Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 019
Code Title:	Emerging Technologies in CpE 2
2nd Semester	AY 2023-2024
ACTIVITY	**Assignment 9.1**
Name	Cuevas, Christian Jay L.
Section	CPE32S3
Date Performed:	4/26/2024
Date Submitted:	4/27/2024
Instructor:	Engr. Roman M. Richard

Instructions:

- Choose any dataset applicable to an image classification problem
- Explain your datasets and the problem being addressed.
- Show evidence that you can do the following:
  - Using your dataset, create a baseline model of the CNN
  - Perform image augmentation
  - Perform feature standardization
  - Perform ZCA whitening of your images
  - Augment data with random rotations, shifts, and flips
  - Save augmented image data to disk
  - Develop a test harness to develop a robust evaluation of a model and establish a baseline of performance for a classification task
  - Explore extensions to a baseline model to improve learning and model capacity.
  - Develop a finalized model, evaluate the performance of the final model, and use it to make predictions on new images.
- Submit the link to your Google Colab (make sure that it is accessible to me) and the link to your dataset/s

Dataset

- The dataset that I will be using is a dataset present in the keras library, it is the fashion mnist dataset. This dataset is very popular because people can easily access it and it is already labeled.
- The goal of this dataset is to classify different types of fashion clothes or items from the Zalando's article. It consists of 10 labels and 60,000 grayscale 28x28 images.
- The labels for each dataset is shown below:

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

Import libraries and dataset

- We will first import the libraries and the dataset that we will be using.
- Loading a dataset in keras is easy because it is a built in dataset that can be instantly used.

In [ ]:

```
!pip install tensorflow
```

In [ ]:

```
import numpy as np
import seaborn as sns
import pandas as pd
from numpy import mean
from numpy import std
from matplotlib import pyplot as plt
from sklearn.model_selection import KFold
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
import tensorflow as tf
from keras.models import load_model
from matplotlib import image as mpimg
from keras.models import load_model
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, precision_recall_curve, roc_auc_score, roc_c
urve, accuracy_score
```

In [ ]:

```
fashion_mnist = fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-lab
els-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-ima
ges-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labe
ls-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-imag
es-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
```

Preprocessing the image dataset

- **We will start by using a function to load the dataset, reshape it to 28x28 scale and convert it to grayscale.**
- **This function returns 4 values (trainX, trainY, testX, testY).**

In [ ]:

```
def preprocess_dataset(dataset):
 (trainX, trainY), (testX, testY) = dataset
 trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
 testX = testX.reshape((testX.shape[0], 28, 28, 1))
 return trainX, trainY, testX, testY
```

In [ ]:

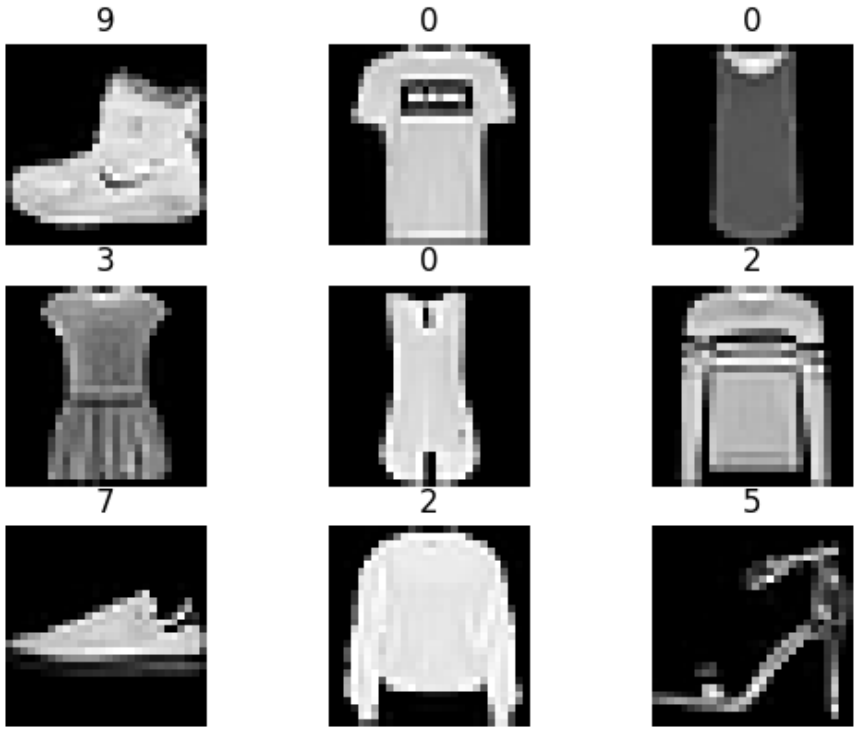
```
def display_images(X,Y):
 print('Train: X=%s, y=%s' % (X.shape, Y.shape))
 print('Test: X=%s, y=%s' % (X.shape, Y.shape))
 for i in range(9):
  plt.subplot(3,3, 1 + i)
```

```
  plt.imshow(X[i], cmap=plt.get_cmap('gray'))
  plt.title(Y[i])
  plt.axis('off')
 plt.show()
```

In [ ]:

```
trainX, trainY, testX, testY = preprocess_dataset(fashion_mnist)
display_images(trainX,trainY)
```

```
Train: X=(60000, 28, 28, 1), y=(60000,)
Test: X=(60000, 28, 28, 1), y=(60000,)
```



Observation:

- **We can observe above the resized grayscale images which was done with the function above. We can observe from the dataset above that there are different fashion items that we use in our everyday life.**
- **We can also observe that above each image is a number, this serves as the label for each image and it will be used later for the predictions of images.**
- **The corresponding names for the fashion items is displayed above in the dataset description.**

- **Normalizing the values of the pixels of an image is very important as this helps in improving the contrast of images and also this makes all the images similar to each other.**

In [ ]:

```
def normalize_pixels(train, test):
 train_norm = train.astype('float32')
 test_norm = test.astype('float32')
 train_norm = train_norm / 255.0
 test_norm = test_norm / 255.0
 return train_norm, test_norm
```

In [ ]:

```
trainX, testX = normalize_pixels(trainX, testX)
```

Training the baseline model

- **In creating a baseline model of convolutional neural network, we need to first understand what are the different layers that we need.**
- **Convolutional layer is a layer which applies a filter to an input and then creates a feature map from it [1]. The parameters of conv2d that we will use are (filters, kernels, activation, kernel_initializer, and input shape). The filter parameter is the number of convolution filters, the kernel specifies the convolution window that is used to scan the image, activation is for the formula that will be used for the values, kernel initializer is for the weight distribution and lastly the input shape is the shape of the input [2].**
- **MaxPooling2D is a layer for pooling which downsamples the feature maps created by the convolutional layer resulting to a robust model that can is not affected by the change in position of the feature. This is also called as local translation invariance [3].**

In [ ]:

```
def define_model():
 model = Sequential()
 model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_sh
ape=(28, 28, 1)))
 model.add(MaxPooling2D((2, 2)))
 model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform'))
 model.add(MaxPooling2D((2, 2)))
 model.add(Flatten())
 model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
 model.add(Dense(10, activation='softmax'))

 opt = SGD(learning_rate=0.01, momentum=0.9)
 model.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy']
)
 return model
```

In [ ]:

```
model = define_model()
model.summary()
fashion_history = model.fit(trainX, trainY, validation_split = 0.2, epochs=10, batch_size=
32, verbose=0)
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_16 (Conv2D)          (None, 26, 26, 32)        320

 max_pooling2d_16 (MaxPooli  (None, 13, 13, 32)        0
 ng2D)

 conv2d_17 (Conv2D)          (None, 11, 11, 64)        18496

 max_pooling2d_17 (MaxPooli  (None, 5, 5, 64)          0
 ng2D)

 flatten_8 (Flatten)         (None, 1600)              0

 dense_16 (Dense)            (None, 100)               160100

 dense_17 (Dense)            (None, 10)                1010

=================================================================
Total params: 179926 (702.84 KB)
Trainable params: 179926 (702.84 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [ ]:

```
model.save('/content/drive/MyDrive/FashionMnist_Augmented/initial_model.h5')
```

In [ ]:

```
, acc = model.evaluate(testX, testY, verbose=0)
```

```
 print('> %.3f' % (acc * 100.0))
```

```
> 90.220
```

In [ ]:

```
#baseline model history
print(fashion_history.history.keys())
```

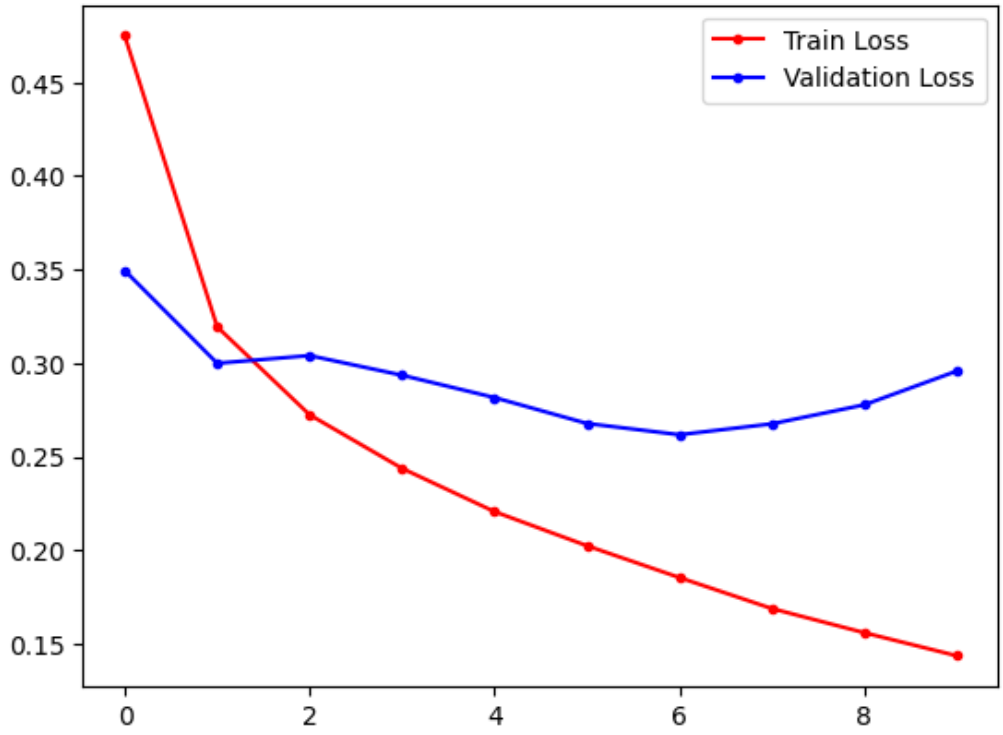
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [ ]:

```
fig, ax = plt.subplots()
ax.plot(fashion_history.history["loss"],'r', marker='.', label="Train Loss")
ax.plot(fashion_history.history["val_loss"],'b', marker='.', label="Validation Loss")
ax.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x78da080d8c10>
```

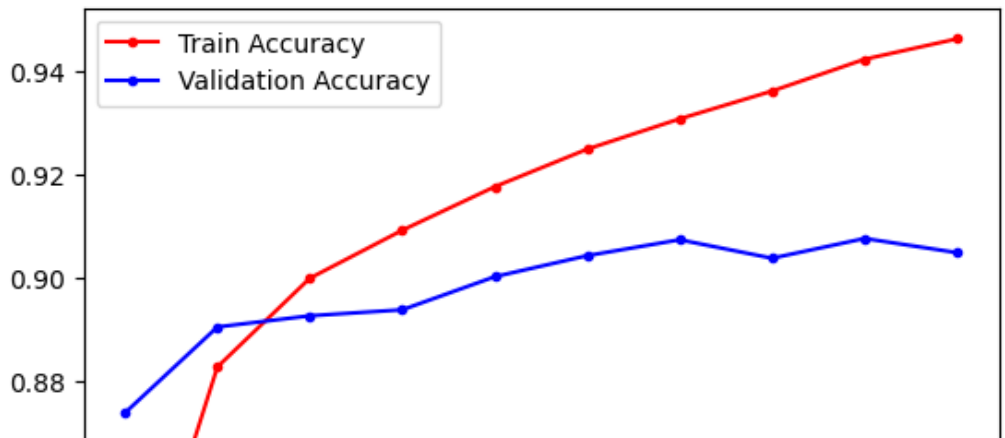


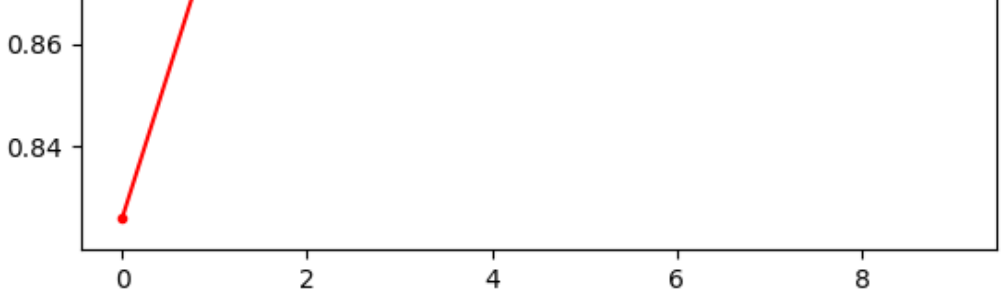
In [ ]:

```
fig, ax = plt.subplots()
ax.plot(fashion_history.history["accuracy"],'r', marker='.', label="Train Accuracy")
ax.plot(fashion_history.history["val_accuracy"],'b', marker='.', label="Validation Accurac
y")
ax.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x78da08515a80>
```





Observation:

- **We can observe above the model summary which outputs the Layers, Output Shape and the Parameters. We can observe that the number of parameters increases from 320 to 160100 until it became only 1010 for the output layer. This shows the frame maps generation of the convolutional layer, which scans the entire image and then uses filter for feature maps.**
- **We can also see that the initial accuracy of our baseline model is 90.22%, which is a very good result considering that we haven't done any image augmentation.**
- **We can observe that the graphs above shows signs of overfitting. This means that the model is overfitted with the training data. We can see that for the loss, the validation loss stopped improving at 0.3 while for the accuracy the validation accuracy stopped improving at 0.9.**
- **This shows the flaw of our model which is overfitting, we can further improve this by introducing new images using image augmentation.**

Image Augmentation

- **Image Augmentation is like the data preparation for ann models, the difference is it is for images and object recognition tasks. Image augmentation API in keras provides us capabilities to do feature standardization, ZCA whitening, Random rotation, shifts, shear, and flips [4]**
- **The first thing that we will do is import ImageDataGenerator to access the API for the image augmentation.**

Feature Standardization

- **In feature standardization, our goal is to standardize the dataset by using featurewise_center and featurewise_std_normalization. These 2 standardizes the dataset by centering the pixels by subtracting the mean from the dataset then it will be divided by the standard deviation.**

In [ ]:

```
def display_dataset(trainX, trainY):
  for X_batch, y_batch in datagen.flow(trainX, trainY, batch_size=9, shuffle=False):
      print(X_batch.min(), X_batch.mean(), X_batch.max())
      # create a grid of 3x3 images
      fig, ax = plt.subplots(3, 3, sharex=True, sharey=True, figsize=(4,4))
      for i in range(3):
          for j in range(3):
              ax[i][j].imshow(X_batch[i*3+j], cmap=plt.get_cmap("gray"))
      # show the plot
      plt.show()
      break
```

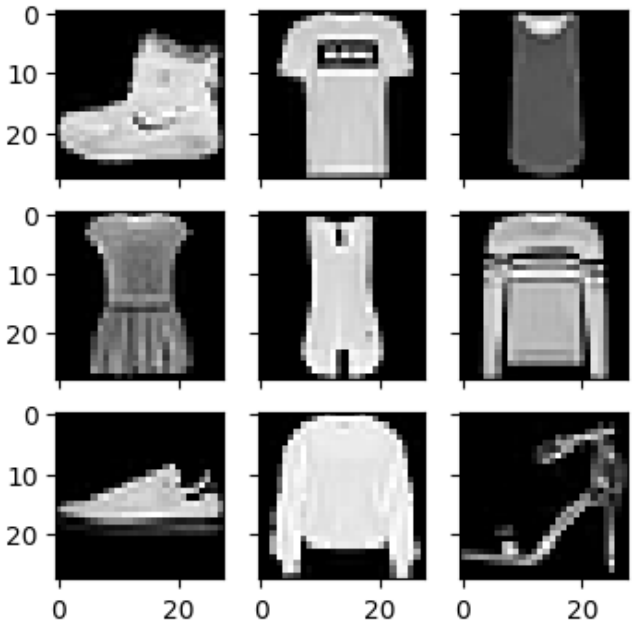
In [ ]:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(featurewise_center=True, featurewise_std_normalization=True)
```

In [ ]:

```
trainX, trainY, testX, testY = preprocess_dataset(fashion_mnist)
trainX, testX = normalize_pixels(trainX, testX)
datagen.fit(trainX)
display_dataset(trainX,trainY)
```

```
-0.810255 0.05422249 2.0224063
```



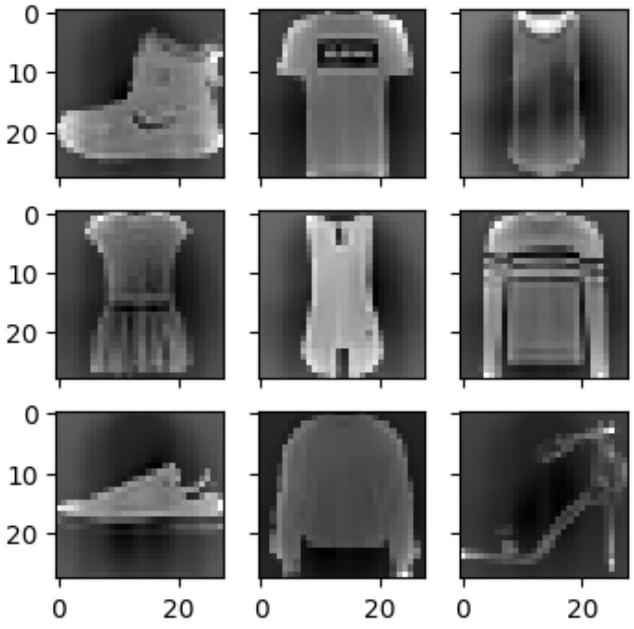
Observation:

- **As we can observe from above, there is no change from the initial imges of the dataset. This is because there is a bug with the keras fit() and it does not do what it is expected to do [4]. So in the next code block, we're manually compute the mean and std of the dataset.**

In [ ]:

```
trainX, trainY, testX, testY = preprocess_dataset(fashion_mnist)
trainX, testX = normalize_pixels(trainX, testX)
datagen.mean = trainX.mean(axis=0)
datagen.std = trainX.std(axis=0)
display_dataset(trainX,trainY)
```

```
-2.4011555 0.05631777 7.557218
```



Observation:

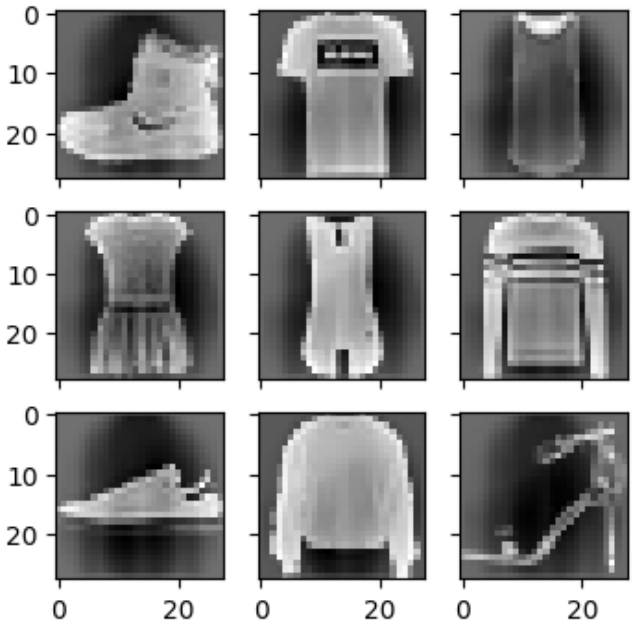
- **Here, by using the manually computed mean and std, we successfully applied the standardization of the images. As you can see, the image became darker especially in the middle part. This can help the network to avoid overfitting especially if there is a change in the feature color.**

ZCA Whitening

In [ ]:

```
trainX, trainY, testX, testY = preprocess_dataset(fashion_mnist)
trainX, testX = normalize_pixels(trainX, testX)
X_mean = trainX.mean(axis=0)
datagen.fit(trainX-X_mean)
for X_batch, y_batch in datagen.flow(trainX - X_mean, trainY, batch_size=9, shuffle=False)
:
    print(X_batch.min(), X_batch.mean(), X_batch.max())
    fig, ax = plt.subplots(3, 3, sharex=True, sharey=True, figsize=(4,4))
    for i in range(3):
        for j in range(3):
            ax[i][j].imshow(X_batch[i*3+j].reshape(28,28), cmap=plt.get_cmap("gray"))
    plt.show()
    break
```

```
-2.1523442 0.0648915 3.1170547
```



Observation:

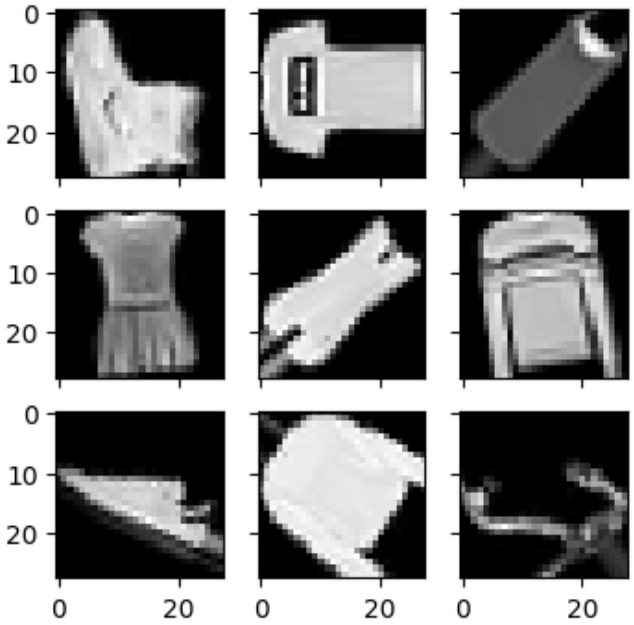
- **Here in ZCA whitening, it is done to also avoid overfitting as it changes the color of pixels around the middle part of the picture. This can avoid correlation of the features and the model can predict a wider range of images especially images in high contrast.**

Random Rotations

In [ ]:

```
trainX, trainY, testX, testY = preprocess_dataset(fashion_mnist)
```

```
trainX, testX = normalize_pixels(trainX, testX)
datagen = ImageDataGenerator(rotation_range=90)

for X_batch, y_batch in datagen.flow(trainX, trainY, batch_size=9, shuffle=False):
    fig, ax = plt.subplots(3, 3, sharex=True, sharey=True, figsize=(4,4))
    for i in range(3):
        for j in range(3):
            ax[i][j].imshow(X_batch[i*3+j].reshape(28,28), cmap=plt.get_cmap("gray"))
    plt.show()
    break
```



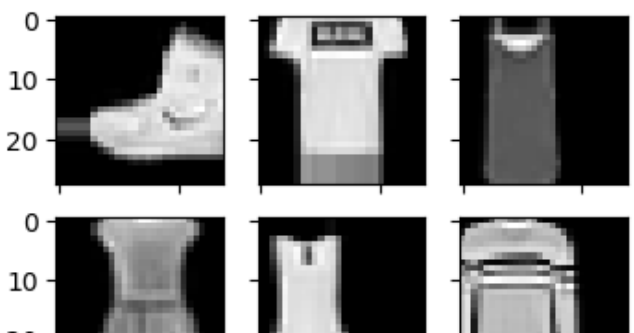
Observations:

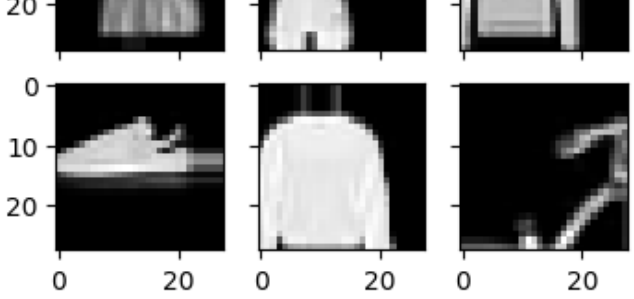
- **Here in random rotations, this is done so that the model will be robust in the change of the location of the feature and it will avoid overfittin especially if all the images have the same orientation. Doing this will also result into an invariance, which means that the model can accurately predict the image in any orientation.**

Random Shifts

In [ ]:

```
trainX, trainY, testX, testY = preprocess_dataset(fashion_mnist)
trainX, testX = normalize_pixels(trainX, testX)
shift = 0.2
datagen = ImageDataGenerator(width_shift_range=shift, height_shift_range=shift)
for X_batch, y_batch in datagen.flow(trainX, trainY, batch_size=9, shuffle=False):
    fig, ax = plt.subplots(3, 3, sharex=True, sharey=True, figsize=(4,4))
    for i in range(3):
        for j in range(3):
            ax[i][j].imshow(X_batch[i*3+j].reshape(28,28), cmap=plt.get_cmap("gray"))
    plt.show()
    break
```





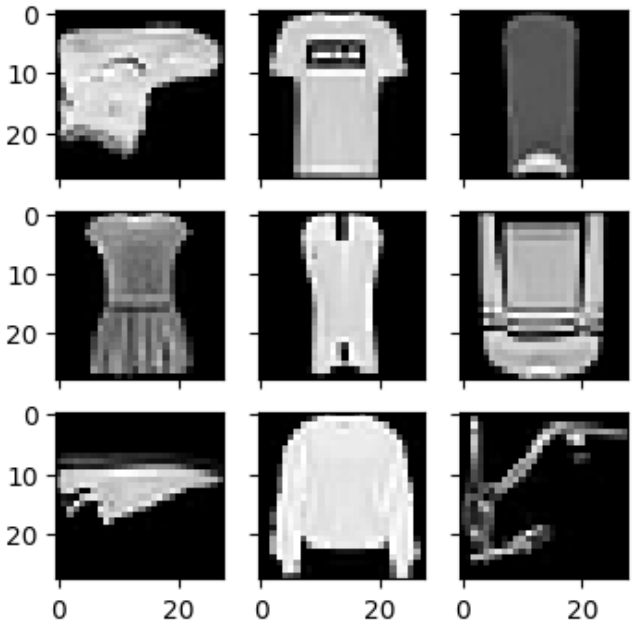
Observation:

- **The same as the random rotations, random shifts is done to avoid overfitting and increase the robustness of the model. It is done to introduce new images but in different positions, this can improve the accuracy of the model especially if done with irregular images where the object is not in the center of the image.**

Random Flips

In [ ]:

```
trainX, trainY, testX, testY = preprocess_dataset(fashion_mnist)
trainX, testX = normalize_pixels(trainX, testX)
datagen = ImageDataGenerator(horizontal_flip=True, vertical_flip=True)
for X_batch, y_batch in datagen.flow(trainX, trainY, batch_size=9, shuffle=False):
    fig, ax = plt.subplots(3, 3, sharex=True, sharey=True, figsize=(4,4))
    for i in range(3):
        for j in range(3):
            ax[i][j].imshow(X_batch[i*3+j].reshape(28,28), cmap=plt.get_cmap("gray"))
    plt.show()
    break
```



Observations:

- **This image augmentation method is done for the model to be invariant to mirroring or flipping of the image. Features can easily be generalized if the images all have the same feature, so introducing flips will let you easily generate new images for the model to train to. This will also help the model predict a wide range of images subjected to it.**

Saving Augmented Images to Disk

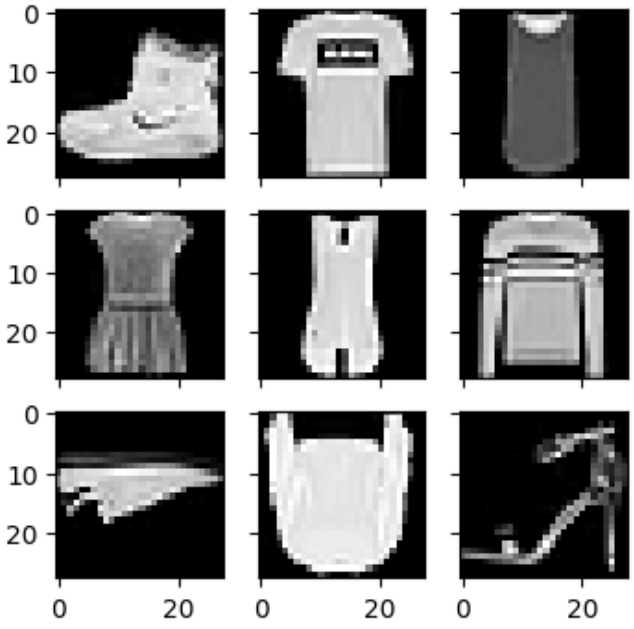
In [ ]:

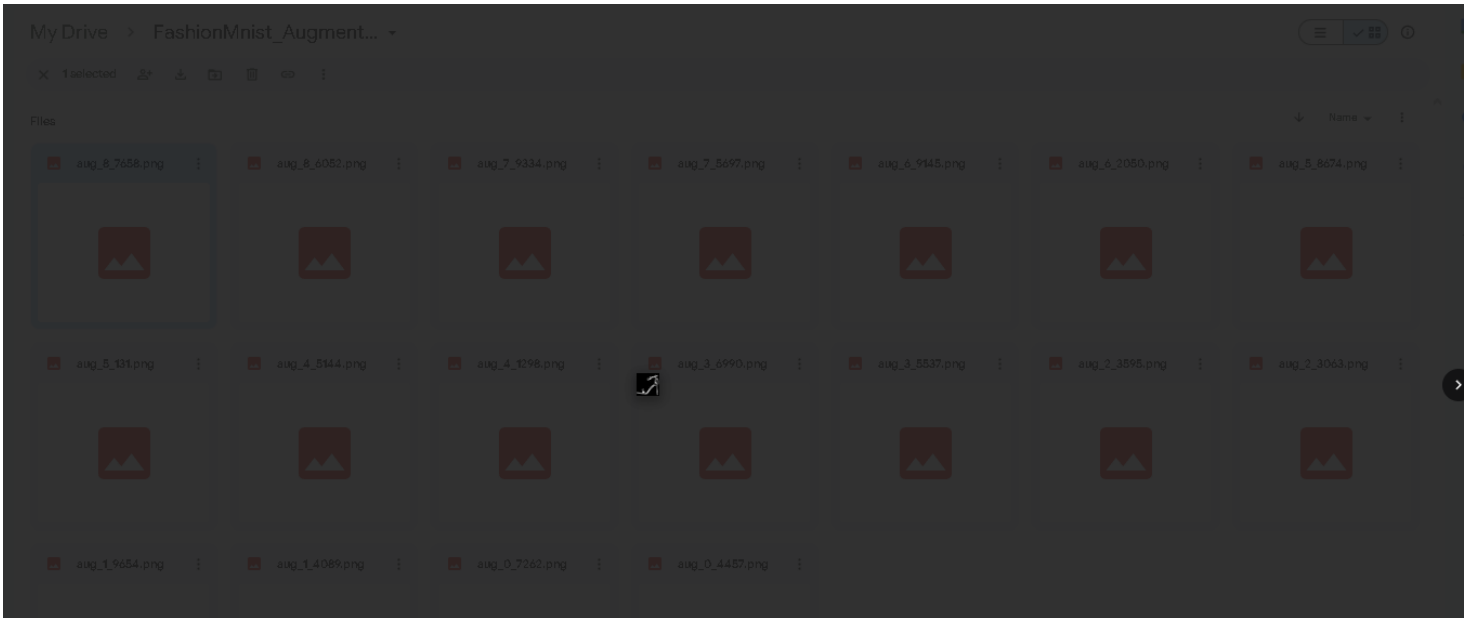
```
from google.colab import drive
drive.mount('/content/drive')
```

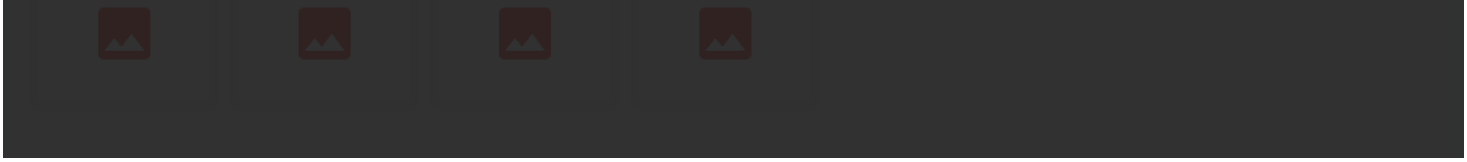
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("
/content/drive", force_remount=True).
```

In [ ]:

```
trainX, trainY, testX, testY = preprocess_dataset(fashion_mnist)
trainX, testX = normalize_pixels(trainX, testX)
datagen = ImageDataGenerator(horizontal_flip=True, vertical_flip=True)

for X_batch, y_batch in datagen.flow(trainX, trainY, batch_size=9, shuffle=False,
                                     save_to_dir='/content/drive/MyDrive/FashionMnist_Augm
ented/', save_prefix='aug', save_format='png'):
    # create a grid of 3x3 images
    fig, ax = plt.subplots(3, 3, sharex=True, sharey=True, figsize=(4,4))
    for i in range(3):
        for j in range(3):
            ax[i][j].imshow(X_batch[i*3+j].reshape(28,28), cmap=plt.get_cmap("gray"))
    # show the plot
    plt.show()
    break
```







Observation:

- **We can see above that we have done a random flip to our image and saved it to our google drive using the datagen.flow() method. The prefix is aug and the format is png. To prove that this code works, I showed an image of my google drive containing the saved augmented images. You can observe that there are 18 images and that is because I ran the code twice. The batch size indicates the number of images that will be augmented and saved.**

Developing Test Harness

- **Developing a test harness is important because it contains all that you need for compiling and testing a model [5]. This can help you save time when running single blocks of codes, instead you run the test harness one time and it does all the work for you. To develop a test harness, it is important for us to know what are the essential tests of a convolutional neural network.**

In [ ]:

```
#Functions to be used in test harness
def evaluate_model(dataX, dataY, n_folds=5):
    scores, histories = list(), list()
    kfold = KFold(n_folds, shuffle=True, random_state=1)
    for train_ix, test_ix in kfold.split(dataX):
        model = define_model()
        trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], d
ataY[test_ix]
        history = model.fit(trainX, trainY, epochs=10, batch_size=1000, validation_data=(t
estX, testY), verbose=0)
        _, acc = model.evaluate(testX, testY, verbose=0)
        print('> %.3f' % (acc * 100.0))
        scores.append(acc)
        histories.append(history)
    return scores, histories

def summarize_diagnostics(histories):
 for i in range(len(histories)):

  plt.subplot(2, 1, 1)
  plt.title('Cross Entropy Loss')
  plt.plot(histories[i].history['loss'], color='blue', label='train')
  plt.plot(histories[i].history['val_loss'], color='orange', label='test')
  plt.subplot(2, 1, 2)
  plt.title('Classification Accuracy')
  plt.plot(histories[i].history['accuracy'], color='blue', label='train')
  plt.plot(histories[i].history['val_accuracy'], color='orange', label='test')
 plt.show()

def summarize_performance(scores):
 print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100, std(scores)*100, len(scor
es)))
 plt.boxplot(scores)
 plt.show()

def plot_roc(y_test, y_pred, model_name):
 fpr, tpr, thr = roc_curve(y_test, y_pred)
 fig, ax = plt.subplots(figsize=(8, 8))
 ax.plot(fpr, tpr, 'k-')
 ax.plot([0, 1], [0, 1], 'k--', linewidth=.5) # roc curve for random model
```

```
 ax.grid(True)
 ax.set(title='ROC Curve for {} on Fashion MNIST'.format(model_name),
 xlim=[-0.01, 1.01], ylim=[-0.01, 1.01])
def display_matrix(matrix):
  names=[0,1,2,3,4,5,6,7,8,9]
  fig, ax = plt.subplots()
  tick_marks = np.arange(len(names))
  plt.xticks(tick_marks,names)
  plt.yticks(tick_marks,names)
  sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="BuPu" ,fmt='g')
  ax.xaxis.set_label_position("top")
  plt.tight_layout()
  plt.title('Confusion matrix', y=1.1)
  plt.ylabel('Actual label')
  plt.xlabel('Predicted label')
```

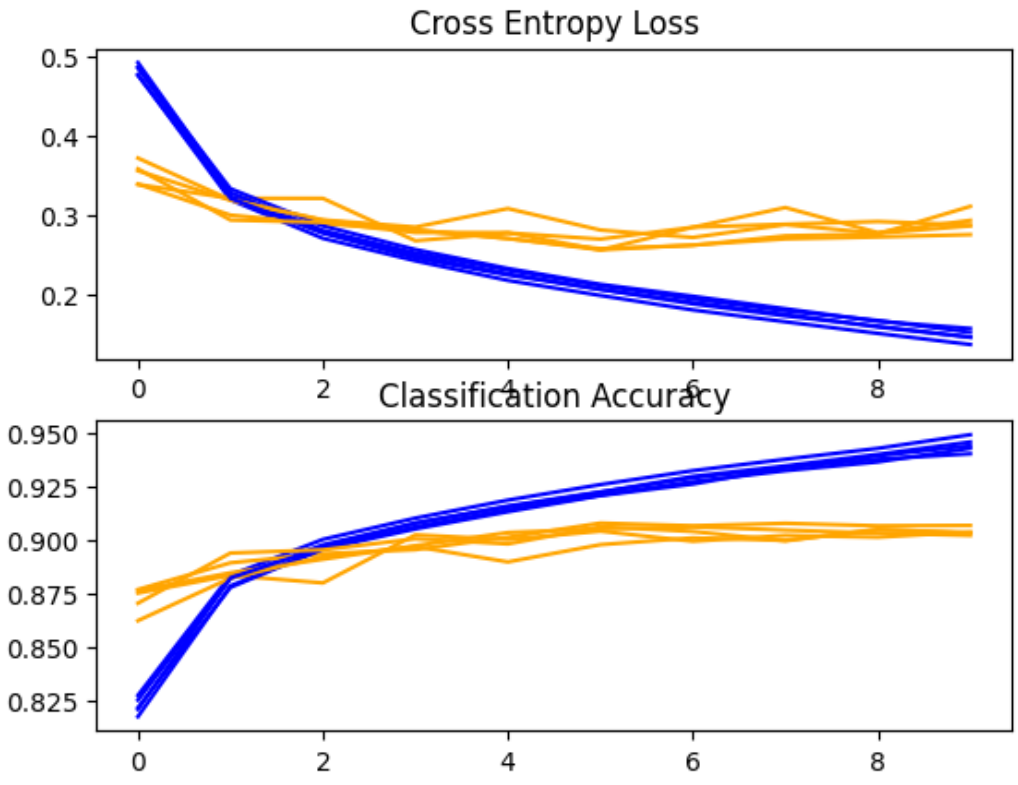
In [ ]:

```
def test_harness():
  trainX, trainY, testX, testY = preprocess_dataset(fashion_mnist)
  trainX, testX = normalize_pixels(trainX, testX)
  scores, histories = evaluate_model(trainX, trainY)
  summarize_diagnostics(histories)
  summarize_performance(scores)
```

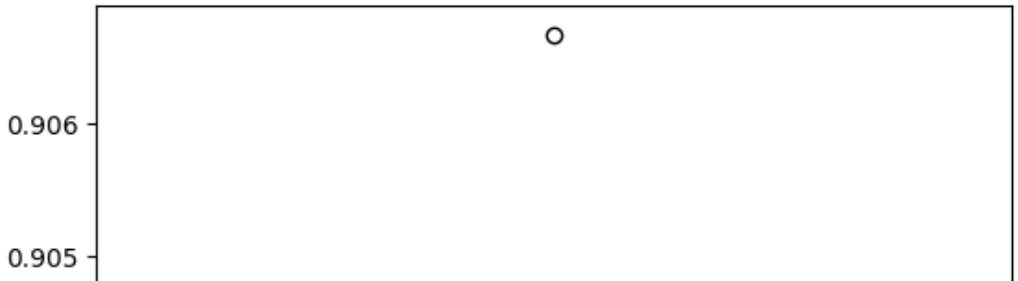
In [ ]:

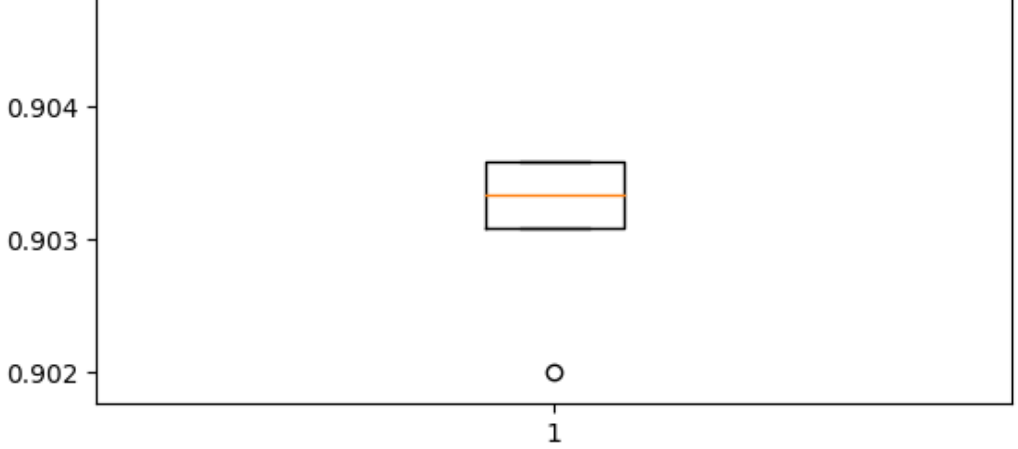
```
test_harness()
```

```
> 90.358
> 90.667
> 90.200
> 90.333
> 90.308
```




```
Accuracy: mean=90.373 std=0.156, n=5
```

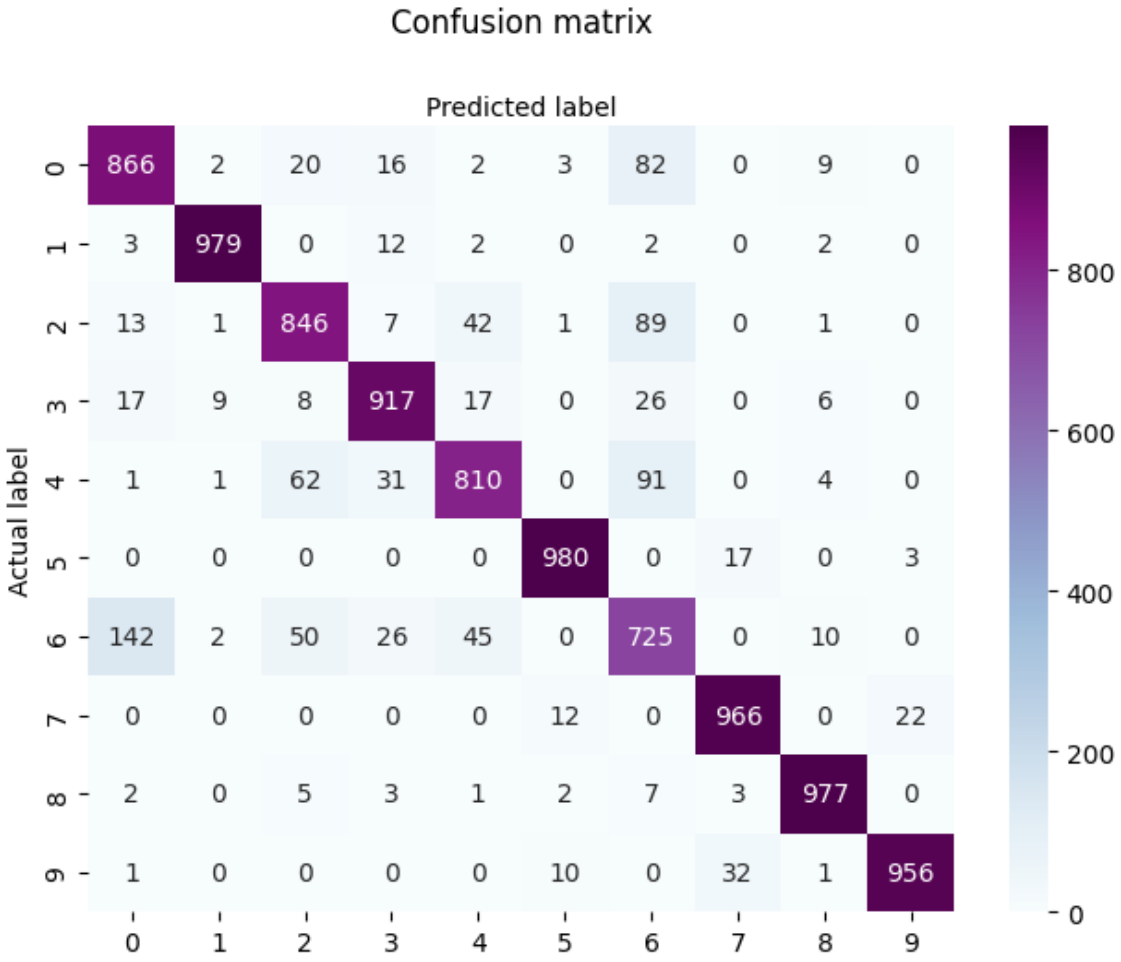




In [ ]:

```
y_pred_prob_nn_2 = np.argmax(model.predict(testX), axis = 1)
matrix = confusion_matrix(testY, y_pred_prob_nn_2)
display_matrix(matrix)
```

```
313/313 [==============================] - 1s 3ms/step
```



In [ ]:

```
y_pred_prob_nn_2 = np.argmax(model.predict(testX), axis = 1)
print(classification_report(testY, y_pred_prob_nn_2))
```

```
313/313 [==============================] - 1s 3ms/step
              precision    recall  f1-score   support

           0       0.83      0.87      0.85      1000
           1       0.98      0.98      0.98      1000
           2       0.85      0.85      0.85      1000
           3       0.91      0.92      0.91      1000
           4       0.88      0.81      0.84      1000
           5       0.97      0.98      0.98      1000
           6       0.71      0.72      0.72      1000
           7       0.95      0.97      0.96      1000
```

```
           8       0.97      0.98      0.97      1000
           9       0.97      0.96      0.97      1000

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

Observation:

- **As you can see in this test harness, I used many tests that can be useful in evaluating the final model. I used kFold cross validation, loss and accuracy plotting, confusion matrix and lastly the classification report.**
- **Overall, you can see the performance of the baseline model here, and we can observe that it is a good model with an average accuracy of 90%.**

Optimizing Baseline Model

- **Here in the optimized model, I decided to add more layers to the model. I added 3 Conv2D layer which are 32, 64 and 128 respectively.**
- **I also added a new Conv2D layer which is the layer with 128 filters. Then I changed the value for the hidden nodes of the hidden layer to 128.**
- **I also added more epochs and changed the batch size to have a faster training of the model.**

In [ ]:

```
def define_model():
  model = Sequential()
  model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',padding=
'same', input_shape=(28, 28, 1)))
  model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',padding=
'same'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform',padding=
'same'))
  model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform',padding=
'same'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform',padding
='same'))
  model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform',padding
='same'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Flatten())
  model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
  model.add(Dense(10, activation='softmax'))

  opt = SGD(learning_rate=0.001, momentum=0.9)
  model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
  return model
```

In [ ]:

```
new_model = define_model()
new_model.summary()
fashion_history2 = new_model.fit(trainX, trainY, validation_split = 0.2, epochs=30, batch_
size=1000, verbose=0)
```

```
Model: "sequential_36"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_79 (Conv2D)          (None, 28, 28, 32)        320

 conv2d_80 (Conv2D)          (None, 28, 28, 32)        9248
```

```
 max_pooling2d_76 (MaxPooli   (None, 14, 14, 32)        0
 ng2D)

 conv2d_81 (Conv2D)          (None, 14, 14, 64)        18496

 conv2d_82 (Conv2D)          (None, 14, 14, 64)        36928

 max_pooling2d_77 (MaxPooli   (None, 7, 7, 64)          0
 ng2D)

 conv2d_83 (Conv2D)          (None, 7, 7, 128)         73856

 conv2d_84 (Conv2D)          (None, 7, 7, 128)         147584

 max_pooling2d_78 (MaxPooli   (None, 3, 3, 128)         0
 ng2D)

 flatten_36 (Flatten)        (None, 1152)              0

 dense_72 (Dense)            (None, 128)               147584

 dense_73 (Dense)            (None, 10)                1290

=================================================================
Total params: 435306 (1.66 MB)
Trainable params: 435306 (1.66 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [ ]:

```
model.save("")
```

In [ ]:

```
_, acc = new_model.evaluate(testX, testY, verbose=0)
 print('> %.3f' % (acc * 100.0))
```

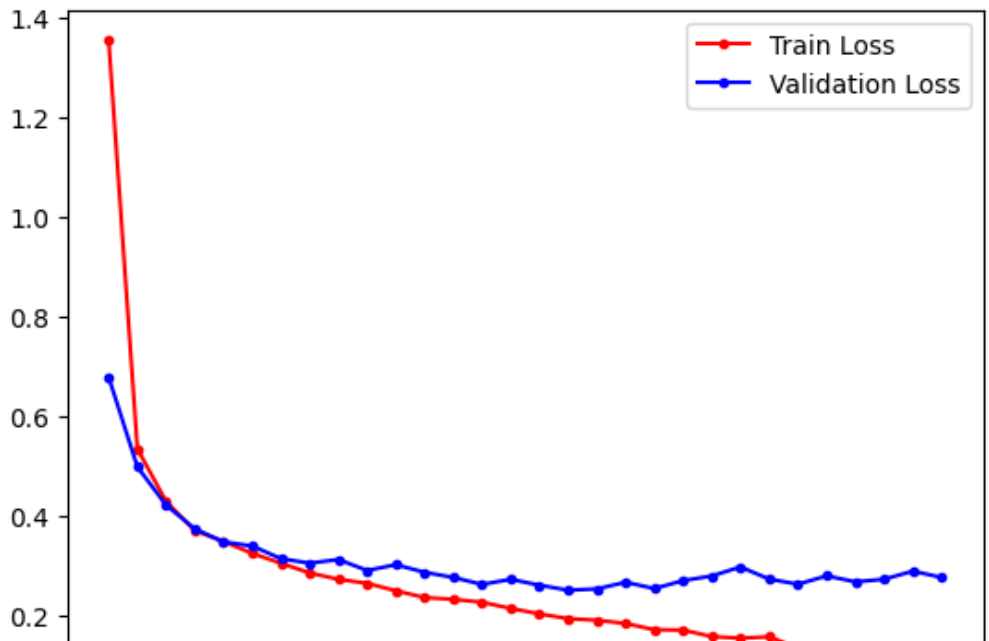
```
> 90.280
```

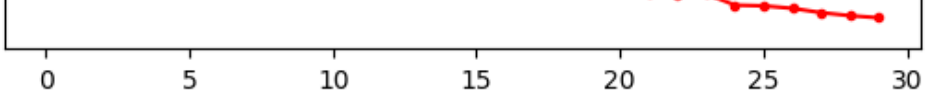
In [ ]:

```
fig, ax = plt.subplots()
ax.plot(fashion_history2.history["loss"],'r', marker='.', label="Train Loss")
ax.plot(fashion_history2.history["val_loss"],'b', marker='.', label="Validation Loss")
ax.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x78da10b34df0>
```



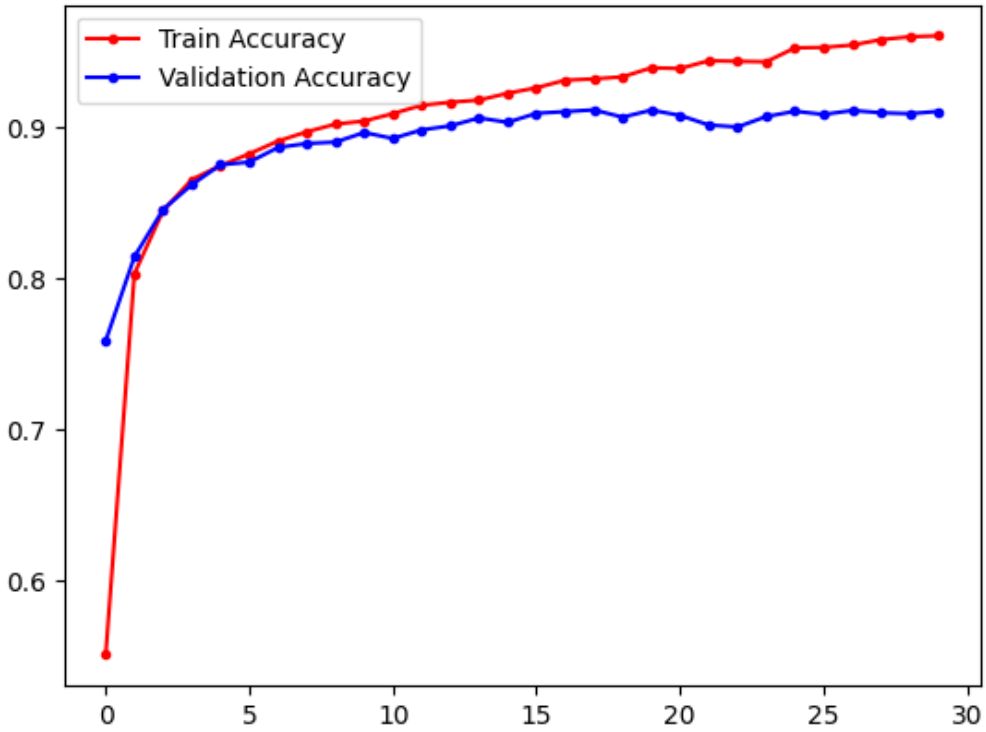


In [ ]:

```
fig, ax = plt.subplots()
ax.plot(fashion_history2.history["accuracy"],'r', marker='.', label="Train Accuracy")
ax.plot(fashion_history2.history["val_accuracy"],'b', marker='.', label="Validation Accura
cy")
ax.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x78da10acbd90>
```



Observation:

- **Here, we can see that the loss and accuracy of the model is stagnant around 15th epoch. This means that for now, 15 epochs is the best number of epoch for our model. The accuracy and loss for this model is the same for the baseline model but we found out that having more layers does not mean that it is good for this dataset.**

Finalizing the model

In [ ]:

```
def define_model():
  model = Sequential()
  model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',padding=
'same', input_shape=(28, 28, 1)))
  model.add(MaxPooling2D((2, 2)))
  model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform',padding=
'same'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform',padding
='same'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Flatten())
  model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
  model.add(Dense(10, activation='softmax'))
```

```
  opt = SGD(learning_rate=0.1, momentum=0.9)
  model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
  return model
```

In [ ]:

```
new_model = define_model()
new_model.summary()
fashion_history2 = new_model.fit(trainX, trainY, validation_split = 0.2, epochs=30, batch_
size=1000, verbose=0)
```

```
Model: "sequential_39"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_91 (Conv2D)          (None, 28, 28, 32)        320

 max_pooling2d_85 (MaxPooli  (None, 14, 14, 32)        0
 ng2D)

 conv2d_92 (Conv2D)          (None, 14, 14, 64)        18496

 max_pooling2d_86 (MaxPooli  (None, 7, 7, 64)          0
 ng2D)

 conv2d_93 (Conv2D)          (None, 7, 7, 128)         73856

 max_pooling2d_87 (MaxPooli  (None, 3, 3, 128)         0
 ng2D)

 flatten_39 (Flatten)        (None, 1152)              0

 dense_78 (Dense)            (None, 128)               147584

 dense_79 (Dense)            (None, 10)                1290

=================================================================
Total params: 241546 (943.54 KB)
Trainable params: 241546 (943.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [ ]:

```
new_model.save("/content/drive/MyDrive/FashionMnist_Augmented/final_model.h5")
```

In [ ]:

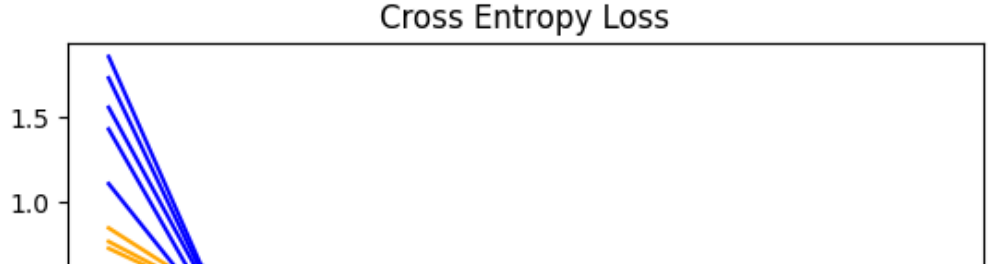
```
_, acc = new_model.evaluate(testX, testY, verbose=0)
 print('> %.3f' % (acc * 100.0))
```

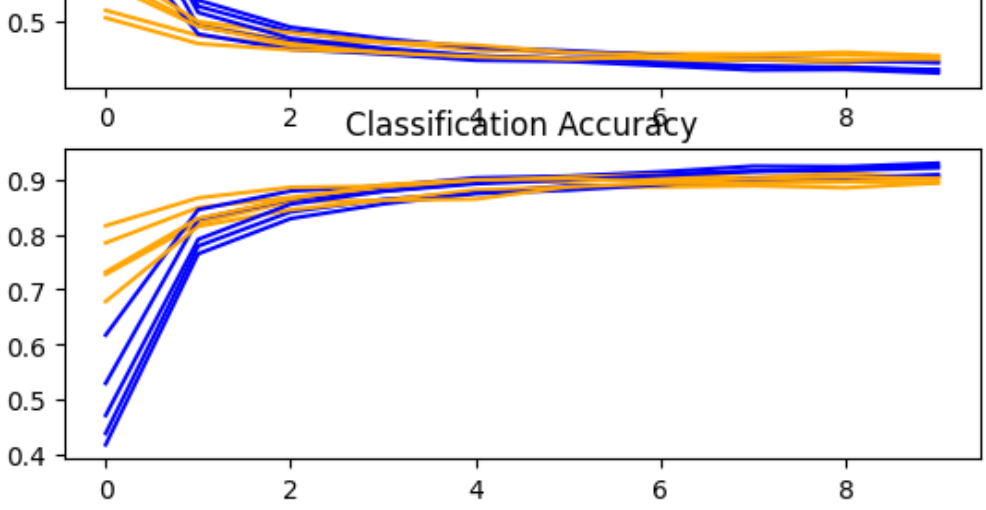
```
> 90.420
```

In [ ]:

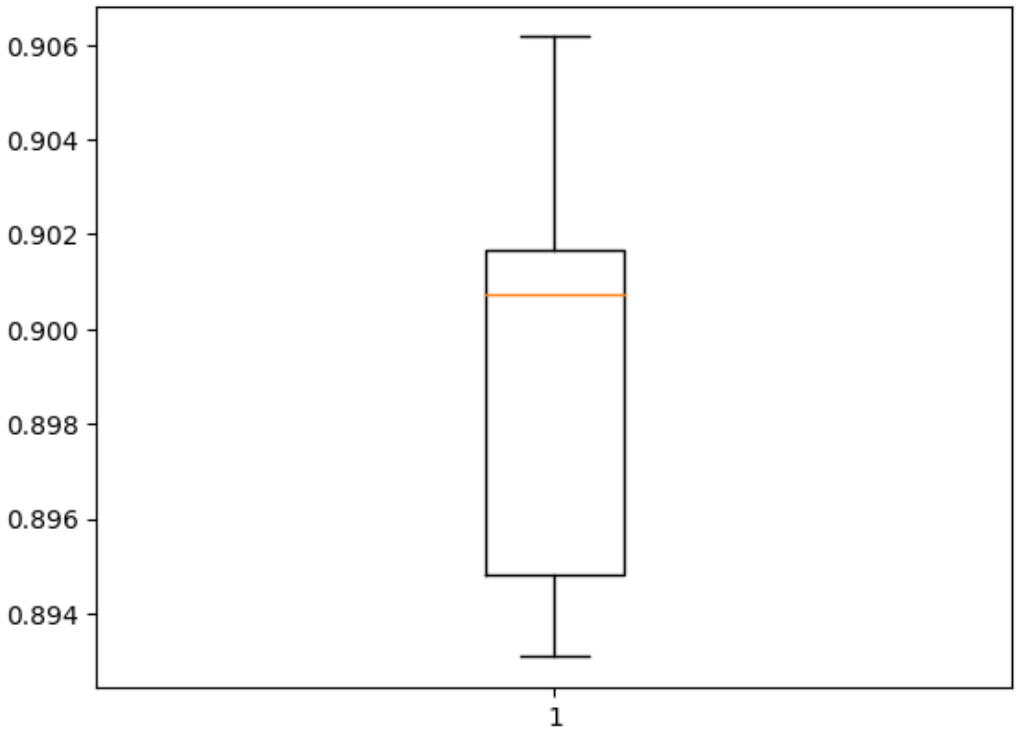
```
test_harness()
```

```
> 89.308
> 90.617
> 90.075
> 89.483
> 90.167
```






```
Accuracy: mean=89.930 std=0.476, n=5
```



Observation:

- **We can see above the final model that I made, it is a combination of the baseline and the optimized model. It has an uplifted accuracy compared to the baseline and the optimized model. Additionally, we can observe from the kfold test that this model is not overfitted and the gap between training and validation is pretty small unlike for the baseline model and the optimized model.**
- **This finalized model was saved to the file drive in an hdf5 format. The entire model was saved including the weights. This is done so that we can easily access this model in the future without running the fit() again.**

Testing the final model using real images

In [ ]:

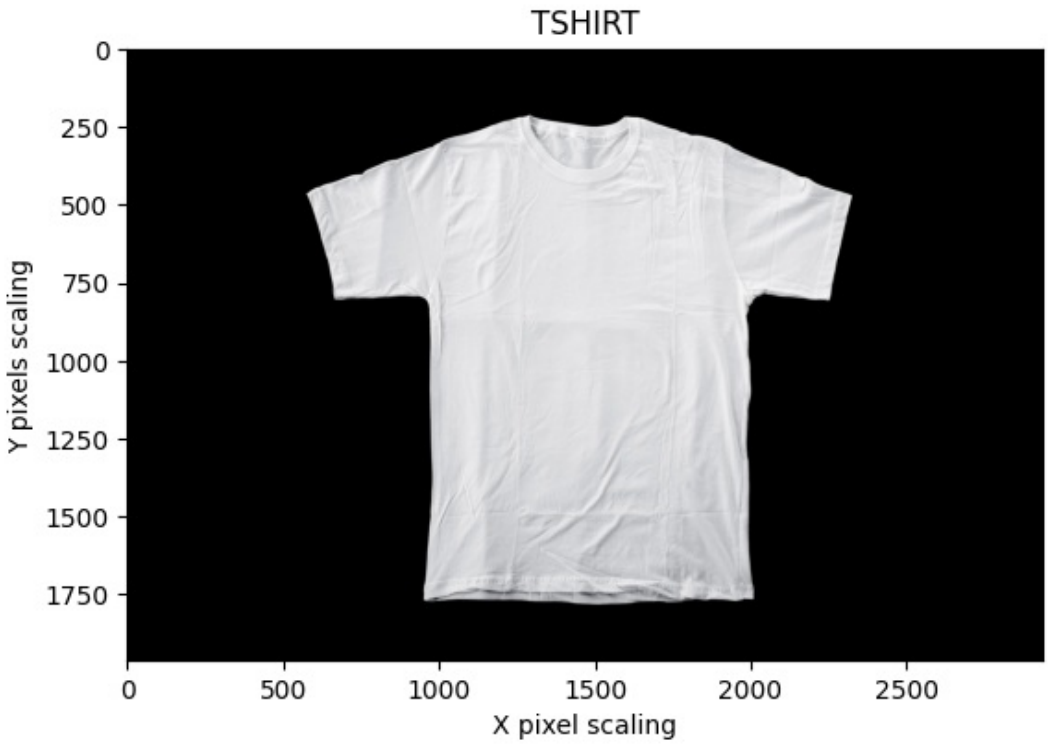
```
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
import tensorflow as tf
from keras.models import load_model

labels = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal",
```

```
          "Shirt", "Sneaker", "Bag", "Ankle boot"]

# load and prepare the image
def load_image(filename):

 img = load_img(filename, target_size=(28,28))
 img = img_to_array(img)
 img = img[:,:,0]
 img = img.reshape(1,28, 28,1)
 img = img.astype('float32')
 img = img / 255.0
 return img

def final_model_test(imgpath):
 img = load_image(imgpath)
 model = tf.keras.models.load_model("/content/drive/MyDrive/FashionMnist_Augmented/final_m
odel.h5")
 result = np.argmax(model.predict(img))
 print("\nThe image is:", labels[result])

def show_img(imgpath, name):
 plt.title(name)
 plt.xlabel("X pixel scaling")
 plt.ylabel("Y pixels scaling")

 image = mpimg.imread(imgpath)
 plt.imshow(image)
 plt.show()
```

In [ ]:

```
TSHIRT = '/content/drive/MyDrive/FashionMnist_Augmented/TSHIRT_TEST.jpg'
final_model_test(TSHIRT)
show_img(TSHIRT, "TSHIRT")
```

```
1/1 [==============================] - 0s 80ms/step

The image is: T-shirt/top
```

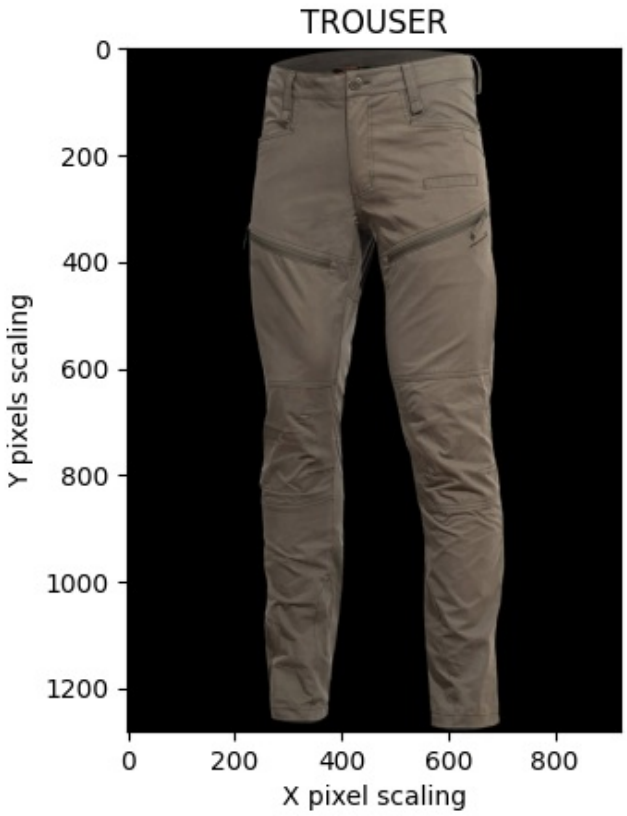


In [ ]:

```
TROUSER = '/content/drive/MyDrive/FashionMnist_Augmented/TROUSER_TEST.jpg'
final_model_test(TROUSER)
show_img(TROUSER, "TROUSER")
```

```
1/1 [==============================] - 0s 125ms/step
```

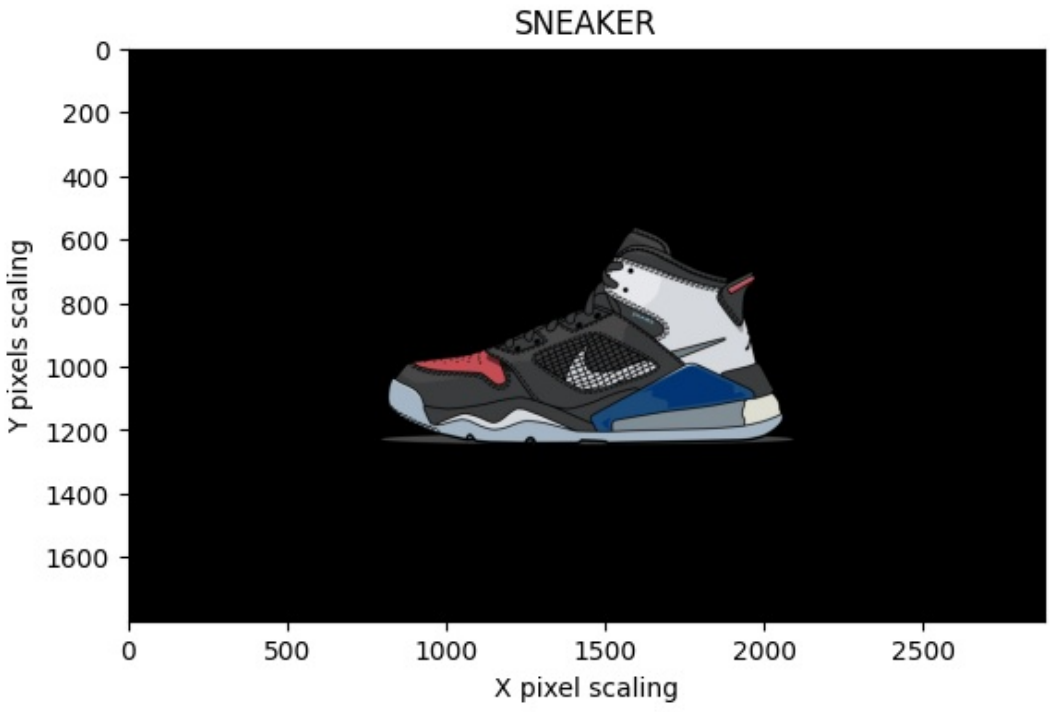
The image is: Trouser



In [ ]:

```
SNEAKER = '/content/drive/MyDrive/FashionMnist_Augmented/SNEAKER_TEST.jpg'
final_model_test(SNEAKER)
show_img(SNEAKER, "SNEAKER")
```

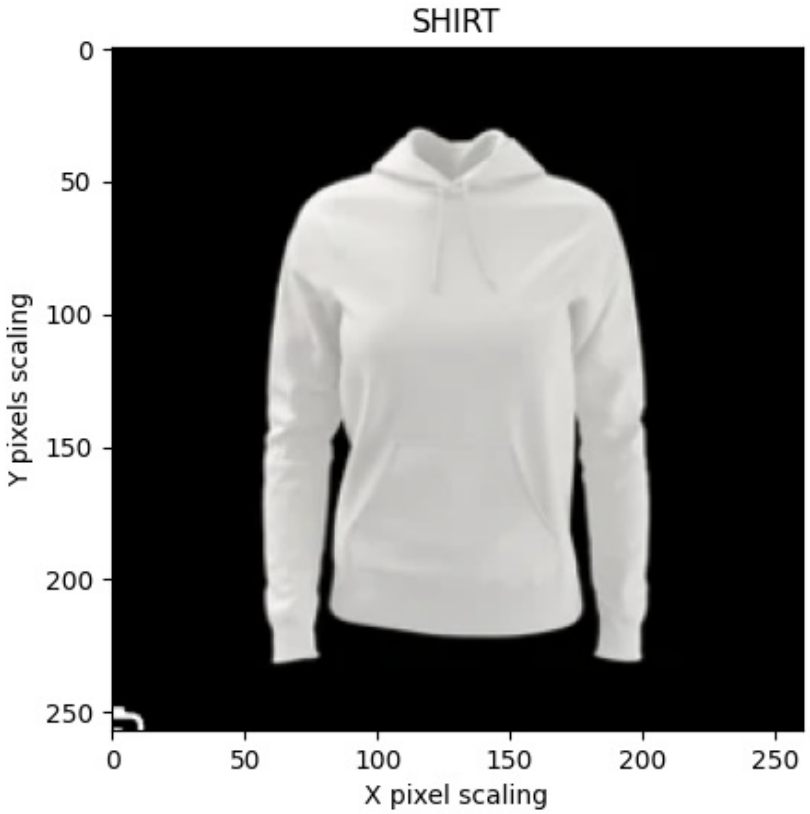
```
1/1 [==============================] - 0s 258ms/step

The image is: Sneaker
```



In [ ]:

```
SHIRT = '/content/drive/MyDrive/FashionMnist_Augmented/SHIRT_TEST.png'
final_model_test(SHIRT)
show_img(SHIRT, "SHIRT")
```

```
1/1 [==============================] - 0s 128ms/step

The image is: Shirt
```



Observation:

- **Here, we can observe that the model has made correct predictions against the real life pictures of the fashion items features of this dataset. The images that I used was from the internet and I saved them to the google drive so I can easily access them. I loaded the final model and used it to predict different images.**
- **There are some examples above like the Tshirt, Trouser, Sneakers and Shirt. These are some of the images present in the dataset and it was correctly predicted by the final model.**
- **I can say that this model is well-fitted and it is a good model since it has an accuracy of almost 91%.**

SUMMARY/CONCLUSION

- **In this activity, we were tasked to create a convolutional neural network and augment images that will be used for the training. The first thing I did was to get a dataset from the keras datasets which is the fashion mnist dataset. This dataset is composed of 10 labels and 60,000 parameters.**
- **The next thing I did was to preprocess the dataset by reshaping it to 28 x 28 and setting the color channel to 1 which will result to grayscale. Then I normalized the pixels by dividing them to 5. After that I created the baseline model by utilizing new layers like Conv2d and Maxpooling. These layers help in applying filters to the image and downscaling them to avoid overfitting.**
- **After training the baseline model, I found out that the baseline model has an accuracy of 0.9 or 90% which is pretty good. I did some evaluation which resulted to a good result.**
- **Then I did some image augmentation like feature standardization, zca whitening, random flips, random rotations, and random shifts. This is done to prevent overfitting and make the model more robust or in other words local translation invariant. After that, I created a test harness in which I utilized different evaluation tests for a CNN model like loss, accuracy, confusion matrix, and model report.**
- **In the latter part of the activity, I did some model optimization from where I changed the layers of the model and also changed the learning rate and batch size. The optimized model is pretty much the same as the baseline model so when I finalized the model, I further optimized it and achieved a pretty good result.**
- **In finalizing the model, I saved my final model in hdf5 format so that I can easily use it and access it in the future. I can also transfer this model or deploy it if I want to. I also did some testing with real images and found out that this final model is well-fitted since it correctly predicted those real-life images that is outside the dataset.**
- **Overall, this activity is very enlightening and it was also very stressful. It was enlightening in a way that it**

introduced me to CNN or convolutional neural networks, convolutional layers, pooling layers, image augmentation and test harness. It is very stressful because of the amount of time it takes to train and test the model especially since this is done in the google colab. It takes an ample amount of time to train a model, and even more amount of time to kfold cross validate a model. Although I learned that there is an available GPU runtime in the google colab, it was only for a limited amount of time.

- **I will certainly recommend this activity to aspiring data scientist CPE students because it will open their eyes regarding the wide use of Neural Networks. I am very happy that I was exposed to this knowledge and I am looking forward for more activities. Thank you for reading.**

References:

[1]Jason Brownlee, "How Do Convolutional Layers Work in Deep Learning Neural Networks?," Machine Learning Mastery, Apr. 16, 2019. https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/

[2]Team, Keras. "Keras Documentation: Conv2D Layer." Keras.io, keras.io/api/layers/convolution_layers/convolution2d/

[3]Brownlee, Jason. "A Gentle Introduction to Pooling Layers for Convolutional Neural Networks." Machine Learning Mastery, 30 Apr. 2019, machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

[4]Brownlee, Jason. "Image Augmentation for Deep Learning with Keras." Machine Learning Mastery, 24 May 2019, machinelearningmastery.com/image-augmentation-deep-learning-keras/.

[5]"Test Harness Structure." Www.ibm.com, www.ibm.com/docs/en/devops-test-embedded/9.0.0?topic=components-test-harness-structure. Accessed 27 Apr. 2024.

Link to dataset

- **https://www.tensorflow.org/datasets/catalog/fashion_mnist**