurity is defined as in (12). Decision trees in the project were implemented using scikit-learn's DecisionTreeClassifier.

∆i(N) = i(N) − PLi(NL) − (1 − PL)i(NR) …(5)

6.Backpropagation Neural Networks (BPNN)

The fundamental building block of neural networks is the perceptron (a single layer network), which are inspired by the neurons of the human brain. They are composed of inputs ~x = x1 x2 . . . xd 1 and weights ~w = w1 w2 . . . wd −w0 , where w0 is the bias. The output z = ~w T ~x. If z > 0, the perceptron outputs 1. Otherwise, it outputs 0.

$$\vec{w}^{k+1} = \vec{w}^{k} + \sum_{i=1}^{n} (T_i - z_i) x_i$$

…(6)

CODE

Fashion MNIST Data Classification Project

Step 1) Import Libraries

```
In [ ]: import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import tensorflow as tf
        import keras
```

Step 2) Load data

```
In [ ]: (X_train, y_train), (X_test, y_test)=tf.keras.datasets.fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datase
ts/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-datase
ts/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-datase
ts/t10k-labels-idx1-ubyte.gz
16384/5148 [==================================================================
============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datase
ts/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
4431872/4422102 [==============================] - 0s 0us/step
```

```
In [ ]: # Print the shape of data
```

```
In [ ]: X_train.shape,y_train.shape, "***************" , X_test.shape,y_test.shape
```

```
Out[ ]: ((60000, 28, 28), (60000,), '***************', (10000, 28, 28), (10000,))
```

```
In [ ]: X_train[0]
```

```
Out[ ]: array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
            0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
            0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
            0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
            0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
            0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
            0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,
            0,   0,  13,  73,   0,   0,   1,   4,   0,   0,   0,   0,   1,
            1,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
            0,  36, 136, 127,  62,  54,   0,   0,   0,   1,   3,   4,   0,
            0,   3],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   6,
            0, 102, 204, 176, 134, 144, 123,  23,   0,   0,   0,   0,  12,
           10,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
            0, 155, 236, 207, 178, 107, 156, 161, 109,  64,  23,  77, 130,
           72,  15],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   0,
           69, 207, 223, 218, 216, 216, 163, 127, 121, 122, 146, 141,  88,
          172,  66],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   1,   1,   0,
          200, 232, 232, 233, 229, 223, 223, 215, 213, 164, 127, 123, 196,
          229,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          183, 225, 216, 223, 228, 235, 227, 224, 222, 224, 221, 223, 245,
          173,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          193, 228, 218, 213, 198, 180, 212, 210, 211, 213, 223, 220, 243,
          202,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   3,   0,  12,
          219, 220, 212, 218, 192, 169, 227, 208, 218, 224, 212, 226, 197,
          209,  52],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   6,   0,  99,
          244, 222, 220, 218, 203, 198, 221, 215, 213, 222, 220, 245, 119,
          167,  56],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   4,   0,   0,  55,
          236, 228, 230, 228, 240, 232, 213, 218, 223, 234, 217, 217, 209,
           92,   0],
       [  0,   0,   1,   4,   6,   7,   2,   0,   0,   0,   0,   0, 237,
          226, 217, 223, 222, 219, 222, 221, 216, 223, 229, 215, 218, 255,
           77,   0],
       [  0,   3,   0,   0,   0,   0,   0,   0,   0,  62, 145, 204, 228,
          207, 213, 221, 218, 208, 211, 218, 224, 223, 219, 215, 224, 244,
          159,   0],
       [  0,   0,   0,   0,  18,  44,  82, 107, 189, 228, 220, 222, 217,
          226, 200, 205, 211, 230, 224, 234, 176, 188, 250, 248, 233, 238,
          215,   0],
       [  0,  57, 187, 208, 224, 221, 224, 208, 204, 214, 208, 209, 200,
          159, 245, 193, 206, 223, 255, 255, 221, 234, 221, 211, 220, 232,
          246,   0],
       [  3, 202, 228, 224, 221, 211, 211, 214, 205, 205, 205, 220, 240,
           80, 150, 255, 229, 221, 188, 154, 191, 210, 204, 209, 222, 228,
          225,   0],
       [ 98, 233, 198, 210, 222, 229, 229, 234, 249, 220, 194, 215, 217,
          241,  65,  73, 106, 117, 168, 219, 221, 215, 217, 223, 223, 224,
          229,  29],
       [ 75, 204, 212, 204, 193, 205, 211, 225, 216, 185, 197, 206, 198,
          213, 240, 195, 227, 245, 239, 223, 218, 212, 209, 222, 220, 221,
          230,  67],
       [ 48, 203, 183, 194, 213, 197, 185, 190, 194, 192, 202, 214, 219,
          221, 220, 236, 225, 216, 199, 206, 186, 181, 177, 172, 181, 205,
          206, 115],
       [  0, 122, 219, 193, 179, 171, 183, 196, 204, 210, 213, 207, 211,
          210, 200, 196, 194, 191, 195, 191, 198, 192, 176, 156, 167, 177,
          210,  92],
       [  0,   0,  74, 189, 212, 191, 175, 172, 175, 181, 185, 188, 189,
          188, 193, 198, 204, 209, 210, 210, 211, 188, 188, 194, 192, 216,
          170,   0],
       [  2,   0,   0,   0,  66, 200, 222, 237, 239, 242, 246, 243, 244,
          221, 220, 193, 191, 179, 182, 182, 181, 176, 166, 168,  99,  58,
            0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  40,  61,  44,  72,  41,  35,
            0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
            0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
            0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
            0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
            0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
            0,   0]], dtype=uint8)
```

In [ ]:
```
y_train[0]
```

Out[ ]:
```
9
```

In [ ]:
```
class_labels = [        "T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "
```

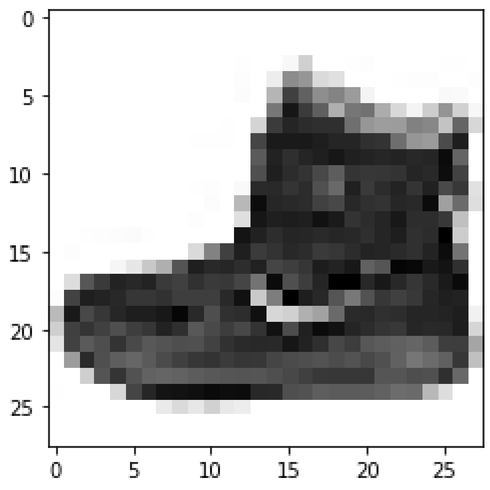
In [ ]:
```
class_labels
```

Out[ ]:
```
['T-shirt/top',
 'Trouser',
 'Pullover',
 'Dress',
 'Coat',
 'Sandal',
 'Shirt',
 'Sneaker',
 'Bag',
 'Ankle boot']
```

In [ ]:
```
# show image
```

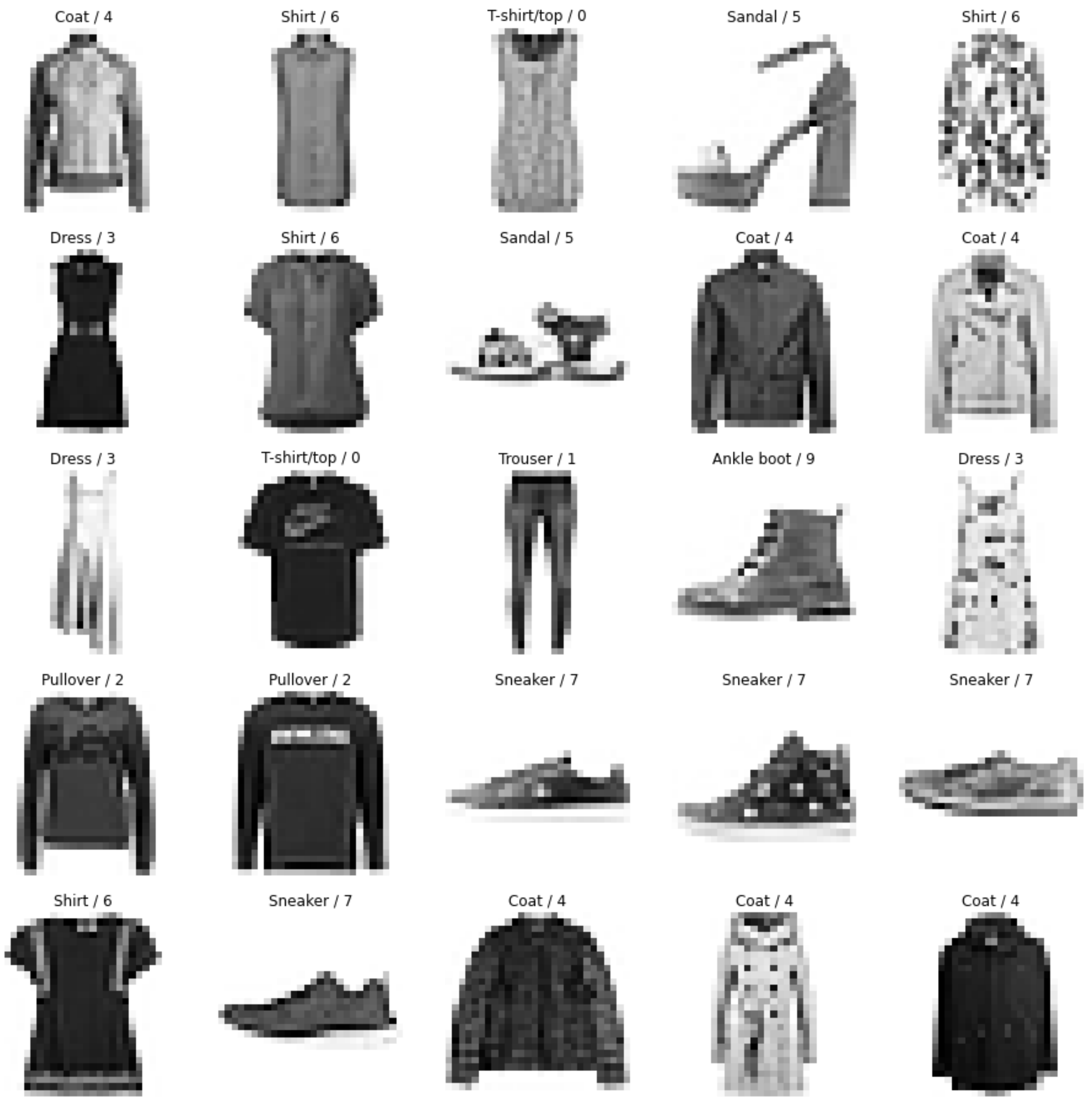
In [ ]:
```
plt.imshow(X_train[0],cmap='Greys')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7f0106f59a90>
```



```
In [ ]: plt.figure(figsize=(16,16))

        j=1
        for  i in np.random.randint(0,1000,25):
          plt.subplot(5,5,j);j+=1
          plt.imshow(X_train[i],cmap='Greys')
          plt.axis('off')
          plt.title('{}  /  {}'.format(class_labels[y_train[i]],y_train[i]))
```



```
In [ ]: X_train.ndim
```

```
Out[ ]: 3
```

```
In [ ]: X_train  =  np.expand_dims(X_train,-1)
```

```
In [ ]: X_train.ndim
```

```
Out[ ]: 4
```

```
In [ ]: X_test=np.expand_dims(X_test,-1)
```

```
In [ ]: # feature scaling
```

```
In [ ]: X_train = X_train/255
        X_test= X_test/255
```

```
In [ ]: # Split dataset
```

```
In [ ]: from  sklearn.model_selection  import   train_test_split
        X_train,X_Validation,y_train,y_Validation=train_test_split(X_train,y_train,test_
```

```
In [ ]: X_train.shape,X_Validation.shape,y_train.shape,y_Validation.shape
```

```
Out[ ]: ((48000, 28, 28, 1), (12000, 28, 28, 1), (48000,), (12000,))
```

```
In [ ]:
```

Step 3) Buiding the CNN model

```
In [ ]: model=keras.models.Sequential([
                                 keras.layers.Conv2D(filters=32,kernel_size=3,strides=(1
                                 keras.layers.MaxPooling2D(pool_size=(2,2)),
                                 keras.layers.Flatten(),
                                 keras.layers.Dense(units=128,activation='relu'),
                                 keras.layers.Dense(units=10,activation='softmax')
        ])
```

```
In [ ]: model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
conv2d (Conv2D)             (None, 26, 26, 32)        320

max_pooling2d (MaxPooling2D) (None, 13, 13, 32)       0

flatten (Flatten)           (None, 5408)              0

dense (Dense)               (None, 128)               692352

dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 693,962
Trainable params: 693,962
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['
```

In [ ]:
```
model.fit(X_train,y_train,epochs=10,batch_size=512,verbose=1,validation_data=(X_
```

```
Epoch 1/10
94/94 [==============================] - 21s 212ms/step - loss: 0.6340 - accura
cy: 0.7849 - val_loss: 0.4368 - val_accuracy: 0.8497
Epoch 2/10
94/94 [==============================] - 20s 217ms/step - loss: 0.3856 - accura
cy: 0.8646 - val_loss: 0.3644 - val_accuracy: 0.8746
Epoch 3/10
94/94 [==============================] - 20s 212ms/step - loss: 0.3325 - accura
cy: 0.8834 - val_loss: 0.3417 - val_accuracy: 0.8810
Epoch 4/10
94/94 [==============================] - 20s 218ms/step - loss: 0.3007 - accura
cy: 0.8935 - val_loss: 0.3207 - val_accuracy: 0.8872
Epoch 5/10
94/94 [==============================] - 21s 222ms/step - loss: 0.2787 - accura
cy: 0.9015 - val_loss: 0.3087 - val_accuracy: 0.8907
Epoch 6/10
94/94 [==============================] - 20s 217ms/step - loss: 0.2653 - accura
cy: 0.9047 - val_loss: 0.3011 - val_accuracy: 0.8937
Epoch 7/10
94/94 [==============================] - 20s 212ms/step - loss: 0.2526 - accura
cy: 0.9106 - val_loss: 0.2921 - val_accuracy: 0.8961
Epoch 8/10
94/94 [==============================] - 20s 215ms/step - loss: 0.2353 - accura
cy: 0.9165 - val_loss: 0.2748 - val_accuracy: 0.9032
Epoch 9/10
94/94 [==============================] - 20s 214ms/step - loss: 0.2225 - accura
cy: 0.9215 - val_loss: 0.2777 - val_accuracy: 0.9026
Epoch 10/10
94/94 [==============================] - 20s 214ms/step - loss: 0.2116 - accura
cy: 0.9246 - val_loss: 0.2771 - val_accuracy: 0.9014
```

Out[ ]:
```
<keras.callbacks.History at 0x7f8049b326d0>
```

In [ ]:
```
y_pred = model.predict(X_test)
y_pred.round(2)
```

```
Out[ ]: array([[0.  , 0.  , 0.  , ..., 0.01, 0.  , 0.98],
               [0.  , 0.  , 1.  , ..., 0.  , 0.  , 0.  ],
               [0.  , 1.  , 0.  , ..., 0.  , 0.  , 0.  ],
               ...,
               [0.  , 0.  , 0.  , ..., 0.  , 0.99, 0.  ],
               [0.  , 1.  , 0.  , ..., 0.  , 0.  , 0.  ],
               [0.  , 0.  , 0.01, ..., 0.23, 0.02, 0.  ]], dtype=float32)
```

```
In [ ]: y_test
```

```
Out[ ]: array([9, 2, 1, ..., 8, 1, 5], dtype=uint8)
```

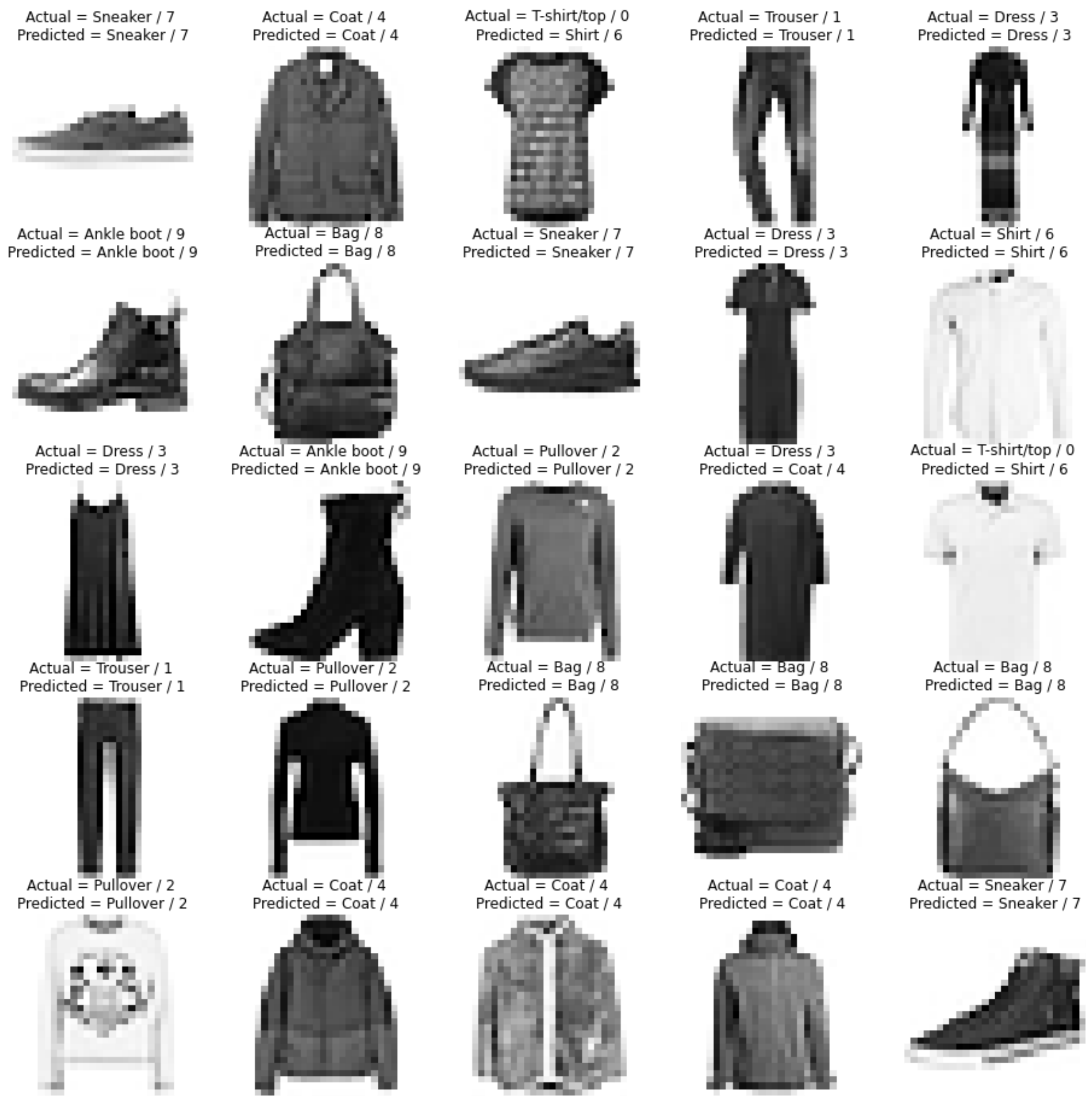
```
In [ ]: model.evaluate(X_test,  y_test)
```

```
313/313 [==============================] - 2s 7ms/step - loss: 0.2797 - accurac
y: 0.8953
```

```
Out[ ]: [0.27970078587532043,  0.8952999711036682]
```

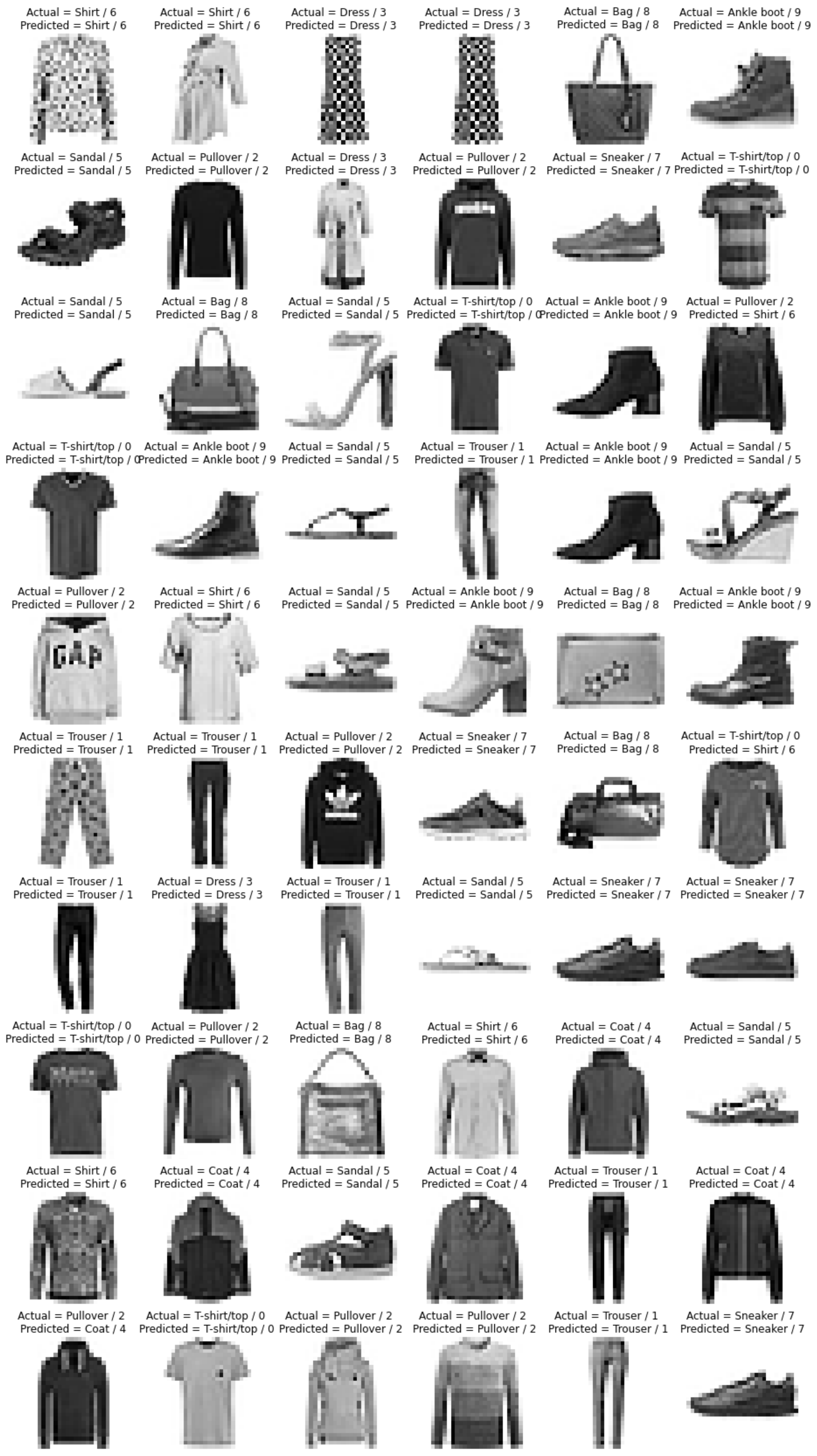
```
In [ ]: plt.figure(figsize=(16,16))

        j=1
        for i in np.random.randint(0, 1000,25):
          plt.subplot(5,5, j); j+=1
          plt.imshow(X_test[i].reshape(28,28),  cmap  =  'Greys')
          plt.title('Actual = {} / {} \nPredicted = {} / {}'.format(class_labels[y_test[
          plt.axis('off')
```



```
In [ ]: plt.figure(figsize=(16,30))

        j=1
        for i in np.random.randint(0, 1000,60):
          plt.subplot(10,6, j); j+=1
          plt.imshow(X_test[i].reshape(28,28), cmap = 'Greys')
          plt.title('Actual = {} / {} \nPredicted = {} / {}'.format(class_labels[y_test[
          plt.axis('off')
```


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

In [ ]:
```
"""## Confusion Matrix"""
```

Out[ ]:
```
'## Confusion Matrix'
```

In [ ]:
```
from sklearn.metrics import confusion_matrix
plt.figure(figsize=(16,9))
y_pred_labels = [ np.argmax(label) for label in y_pred ]
cm = confusion_matrix(y_test, y_pred_labels)
```

```
<Figure size 1152x648 with 0 Axes>
```

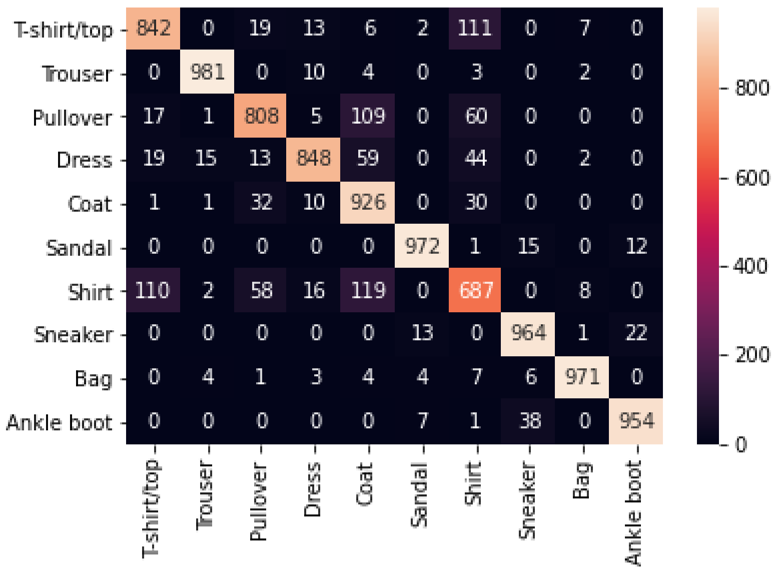
In [ ]:
```
sns.heatmap(cm, annot=True, fmt='d',xticklabels=class_labels, yticklabels=class_

from sklearn.metrics import classification_report
cr= classification_report(y_test, y_pred_labels, target_names=class_labels)
print(cr)
```

```
              precision    recall  f1-score   support

 T-shirt/top       0.85      0.84      0.85      1000
     Trouser       0.98      0.98      0.98      1000
    Pullover       0.87      0.81      0.84      1000
       Dress       0.94      0.85      0.89      1000
        Coat       0.75      0.93      0.83      1000
      Sandal       0.97      0.97      0.97      1000
       Shirt       0.73      0.69      0.71      1000
     Sneaker       0.94      0.96      0.95      1000
         Bag       0.98      0.97      0.98      1000
  Ankle boot       0.97      0.95      0.96      1000

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000
```



In [ ]:
```
"""# Save Model"""
```

In [ ]:
```
model.save('fashion_mnist_cnn_model.h5')
```

Build 2 complex CNN

In [ ]:
```
#Building CNN model
cnn_model2 = keras.models.Sequential([
                         keras.layers.Conv2D(filters=32, kernel_size=3, strides=
                         keras.layers.MaxPooling2D(pool_size=(2,2)),
                         keras.layers.Conv2D(filters=64, kernel_size=3, strides=
                         keras.layers.MaxPooling2D(pool_size=(2,2)),
                         keras.layers.Flatten(),
                         keras.layers.Dense(units=128, activation='relu'),
                         keras.layers.Dropout(0.25),
                         keras.layers.Dense(units=256, activation='relu'),
                         keras.layers.Dropout(0.25),
                         keras.layers.Dense(units=128, activation='relu'),
                         keras.layers.Dense(units=10, activation='softmax')
                         ])

# complie the model
cnn_model2.compile(optimizer='adam', loss= 'sparse_categorical_crossentropy', me

#Train the Model
cnn_model2.fit(X_train, y_train, epochs=20, batch_size=512, verbose=1, validatio

cnn_model2.save('fashion_mnist_cnn_model2.h5')

"""######## very complex model"""

#Building CNN model
cnn_model3 = keras.models.Sequential([
                         keras.layers.Conv2D(filters=64, kernel_size=3, strides=
                         keras.layers.MaxPooling2D(pool_size=(2,2)),
                         keras.layers.Conv2D(filters=128, kernel_size=3, strides
                         keras.layers.MaxPooling2D(pool_size=(2,2)),
                         keras.layers.Conv2D(filters=64, kernel_size=3, strides=
                         keras.layers.MaxPooling2D(pool_size=(2,2)),
                         keras.layers.Flatten(),
                         keras.layers.Dense(units=128, activation='relu'),
                         keras.layers.Dropout(0.25),
                         keras.layers.Dense(units=256, activation='relu'),
                         keras.layers.Dropout(0.5),
                         keras.layers.Dense(units=256, activation='relu'),
                         keras.layers.Dropout(0.25),
                         keras.layers.Dense(units=128, activation='relu'),
                         keras.layers.Dropout(0.10),
                         keras.layers.Dense(units=10,  activation='softmax')
                         ])

# complie the model
cnn_model3.compile(optimizer='adam', loss= 'sparse_categorical_crossentropy', me

#Train the Model
cnn_model3.fit(X_train, y_train, epochs=50, batch_size=512, verbose=1, validatio

cnn_model3.save('fashion_mnist_cnn_model3.h5')

cnn_model3.evaluate(X_test, y_test)
```

```
Epoch 1/20
94/94 [==============================] - 25s 263ms/step - loss: 1.0185 - accura
cy: 0.6177 - val_loss: 0.5983 - val_accuracy: 0.7672
Epoch 2/20
94/94 [==============================] - 25s 269ms/step - loss: 0.5628 - accura
cy: 0.7879 - val_loss: 0.4714 - val_accuracy: 0.8256
Epoch 3/20
94/94 [==============================] - 25s 266ms/step - loss: 0.4650 - accura
cy: 0.8279 - val_loss: 0.4083 - val_accuracy: 0.8487
Epoch 4/20
94/94 [==============================] - 25s 271ms/step - loss: 0.4093 - accura
cy: 0.8514 - val_loss: 0.3694 - val_accuracy: 0.8657
Epoch 5/20
94/94 [==============================] - 25s 265ms/step - loss: 0.3726 - accura
cy: 0.8644 - val_loss: 0.3530 - val_accuracy: 0.8711
Epoch 6/20
94/94 [==============================] - 24s 260ms/step - loss: 0.3458 - accura
cy: 0.8751 - val_loss: 0.3337 - val_accuracy: 0.8739
Epoch 7/20
94/94 [==============================] - 25s 269ms/step - loss: 0.3274 - accura
cy: 0.8804 - val_loss: 0.3217 - val_accuracy: 0.8813
Epoch 8/20
94/94 [==============================] - 26s 274ms/step - loss: 0.3132 - accura
cy: 0.8864 - val_loss: 0.3090 - val_accuracy: 0.8874
Epoch 9/20
94/94 [==============================] - 26s 273ms/step - loss: 0.2963 - accura
cy: 0.8905 - val_loss: 0.3016 - val_accuracy: 0.8891
Epoch 10/20
94/94 [==============================] - 26s 274ms/step - loss: 0.2845 - accura
cy: 0.8959 - val_loss: 0.2954 - val_accuracy: 0.8915
Epoch 11/20
94/94 [==============================] - 26s 274ms/step - loss: 0.2816 - accura
cy: 0.8961 - val_loss: 0.2937 - val_accuracy: 0.8932
Epoch 12/20
94/94 [==============================] - 26s 276ms/step - loss: 0.2646 - accura
cy: 0.9025 - val_loss: 0.2803 - val_accuracy: 0.8990
Epoch 13/20
94/94 [==============================] - 26s 276ms/step - loss: 0.2609 - accura
cy: 0.9041 - val_loss: 0.2748 - val_accuracy: 0.8985
Epoch 14/20
94/94 [==============================] - 26s 275ms/step - loss: 0.2506 - accura
cy: 0.9075 - val_loss: 0.2814 - val_accuracy: 0.8990
Epoch 15/20
94/94 [==============================] - 26s 275ms/step - loss: 0.2402 - accura
cy: 0.9116 - val_loss: 0.2763 - val_accuracy: 0.9021
Epoch 16/20
94/94 [==============================] - 26s 272ms/step - loss: 0.2346 - accura
cy: 0.9144 - val_loss: 0.2868 - val_accuracy: 0.8963
Epoch 17/20
94/94 [==============================] - 25s 267ms/step - loss: 0.2325 - accura
cy: 0.9138 - val_loss: 0.2759 - val_accuracy: 0.9022
Epoch 18/20
94/94 [==============================] - 26s 273ms/step - loss: 0.2233 - accura
cy: 0.9175 - val_loss: 0.2800 - val_accuracy: 0.9028
Epoch 19/20
94/94 [==============================] - 26s 278ms/step - loss: 0.2175 - accura
cy: 0.9203 - val_loss: 0.2780 - val_accuracy: 0.9018
Epoch 20/20
94/94 [==============================] - 26s 280ms/step - loss: 0.2079 - accura
cy: 0.9231 - val_loss: 0.2685 - val_accuracy: 0.9071
```

```
Epoch 1/50
94/94 [==============================] - 58s 604ms/step - loss: 1.1713 - accura
cy: 0.5471 - val_loss: 0.5933 - val_accuracy: 0.7611
Epoch 2/50
94/94 [==============================] - 57s 602ms/step - loss: 0.5709 - accura
cy: 0.7840 - val_loss: 0.4601 - val_accuracy: 0.8317
Epoch 3/50
94/94 [==============================] - 56s 595ms/step - loss: 0.4617 - accura
cy: 0.8315 - val_loss: 0.3918 - val_accuracy: 0.8551
Epoch 4/50
94/94 [==============================] - 55s 588ms/step - loss: 0.3932 - accura
cy: 0.8592 - val_loss: 0.3528 - val_accuracy: 0.8698
Epoch 5/50
94/94 [==============================] - 55s 588ms/step - loss: 0.3556 - accura
cy: 0.8742 - val_loss: 0.3248 - val_accuracy: 0.8815
Epoch 6/50
94/94 [==============================] - 55s 590ms/step - loss: 0.3228 - accura
cy: 0.8860 - val_loss: 0.3310 - val_accuracy: 0.8810
Epoch 7/50
94/94 [==============================] - 55s 587ms/step - loss: 0.3028 - accura
cy: 0.8926 - val_loss: 0.3019 - val_accuracy: 0.8916
Epoch 8/50
94/94 [==============================] - 56s 594ms/step - loss: 0.2823 - accura
cy: 0.8993 - val_loss: 0.3074 - val_accuracy: 0.8879
Epoch 9/50
94/94 [==============================] - 58s 618ms/step - loss: 0.2720 - accura
cy: 0.9033 - val_loss: 0.2821 - val_accuracy: 0.8973
Epoch 10/50
94/94 [==============================] - 58s 616ms/step - loss: 0.2541 - accura
cy: 0.9091 - val_loss: 0.2983 - val_accuracy: 0.8947
Epoch 11/50
94/94 [==============================] - 58s 617ms/step - loss: 0.2381 - accura
cy: 0.9154 - val_loss: 0.2871 - val_accuracy: 0.8964
Epoch 12/50
94/94 [==============================] - 57s 609ms/step - loss: 0.2236 - accura
cy: 0.9201 - val_loss: 0.2735 - val_accuracy: 0.9038
Epoch 13/50
94/94 [==============================] - 57s 609ms/step - loss: 0.2165 - accura
cy: 0.9226 - val_loss: 0.2980 - val_accuracy: 0.8990
Epoch 14/50
94/94 [==============================] - 57s 609ms/step - loss: 0.2061 - accura
cy: 0.9271 - val_loss: 0.2732 - val_accuracy: 0.9076
Epoch 15/50
94/94 [==============================] - 57s 610ms/step - loss: 0.1949 - accura
cy: 0.9314 - val_loss: 0.2681 - val_accuracy: 0.9072
Epoch 16/50
94/94 [==============================] - 63s 671ms/step - loss: 0.1884 - accura
cy: 0.9325 - val_loss: 0.2850 - val_accuracy: 0.9072
Epoch 17/50
94/94 [==============================] - 65s 693ms/step - loss: 0.1777 - accura
cy: 0.9377 - val_loss: 0.2775 - val_accuracy: 0.9064
Epoch 18/50
94/94 [==============================] - 63s 671ms/step - loss: 0.1714 - accura
cy: 0.9392 - val_loss: 0.2762 - val_accuracy: 0.9115
Epoch 19/50
94/94 [==============================] - 59s 633ms/step - loss: 0.1609 - accura
cy: 0.9435 - val_loss: 0.2955 - val_accuracy: 0.9052
Epoch 20/50
94/94 [==============================] - 59s 624ms/step - loss: 0.1591 - accura
cy: 0.9444 - val_loss: 0.3144 - val_accuracy: 0.9043
```

```
Epoch 21/50
94/94 [==============================] - 60s 641ms/step - loss: 0.1539 - accura
cy: 0.9450 - val_loss: 0.2845 - val_accuracy: 0.9105
Epoch 22/50
94/94 [==============================] - 58s 616ms/step - loss: 0.1407 - accura
cy: 0.9510 - val_loss: 0.2957 - val_accuracy: 0.9105
Epoch 23/50
94/94 [==============================] - 56s 593ms/step - loss: 0.1306 - accura
cy: 0.9535 - val_loss: 0.3419 - val_accuracy: 0.9021
Epoch 24/50
94/94 [==============================] - 57s 602ms/step - loss: 0.1287 - accura
cy: 0.9555 - val_loss: 0.2974 - val_accuracy: 0.9087
Epoch 25/50
94/94 [==============================] - 57s 612ms/step - loss: 0.1268 - accura
cy: 0.9540 - val_loss: 0.3154 - val_accuracy: 0.9097
Epoch 26/50
94/94 [==============================] - 58s 618ms/step - loss: 0.1215 - accura
cy: 0.9567 - val_loss: 0.3280 - val_accuracy: 0.9078
Epoch 27/50
94/94 [==============================] - 58s 615ms/step - loss: 0.1139 - accura
cy: 0.9597 - val_loss: 0.3438 - val_accuracy: 0.9019
Epoch 28/50
94/94 [==============================] - 61s 651ms/step - loss: 0.1067 - accura
cy: 0.9626 - val_loss: 0.3481 - val_accuracy: 0.9075
Epoch 29/50
94/94 [==============================] - 58s 620ms/step - loss: 0.1113 - accura
cy: 0.9602 - val_loss: 0.3404 - val_accuracy: 0.9057
Epoch 30/50
94/94 [==============================] - 69s 738ms/step - loss: 0.0990 - accura
cy: 0.9651 - val_loss: 0.3442 - val_accuracy: 0.9120
Epoch 31/50
12/94 [==>...........................] - ETA: 57s - loss: 0.0810 - accuracy: 0.
9692
```

```
In [ ]:
```

```
Epoch 21/50
```

CONCLUSION

We found that although Resnet is claimed to make complicated models perform not worse than simple models, the fact that shortcuts have weights still requires us to tune the model size. The shortcut of Resnet is mostly effective at deep layers of the network. As for traditional machine learning methods, the accuracy scores are competitive to those of neural networks. However, for SVM, the time cost is much higher. Also, it is observed that Adaboost can help improve the results of other learners. If implemented with CNN, Resnet or VGG, Adaboost may help reach a much satisfied accuracy. As for CNN and VGG, we also achieved good result. Through careful parameter tuning, the ability of CNN on image classification is shown.

We address the problem of distinguishing clothing elements in fashion photographs using a CNN. This is accomplished using the fashion MNIST dataset, which contains many images, together with CNN and ANN algorithms for accurate and effective image classification. Image recognition is becoming increasingly crucial as deep learning algorithms progress. CNN recognition is commonly used when it comes to fashion-related applications, such as garment classification, retrieval, and computerised clothing labelling. In this study, we employ CNN architecture on the Fashion MNIST dataset. We would like to compare this with different datasets. Fashion MNIST is a dataset having low-resolution images. We would like to try these highresolution images in the future, and we also plan to put CNN architecture to the test on a dataset of real-life apparel pictures that we amassed ourselves.