fashion-mnist-classification-1

March 11, 2024

Fashion_mnist_classification_project by:-Ameet raj

IMPORT LIBRARIES

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import tensorflow as tf
from tensorflow.python import keras
```

LOAD THE FASHION MNIST DATASET

[3]:
```
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.fashion_mnist.
↪load_data()
```

"Exploring the Dimensions"

[5]:
```
X_train.shape,y_train.shape
```

[5]:
```
((60000, 28, 28), (60000,))
```

[6]:
```
X_test.shape,y_test.shape
```

[6]:
```
((10000, 28, 28), (10000,))
```

INFRENCES:-X_train contains 60,000 images, each with dimensions 28x28 pixels. y_train contains 60,000 labels, each corresponding to one of the images in X_train. similerly X_test has 10,000 images,each 28x28 pixels. y_test has 10,000 corresponding labels.

[7]:
```
X_train.ndim,y_train.ndim
```

[7]:
```
(3, 1)
```

[8]:
```
X_test.ndim,y_test.ndim
```

[8]:
```
(3, 1)
```

[9]:
```
X_train
```

```
[9]: array([[[0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             …,
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0]],

            [[0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             …,
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0]],

            [[0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             …,
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0]],

            …,

            [[0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             …,
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0]],

            [[0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             …,
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0]],

            [[0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             …,
             [0, 0, 0, …, 0, 0, 0],
```

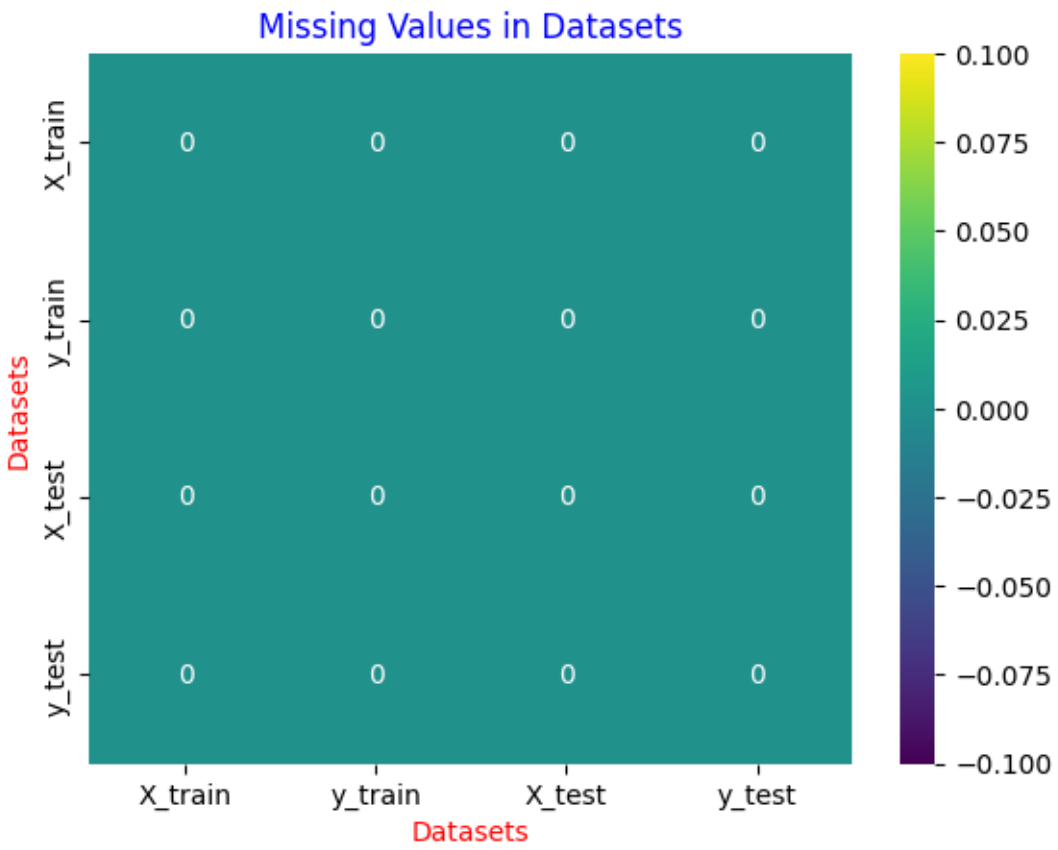
```
        [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0]]], dtype=uint8)
```

Checking Missing Values:-

[10]:
```
print("missing values in X_train:", np.isnan(X_train).sum())
print("missing values in y_train:", np.isnan(y_train).sum())
print("missing values in X_test:", np.isnan(X_test).sum())
print("missing values in y_test:", np.isnan(y_test).sum())
```

```
missing values in X_train: 0
missing values in y_train: 0
missing values in X_test: 0
missing values in y_test: 0
```

[11]:
```
missing_matrix = np.zeros((4, 4))
sns.heatmap(missing_matrix, annot=True, cmap='viridis', xticklabels=['X_train',␣
 ↪'y_train', 'X_test', 'y_test'], yticklabels=['X_train', 'y_train', 'X_test',␣
 ↪'y_test'])
plt.xlabel('Datasets',color='red')
plt.ylabel('Datasets',color ='red')
plt.title('Missing Values in Datasets',color= 'blue')
plt.show()
```



INFRENCES:-There is no missing values.

[12]:
```
class_labels = {
    0: "T-shirt/top",
    1: "Trouser",
    2: "Pullover",
    3: "Dress",
    4: "Coat",
    5: "Sandal",
    6: "Shirt",
    7: "Sneaker",
    8: "Bag",
    9: "Ankle boot"
}
```

[13]:
```
X_train [0]
```

[13]:
```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,
          0,   0,  13,  73,   0,   0,   1,   4,   0,   0,   0,   0,   1,
          1,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
          0,  36, 136, 127,  62,  54,   0,   0,   0,   1,   3,   4,   0,
          0,   3],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   6,
          0, 102, 204, 176, 134, 144, 123,  23,   0,   0,   0,   0,  12,
         10,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0, 155, 236, 207, 178, 107, 156, 161, 109,  64,  23,  77, 130,
         72,  15],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   0,
         69, 207, 223, 218, 216, 216, 163, 127, 121, 122, 146, 141,  88,
        172,  66],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   1,   1,   0,
        200, 232, 232, 233, 229, 223, 223, 215, 213, 164, 127, 123, 196,
        229,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
 183, 225, 216, 223, 228, 235, 227, 224, 222, 224, 221, 223, 245,
 173,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
 193, 228, 218, 213, 198, 180, 212, 210, 211, 213, 223, 220, 243,
 202,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   3,   0,  12,
 219, 220, 212, 218, 192, 169, 227, 208, 218, 224, 212, 226, 197,
 209,  52],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   6,   0,  99,
 244, 222, 220, 218, 203, 198, 221, 215, 213, 222, 220, 245, 119,
 167,  56],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   4,   0,   0,  55,
 236, 228, 230, 228, 240, 232, 213, 218, 223, 234, 217, 217, 209,
  92,   0],
[  0,   0,   1,   4,   6,   7,   2,   0,   0,   0,   0,   0, 237,
 226, 217, 223, 222, 219, 222, 221, 216, 223, 229, 215, 218, 255,
  77,   0],
[  0,   3,   0,   0,   0,   0,   0,   0,   0,  62, 145, 204, 228,
 207, 213, 221, 218, 208, 211, 218, 224, 223, 219, 215, 224, 244,
 159,   0],
[  0,   0,   0,   0,  18,  44,  82, 107, 189, 228, 220, 222, 217,
 226, 200, 205, 211, 230, 224, 234, 176, 188, 250, 248, 233, 238,
 215,   0],
[  0,  57, 187, 208, 224, 221, 224, 208, 204, 214, 208, 209, 200,
 159, 245, 193, 206, 223, 255, 255, 221, 234, 221, 211, 220, 232,
 246,   0],
[  3, 202, 228, 224, 221, 211, 211, 214, 205, 205, 205, 220, 240,
  80, 150, 255, 229, 221, 188, 154, 191, 210, 204, 209, 222, 228,
 225,   0],
[ 98, 233, 198, 210, 222, 229, 229, 234, 249, 220, 194, 215, 217,
 241,  65,  73, 106, 117, 168, 219, 221, 215, 217, 223, 223, 224,
 229,  29],
[ 75, 204, 212, 204, 193, 205, 211, 225, 216, 185, 197, 206, 198,
 213, 240, 195, 227, 245, 239, 223, 218, 212, 209, 222, 220, 221,
 230,  67],
[ 48, 203, 183, 194, 213, 197, 185, 190, 194, 192, 202, 214, 219,
 221, 220, 236, 225, 216, 199, 206, 186, 181, 177, 172, 181, 205,
 206, 115],
[  0, 122, 219, 193, 179, 171, 183, 196, 204, 210, 213, 207, 211,
 210, 200, 196, 194, 191, 195, 191, 198, 192, 176, 156, 167, 177,
 210,  92],
[  0,   0,  74, 189, 212, 191, 175, 172, 175, 181, 185, 188, 189,
 188, 193, 198, 204, 209, 210, 210, 211, 188, 188, 194, 192, 216,
 170,   0],
[  2,   0,   0,   0,  66, 200, 222, 237, 239, 242, 246, 243, 244,
 221, 220, 193, 191, 179, 182, 182, 181, 176, 166, 168,  99,  58,
   0,   0],
```

```
[  0,   0,   0,   0,   0,   0,   0,  40,  61,  44,  72,  41,  35,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0]], dtype=uint8)
```

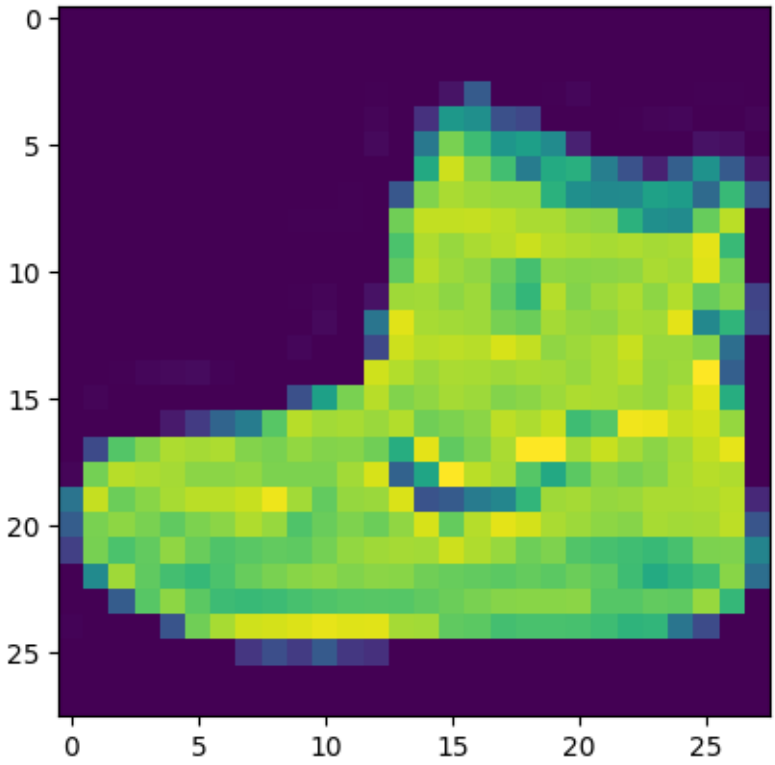
[14]:
```
y_train[0]
```

[14]:
```
9
```

INFRENCES:-The first image in the training dataset is a grayscale image with dimensions 28x28 pixels, and its corresponding label is 9, representing the class “Ankle boot”.

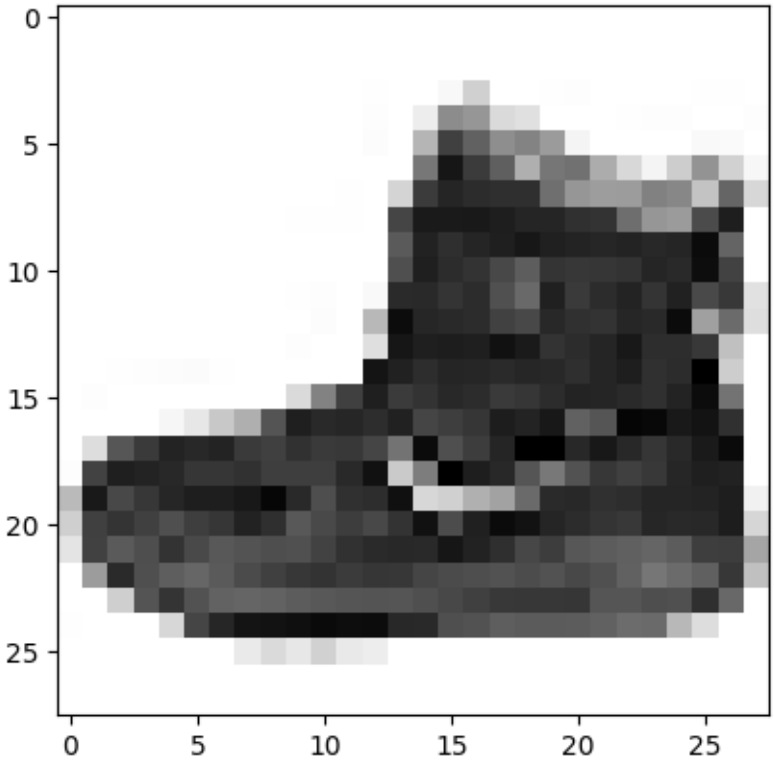
[15]:
```
plt.imshow(X_train[0])
```

[15]:
```
<matplotlib.image.AxesImage at 0x7fc6e1b943d0>
```



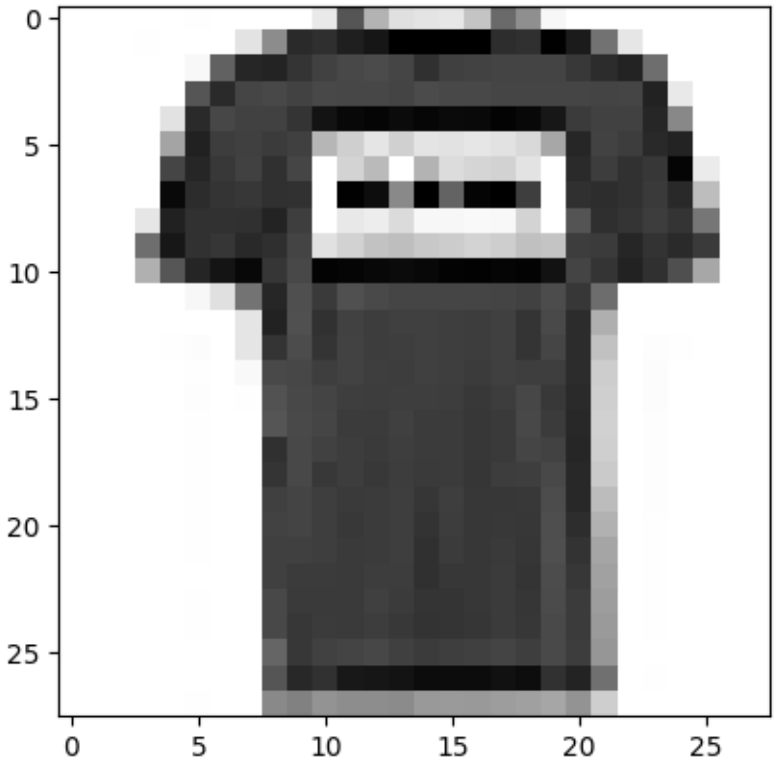
[16]:
```
plt.imshow(X_train[0], cmap='Greys')
```

[16]: <matplotlib.image.AxesImage at 0x7fc6dfa6d0c0>



[17]:
```
plt.imshow(X_train[1], cmap='Greys')
```

[17]: <matplotlib.image.AxesImage at 0x7fc6e1b63130>



NOTE:- Grayscale images are preferred due to Simplicity,Reduced Dimensionality,Focus on Structure,Compatibility.Image classification is my task ,not color information So i will go with greyscale.

[18]:
```
y_test[0]
```

[18]: 9

[19]:
```
y_test[1]
```

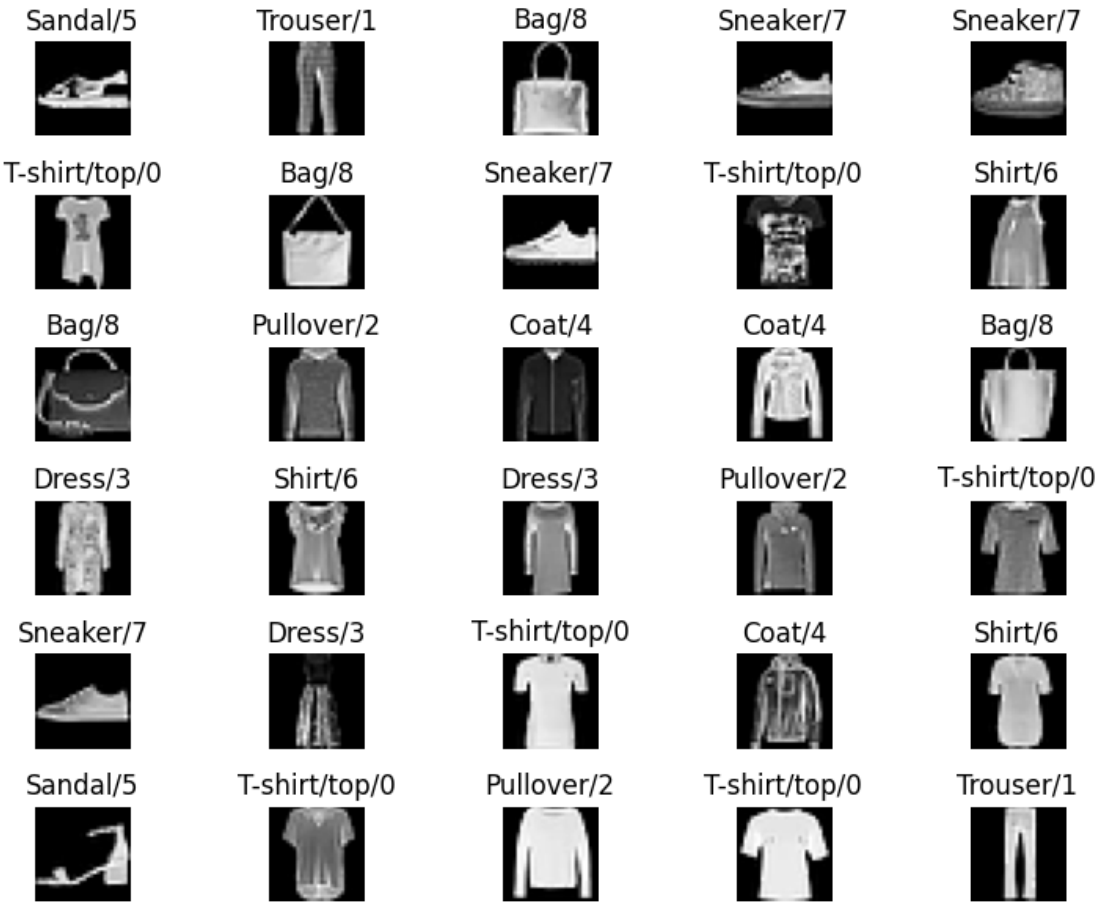
[19]: 2

INFRENCES:- 9 represents an “Ankle boot”. 2 represents a “Pullover”. (provided in class labels)

“Visualizing 30 Test Images with Labels and Descriptions”

[20]:
```
from tensorflow.keras.datasets import fashion_mnist
# Visualize a grid of 25 random images from the training set
plt.figure(figsize=(8, 6))
random_indices = np.random.randint(0, len(X_train), 30)
for i, index in enumerate(random_indices, 1):
    plt.subplot(6, 5, i)
    plt.imshow(X_train[index], cmap="gray")
    plt.axis('off')
```

```
    plt.title("{}/{}".format(class_labels[y_train[index]], y_train[index]))
plt.tight_layout()
plt.show()
```



Change Dimension

[21]:
```
X_train.ndim
```

[21]: 3

[25]:
```
X_train= np.expand_dims(X_train,-1)
X_test= np.expand_dims(X_test,-1)
```

[26]:
```
X_train.ndim
```

[26]: 4

[27]:
```
X_train.shape
```

[27]:
```
(60000, 28, 28, 1)
```

[28]:
```
X_test.shape
```

[28]:
```
(10000, 28, 28, 1)
```

INFRENCES:-Working with Convolutional Neural Networks (CNNs) in deep learning, it is necessary to provide the input data in a 4-dimensional format.like (60000, 28, 28, 1) indicates that i have a dataset with 60,000 grayscale images, where each image is 28 pixels tall, 28 pixels wide, and has a single channel.

Data Pre-Processing Feature Scaling / Normalization

[30]:
```
X_train_Scaled = X_train/255
X_test_Scaled = X_test/255
```

Split Dataset

[32]:
```
from sklearn.model_selection import train_test_split
```

[33]:
```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
↪2, random_state=42)
```

[34]:
```
X_train.shape,y_train.shape
```

[34]:
```
((48000, 28, 28, 1), (48000,))
```

[35]:
```
X_val.shape,y_val.shape
```

[35]:
```
((12000, 28, 28, 1), (12000,))
```

Model building:- Convolutional Neural Network

[36]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Define the CNN architecture
model = Sequential([
    Conv2D(filters=32, kernel_size=3, strides=(1, 1), padding='valid',␣
 ↪activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(units=128, activation='relu'),
    Dense(units=10, activation='softmax')
])
```

[37]:
```
model.summary()
```

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

 max_pooling2d (MaxPooling2  (None, 13, 13, 32)        0
 D)

 flatten (Flatten)           (None, 5408)              0

 dense (Dense)               (None, 128)               692352

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 693962 (2.65 MB)
Trainable params: 693962 (2.65 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Infrences:-Model Architecture Overview: The CNN architecture features convolutional and pooling layers for feature extraction, followed by fully connected layers for classification. With 693,962 trainable parameters, the model learns intricate patterns to accurately categorize images, making it versatile for various real-world applications.

Compiling the model

[38]:
```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
↪metrics=['accuracy'])
```

Training the model

[43]:
```
model.fit(X_train, y_train, batch_size=128, epochs=5, verbose=1)
```

```
Epoch 1/5
375/375 [==============================] - 21s 56ms/step - loss: 0.1609 -
accuracy: 0.9410
Epoch 2/5
375/375 [==============================] - 20s 54ms/step - loss: 0.1399 -
accuracy: 0.9497
Epoch 3/5
375/375 [==============================] - 20s 54ms/step - loss: 0.1226 -
accuracy: 0.9555
Epoch 4/5
375/375 [==============================] - 21s 56ms/step - loss: 0.1135 -
accuracy: 0.9588
Epoch 5/5
375/375 [==============================] - 23s 62ms/step - loss: 0.1015 -
```

```
accuracy: 0.9642
```

[43]:
```
<keras.src.callbacks.History at 0x7fc6df981ae0>
```

Evaluating The Model :- Evaluate the model against the Testing dataset of 12,000 images.

[51]:
```
score = model.evaluate(X_test, y_test)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
313/313 [==============================] - 2s 6ms/step - loss: 0.2791 -
accuracy: 0.9136
Test score: 0.2790828347206116
Test accuracy: 0.9136000275611877
```

INFRENCES:- The model showcases robust performance across both the training and test datasets, suggesting a keen ability to generalize from the training data to unseen examples. This proficiency underscores its efficacy in learning intricate patterns and nuances from the training set, ultimately translating into strong predictive capabilities when confronted with new, previously unseen data.

Make prediction

[46]:
```
model.predict(np.expand_dims(X_test[0],axis=0)).round(2)
```

```
1/1 [==============================] - 0s 63ms/step
```

[46]:
```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]], dtype=float32)
```

Finding the indices of the maximum values along an axis of an array.

[47]:
```
np.argmax(model.predict(np.expand_dims(X_test[0],axis=0)).round(2))
```

```
1/1 [==============================] - 0s 21ms/step
```

[47]:
```
9
```

Cross checking

[48]:
```
y_test[0]
```

[48]:
```
9
```

[53]:
```
y_pred = model.predict(X_test).round(2)
y_pred
```

```
313/313 [==============================] - 2s 6ms/step
```

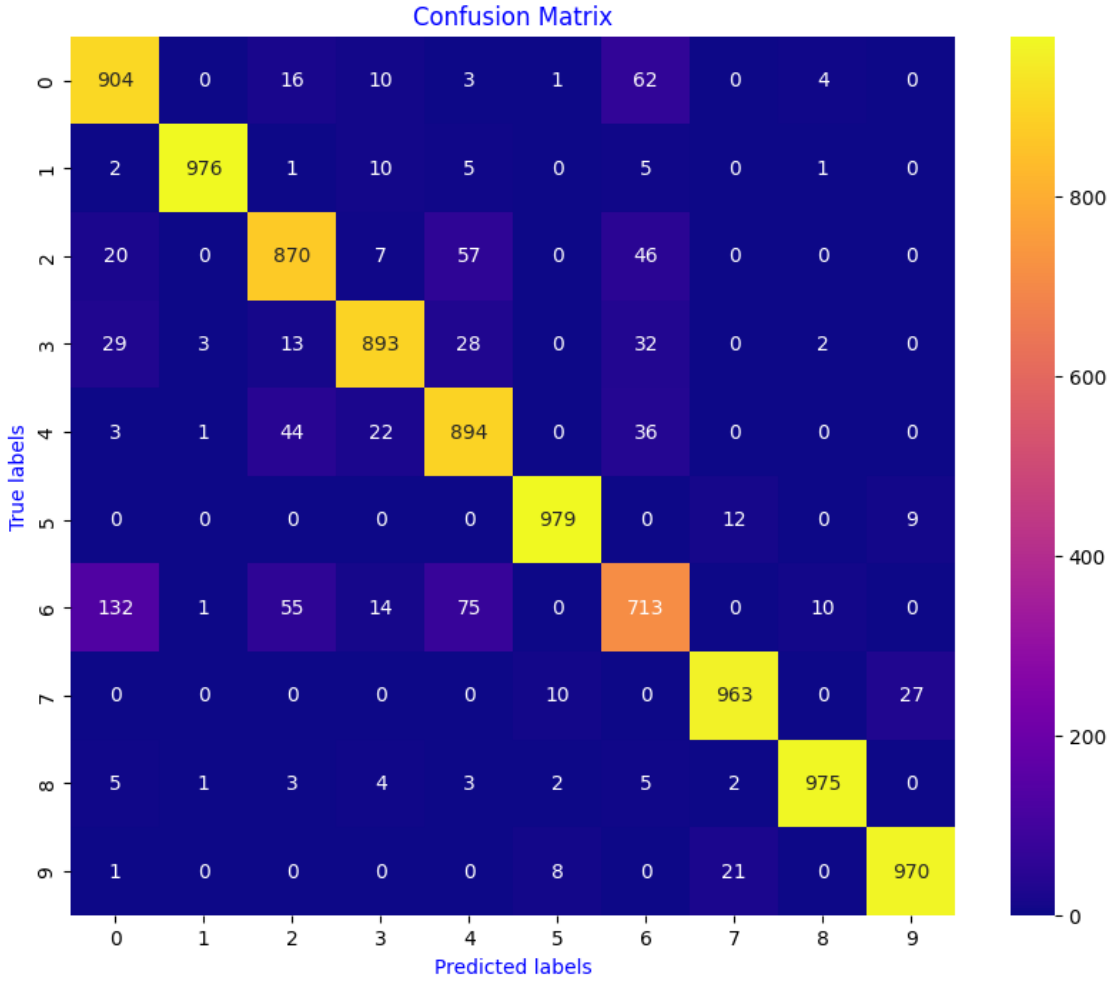
[53]:
```
array([[0.  , 0.  , 0.  , …, 0.  , 0.  , 1.  ],
       [0.  , 0.  , 1.  , …, 0.  , 0.  , 0.  ],
```

```
[0.  , 1.  , 0.  , …, 0.  , 0.  , 0.  ],
…,
[0.  , 0.  , 0.  , …, 0.  , 1.  , 0.  ],
[0.  , 1.  , 0.  , …, 0.  , 0.  , 0.  ],
[0.  , 0.  , 0.  , …, 0.11, 0.01, 0.  ]], dtype=float32)
```

Confusion Matrix

[58]:
```
from sklearn.metrics import confusion_matrix
y_pred_labels = [np.argmax(label) for label in y_pred]
conf_matrix = confusion_matrix(y_test, y_pred_labels)

plt.figure(figsize=(10,8))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="plasma",␣
 ↪xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted labels',color='blue')
plt.ylabel('True labels',color='blue')
plt.title('Confusion Matrix',color='blue')
plt.show()
```



Classification Report

[65]:
```
from sklearn.metrics import classification_report
y_pred_labels = [np.argmax(label) for label in y_pred]

report = classification_report(y_test, y_pred_labels)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86      1000
           1       0.99      0.98      0.98      1000
           2       0.87      0.87      0.87      1000
           3       0.93      0.89      0.91      1000
           4       0.84      0.89      0.87      1000
           5       0.98      0.98      0.98      1000
           6       0.79      0.71      0.75      1000
           7       0.96      0.96      0.96      1000
           8       0.98      0.97      0.98      1000
           9       0.96      0.97      0.97      1000

    accuracy                           0.91     10000
   macro avg       0.91      0.91      0.91     10000
weighted avg       0.91      0.91      0.91     10000
```

INFRENCES:-Overall, the classification report indicates that the model achieves high precision, recall, and F1-score for most classes, with an overall accuracy of 91%. The macro and weighted averages also suggest consistent performance across all classes.

[ ]:

[ ]:

[ ]: