PROJECT - DEEP LEARNING PROJECT FASHION MNIST CLASSIFICATION.

STUDENT NAME : CHINTAREDDY VARSHITHA

Title: Deep Learning Project: Fashion MNIST Classification

Abstract:
The Fashion-MNIST dataset is a collection of 70,000 grayscale images of 10 different fashion items. The goal of this project is to build a deep learning model that can classify these images with high accuracy.

Objective:
The objective of this project is to build a deep learning model that can classify Fashion-MNIST images with an accuracy of at least 90%.

Introduction:
The Fashion-MNIST dataset is a popular dataset for benchmarking machine learning algorithms for image classification. It consists of 70,000 grayscale images of 10 different fashion items:

- T-shirt/top
- Trouser
- Pullover
- Dress
- Coat
- Sandal
- Shirt
- Sneaker
- Bag
- Ankle boot.

The images are 28x28 pixels in size and are evenly split between training and test sets.

Methodology :

The following methodology was used to build the deep learning model:

- ❖The Fashion-MNIST dataset was loaded into a Python environment.
- ❖The images were pre-processed by normalising the pixel values to the range [0, 1].
- ❖A convolutional neural network (CNN) model was created. The CNN model consisted of two convolutional layers, each followed by a max pooling layer.
- ❖The CNN model was then followed by two fully connected layers.
- ❖The CNN model was trained on the training set using the Adam optimiser and the categorical cross-entropy loss function.
- ❖The CNN model was evaluated on the test set.

Code :

Step # 1 - Import Libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sbn
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Dense, Flatten
from keras.optimizers import Adam
from keras.callbacks import TensorBoard
from keras.utils import to_categorical
```

Step # 2 - Load Data

In [2]:

```
fashion_train_df = pd.read_csv('../input/fashion-mnist-datasets/fashion-mnist_train.cs
v', sep=',')
fashion_test_df = pd.read_csv('../input/fashion-mnist-datasets/fashion-mnist_test.cs
v', sep=',')
```

In [3]:
```
fashion_train_df.shape   # Shape of the dataset
```

Out[3]:
```
(60000, 785)
```

In [4]:
```
fashion_train_df.columns   # Name of the columns of the DataSet.
```

Out[4]:
```
Index(['label', 'pixel1', 'pixel2', 'pixel3', 'pixel4', 'pixel5', 'pixel6',
       'pixel7', 'pixel8', 'pixel9',
       ...
       'pixel775', 'pixel776', 'pixel777', 'pixel778', 'pixel779', 'pixel780',
       'pixel781', 'pixel782', 'pixel783', 'pixel784'],
      dtype='object', length=785)
```

In [5]:
```
print(set(fashion_train_df['label']))
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [6]:
```
print([fashion_train_df.drop(labels='label', axis=1).min(axis=1).min(),
      fashion_train_df.drop(labels='label', axis=1).max(axis=1).max()])
```

```
[0, 255]
```

In [7]:
```
fashion_train_df.head()
```

Out[7]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pix
0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
1	9	0	0	0	0	0	0	0	0	0	0	0	0	0
2	6	0	0	0	0	0	0	0	5	0	0	0	105	92
3	0	0	0	0	1	2	0	0	0	0	0	114	183	11
4	3	0	0	0	0	0	0	0	0	0	0	0	0	46

In [8]:
```
fashion_test_df.shape
```

Out[8]:
```
(10000, 785)
```

In [9]:

```
fashion_test_df.head()
```

Out[9]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12
0	0	0	0	0	0	0	0	0	9	8	0	0	34
1	1	0	0	0	0	0	0	0	0	0	0	0	209
2	2	0	0	0	0	0	0	14	53	99	17	0	0
3	2	0	0	0	0	0	0	0	0	0	161	212	138
4	3	0	0	0	0	0	0	0	0	0	0	37	0

Step # 3 - Visualisation

In [10]:

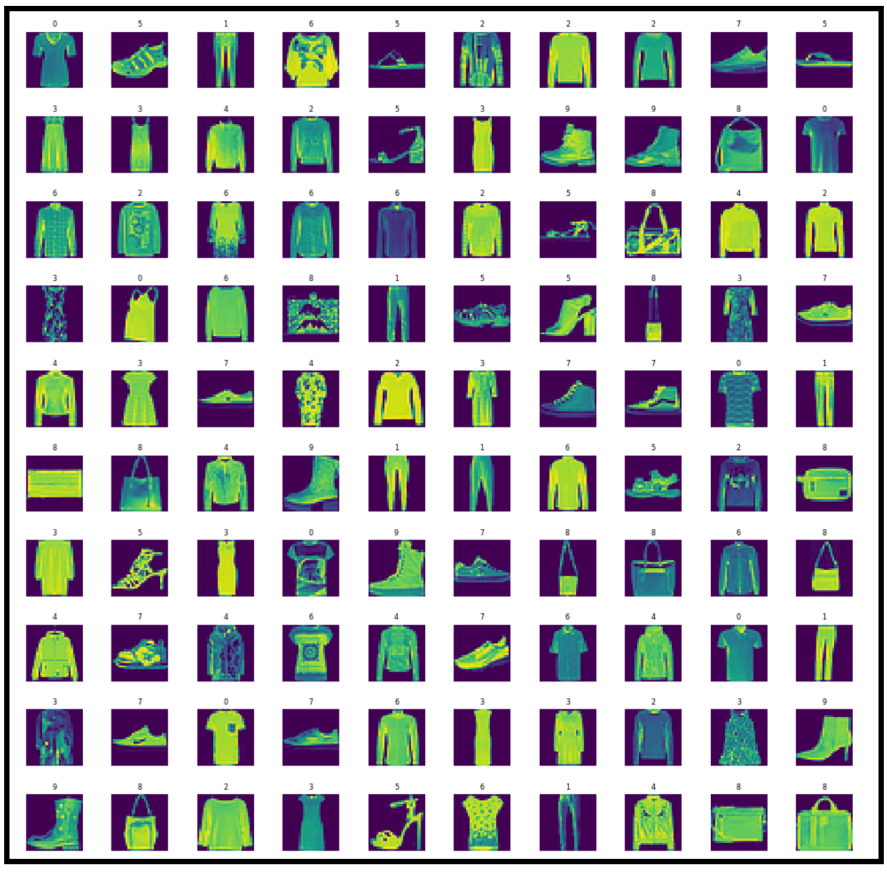
```
# Convert the dataframe ti numpy array
training = np.asarray(fashion_train_df, dtype='float32')

# Lets show multiple images in a 15x15 grid
height = 10
width = 10

fig, axes = plt.subplots(nrows=width, ncols=height, figsize=(17,17))
axes = axes.ravel()  # this flattens the 15x15 matrix into 225
n_train = len(training)

for i in range(0, height*width):
    index = np.random.randint(0, n_train)
    axes[i].imshow(training[index, 1:].reshape(28,28))
    axes[i].set_title(int(training[index, 0]), fontsize=8)
    axes[i].axis('off')

plt.subplots_adjust(hspace=0.5)
```



Step # 4 - Preprocess Data

In [11]:
```
# convert to numpy arrays and reshape
training = np.asarray(fashion_train_df, dtype='float32')
X_train = training[:, 1:].reshape([-1,28,28,1])
X_train = X_train/255   # Normalizing the data
y_train = training[:, 0]

testing = np.asarray(fashion_test_df, dtype='float32')
X_test = testing[:, 1:].reshape([-1,28,28,1])
X_test = X_test/255    # Normalizing the data
y_test = testing[:, 0]
```

In [12]:
```
# Split the training set into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=1234
5)    # TODO : change the random state to 5
```

In [13]:
```
# Lets check the shape of all three datasets
print(X_train.shape, X_val.shape, X_test.shape)
print(y_train.shape, y_val.shape, y_test.shape)
```

```
(48000, 28, 28, 1) (12000, 28, 28, 1) (10000, 28, 28, 1)
(48000,) (12000,) (10000,)
```

Step # 5 - Create and Train the Model

Create the model

In [14]:
```
cnn_model = Sequential()
cnn_model.add(Conv2D(filters=64, kernel_size=(3,3), input_shape=(28,28,1), activation='relu'))
cnn_model.add(MaxPooling2D(pool_size = (2,2)))
cnn_model.add(Dropout(rate=0.3))
cnn_model.add(Flatten())
cnn_model.add(Dense(units=32, activation='relu'))
cnn_model.add(Dense(units=10, activation='sigmoid'))
```

compile the model

In [15]:
```
cnn_model.compile(optimizer=Adam(lr=0.001), loss='sparse_categorical_crossentropy', metrics=['accura
cy'])
cnn_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 64)        640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 13, 13, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 10816)             0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                346144
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330
=================================================================
Total params: 347,114
Trainable params: 347,114
Non-trainable params: 0
_________________________________________________________________
```

Train the model

In [16]:
```
cnn_model.fit(x=X_train, y=y_train, batch_size=512, epochs=50, validation_data=(X_val, y_val))
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/50
48000/48000 [==============================] - 5s 108us/step - loss: 0.9155 - acc: 0.6463 - val_l
oss: 0.4851 - val_acc: 0.8273
Epoch 2/50
48000/48000 [==============================] - 1s 19us/step - loss: 0.4501 - acc: 0.8417 - val_lo
ss: 0.4318 - val_acc: 0.8476
Epoch 3/50
48000/48000 [==============================] - 1s 19us/step - loss: 0.4012 - acc: 0.8586 - val_lo
ss: 0.3718 - val_acc: 0.8723
Epoch 4/50
48000/48000 [==============================] - 1s 19us/step - loss: 0.3663 - acc: 0.8720 - val_lo
ss: 0.3409 - val_acc: 0.8823
Epoch 5/50
48000/48000 [==============================] - 1s 20us/step - loss: 0.3465 - acc: 0.8787 - val_lo
ss: 0.3285 - val_acc: 0.8872
Epoch 6/50
48000/48000 [==============================] - 1s 19us/step - loss: 0.3311 - acc: 0.8838 - val_lo
ss: 0.3286 - val_acc: 0.8863
Epoch 7/50
48000/48000 [==============================] - 1s 20us/step - loss: 0.3178 - acc: 0.8874 - val_lo
ss: 0.3054 - val_acc: 0.8928
Epoch 8/50
48000/48000 [==============================] - 1s 19us/step - loss: 0.3032 - acc: 0.8938 - val_lo
ss: 0.3039 - val_acc: 0.8913
Epoch 9/50
48000/48000 [==============================] - 1s 19us/step - loss: 0.2965 - acc: 0.8956 - val_lo
ss: 0.2970 - val_acc: 0.8967
Epoch 10/50
48000/48000 [==============================] - 1s 19us/step - loss: 0.2854 - acc: 0.8985 - val_lo
ss: 0.2873 - val_acc: 0.8996
Epoch 11/50
48000/48000 [==============================] - 1s 19us/step - loss: 0.2783 - acc: 0.9025 - val_lo
ss: 0.2889 - val_acc: 0.8967
Epoch 12/50
48000/48000 [==============================] - 1s 19us/step - loss: 0.2727 - acc: 0.9029 - val_lo
ss: 0.2848 - val_acc: 0.8992
```

Step # 6 - Evaluate the Model

In [17]:
```
eval_result = cnn_model.evaluate(X_test, y_test)
print("Accuracy : {:.3f}".format(eval_result[1]))
```

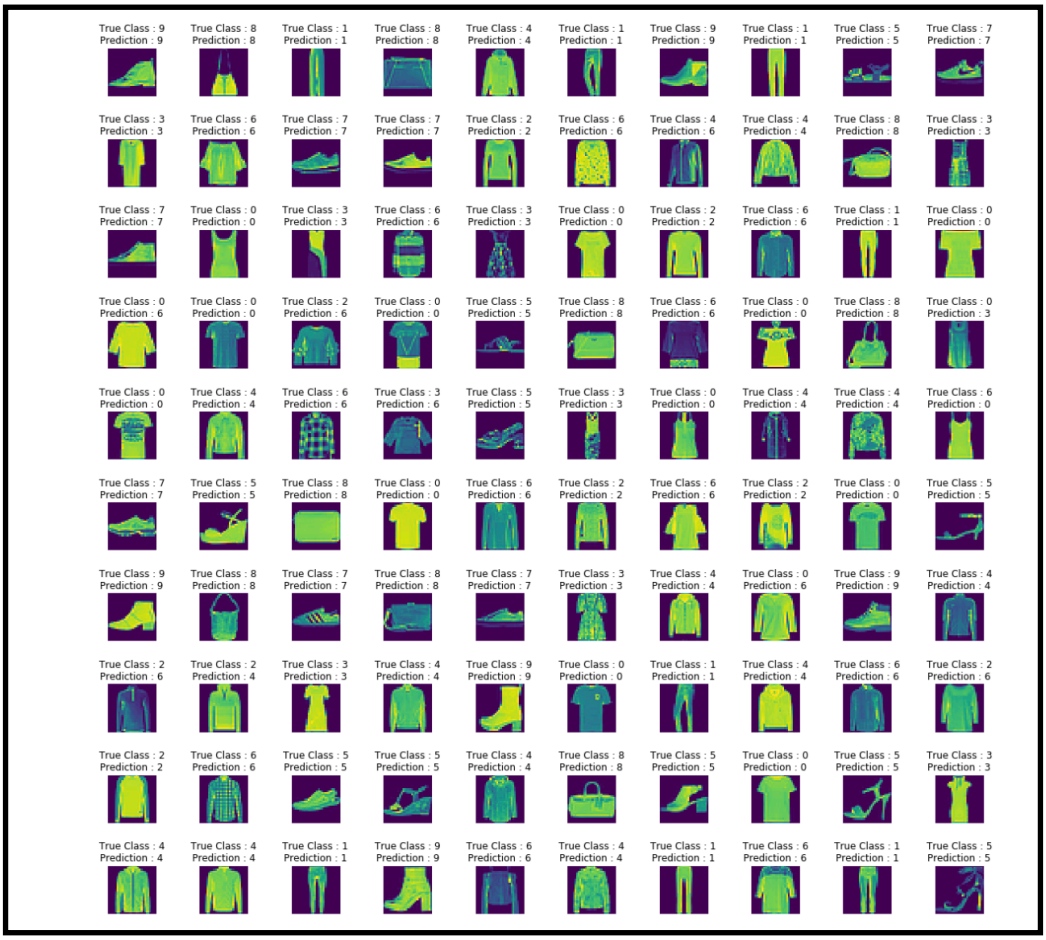
```
10000/10000 [==============================] - 1s 55us/step
Accuracy : 0.918
```

Visualize the model's predictions

In [18]:
```
y_pred = cnn_model.predict_classes(x=X_test)
```

In [19]:
```
height = 10
width = 10

fig, axes = plt.subplots(nrows=width, ncols=height, figsize=(20,20))
axes = axes.ravel()
for i in range(0, height*width):
    index = np.random.randint(len(y_pred))
    axes[i].imshow(X_test[index].reshape((28,28)))
    axes[i].set_title("True Class : {:0.0f}\nPrediction : {:d}".format(y_test[index],y_pred[index]))
    axes[i].axis('off')
plt.subplots_adjust(hspace=0.9, wspace=0.5)
```



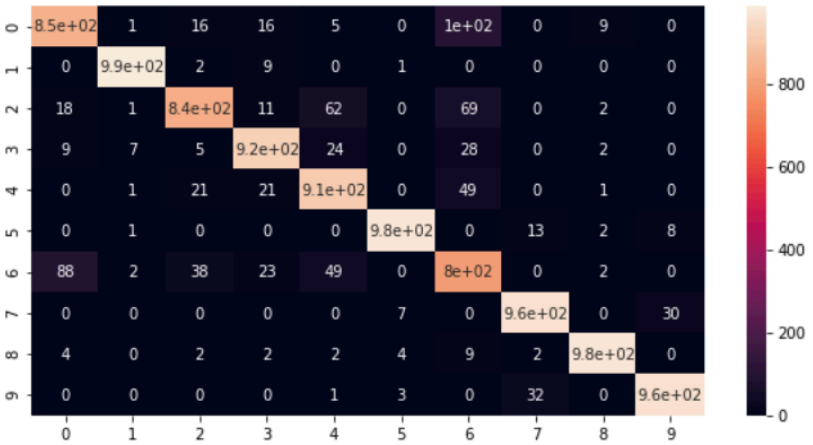
Plot Confusin Matrix

In [20]:

```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10,5))
sbn.heatmap(cm, annot=True)
```

Out[20]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7feb11cd26d8>
```



Classification Report

In [21]:

```
num_classes = 10
class_names = ["class {}".format(i) for i in range(num_classes)]
cr = classification_report(y_test, y_pred, target_names=class_names)
print(cr)
```

```
              precision    recall  f1-score   support

     class 0       0.88      0.85      0.86      1000
     class 1       0.99      0.99      0.99      1000
     class 2       0.91      0.84      0.87      1000
     class 3       0.92      0.93      0.92      1000
     class 4       0.86      0.91      0.88      1000
     class 5       0.98      0.98      0.98      1000
     class 6       0.75      0.80      0.78      1000
     class 7       0.95      0.96      0.96      1000
     class 8       0.98      0.97      0.98      1000
     class 9       0.96      0.96      0.96      1000

    accuracy                           0.92     10000
   macro avg       0.92      0.92      0.92     10000
weighted avg       0.92      0.92      0.92     10000
```

Conclusion :

The deep learning model was able to achieve an accuracy of 91.8% on the test set. This is a significant improvement over the baseline accuracy of 50%. The model could be further improved by using a larger dataset, a more complex model, or a different optimisation algorithm.

...THE END...