Fashion MNIST

By: Adi Hoftman-322267824 , Orya Spiegel-207018524

Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Zalando intends Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

- Each row is a separate image
- Column 1 is the class label.
- Remaining columns are pixel numbers (784 total).
- Each value is the darkness of the pixel (1 to 255)

Questions we asked about our data

- How well can we predict the label of each item?
- Is it possible to get good predictions based on the colors of the item?
- Can we improve the results if we create categories for similar items?

Techniques we used

- ### VGG - New!
- ADABOOST
- RANDOM FOREST
- NAIVE BAYES
- KNN

Challenges we had

- We wanted to know if it's possible to classify using only the gray levels of each item. We didn't get good results for any of the models, and we realized this data is not enough to make the predictions, since there isn't a lot of difference between the average of the different labels.
- We wanted to check if creating categories for similar items will give us better results. For that we tried a few different ways to unite labels. On the one hand we wanted to get good classification results, at least like the ones we got with the original data, so we wanted to use as little groups as possible. On the other hand, we still want to keep the original information as much as possible, and uniting too many labels together is going to affect this. So we tried to keep as many of the original labels as possible while improving the labeling for similar items.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
import matplotlib
import seaborn as sns
%matplotlib inline
from sklearn import metrics
from sklearn.metrics import classification_report,confusion_matrix,ConfusionMatrixDispl
import random
```

In [2]:
```
df_train = pd.read_csv('../data/fashion-mnist_train.csv')
df_test = pd.read_csv('../data/fashion-mnist_test.csv')
```

In [3]:
```
df_train.head()
```

Out[3]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	p
0	2	0	0	0	0	0	0	0	0	0	...	0	0	
1	9	0	0	0	0	0	0	0	0	0	...	0	0	
2	6	0	0	0	0	0	0	0	5	0	...	0	0	
3	0	0	0	0	1	2	0	0	0	0	...	3	0	
4	3	0	0	0	0	0	0	0	0	0	...	0	0	

5 rows × 785 columns

In [4]:
```
print("train", df_train.shape)
print("test", df_test.shape)
```

```
train (60000, 785)
test (10000, 785)
```

In [5]:
```
#Check for missing data
df_train.isnull().sum().sum()
```

Out[5]:
```
0
```

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255. The training and test data sets have 785 columns. The first column consists of the class labels (see above), and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image.

To locate a pixel on the image, suppose that we have decomposed x as x = i * 28 + j, where i and j are integers between 0 and 27. The pixel is located on row i and column j of a 28 x 28 matrix.

Labels

Each training and test example is assigned to one of the following labels:

0 - T-shirt/top

1 - Trouser

2 - Pullover

3 - Dress

4 - Coat

5 - Sandal

6 - Shirt

7 - Sneaker

8 - Bag

9 - Ankle boot

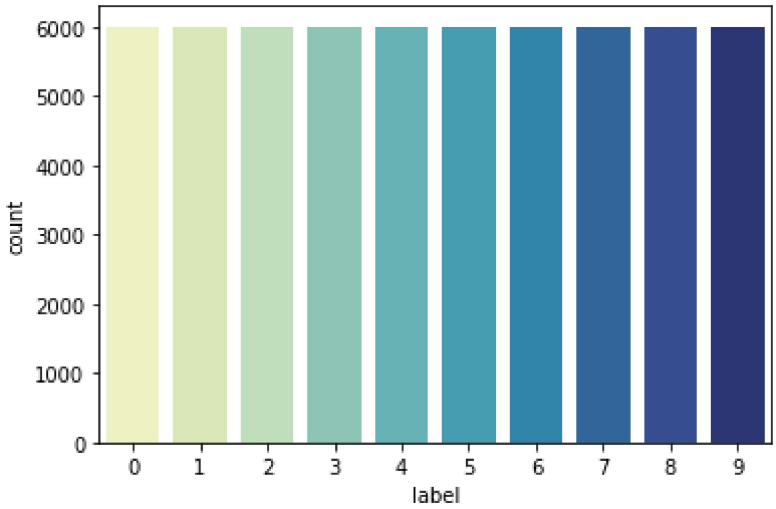
In [6]:
```
df_train.label.value_counts()
```

Out[6]:
```
0    6000
1    6000
2    6000
3    6000
4    6000
5    6000
6    6000
7    6000
8    6000
9    6000
Name: label, dtype: int64
```

In [7]:
```
sns.countplot(df_train.label , palette='YlGnBu')
```

```
C:\Users\oryas\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\_decorat
ors.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

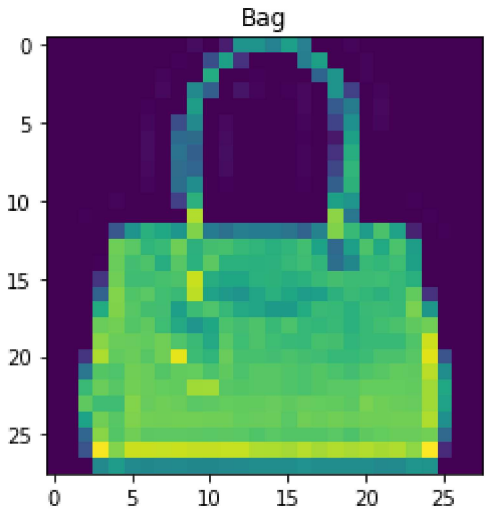
Out[7]:
```
<AxesSubplot:xlabel='label', ylabel='count'>
```



In [8]:
```
class_names = ['T_shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt'
```

In [9]:
```
#Visualization
i = random.randint(1,60000) #select any random index from 1 to 60,000
train= np.array(df_train, dtype = 'float32')
plt.imshow(train[i,1:].reshape((28,28))) #reshape and plot the image

label_index = df_train["label"][i]
plt.title(f"{class_names[label_index]}")
```

Out[9]:
```
Text(0.5, 1.0, 'Bag')
```



In [ ]:
```
```

Making Predictions

VGG NEW

In [10]:
```
# VGG models are a type of CNN Architecture proposed by Karen Simonyan & Andrew Zisserm
```

```
# Oxford University. (CNN is a specialized deep neural network model for handling image
```

In [11]:
```
from keras.applications.vgg16 import VGG16
from keras import models
from keras.models import Model
from keras import layers
from tensorflow.keras import optimizers
```

Convert the images into 3 channels

In [12]:
```
X_train = np.array(df_train.iloc[:,1:])
X_test = np.array(df_test.iloc[:,1:])
y_train = np.array(df_train.iloc[:,0])
y_test = np.array(df_test.iloc[:,0])

X_train = np.dstack([X_train]*3)
X_test = np.dstack([X_test]*3)
X_train.shape, X_test.shape
```

Out[12]:
```
((60000, 784, 3), (10000, 784, 3))
```

Reshape images to the tensor format required by tensorflow

In [13]:
```
X_train = X_train.reshape(-1, 28, 28, 3)
X_test = X_test.reshape(-1, 28, 28, 3)
X_train.shape, X_test.shape
```

Out[13]:
```
((60000, 28, 28, 3), (10000, 28, 28, 3))
```

Resize the images 48*48 as required by VGG16

In [14]:
```
from keras.preprocessing.image import img_to_array, array_to_img
X_train = np.asarray([img_to_array(array_to_img(im, scale=False).resize((48,48))) for i
X_test = np.asarray([img_to_array(array_to_img(im, scale=False).resize((48,48))) for im

X_train.shape, X_test.shape
```

Out[14]:
```
((60000, 48, 48, 3), (10000, 48, 48, 3))
```

Normalize the data and change data type

In [15]:
```
X_train = X_train / 255.
X_test = X_test / 255.
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
```

Converting Labels to one hot encoded format

In [16]:
```
from tensorflow.keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

Check the data size whether it is as per tensorflow and VGG16 requirement

In [17]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[17]:
```
((60000, 48, 48, 3), (10000, 48, 48, 3), (60000, 10), (10000, 10))
```

In [18]:
```
from keras.applications.vgg16 import preprocess_input
X_train = preprocess_input(X_train)
X_test = preprocess_input(X_test)
```

In [19]:
```
IMG_WIDTH = 48
IMG_HEIGHT = 48
IMG_DEPTH = 3
BATCH_SIZE = 16
```

In [20]:
```
conv_base = VGG16(include_top=False, input_shape=(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH))
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 48, 48, 3)]       0

 block1_conv1 (Conv2D)       (None, 48, 48, 64)        1792

 block1_conv2 (Conv2D)       (None, 48, 48, 64)        36928

 block1_pool (MaxPooling2D)  (None, 24, 24, 64)        0

 block2_conv1 (Conv2D)       (None, 24, 24, 128)       73856

 block2_conv2 (Conv2D)       (None, 24, 24, 128)       147584

 block2_pool (MaxPooling2D)  (None, 12, 12, 128)       0

 block3_conv1 (Conv2D)       (None, 12, 12, 256)       295168

 block3_conv2 (Conv2D)       (None, 12, 12, 256)       590080

 block3_conv3 (Conv2D)       (None, 12, 12, 256)       590080

 block3_pool (MaxPooling2D)  (None, 6, 6, 256)         0

 block4_conv1 (Conv2D)       (None, 6, 6, 512)         1180160

 block4_conv2 (Conv2D)       (None, 6, 6, 512)         2359808

 block4_conv3 (Conv2D)       (None, 6, 6, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 3, 3, 512)         0

 block5_conv1 (Conv2D)       (None, 3, 3, 512)         2359808

 block5_conv2 (Conv2D)       (None, 3, 3, 512)         2359808
```

```
 block5_conv3 (Conv2D)       (None, 3, 3, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 1, 1, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Extracting and flattening features

In [21]:

```
train_features = conv_base.predict(np.array(X_train), batch_size=BATCH_SIZE, verbose=1)
test_features = conv_base.predict(np.array(X_test), batch_size=BATCH_SIZE, verbose=1)

train_features_flat = np.reshape(train_features, (60000, 1*1*512))
test_features_flat = np.reshape(test_features, (10000, 1*1*512))
```

```
3750/3750 [==============================] - 623s 166ms/step
625/625 [==============================] - 102s 163ms/step
```

In [22]:

```
# We tried to add more layers to improve accuracy but got over-fitting, so we used only
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_dim=(1*1*512)))
model.add(layers.LeakyReLU(alpha=0.1))
model.add(layers.Dense(10, activation='softmax'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               262656

 leaky_re_lu (LeakyReLU)     (None, 512)               0

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 267,786
Trainable params: 267,786
Non-trainable params: 0
_________________________________________________________________
```

In [23]:

```
model.compile(loss='categorical_crossentropy', optimizer=optimizers.Adam(), metrics=['a
```

In [24]:

```
history = model.fit(train_features_flat, y_train, epochs= 150, validation_split=0.2, sh
```

```
Epoch 1/150
1500/1500 [==============================] - 5s 3ms/step - loss: 1.4618 - accuracy: 0.45
88 - val_loss: 1.1887 - val_accuracy: 0.5511.0531  - ETA: 2s - los - ETA - ETA: 0s - los
s: 1.4879 -
Epoch 2/150
1500/1500 [==============================] - 5s 3ms/step - loss: 1.0946 - accuracy: 0.59
65 - val_loss: 1.0148 - val_accuracy: 0.6346
```

```
Epoch 3/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.9881 - accuracy: 0.63
63 - val_loss: 0.9769 - val_accuracy: 0.6449: 2s - loss: 0.9923 -  - ETA: 1s - l - ETA:
0s - loss: 0.9921 - accuracy: 0.63 - ETA:
Epoch 4/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.9408 - accuracy: 0.65
55 - val_loss: 0.9516 - val_accuracy: 0.6462
Epoch 5/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.9059 - accuracy: 0.66
76 - val_loss: 0.8836 - val_accuracy: 0.6734oss: 0.9081 -
Epoch 6/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.8860 - accuracy: 0.67
56 - val_loss: 0.9285 - val_accuracy: 0.6502s: 0.8865 - accuracy - ETA: 0s - loss: 0.8 -
ETA: 0s - loss: 0.8871 - accuracy
Epoch 7/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.8649 - accuracy: 0.68
33 - val_loss: 0.8794 - val_accuracy: 0.6858
Epoch 8/150
1500/1500 [==============================] - 6s 4ms/step - loss: 0.8534 - accuracy: 0.68
71 - val_loss: 0.8695 - val_accuracy: 0.6822
Epoch 9/150
1500/1500 [==============================] - 5s 4ms/step - loss: 0.8379 - accuracy: 0.69
48 - val_loss: 0.8461 - val_accuracy: 0.6930 0.8378 -  - ETA: 1s - loss: 0.839 - ETA: 0s
- loss: 0.8397 -
Epoch 10/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.8252 - accuracy: 0.69
86 - val_loss: 0.8088 - val_accuracy: 0.7032
Epoch 11/150
1500/1500 [==============================] - 6s 4ms/step - loss: 0.8184 - accuracy: 0.69
91 - val_loss: 0.8073 - val_accuracy: 0.7106
Epoch 12/150
1500/1500 [==============================] - 7s 5ms/step - loss: 0.8115 - accuracy: 0.70
32 - val_loss: 0.8163 - val_accuracy: 0.6965
Epoch 13/150
1500/1500 [==============================] - 6s 4ms/step - loss: 0.8049 - accuracy: 0.70
49 - val_loss: 0.8211 - val_accuracy: 0.6937
Epoch 14/150
1500/1500 [==============================] - 6s 4ms/step - loss: 0.7983 - accuracy: 0.70
61 - val_loss: 0.8253 - val_accuracy: 0.6956
Epoch 15/150
1500/1500 [==============================] - 6s 4ms/step - loss: 0.7921 - accuracy: 0.70
98 - val_loss: 0.7890 - val_accuracy: 0.7122
Epoch 16/150
1500/1500 [==============================] - 5s 4ms/step - loss: 0.7865 - accuracy: 0.71
14 - val_loss: 0.7998 - val_accuracy: 0.7009
Epoch 17/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7817 - accuracy: 0.71
25 - val_loss: 0.7893 - val_accuracy: 0.7125 - loss: 0.7 - ETA:  - ETA: 0s - loss: 0.781
5 - accuracy:
Epoch 18/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7782 - accuracy: 0.71
51 - val_loss: 0.7693 - val_accuracy: 0.7158
Epoch 19/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7745 - accuracy: 0.71
87 - val_loss: 0.7911 - val_accuracy: 0.7046 loss: 0.7794 - accu - ETA: 0s - loss: - ET
A: 0s - loss: 0.7763 - accuracy
Epoch 20/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7764 - accuracy: 0.71
52 - val_loss: 0.7968 - val_accuracy: 0.7162 loss: 0.7799  - ETA: 2s - loss: 0.7 - ETA:
1s - loss: 0.7726 - accuracy - ETA: 1s - los - ETA: 0s - loss: 0.775 - ETA: 0s - loss:
```

```
0.7748 -
Epoch 21/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7678 - accuracy: 0.71
89 - val_loss: 0.7443 - val_accuracy: 0.7267- accuracy - ETA: 0s - loss: 0.7679 - ac
Epoch 22/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7689 - accuracy: 0.71
86 - val_loss: 0.7594 - val_accuracy: 0.7292
Epoch 23/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7584 - accuracy: 0.72
33 - val_loss: 0.8219 - val_accuracy: 0.6999
Epoch 24/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7520 - accuracy: 0.72
43 - val_loss: 0.7957 - val_accuracy: 0.7128 loss: 0.7386 - ac - ETA: 1s - l - ETA: 0s -
loss: 0.7536 - accu
Epoch 25/150
1500/1500 [==============================] - 5s 4ms/step - loss: 0.7523 - accuracy: 0.72
51 - val_loss: 0.7563 - val_accuracy: 0.7157
Epoch 26/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7536 - accuracy: 0.72
29 - val_loss: 0.8047 - val_accuracy: 0.6987
Epoch 27/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7466 - accuracy: 0.72
56 - val_loss: 0.8019 - val_accuracy: 0.7129471 - accu
Epoch 28/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7457 - accuracy: 0.72
59 - val_loss: 0.7823 - val_accuracy: 0.7165s: 0.761 - ETA: 0s - loss: 0.7 - ETA: 0s - l
oss: 0.7483 - ac
Epoch 29/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7423 - accuracy: 0.72
69 - val_loss: 0.7654 - val_accuracy: 0.7227485 - accura - ETA: 2s - loss: 0.7525 - accu
racy:  - ETA: 2s - loss: 0.7537 - accuracy: 0.72 - ETA: 2s - loss: 0.7531 - accura - ET
A: 1s - loss: 0.7434  - ETA: 0s - - ETA: 0s - loss: 0.7440 - accuracy: 0.
Epoch 30/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7348 - accuracy: 0.73
17 - val_loss: 0.7882 - val_accuracy: 0.7128
Epoch 31/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7362 - accuracy: 0.72
91 - val_loss: 0.7338 - val_accuracy: 0.7306 - ETA: 0s - los - ETA: 0s - loss: 0.7359 -
accuracy: 0.72 - ETA: 0s - loss: 0.7358 - accuracy: 0.72
Epoch 32/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7284 - accuracy: 0.73
23 - val_loss: 0.7715 - val_accuracy: 0.7174oss: 0.7293 - ac - ETA:
Epoch 33/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7314 - accuracy: 0.73
06 - val_loss: 0.7518 - val_accuracy: 0.7257
Epoch 34/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7251 - accuracy: 0.73
25 - val_loss: 0.7041 - val_accuracy: 0.7473
Epoch 35/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7262 - accuracy: 0.73
58 - val_loss: 0.7672 - val_accuracy: 0.7226- accuracy:  - ETA: 1s - loss: 0.7164 - accu
racy: 0.73 - ETA: 1s - loss: 0.7173 - accuracy:  - ETA: 1s - loss: 0.7199 - accu - ETA:
1s - loss: 0.7 - ETA: 0s - loss: 0
Epoch 36/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7267 - accuracy: 0.73
47 - val_loss: 0.7312 - val_accuracy: 0.7353 loss: 0.7293  - ETA: 0s - loss: 0.7268 - ac
curacy: 0.73 - ETA: 0s - loss: 0 - ETA: 0s - loss: 0.7263 - accuracy: 0.
Epoch 37/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7240 - accuracy: 0.73
36 - val_loss: 0.7092 - val_accuracy: 0.7411
```

```
Epoch 38/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7206 - accuracy: 0.73
46 - val_loss: 0.7498 - val_accuracy: 0.7341 loss: 0.7202 -
Epoch 39/150
1500/1500 [==============================] - 5s 4ms/step - loss: 0.7203 - accuracy: 0.73
59 - val_loss: 0.7056 - val_accuracy: 0.7476
Epoch 40/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7157 - accuracy: 0.73
82 - val_loss: 0.8012 - val_accuracy: 0.7101 - loss: 0.7219 - accuracy - ETA: 1s - loss:
0.7212 -  - E
Epoch 41/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7165 - accuracy: 0.73
56 - val_loss: 0.8142 - val_accuracy: 0.7122
Epoch 42/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7117 - accuracy: 0.73
74 - val_loss: 0.6999 - val_accuracy: 0.7466
Epoch 43/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7129 - accuracy: 0.73
81 - val_loss: 0.7268 - val_accuracy: 0.74012s - - ETA: 0s - loss: 0.7093
Epoch 44/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7126 - accuracy: 0.74
00 - val_loss: 0.7409 - val_accuracy: 0.7320cy: 0.74 - ETA: 1s - loss: 0.7086 - accura -
ETA: 1s - loss: 0.7105 - accuracy - ETA: 1s - los - ETA: 0s - loss: 0.7
Epoch 45/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7129 - accuracy: 0.73
60 - val_loss: 0.7364 - val_accuracy: 0.73593s - loss: 0.7312 - accu - - ETA: 0s - loss:
0.7135 - accuracy: 0.73 - ETA: 0s - loss: 0.7131 - accuracy: 0.73
Epoch 46/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7047 - accuracy: 0.74
36 - val_loss: 0.8470 - val_accuracy: 0.7023cy: 0.74 - ETA: 0s - loss: 0.7078 - accurac
y:  - ETA: 0s - loss: 0.7067 - accura - ETA: 0s - loss: 0.7043 - accura
Epoch 47/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7034 - accuracy: 0.74
34 - val_loss: 0.6961 - val_accuracy: 0.7516oss: - ETA: 1s - loss: 0.6998 - accuracy: 0.
- ETA: 1s - loss: 0.6998 - accuracy - ETA: 1s - loss: - ETA: 0s - loss: 0.7053  - ETA: 0
s - loss: 0.7034 - accuracy:
Epoch 48/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7032 - accuracy: 0.74
21 - val_loss: 0.7497 - val_accuracy: 0.72880s - loss: 0.7066 - accuracy: 0.73 - ETA: 0s
- loss: 0.7057 - ac - ETA: 0s - loss: 0.7
Epoch 49/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7048 - accuracy: 0.74
15 - val_loss: 0.7093 - val_accuracy: 0.7402: 0s - loss: 0.7052 - accuracy: 0.
Epoch 50/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7020 - accuracy: 0.74
24 - val_loss: 0.6922 - val_accuracy: 0.7542
Epoch 51/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.7037 - accuracy: 0.74
15 - val_loss: 0.6952 - val_accuracy: 0.7513 - loss: - ETA: 1s - loss: 0.7030 -  - ETA:
1s - loss: 0.703 - ETA: 0s - loss: 0.7033 - accuracy - ETA: 0s - loss: 0.7036 - ac
Epoch 52/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6989 - accuracy: 0.74
61 - val_loss: 0.7080 - val_accuracy: 0.7444racy: 0. - ETA: 1s - los - ETA: 0s - loss:
0.6994 - accuracy:
Epoch 53/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6971 - accuracy: 0.74
51 - val_loss: 0.7600 - val_accuracy: 0.7286
Epoch 54/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6970 - accuracy: 0.74
67 - val_loss: 0.6925 - val_accuracy: 0.74983s - loss: 0.683 - ETA: 3s - loss: 0.6856 -
```

```
accuracy: 0.75 - ETA: 3s - ETA: 1s - loss: 0.6995 -  - ETA: 0s - loss: 0.6977 - accurac
y: 0. - ETA: 0s - loss: 0.6980 -  - ETA: 0s - loss: 0.6966 - accuracy
Epoch 55/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6943 - accuracy: 0.74
45 - val_loss: 0.7148 - val_accuracy: 0.7433 los - - ETA: 1s - ETA: 0s - loss: 0.6969 -
accuracy:  - ETA: 0s -
Epoch 56/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6921 - accuracy: 0.74
74 - val_loss: 0.7152 - val_accuracy: 0.7366 ETA: 2s - loss: 0.6837 - accuracy: 0.75 - E
- ETA: 1s - loss: 0.6836 - accuracy: 0.75 - ETA: 1s - - ETA: 0s - loss: 0.6905 - accurac
y: 0. - ETA: 0s - loss: 0.6903 - ac
Epoch 57/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6923 - accuracy: 0.74
57 - val_loss: 0.7080 - val_accuracy: 0.7407966 - accuracy:  - ETA:  - ETA: 1s - loss: -
ETA: 1s - loss: 0.690 - ETA: 0s - loss: 0.6917 - accuracy: 0. - ETA: 0s - loss: 0.6911 -
accuracy:  - ETA: 0s - loss: 0.6904 - accuracy - ETA: 0s - loss: 0.6922 - accura
Epoch 58/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6937 - accuracy: 0.74
54 - val_loss: 0.7179 - val_accuracy: 0.7415.6978 - accuracy:  - ETA: 1s - loss: 0.695 -
ETA: 1s - loss: - ETA: 0s - loss: 0.6916 - accuracy:  - ETA: 0s - loss: 0.6929 - ac
Epoch 59/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6920 - accuracy: 0.74
52 - val_loss: 0.6981 - val_accuracy: 0.7476
Epoch 60/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6888 - accuracy: 0.74
67 - val_loss: 0.7070 - val_accuracy: 0.7468 - loss: 0.6905 - accuracy: 0.74 - ETA: 3s -
loss: - ETA: 2s - loss: 0.6925 - accuracy - ETA: 2s - loss: 0.6926 -  - ETA: 2s - loss:
0.6923 - accu - ETA: 1s -
Epoch 61/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6888 - accuracy: 0.74
78 - val_loss: 0.7011 - val_accuracy: 0.7482oss: 0.6917 - accuracy: 0.74 - ETA: 1s - los
s: 0.6 - ETA: 0s - loss: 0.6908 -  - ETA: 0s - loss: 0.6889 - ac
Epoch 62/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6861 - accuracy: 0.74
76 - val_loss: 0.6987 - val_accuracy: 0.7483 0.6959 - accuracy - ETA: 3s - loss: 0.6840
- accuracy: 0.74 - ETA: 3s - loss: - ETA: 1s - loss: 0.6830 - accuracy - ETA: 1s - loss:
0.6842 - accu - ETA: 0s - loss: 0.6836 - accuracy: 0.74 - ETA: 0s - loss: 0.6 - ETA: 0s
- loss: 0.6833 - accuracy:  - ETA: 0s - loss: 0.6844 - accuracy: 0.74 - ETA: 0s - loss:
0.6845 - accuracy:
Epoch 63/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6886 - accuracy: 0.74
74 - val_loss: 0.7333 - val_accuracy: 0.7389 loss: - ETA:
Epoch 64/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6890 - accuracy: 0.74
78 - val_loss: 0.7823 - val_accuracy: 0.7228 - los - ETA: 2s - loss: 0.6927  - ETA: 2s -
loss: - ETA: 1s - loss: 0 - ETA: 0s - loss: 0.6885 - accuracy: 0.
Epoch 65/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6835 - accuracy: 0.74
99 - val_loss: 0.6723 - val_accuracy: 0.7574
Epoch 66/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6836 - accuracy: 0.74
95 - val_loss: 0.6983 - val_accuracy: 0.7475TA: 1s - loss: 0.6836 - accuracy: 0.74 - ET
A: 1s - loss: 0.6832 - accuracy:  - ETA: 0s - loss: 0.6837 - accuracy:  - ETA: 0s - los
s: 0.684 - ETA: 0s - loss: 0.6844 - ac
Epoch 67/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6844 - accuracy: 0.74
93 - val_loss: 0.6946 - val_accuracy: 0.7477cy - ETA: 2s - loss: 0 - E - ETA: 1s - loss:
0.6867  - ETA: 0s - loss: 0.687 - ETA: 0s - loss: 0.6858 - accura
Epoch 68/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6858 - accuracy: 0.74
```

```
81 - val_loss: 0.6718 - val_accuracy: 0.7560
Epoch 69/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6822 - accuracy: 0.74
93 - val_loss: 0.7076 - val_accuracy: 0.7400s: 0.6822 - accuracy - ETA: 1s - loss: 0.680
2 - accuracy:  - ETA: 0s
Epoch 70/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6810 - accuracy: 0.74
95 - val_loss: 0.7083 - val_accuracy: 0.7470s: 0.6672 - accura - ETA: 3s - loss: 0.6706
 - accuracy - ETA: 2s - loss: 0.6699 - accu - ETA: 2s - loss: 0.6761 - accura - ETA:  - E
TA: 1s - loss: 0.6 - ETA: 0s - los - ETA: 0s - loss: 0.6804 - accuracy:
Epoch 71/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6790 - accuracy: 0.75
21 - val_loss: 0.6856 - val_accuracy: 0.7506
Epoch 72/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6820 - accuracy: 0.74
95 - val_loss: 0.6848 - val_accuracy: 0.7514
Epoch 73/150
1500/1500 [==============================] - 5s 4ms/step - loss: 0.6733 - accuracy: 0.75
36 - val_loss: 0.7120 - val_accuracy: 0.7411
Epoch 74/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6772 - accuracy: 0.75
15 - val_loss: 0.7015 - val_accuracy: 0.7434
Epoch 75/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6765 - accuracy: 0.75
08 - val_loss: 0.6820 - val_accuracy: 0.7553
Epoch 76/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6764 - accuracy: 0.75
10 - val_loss: 0.6808 - val_accuracy: 0.7566
Epoch 77/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6746 - accuracy: 0.75
32 - val_loss: 0.7096 - val_accuracy: 0.7403
Epoch 78/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6741 - accuracy: 0.75
08 - val_loss: 0.6739 - val_accuracy: 0.7517
Epoch 79/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6725 - accuracy: 0.75
24 - val_loss: 0.6745 - val_accuracy: 0.7598
Epoch 80/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6741 - accuracy: 0.75
26 - val_loss: 0.6879 - val_accuracy: 0.7537
Epoch 81/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6729 - accuracy: 0.75
26 - val_loss: 0.7067 - val_accuracy: 0.7402 - loss: 0.6586 - accu - E
Epoch 82/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6732 - accuracy: 0.75
37 - val_loss: 0.6896 - val_accuracy: 0.7499- accuracy:  - ETA: 1s - loss: 0 - ETA: 0s -
loss: 0.671
Epoch 83/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6705 - accuracy: 0.75
28 - val_loss: 0.6811 - val_accuracy: 0.7485oss: 0.6756 -
Epoch 84/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6715 - accuracy: 0.75
37 - val_loss: 0.6711 - val_accuracy: 0.7584: 0s - los
Epoch 85/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6714 - accuracy: 0.75
37 - val_loss: 0.6992 - val_accuracy: 0.7487
Epoch 86/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6706 - accuracy: 0.75
42 - val_loss: 0.6823 - val_accuracy: 0.7542-  - ETA - ETA: 0s -
Epoch 87/150
```

```
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6676 - accuracy: 0.75
46 - val_loss: 0.6872 - val_accuracy: 0.7527 - loss: 0 - ETA: 2s - loss: 0.6649 - accura
cy: 0.75 - ETA: 2s - loss: 0.6650 - accuracy: 0. - ETA: 2s - l - ETA: 1s - loss: 0.662 -
ETA: 0s -
Epoch 88/150
1500/1500 [==============================] - 8s 6ms/step - loss: 0.6695 - accuracy: 0.75
51 - val_loss: 0.7334 - val_accuracy: 0.7237
Epoch 89/150
1500/1500 [==============================] - 8s 5ms/step - loss: 0.6661 - accuracy: 0.75
44 - val_loss: 0.7192 - val_accuracy: 0.7396
Epoch 90/150
1500/1500 [==============================] - 6s 4ms/step - loss: 0.6693 - accuracy: 0.75
43 - val_loss: 0.7390 - val_accuracy: 0.7314
Epoch 91/150
1500/1500 [==============================] - 5s 4ms/step - loss: 0.6653 - accuracy: 0.75
57 - val_loss: 0.6748 - val_accuracy: 0.7558
Epoch 92/150
1500/1500 [==============================] - 6s 4ms/step - loss: 0.6623 - accuracy: 0.75
71 - val_loss: 0.6998 - val_accuracy: 0.7485
Epoch 93/150
1500/1500 [==============================] - 7s 5ms/step - loss: 0.6649 - accuracy: 0.75
72 - val_loss: 0.6845 - val_accuracy: 0.7597
Epoch 94/150
1500/1500 [==============================] - 7s 4ms/step - loss: 0.6633 - accuracy: 0.75
75 - val_loss: 0.7348 - val_accuracy: 0.7352
Epoch 95/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6643 - accuracy: 0.75
76 - val_loss: 0.6773 - val_accuracy: 0.7548
Epoch 96/150
1500/1500 [==============================] - 6s 4ms/step - loss: 0.6598 - accuracy: 0.75
85 - val_loss: 0.6653 - val_accuracy: 0.7613
Epoch 97/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6656 - accuracy: 0.75
46 - val_loss: 0.7201 - val_accuracy: 0.7436
Epoch 98/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6604 - accuracy: 0.75
60 - val_loss: 0.6757 - val_accuracy: 0.7580
Epoch 99/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6614 - accuracy: 0.75
67 - val_loss: 0.6569 - val_accuracy: 0.7623.6575 - accuracy: 0. - ETA: 0s - loss: 0.658
Epoch 100/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6601 - accuracy: 0.75
74 - val_loss: 0.6785 - val_accuracy: 0.7551s: 0.6610 - accuracy
Epoch 101/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6604 - accuracy: 0.75
73 - val_loss: 0.6830 - val_accuracy: 0.7541
Epoch 102/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6588 - accuracy: 0.75
91 - val_loss: 0.6896 - val_accuracy: 0.7485
Epoch 103/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6613 - accuracy: 0.75
63 - val_loss: 0.6783 - val_accuracy: 0.7525
Epoch 104/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6629 - accuracy: 0.75
64 - val_loss: 0.6876 - val_accuracy: 0.7547.6733 -  - ETA: 2s - ETA: 2s - loss: 0 - ET
A: 1s - loss: 0.6666 - accuracy:  - ETA: 1s - loss: 0 - ETA: 0s - los
Epoch 105/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6601 - accuracy: 0.76
07 - val_loss: 0.7004 - val_accuracy: 0.7437
Epoch 106/150
```

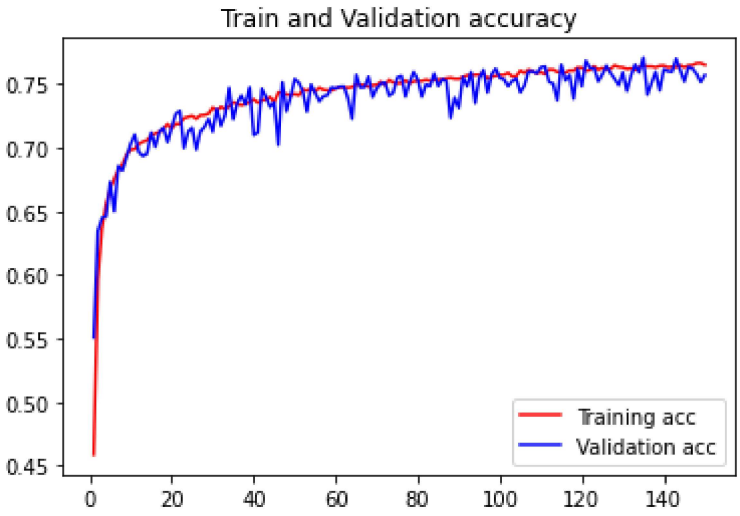
```
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6552 - accuracy: 0.75
90 - val_loss: 0.6923 - val_accuracy: 0.7491 ETA: 0s - loss: 0.6537
Epoch 107/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6551 - accuracy: 0.76
15 - val_loss: 0.6581 - val_accuracy: 0.7600 - loss: 0.6521 - accuracy - ETA: 3s - loss:
0.6660 - accu - ETA: 3s - loss: 0 - ETA: 1s - loss: 0.6 - ETA: 0s -
Epoch 108/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6544 - accuracy: 0.76
03 - val_loss: 0.6638 - val_accuracy: 0.7602
Epoch 109/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6569 - accuracy: 0.75
86 - val_loss: 0.6630 - val_accuracy: 0.7587 ETA: 1s - loss: 0.6578 - ac
Epoch 110/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6558 - accuracy: 0.75
96 - val_loss: 0.6525 - val_accuracy: 0.7634
Epoch 111/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6540 - accuracy: 0.75
81 - val_loss: 0.6585 - val_accuracy: 0.7641
Epoch 112/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6503 - accuracy: 0.76
16 - val_loss: 0.6700 - val_accuracy: 0.7519
Epoch 113/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6497 - accuracy: 0.76
15 - val_loss: 0.6822 - val_accuracy: 0.75061s - los - ETA: 0s - loss: 0.6466 - accurac
y: 0. - ETA: 0s - loss: 0.6 - ETA: 0s - loss: 0.6499 - accuracy: 0.
Epoch 114/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6569 - accuracy: 0.75
98 - val_loss: 0.7199 - val_accuracy: 0.7374 - loss: 0.667 - ETA: 1s - loss: 0.6665 - ac
- ETA: 1s - loss: 0.6631 - ac - ETA: 1s - loss: 0.663 - ETA: 0s - loss: 0.660 - ETA: 0s
- loss: 0.6572 - accura
Epoch 115/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6554 - accuracy: 0.75
97 - val_loss: 0.6593 - val_accuracy: 0.7653ac - ETA: 0s - loss:
Epoch 116/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6539 - accuracy: 0.75
92 - val_loss: 0.6810 - val_accuracy: 0.7531s: 0.6560 -  - ETA: 1s - loss: 0.6564 -  - E
TA: 0s - loss: 0.654 - ETA: 0s - loss: 0.6531 - accuracy: 0.75 - ETA: 0s - loss: 0.6533
- accuracy: 0.75 - ETA: 0s - loss: 0.6535 - accu
Epoch 117/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6513 - accuracy: 0.76
09 - val_loss: 0.6766 - val_accuracy: 0.7573
Epoch 118/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6537 - accuracy: 0.76
08 - val_loss: 0.7187 - val_accuracy: 0.7386s: 0.6506 - accuracy: 0.76 - ETA: 3s - loss:
0.6518 - accuracy - E - ETA: 1s - loss: 0.6562 -  - ETA: 1s - loss: 0.6558 - accuracy:
- E - ETA: 0s - loss: 0.6547 - accuracy
Epoch 119/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6503 - accuracy: 0.76
32 - val_loss: 0.6516 - val_accuracy: 0.7619 - loss: 0.6472 -  - ETA: 1s - loss: 0 - ET
A: 0s - los - ETA: 0s - loss: 0.6494 - accu
Epoch 120/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6503 - accuracy: 0.76
30 - val_loss: 0.6905 - val_accuracy: 0.7485 loss: 0.6 - ETA: 0s - loss: 0.6514 - accura
Epoch 121/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6509 - accuracy: 0.76
06 - val_loss: 0.6493 - val_accuracy: 0.7684
Epoch 122/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6485 - accuracy: 0.76
14 - val_loss: 0.6524 - val_accuracy: 0.7628 - ETA: 2s - loss: 0.6495 - accura - ETA: 2s
- loss: 0.6469 - accu - E
```

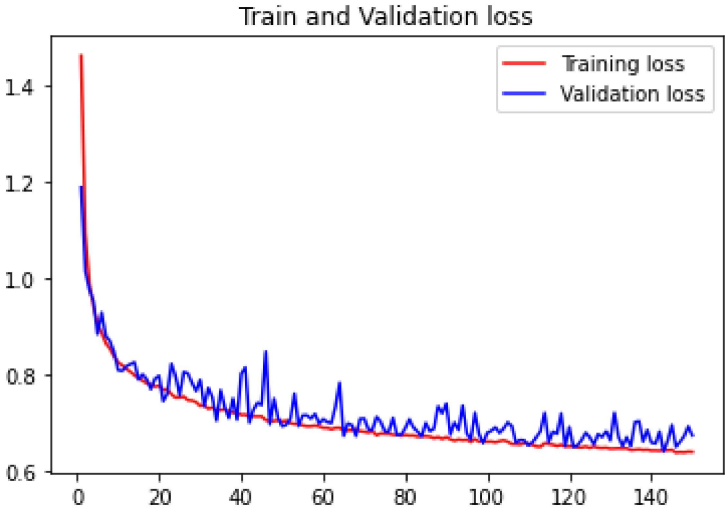
```
Epoch 123/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6483 - accuracy: 0.76
43 - val_loss: 0.6637 - val_accuracy: 0.7626s: 0.6 - ETA:  - ETA: 1s - loss: 0.6461 - ac
- ETA: 0s - loss: 0.6466 - accura - ETA: 0s - los
Epoch 124/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6500 - accuracy: 0.76
15 - val_loss: 0.6790 - val_accuracy: 0.7523
Epoch 125/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6474 - accuracy: 0.76
29 - val_loss: 0.6689 - val_accuracy: 0.7577
Epoch 126/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6502 - accuracy: 0.76
20 - val_loss: 0.6543 - val_accuracy: 0.7648
Epoch 127/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6476 - accuracy: 0.76
21 - val_loss: 0.6757 - val_accuracy: 0.7588 0.6452 - ac - ETA: 0s - loss: 0.6459 - accu
racy:  - ETA: 0s - loss: 0.6466 - accuracy: 0. - ETA: 0s - loss: 0.6470 -  - ETA: 0s - l
oss: 0.6478 - accuracy:
Epoch 128/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6472 - accuracy: 0.76
50 - val_loss: 0.6697 - val_accuracy: 0.7544
Epoch 129/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6477 - accuracy: 0.76
36 - val_loss: 0.6817 - val_accuracy: 0.7494 loss: 0.6446 - ac - ETA: 3s - loss: 0.6522
- accuracy:  - ETA: 3s - loss: 0.6566 - accuracy: 0. - ETA: 3s - loss: 0.6541 - accurac
y:  - ETA: 1s - - ETA: 1s - loss: 0.6497 - accuracy - ETA: 0s - loss: 0.6497 - accu - ET
A: 0s - loss: 0
Epoch 130/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6456 - accuracy: 0.76
34 - val_loss: 0.6627 - val_accuracy: 0.7595 loss:
Epoch 131/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6462 - accuracy: 0.76
21 - val_loss: 0.7199 - val_accuracy: 0.7448 ETA: 0s - loss: 0.648 - ETA: 0s - loss: 0.6
461 - accuracy: 0.76 - ETA: 0s - loss: 0.6464 - accuracy:
Epoch 132/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6464 - accuracy: 0.76
26 - val_loss: 0.6699 - val_accuracy: 0.7568.6471 - accuracy: 0.
Epoch 133/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6461 - accuracy: 0.76
14 - val_loss: 0.6513 - val_accuracy: 0.7656 0.6463 - accu - ETA: 0s - loss: 0.6460
Epoch 134/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6464 - accuracy: 0.76
35 - val_loss: 0.6694 - val_accuracy: 0.7593s: 0.6478 - accuracy - ETA: 0s -
Epoch 135/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6427 - accuracy: 0.76
49 - val_loss: 0.6511 - val_accuracy: 0.7709
Epoch 136/150
1500/1500 [==============================] - 5s 4ms/step - loss: 0.6437 - accuracy: 0.76
41 - val_loss: 0.7006 - val_accuracy: 0.7419
Epoch 137/150
1500/1500 [==============================] - 5s 4ms/step - loss: 0.6443 - accuracy: 0.76
41 - val_loss: 0.7025 - val_accuracy: 0.7512
Epoch 138/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6429 - accuracy: 0.76
34 - val_loss: 0.6602 - val_accuracy: 0.7598
Epoch 139/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6435 - accuracy: 0.76
46 - val_loss: 0.6843 - val_accuracy: 0.7452
Epoch 140/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6417 - accuracy: 0.76
```

```
48 - val_loss: 0.6582 - val_accuracy: 0.7617: 3s - loss: 0.6195 - accu - E - ETA: 2s - l
os - ETA: 1s - loss: 0.6463 - accu - ETA: 1s - loss: 0.645 - ETA: 0s - loss: 0.6424 - ac
cu - ETA: 0s - loss: 0
Epoch 141/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6417 - accuracy: 0.76
33 - val_loss: 0.6563 - val_accuracy: 0.76033s - loss: 0.6420 -  - ETA: 2s - loss: 0.634
9 - accuracy:  - ETA: 2s - loss: 0.6377  - ETA: 2s - loss: 0.633 - ETA: 1s - loss: 0
Epoch 142/150
1500/1500 [==============================] - 6s 4ms/step - loss: 0.6412 - accuracy: 0.76
38 - val_loss: 0.6813 - val_accuracy: 0.7599
Epoch 143/150
1500/1500 [==============================] - 7s 4ms/step - loss: 0.6430 - accuracy: 0.76
34 - val_loss: 0.6395 - val_accuracy: 0.7702
Epoch 144/150
1500/1500 [==============================] - 5s 4ms/step - loss: 0.6423 - accuracy: 0.76
50 - val_loss: 0.6620 - val_accuracy: 0.7609
Epoch 145/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6431 - accuracy: 0.76
34 - val_loss: 0.6950 - val_accuracy: 0.7522
Epoch 146/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6375 - accuracy: 0.76
49 - val_loss: 0.6496 - val_accuracy: 0.7636
Epoch 147/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6385 - accuracy: 0.76
51 - val_loss: 0.6601 - val_accuracy: 0.7624
Epoch 148/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6379 - accuracy: 0.76
67 - val_loss: 0.6714 - val_accuracy: 0.7581
Epoch 149/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6394 - accuracy: 0.76
68 - val_loss: 0.6918 - val_accuracy: 0.7517
Epoch 150/150
1500/1500 [==============================] - 5s 3ms/step - loss: 0.6389 - accuracy: 0.76
53 - val_loss: 0.6731 - val_accuracy: 0.7572
```

In [25]:

```python
# plot the loss and accuracy

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

plt.title('Train and Validation accuracy')
plt.plot(epochs, acc, 'red', label='Training acc')
plt.plot(epochs, val_acc, 'blue', label='Validation acc')
plt.legend()

plt.figure()
plt.title('Train and Validation loss')
plt.plot(epochs, loss, 'red', label='Training loss')
plt.plot(epochs, val_loss, 'blue', label='Validation loss')
plt.legend()

plt.show()
```





VGG got us good results, but Random Forest and KNN have better scores

In [ ]:

In [ ]:

In [26]:
```
#Prepare the training and testing dataset
X_train = df_train.drop(columns='label')
y_train = df_train['label']

X_test = df_test.drop(columns='label')
y_test = df_test['label']
```

ADABOOST

In [27]:
```
# Boosting refers to any method that can combine several weak learners into a strong le
# of mostboosting methods is to train predictors sequentially, each trying to correct i

# The AdaBoost classifier begins by fitting a classifier on the original dataset and th
```

```
# copies of the classifier on the same dataset but where the weights of incorrectly cla
# are adjusted such that subsequent classifiers focus more on difficult cases.
```

In [28]:
```
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier(n_estimators=100)
ada.fit(X_train, y_train)
ada_predictions= ada.predict(X_test)
```

In [29]:
```
#Classification report
print(classification_report(y_test, ada_predictions, target_names= class_names))
```

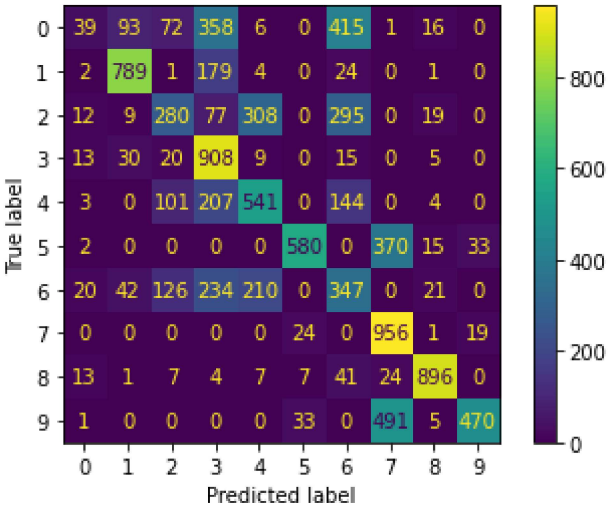
```
              precision    recall  f1-score   support

 T_shirt/top       0.37      0.04      0.07      1000
     Trouser       0.82      0.79      0.80      1000
    Pullover       0.46      0.28      0.35      1000
       Dress       0.46      0.91      0.61      1000
        Coat       0.50      0.54      0.52      1000
      Sandal       0.90      0.58      0.71      1000
       Shirt       0.27      0.35      0.30      1000
     Sneaker       0.52      0.96      0.67      1000
         Bag       0.91      0.90      0.90      1000
  Ankle boot       0.90      0.47      0.62      1000

    accuracy                           0.58     10000
   macro avg       0.61      0.58      0.56     10000
weighted avg       0.61      0.58      0.56     10000
```

In [30]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, ada_predictions)
```

Out[30]:
```
0.5806
```

In [31]:
```
ConfusionMatrixDisplay.from_estimator(ada, X_test, y_test)#, cmap='magma')
```

Out[31]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21816c63c40>
```



In [ ]:

In [ ]:

RANDOM FOREST

In [32]:
```
# Random Forest is a model made up of many decision trees. When using a decision tree c
# overfitting if the data has high variance. As an alternative to limiting the depth of
# variance and increases bias, we combine many decision trees into a random forest.

# When training, each tree in a random forest learns from a random sample of the data p
# that some samples will be used multiple times in a single tree. The idea is that by t
# different samples, although each tree might have high variance, the entire forest wil
# not at the cost of increasing the bias.
```

In [33]:
```
#Creating the model with 100 trees
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)
rf_predictions= rf.predict(X_test)
```

In [34]:
```
print(classification_report(y_test, rf_predictions, target_names= class_names))
```

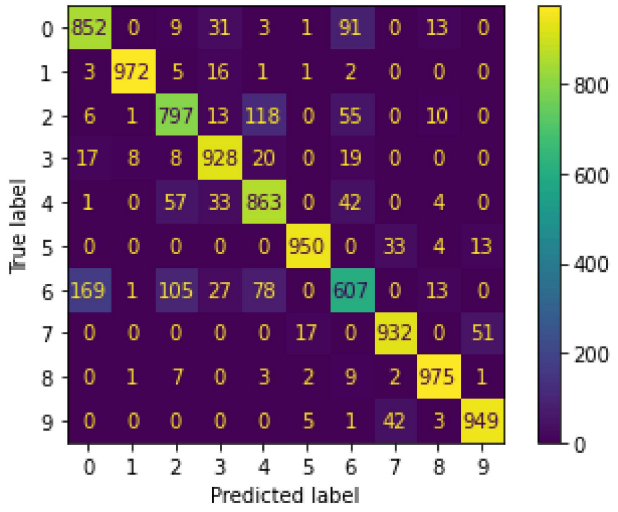
```
              precision    recall  f1-score   support

 T_shirt/top       0.81      0.85      0.83      1000
     Trouser       0.99      0.97      0.98      1000
    Pullover       0.81      0.80      0.80      1000
       Dress       0.89      0.93      0.91      1000
        Coat       0.79      0.86      0.83      1000
      Sandal       0.97      0.95      0.96      1000
       Shirt       0.73      0.61      0.66      1000
     Sneaker       0.92      0.93      0.93      1000
         Bag       0.95      0.97      0.96      1000
  Ankle boot       0.94      0.95      0.94      1000

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

In [35]:
```
accuracy_score(y_test, rf_predictions)
```

Out[35]:
```
0.8825
```

In [36]:
```
ConfusionMatrixDisplay.from_estimator(rf, X_test, y_test)#, cmap='magma')
```

Out[36]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x217f54d27f0>
```



In [ ]:

In [ ]:

NAÏVE BAYES

In [37]:
```
# In probability theory and statistics, Bayes' theorem describes the probability of an
# knowledge of conditions that might be related to the event.

# Naive Bayes is based on applying Bayes' theorem with strong (naive) independence assu
# The fundamental Naive Bayes assumption is that each feature makes an independent cont
```

In [38]:
```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train,y_train)
nb_predictions= nb.predict(X_test)
```

In [39]:
```
print(classification_report(y_test, nb_predictions, target_names= class_names))
```

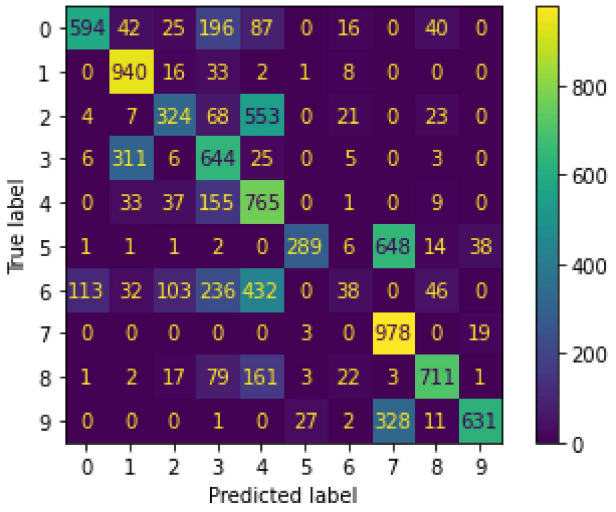
```
              precision    recall  f1-score   support

 T_shirt/top       0.83      0.59      0.69      1000
     Trouser       0.69      0.94      0.79      1000
    Pullover       0.61      0.32      0.42      1000
       Dress       0.46      0.64      0.53      1000
        Coat       0.38      0.77      0.51      1000
      Sandal       0.89      0.29      0.44      1000
       Shirt       0.32      0.04      0.07      1000
     Sneaker       0.50      0.98      0.66      1000
         Bag       0.83      0.71      0.77      1000
  Ankle boot       0.92      0.63      0.75      1000

    accuracy                           0.59     10000
   macro avg       0.64      0.59      0.56     10000
weighted avg       0.64      0.59      0.56     10000
```

In [40]:
```
accuracy_score(y_test, nb_predictions)
```

Out[40]:
```
0.5914
```

In [41]:
```
ConfusionMatrixDisplay.from_estimator(nb, X_test, y_test)#, cmap='magma')
```

Out[41]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21819835c40>
```



In [ ]:

In [ ]:

KNN

In [42]:
```
# The k-nearest neighbors algorithm, also known as KNN, is a supervised learning classi
# which uses proximity to make classifications or predictions about the grouping of an
# A class label is assigned on the basis of a majority vote—i.e. the label that is most
# around a given data point will be assigned to it.
```

In [43]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)
knn_predictions= knn.predict(X_test)
```

In [44]:
```
print(classification_report(y_test, knn_predictions, target_names= class_names))
```

```
              precision    recall  f1-score   support

 T_shirt/top       0.75      0.87      0.80      1000
     Trouser       0.99      0.97      0.98      1000
    Pullover       0.74      0.82      0.78      1000
       Dress       0.91      0.87      0.89      1000
        Coat       0.81      0.77      0.79      1000
      Sandal       0.99      0.82      0.90      1000
```

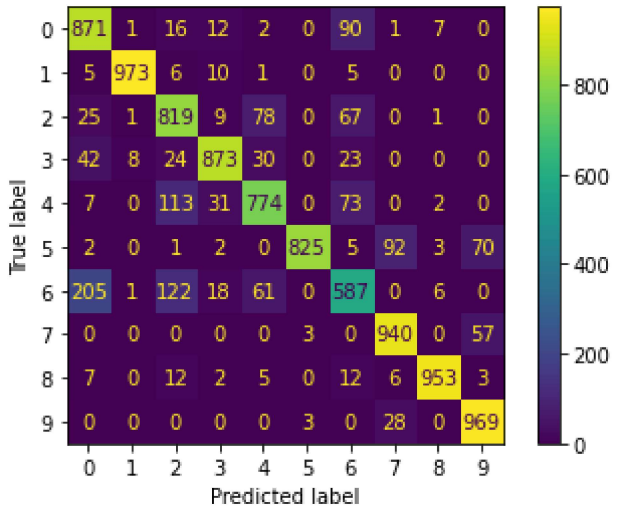
```
              Shirt       0.68      0.59      0.63      1000
            Sneaker       0.88      0.94      0.91      1000
                Bag       0.98      0.95      0.97      1000
         Ankle boot       0.88      0.97      0.92      1000

           accuracy                           0.86     10000
          macro avg       0.86      0.86      0.86     10000
       weighted avg       0.86      0.86      0.86     10000
```

In [45]:
```
accuracy_score(y_test, knn_predictions)
```

Out[45]:
```
0.8584
```

In [46]:
```
ConfusionMatrixDisplay.from_estimator(knn, X_test, y_test)#, cmap='magma')
```

Out[46]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2181986aeb0>
```



In [ ]:

In [ ]:

In [ ]:

Q2 -

We want to check if it's possible to get good predictions based on the colors of the item

If we can get good results that could save a lot of space and a lot of time, since we will only need to save one column for each item instead of 784. We will do this using the mean() command which will give us the average of the gray levels for each item.

In [47]:

```
df_train2 = df_train.copy(deep=True)
df_train2_label = df_train2[['label']]
df_train2 = df_train2.drop(columns='label')
df_train2['average'] = df_train2.mean(axis=1)
df_train2 = df_train2[['average']]
df_train2 = df_train2.join(df_train2_label)


df_test2 = df_test.copy(deep=True)
df_test2_label = df_test2[['label']]
df_test2 = df_test2.drop(columns='label')
df_test2['average'] = df_test2.mean(axis=1)
df_test2 = df_test2[['average']]
df_test2 = df_test2.join(df_test2_label)


df_train2
```

Out[47]:

	average	label
0	115.346939	2
1	58.885204	9
2	120.631378	6
3	86.492347	0
4	85.084184	3
...	...	...
59995	91.038265	9
59996	34.173469	1
59997	87.053571	8
59998	110.553571	8
59999	74.179847	7

60000 rows × 2 columns

Now we can use the same models to try and predict the labels with the new data

In [48]:

```
#Prepare the training and testing dataset
X_train2 = df_train2.drop(columns='label')
y_train2 = df_train2['label']

X_test2 = df_test2.drop(columns='label')
y_test2 = df_test2['label']
```

ADABOOST

In [49]:

```
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier(n_estimators=100)
```

```
ada.fit(X_train2, y_train2)
ada_predictions= ada.predict(X_test2)
```

In [50]:

```
#Classification report
print(classification_report(y_test2, ada_predictions, target_names= class_names, zero_d
```

```
              precision    recall  f1-score   support

 T_shirt/top       0.17      0.01      0.01      1000
     Trouser       0.26      0.55      0.35      1000
    Pullover       0.28      0.27      0.27      1000
       Dress       1.00      0.00      0.00      1000
        Coat       0.23      0.41      0.29      1000
      Sandal       0.55      0.46      0.50      1000
       Shirt       1.00      0.00      0.00      1000
     Sneaker       0.30      0.43      0.35      1000
         Bag       1.00      0.00      0.00      1000
  Ankle boot       0.20      0.57      0.29      1000

    accuracy                           0.27     10000
   macro avg       0.50      0.27      0.21     10000
weighted avg       0.50      0.27      0.21     10000
```

In [51]:

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test2, ada_predictions)
```

Out[51]:

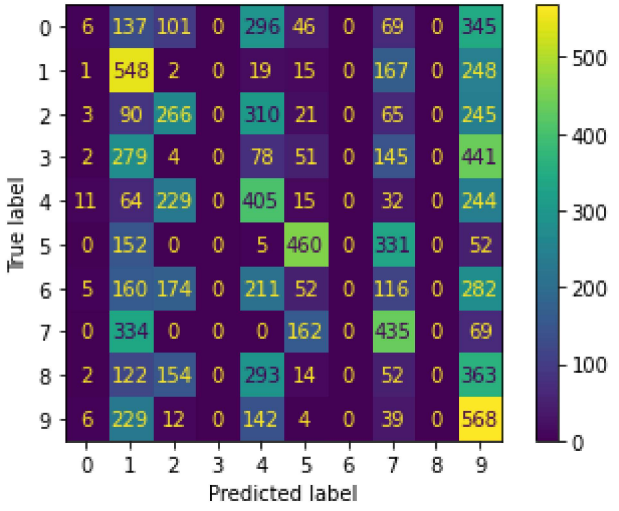
```
0.2688
```

In [52]:

```
ConfusionMatrixDisplay.from_estimator(ada, X_test2, y_test2)
```

Out[52]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21842c3efa0>
```



In [ ]:

In [ ]:

RANDOM FOREST

In [53]:
```
#Creating the model with 100 trees
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train2, y_train2)
rf_predictions= rf.predict(X_test2)
```

In [54]:
```
print(classification_report(y_test2, rf_predictions, target_names= class_names))
```

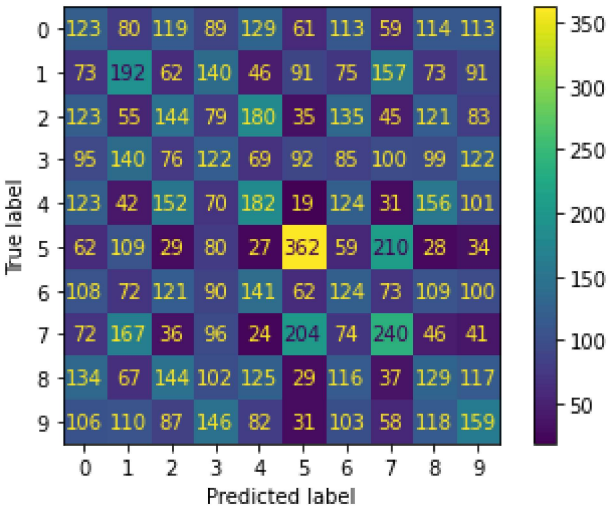
```
              precision    recall  f1-score   support

 T_shirt/top       0.12      0.12      0.12      1000
     Trouser       0.19      0.19      0.19      1000
    Pullover       0.15      0.14      0.15      1000
       Dress       0.12      0.12      0.12      1000
        Coat       0.18      0.18      0.18      1000
      Sandal       0.37      0.36      0.36      1000
       Shirt       0.12      0.12      0.12      1000
     Sneaker       0.24      0.24      0.24      1000
         Bag       0.13      0.13      0.13      1000
  Ankle boot       0.17      0.16      0.16      1000

    accuracy                           0.18     10000
   macro avg       0.18      0.18      0.18     10000
weighted avg       0.18      0.18      0.18     10000
```

In [55]:
```
accuracy_score(y_test2, rf_predictions)
```

Out[55]:
```
0.1777
```

In [56]:
```
ConfusionMatrixDisplay.from_estimator(rf, X_test2, y_test2)
```

Out[56]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21842d9b2e0>
```



In [ ]:

In [ ]:

NAÏVE BAYES

In [57]:
```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train2,y_train2)
nb_predictions= nb.predict(X_test2)
```

In [58]:
```
print(classification_report(y_test2, nb_predictions, target_names= class_names, zero_di
```

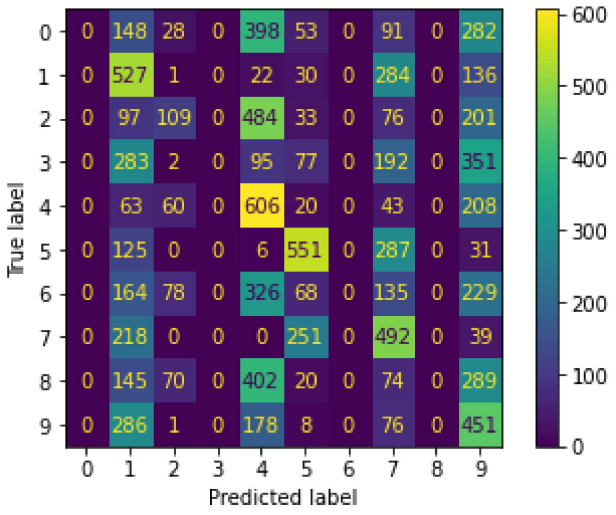
```
              precision    recall  f1-score   support

 T_shirt/top       1.00      0.00      0.00      1000
     Trouser       0.26      0.53      0.34      1000
    Pullover       0.31      0.11      0.16      1000
       Dress       1.00      0.00      0.00      1000
        Coat       0.24      0.61      0.34      1000
      Sandal       0.50      0.55      0.52      1000
       Shirt       1.00      0.00      0.00      1000
     Sneaker       0.28      0.49      0.36      1000
         Bag       1.00      0.00      0.00      1000
  Ankle boot       0.20      0.45      0.28      1000

    accuracy                           0.27     10000
   macro avg       0.58      0.27      0.20     10000
weighted avg       0.58      0.27      0.20     10000
```

In [59]:
```
accuracy_score(y_test2, nb_predictions)
```

Out[59]:
```
0.2736
```

In [60]:
```
ConfusionMatrixDisplay.from_estimator(nb, X_test2, y_test2)
```

Out[60]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21817a4c1f0>
```



In [ ]:

In [ ]:

In [ ]:

KNN

In [61]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train2,y_train2)
knn_predictions= knn.predict(X_test2)
```

In [62]:
```
print(classification_report(y_test2, knn_predictions, target_names= class_names))
```

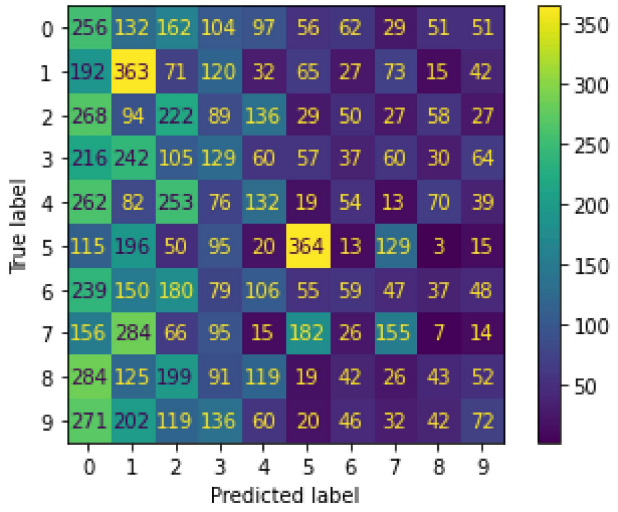
```
              precision    recall  f1-score   support

 T_shirt/top       0.11      0.26      0.16      1000
     Trouser       0.19      0.36      0.25      1000
    Pullover       0.16      0.22      0.18      1000
       Dress       0.13      0.13      0.13      1000
        Coat       0.17      0.13      0.15      1000
      Sandal       0.42      0.36      0.39      1000
       Shirt       0.14      0.06      0.08      1000
     Sneaker       0.26      0.15      0.19      1000
         Bag       0.12      0.04      0.06      1000
  Ankle boot       0.17      0.07      0.10      1000

    accuracy                           0.18     10000
   macro avg       0.19      0.18      0.17     10000
weighted avg       0.19      0.18      0.17     10000
```

In [63]:
```
accuracy_score(y_test2, knn_predictions)
```

Out[63]:
```
0.1795
```

In [64]:
```
ConfusionMatrixDisplay.from_estimator(knn, X_test2, y_test2)
```

Out[64]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21818522970>
```



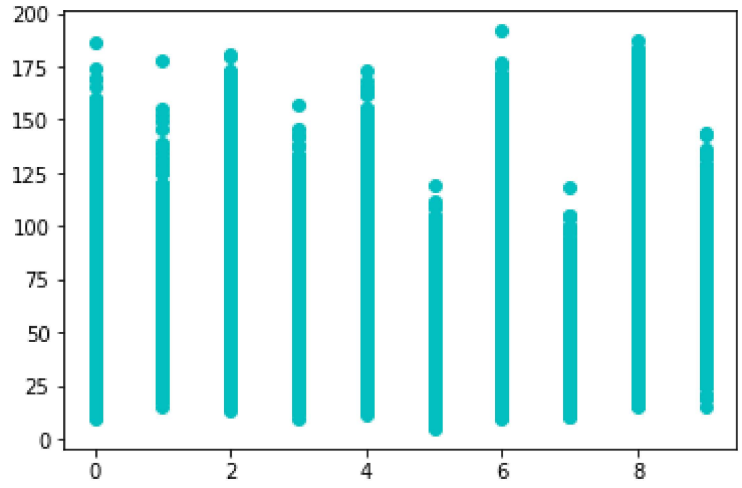
In [ ]:

In [ ]:

We can see that classifying the items based on the everage gray level is not working, the results are a lot worse than the original results. When we looked at the data using the scatter plot we realised that most of the items have the same average of gray level which explains th results.

In [65]:
```
plt.scatter(df_train2.label, df_train2.average, c='c')
```

Out[65]:
```
<matplotlib.collections.PathCollection at 0x218186c5df0>
```



In [ ]:

In [ ]:

Q3 -

We want to check if it's possible to improve the results if we create categories for similar items

Our new labels will represent:

0 - T-shirt/top, shirt

1 - Trouser

2 - Pullover, Coat

3 - Dress

4 - Sandal, Sneaker, Ankle boot

5 - Bag

In [66]:

```
label_map = {0:0, 1:1, 2:2, 3:3, 4:2, 5:4, 6:0, 7:4, 8:5, 9:4}
class_names = ['T_shirt/top, Shirt', 'Trouser', 'Pullover, Coat', 'Dress', 'Sandal, Sne

df_train3 = df_train.copy(deep=True)
df_train3['category'] = df_train3["label"].map(label_map)

df_test3 = df_test.copy(deep=True)
df_test3['category'] = df_test3["label"].map(label_map)


sec_col = df_train3.pop('category')
df_train3.insert(1, 'category', sec_col)
df_train3
```

Out[66]:

	label	category	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel775	pixe
0	2	2	0	0	0	0	0	0	0	0	...	0	
1	9	4	0	0	0	0	0	0	0	0	...	0	
2	6	0	0	0	0	0	0	0	0	5	...	0	
3	0	0	0	0	0	1	2	0	0	0	...	3	
4	3	3	0	0	0	0	0	0	0	0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
59995	9	4	0	0	0	0	0	0	0	0	...	0	
59996	1	1	0	0	0	0	0	0	0	0	...	73	
59997	8	5	0	0	0	0	0	0	0	0	...	160	
59998	8	5	0	0	0	0	0	0	0	0	...	0	
59999	7	4	0	0	0	0	0	0	0	0	...	0	

60000 rows × 786 columns

Now we will use the same models to try and predict the categories with the new data

In [67]:
```
#Prepare the training and testing dataset
X_train3 = df_train3.drop(columns=['label', 'category'])
y_train3 = df_train3['category']

X_test3 = df_test3.drop(columns=['label', 'category'])
y_test3 = df_test3['category']
```

ADABOOST

In [68]:
```
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier(n_estimators=100)
ada.fit(X_train3, y_train3)
ada_predictions = ada.predict(X_test3)
```

In [69]:
```
#Classification report
print(classification_report(y_test3, ada_predictions, target_names=class_names, zero_di
```

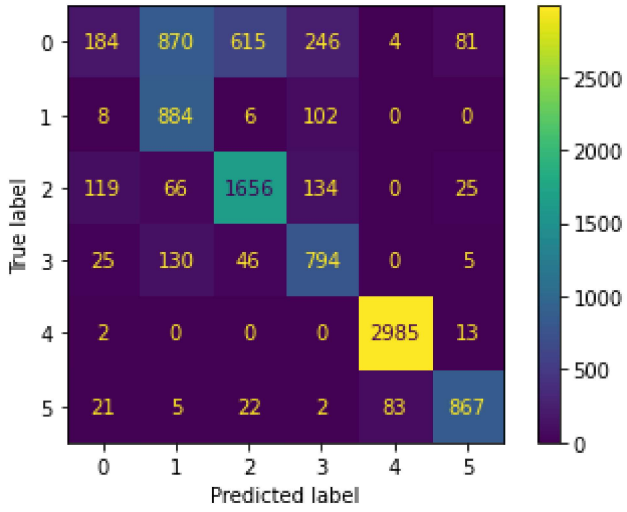
```
                           precision    recall  f1-score   support

        T_shirt/top, Shirt       0.51      0.09      0.16      2000
                   Trouser       0.45      0.88      0.60      1000
           Pullover, Coat        0.71      0.83      0.76      2000
                     Dress       0.62      0.79      0.70      1000
Sandal, Sneaker, Ankle boot      0.97      0.99      0.98      3000
                       Bag       0.87      0.87      0.87      1000

                  accuracy                           0.74     10000
                 macro avg       0.69      0.74      0.68     10000
              weighted avg       0.73      0.74      0.70     10000
```

In [70]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test3, ada_predictions)
```

Out[70]:
```
0.737
```

In [71]:
```
ConfusionMatrixDisplay.from_estimator(ada, X_test3, y_test3)
```

Out[71]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x217c97b4880>
```



In [ ]:

RANDOM FOREST

In [72]:
```
#Creating the model with 100 trees
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train3, y_train3)
rf_predictions= rf.predict(X_test3)
```

In [73]:
```
print(classification_report(y_test3, rf_predictions, target_names= class_names))
```

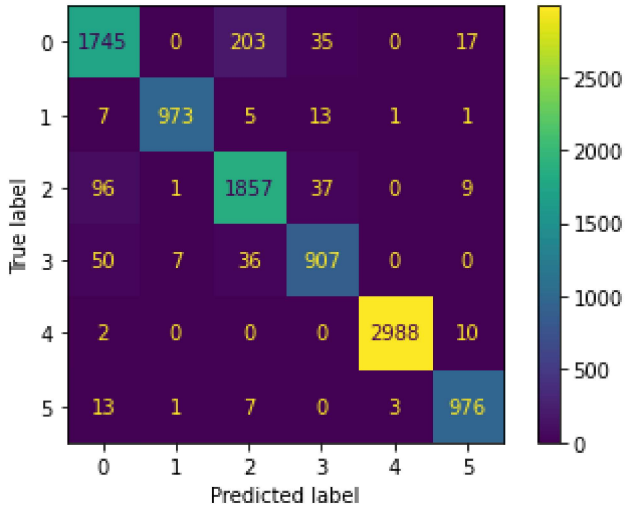
```
                           precision    recall  f1-score   support

        T_shirt/top, Shirt       0.91      0.87      0.89      2000
                  Trouser       0.99      0.97      0.98      1000
             Pullover, Coat       0.88      0.93      0.90      2000
                    Dress       0.91      0.91      0.91      1000
Sandal, Sneaker, Ankle boot       1.00      1.00      1.00      3000
                      Bag       0.96      0.98      0.97      1000

                 accuracy                           0.94     10000
                macro avg       0.94      0.94      0.94     10000
             weighted avg       0.95      0.94      0.94     10000
```

In [74]:
```
accuracy_score(y_test3, rf_predictions)
```

Out[74]:
```
0.9446
```

In [75]:
```
ConfusionMatrixDisplay.from_estimator(rf, X_test3, y_test3)
```

Out[75]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21818688940>
```



In [ ]:

```
```

NAÏVE BAYES

In [76]:

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train3,y_train3)
nb_predictions= nb.predict(X_test3)
```

In [77]:

```
print(classification_report(y_test3, nb_predictions, target_names= class_names, zero_di
```

```
                           precision    recall  f1-score   support

          T_shirt/top, Shirt       0.85      0.35      0.49      2000
                      Trouser       0.69      0.94      0.79      1000
             Pullover, Coat       0.66      0.80      0.72      2000
                       Dress       0.42      0.65      0.51      1000
Sandal, Sneaker, Ankle boot       1.00      0.97      0.98      3000
                         Bag       0.80      0.72      0.76      1000

                    accuracy                           0.75     10000
                   macro avg       0.73      0.74      0.71     10000
                weighted avg       0.79      0.75      0.74     10000
```

In [78]:

```
accuracy_score(y_test3, nb_predictions)
```

Out[78]:

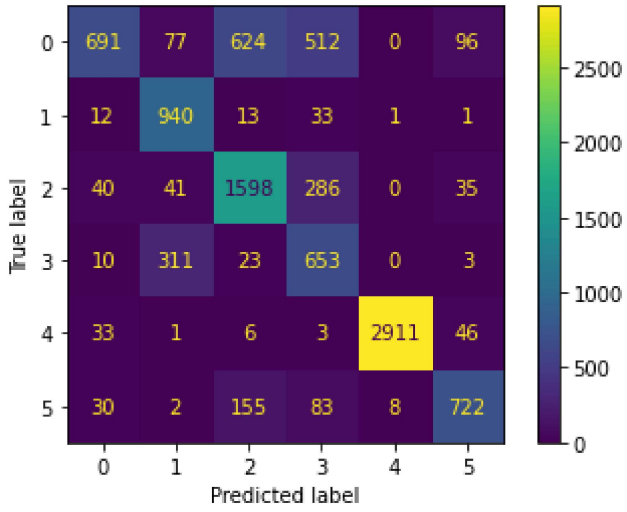
```
0.7515
```

In [79]:

```
ConfusionMatrixDisplay.from_estimator(nb, X_test3, y_test3)
```

Out[79]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21818888040>
```



In [ ]:

KNN

In [80]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train3,y_train3)
knn_predictions= knn.predict(X_test3)
```

In [81]:
```
print(classification_report(y_test3, knn_predictions, target_names= class_names))
```

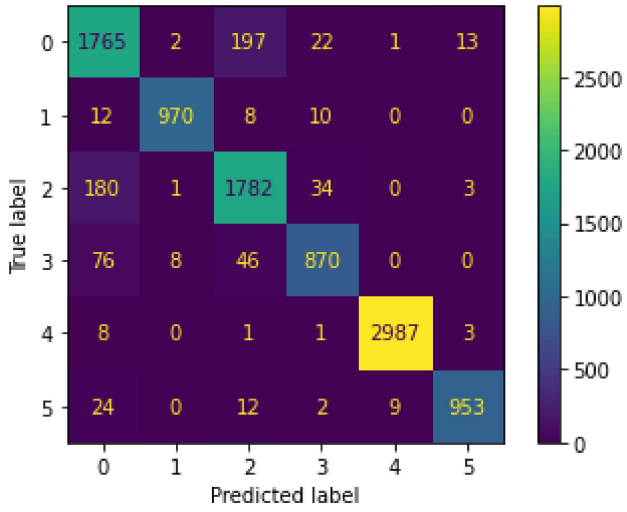
```
                           precision    recall  f1-score   support

        T_shirt/top, Shirt       0.85      0.88      0.87      2000
                   Trouser       0.99      0.97      0.98      1000
       Pullover, Coat        0.87      0.89      0.88      2000
                     Dress       0.93      0.87      0.90      1000
Sandal, Sneaker, Ankle boot       1.00      1.00      1.00      3000
                       Bag       0.98      0.95      0.97      1000

                  accuracy                           0.93     10000
                 macro avg       0.94      0.93      0.93     10000
              weighted avg       0.93      0.93      0.93     10000
```

In [82]:
```
accuracy_score(y_test3, knn_predictions)
```

Out[82]:
```
0.9327
```

In [83]:
```
ConfusionMatrixDisplay.from_estimator(knn, X_test3, y_test3)
```

Out[83]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x218178ebb20>
```



In [ ]:

We can see that the results have improved for all models, and the models that had low results in the original data have improved significantly

In [ ]:

In [ ]:

In [84]:

```
#Visualization
plt.figure(figsize=(10,10))
for i in range(16):
    plt.subplot(4,4,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train[i,1:].reshape((28,28)), cmap=plt.cm.binary)
    plt.title("Image label is: {}".format(y_train[i]))
plt.show()
```


Image label is: 2
Image label is: 9
Image label is: 6
Image label is: 0
Image label is: 3
Image label is: 4
Image label is: 4
Image label is: 5
Image label is: 4
Image label is: 8
Image label is: 0
Image label is: 8
Image label is: 9
Image label is: 0
Image label is: 2
Image label is: 2

In [ ]: