AML_Project

May 9, 2021

Importing Libraries

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
import tensorflow as tf
from tensorflow import keras
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.manifold import TSNE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
%matplotlib inline
```

Loading DataSet

[2]:
```
df_train = pd.read_csv('train.csv')
df_test = pd.read_csv('test.csv')
```

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

[3]:
```
   label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0      1       0       0       0       0       0       0       0       0
1      0       0       0       0       0       0       0       0       0
2      1       0       0       0       0       0       0       0       0
3      4       0       0       0       0       0       0       0       0
4      0       0       0       0       0       0       0       0       0

   pixel8  …  pixel774  pixel775  pixel776  pixel777  pixel778  pixel779  \
0       0  …         0         0         0         0         0         0
1       0  …         0         0         0         0         0         0
```

```
2       0  ...         0         0         0         0         0         0
3       0  ...         0         0         0         0         0         0
4       0  ...         0         0         0         0         0         0

   pixel780  pixel781  pixel782  pixel783
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0

[5 rows x 785 columns]
```

[4]:
```
df_test.head()
```

[4]:
```
   pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
0       0       0       0       0       0       0       0       0       0
1       0       0       0       0       0       0       0       0       0
2       0       0       0       0       0       0       0       0       0
3       0       0       0       0       0       0       0       0       0
4       0       0       0       0       0       0       0       0       0

   pixel9  ...  pixel774  pixel775  pixel776  pixel777  pixel778  pixel779  \
0       0  ...         0         0         0         0         0         0
1       0  ...         0         0         0         0         0         0
2       0  ...         0         0         0         0         0         0
3       0  ...         0         0         0         0         0         0
4       0  ...         0         0         0         0         0         0

   pixel780  pixel781  pixel782  pixel783
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0

[5 rows x 784 columns]
```

[5]:
```
df_train.shape
```

[5]:
```
(42000, 785)
```

[6]:
```
df_test.shape
```

[6]:
```
(28000, 784)
```

Exploratory Data Analysis

[7]:
```
df_train.isnull().any().describe()
```

[7]:
```
count       785
unique        1
top       False
freq        785
dtype: object
```

[8]:
```
df_test.isnull().any().describe()
```

[8]:
```
count       784
unique        1
top       False
freq        784
dtype: object
```

[9]:
```
df_train["label"].value_counts()
```

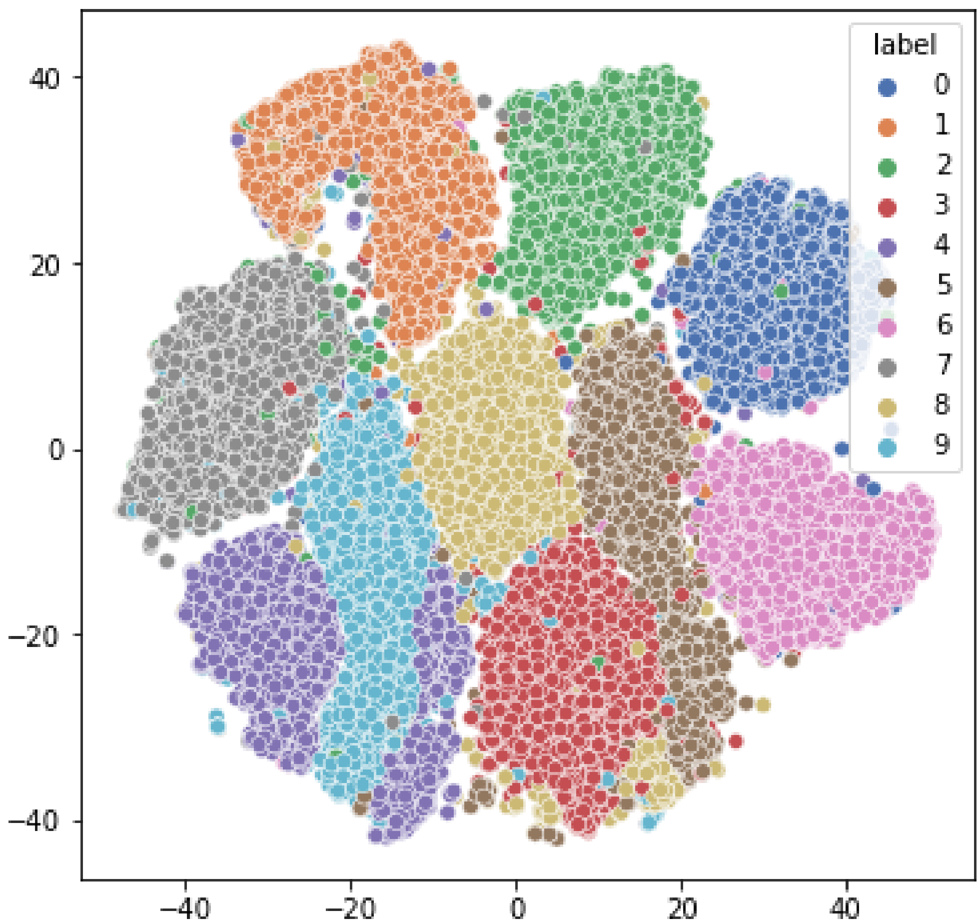
[9]:
```
1    4684
7    4401
3    4351
9    4188
2    4177
6    4137
0    4132
4    4072
8    4063
5    3795
Name: label, dtype: int64
```

[10]:
```
y = df_train['label']
X = df_train.drop('label',axis=1)
```

[11]:
```
#df_train = df_train.to_numpy() / 255.0
#train_labels = train_labels.to_numpy()
X = X/255
df_test = df_test/255
```

[12]:
```
X_embedded = TSNE(n_components=2, perplexity = 30, early_exaggeration = 12,␣
 ↪learning_rate = 100).fit_transform(X)
plt.figure(figsize = (6,6))
sns.scatterplot(x = X_embedded[:,0], y = X_embedded[:,1], hue = y, palette =␣
 ↪'deep')
```

[12]:
```
<AxesSubplot:>
```

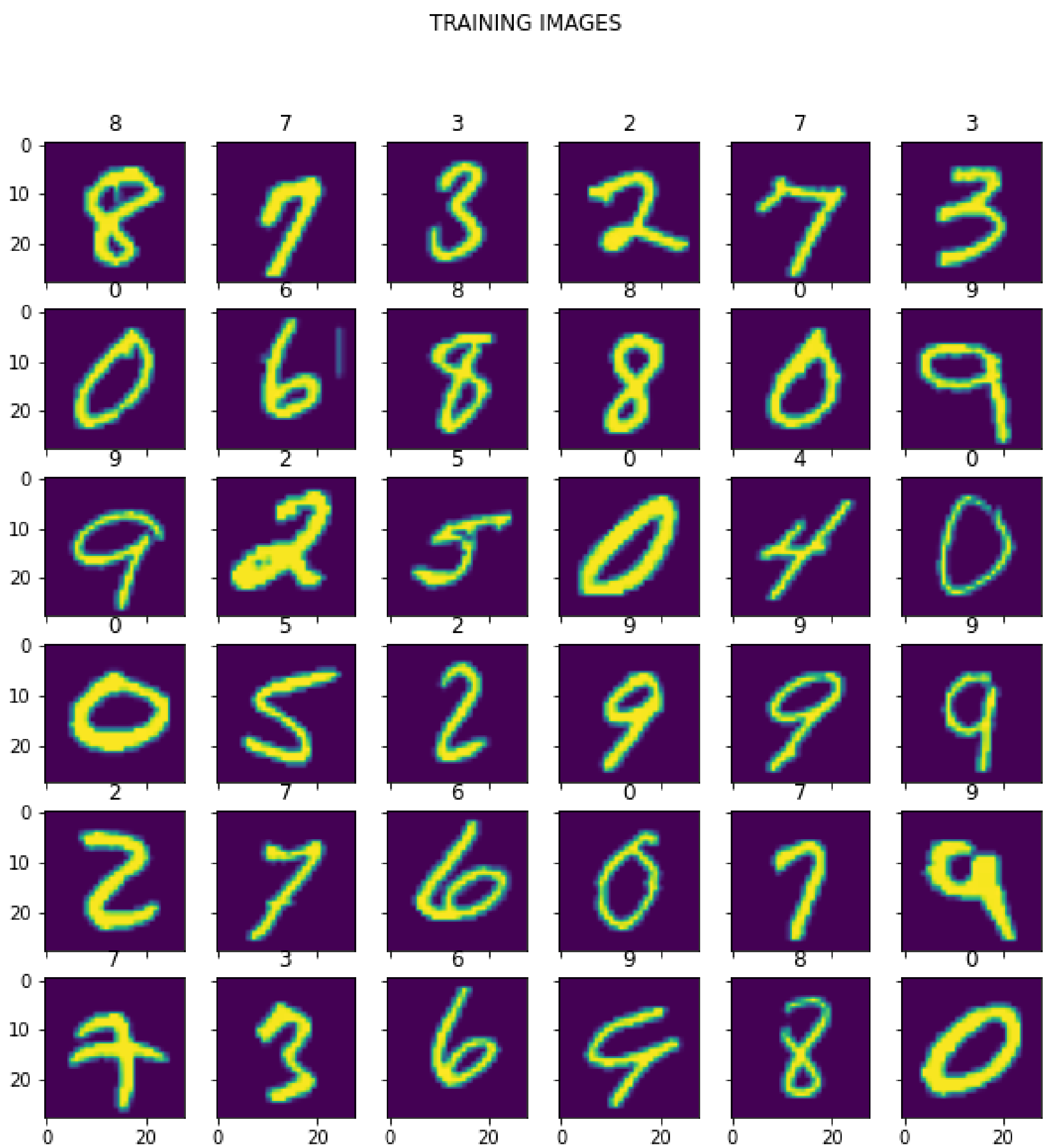


[13]:
```
x_train, x_val, y_train, y_val = train_test_split(X,y,test_size=0.2)
print(f'x_train.shape: {x_train.shape}, x_val.shape: {x_val.shape}')
```

```
x_train.shape: (33600, 784), x_val.shape: (8400, 784)
```

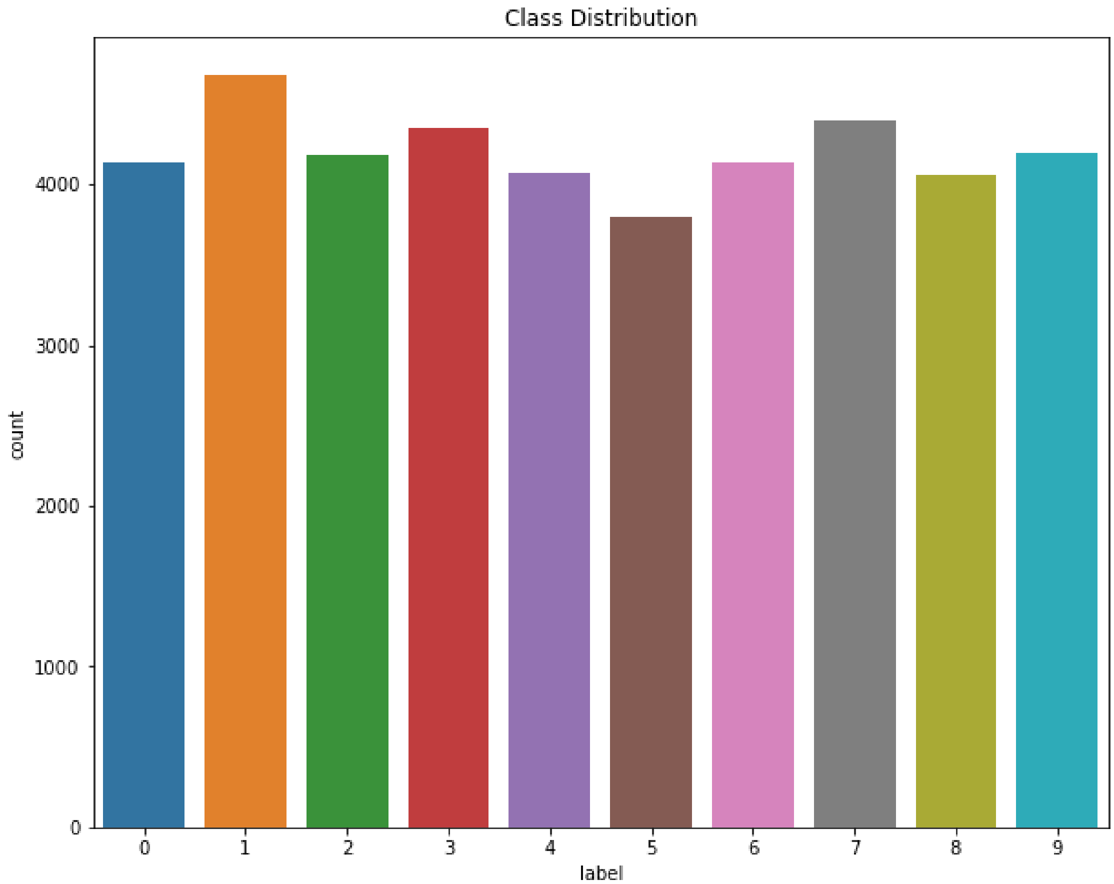
[14]:
```
def random_view(X, title, y = None):
    fig, axs = plt.subplots(6, 6, sharex= True, sharey = True, figsize = (10,␣
↪10))
    fig.suptitle(title)

    for i in range(6):
        for j in range(6):
            n = random.randint(0, len(X))
            axs[i][j].imshow(X.iloc[n].values.reshape(28, 28,1))
            if y is not None:
                axs[i][j].set_title(y.iloc[n])
```

[15]:
```
random_view(X, "TRAINING IMAGES", y)
```



[16]:
```
plt.figure(figsize=(10,8))
sns.countplot(x=y)
plt.title('Class Distribution')
```

[16]: Text(0.5, 1.0, 'Class Distribution')



Distributing Data into train and test data

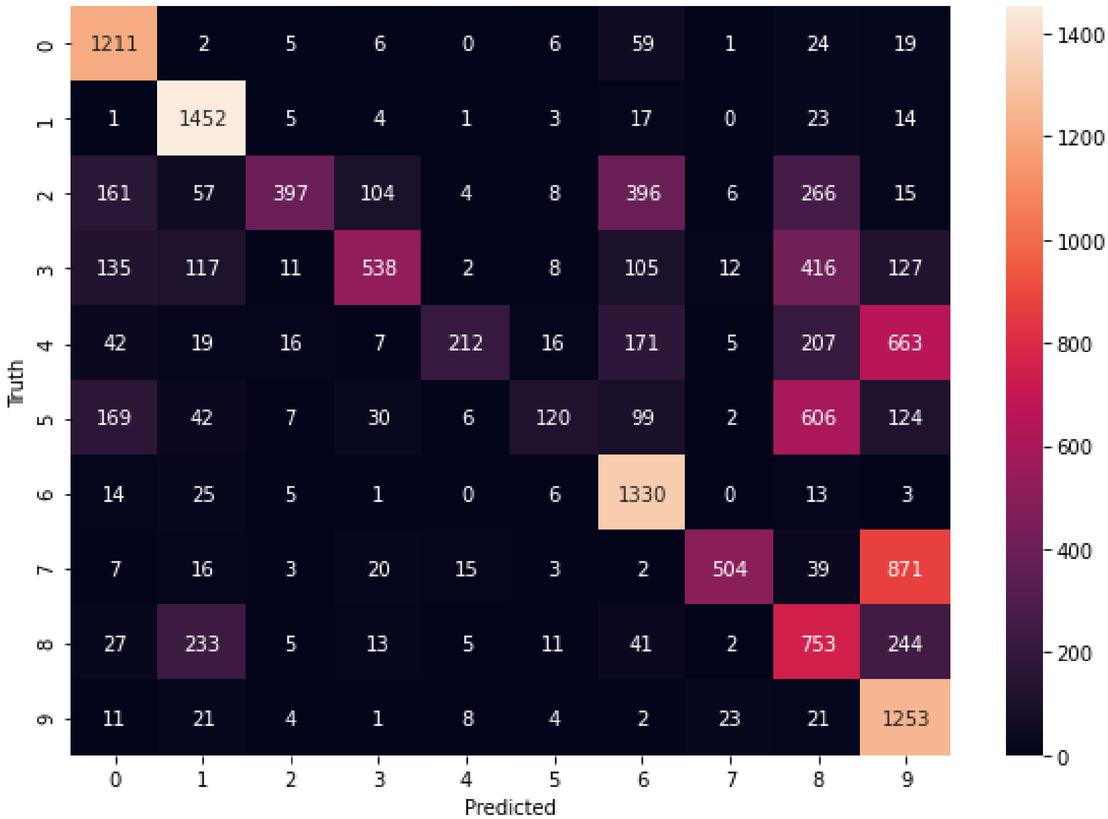
```
[17]: # Splitting Data
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.33,␣
 ↪random_state=42)
```

Naive Bayes Classifier

```
[18]: clf = GaussianNB()
clf.fit(X_train,Y_train)
Y_predict_nb = clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(Y_test, Y_predict_nb) * 100)
cmNB = confusion_matrix(Y_test, Y_predict_nb)
plt.figure(figsize=(10,7))
sns.heatmap(cmNB,annot=True , fmt = 'd')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Accuracy: 56.06060606060606055
```

[18]: Text(69.0, 0.5, 'Truth')



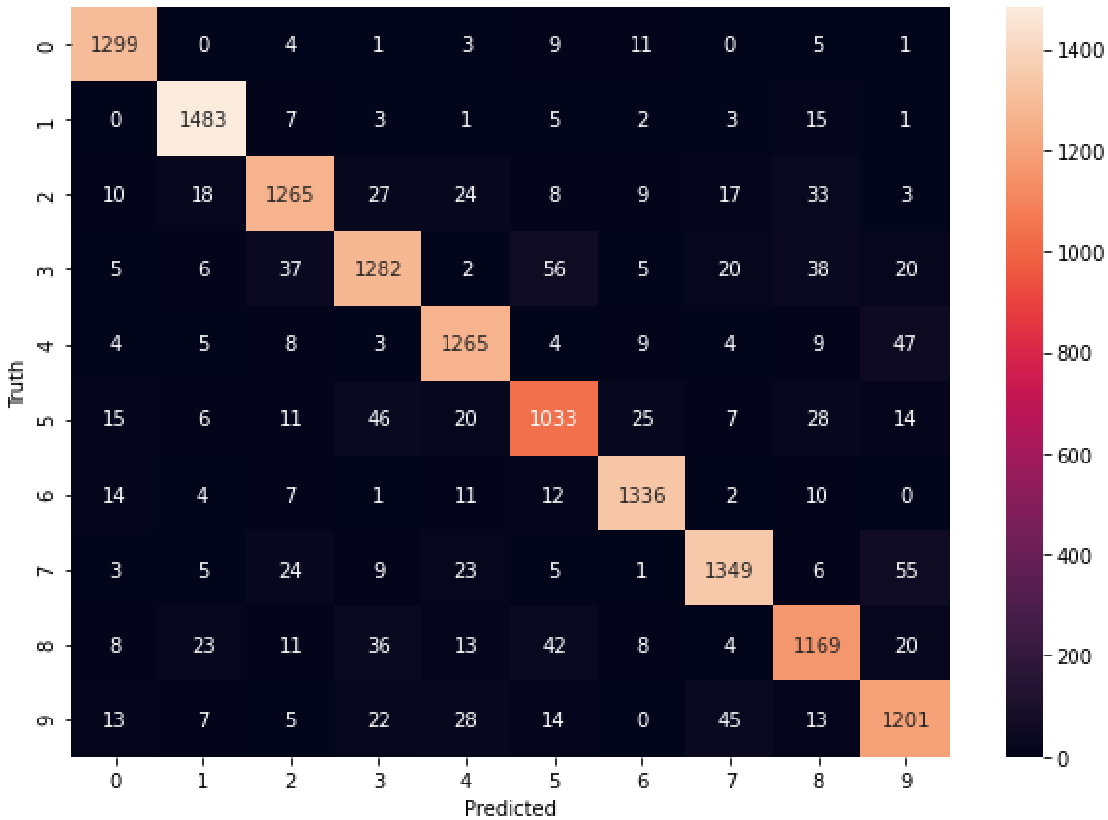
Accuracy for Naive Bayes Classifier **Accuracy: 56.060606060606055**

Logistic Regression

```
[19]: ## Logistic Regression
clf = LogisticRegression(random_state=0,solver='liblinear').fit(X_train,␣
 ↪Y_train)
Y_predict_lr = clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(Y_test, Y_predict_lr) * 100)
#confusion_matrix(Y_test, Y_predict_lr)
cmLR = confusion_matrix(Y_test, Y_predict_lr)
plt.figure(figsize=(10,7))
sns.heatmap(cmLR,annot=True , fmt = 'd')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Accuracy: 91.50072150072151

[19]: Text(69.0, 0.5, 'Truth')

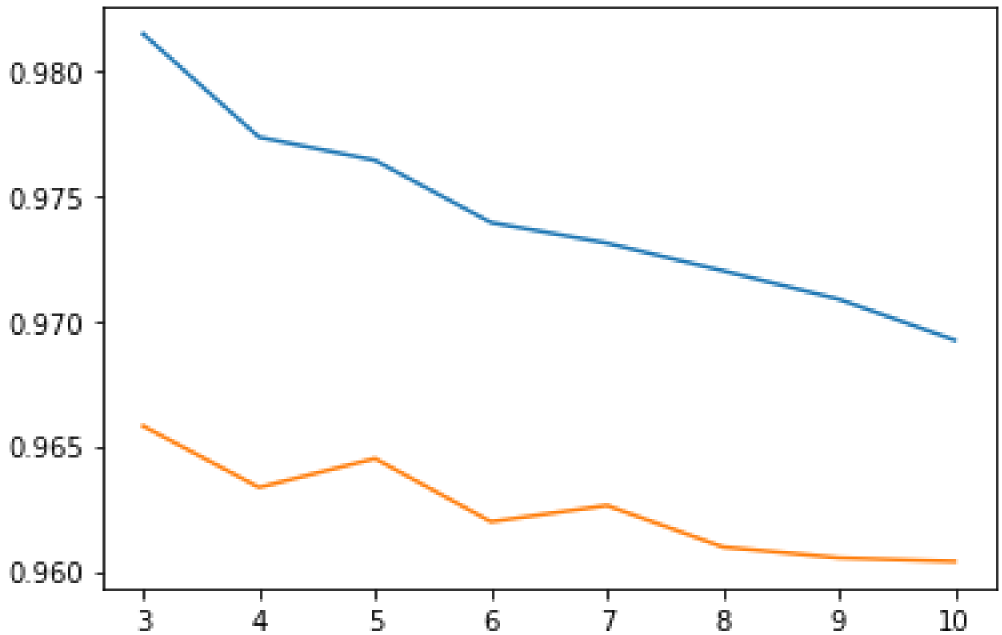


Accuracy for Logistic Regression **Accuracy: 91.50072150072151**

KNN classifier

[20]:
```
train_scores = []
test_score = []
for i in range(3,11):
    clf = KNeighborsClassifier(i)
    clf.fit(X_train,Y_train)
    train_scores.append(clf.score(X_train,Y_train))
    test_score.append(clf.score(X_test,Y_test))

plt.plot(list(range(3,11)),train_scores)
plt.plot(list(range(3,11)),test_score)

print("Accuracy:",max(test_score))
```

```
Accuracy: 0.9658008658008658
```

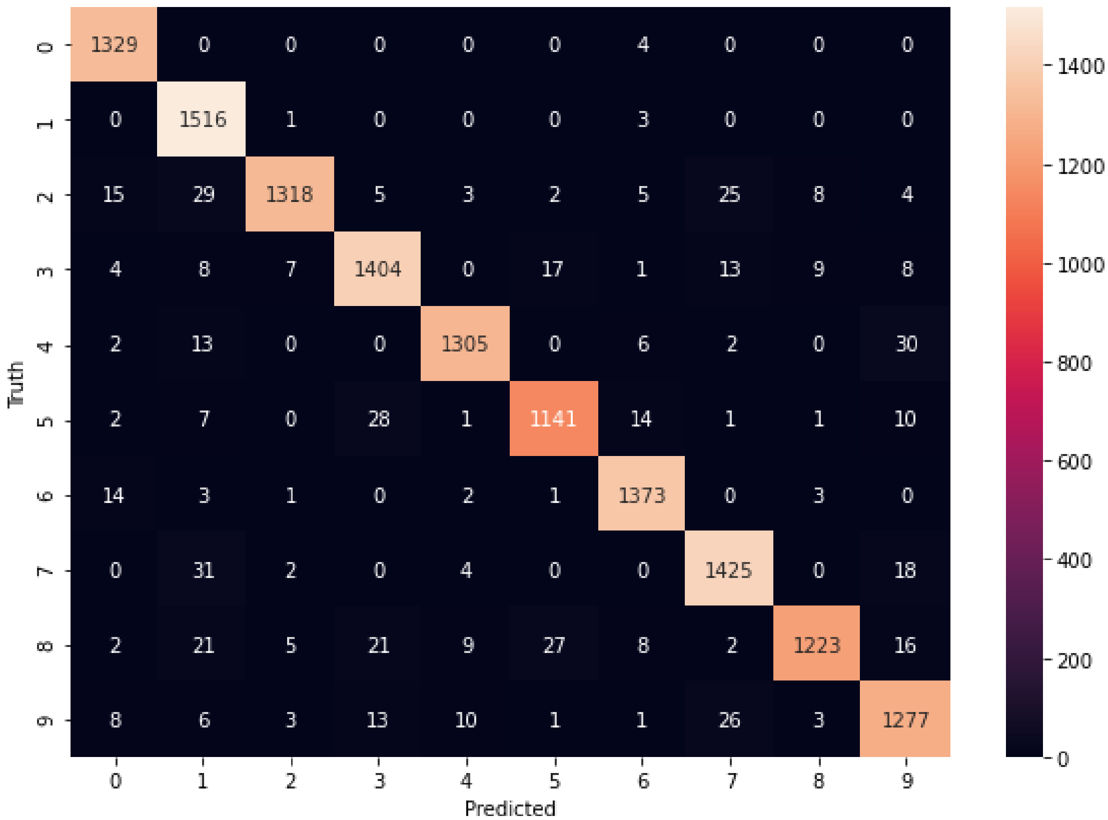


Accuracy for KNN classifier **Accuracy: 96.58008658008658**

Confusion Matrix for KNN Classifier

[21]:
```
cmKN = confusion_matrix(Y_test,clf.predict(X_test))
plt.figure(figsize=(10,7))
sns.heatmap(cmKN ,annot=True , fmt = 'd')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

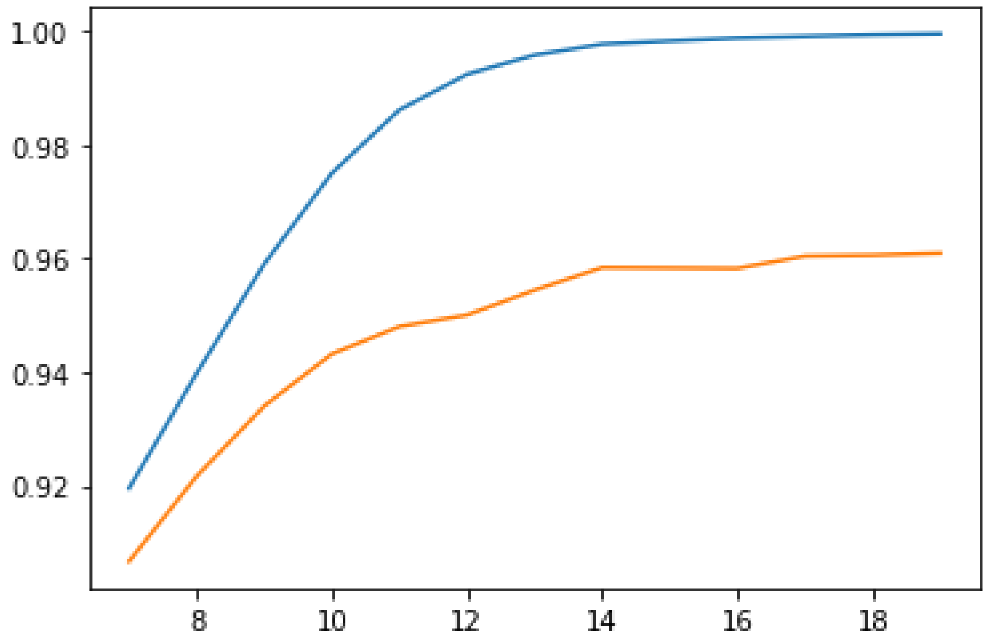
[21]:
```
Text(69.0, 0.5, 'Truth')
```



Random Forest

[22]:
```
train_scores = []
test_score = []
for i in range(7,20):
    forest_model = RandomForestClassifier(max_depth= i)
    forest_model.fit(X_train,Y_train)
    train_scores.append(forest_model.score(X_train,Y_train))
    test_score.append(forest_model.score(X_test,Y_test))

plt.plot(list(range(7,20)),train_scores)
plt.plot(list(range(7,20)),test_score)

print(max(test_score))
```

```
0.9609668109668109
```

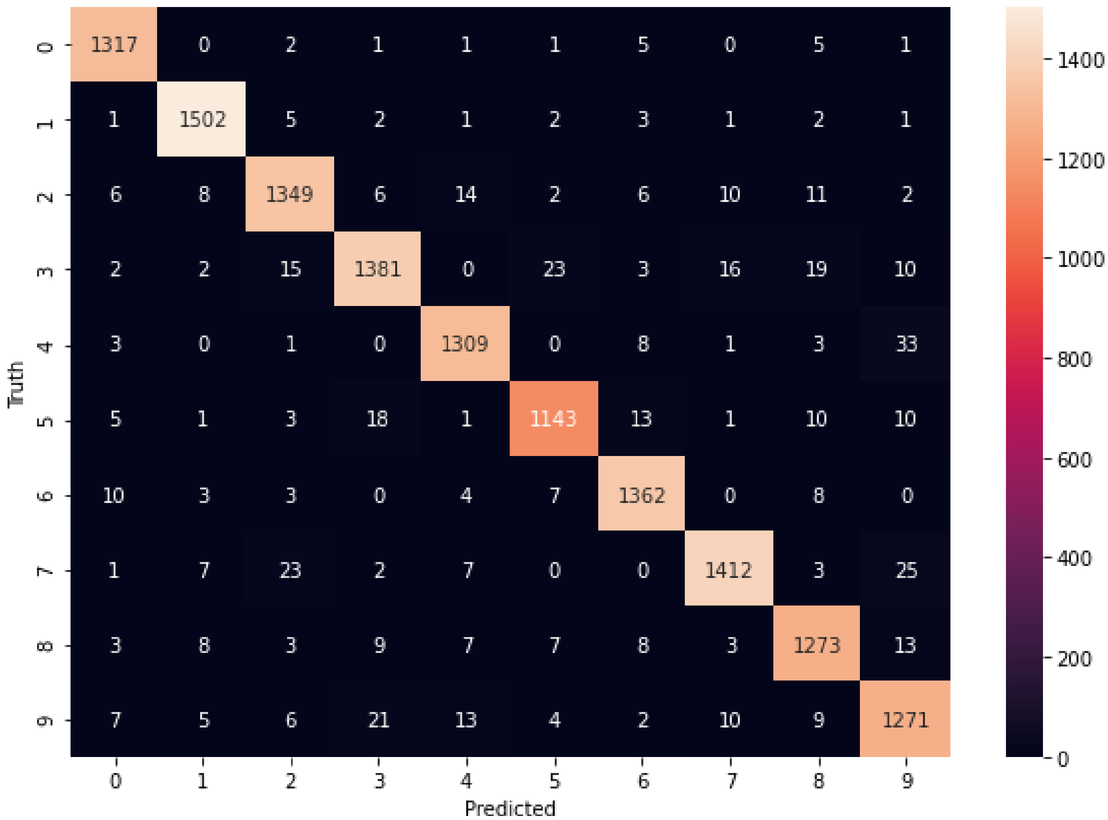


Accuracy for KNN classifier **Accuracy: 96.11111111111111**

Confusion Matrix for Random Forest

[23]:
```
cmRF = confusion_matrix(Y_test,forest_model.predict(X_test))
plt.figure(figsize=(10,7))
sns.heatmap(cmRF ,annot=True , fmt = 'd')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

[23]: Text(69.0, 0.5, 'Truth')



ANN

[24]:
```
X,df_test=X.values.reshape(-1,28,28),df_test.values.reshape(-1,28,28)
```

[25]:
```
X.shape,df_test.shape
```

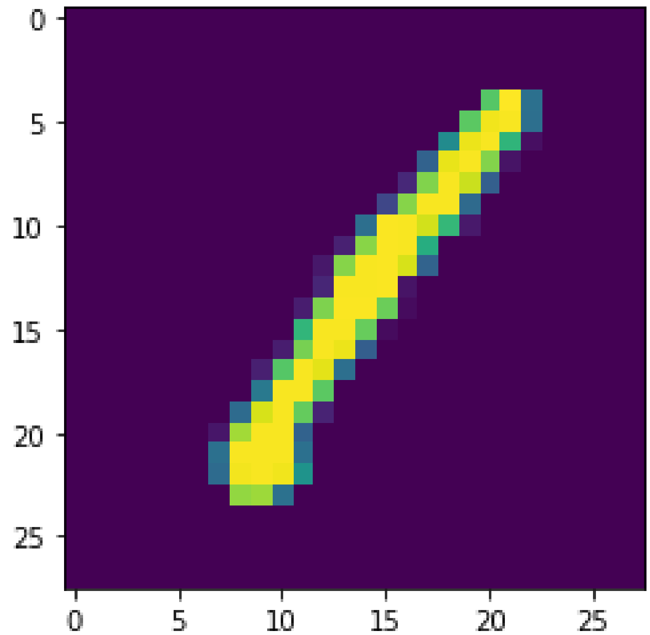
[25]: ((42000, 28, 28), (28000, 28, 28))

[26]:
```
X[0].shape
```

[26]: (28, 28)

[27]:
```
plt.imshow(X[0])
```

[27]: <matplotlib.image.AxesImage at 0x1808c5a7af0>



[28]:
```
X,df_test=X/255,df_test/255
```

[29]:
```
model=keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),
    keras.layers.Dense(100,activation='relu'),
    keras.layers.Dropout(0.20),
    keras.layers.Dense(50,activation='relu'),
    keras.layers.Dense(25,activation='relu'),
    keras.layers.Dropout(0.10),
    keras.layers.Dense(12,activation='relu'),
    keras.layers.Dense(10,activation='relu'),
    keras.layers.Dense(10,activation='softmax')
])
tensorboard=tf.keras.callbacks.TensorBoard(log_dir='logs/
 ↪Digit_Recog',histogram_freq=1)
model.
 ↪compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
history=model.
 ↪fit(X,y,epochs=100,validation_data=(X_test,Y_test),callbacks=tensorboard)
```

```
Epoch 1/100
1298/1313 [============================>.] - ETA: 0s - loss: 2.0012 - accuracy:
0.2386WARNING:tensorflow:Model was constructed with shape (None, 28, 28) for
input KerasTensor(type_spec=TensorSpec(shape=(None, 28, 28), dtype=tf.float32,
name='flatten_input'), name='flatten_input', description="created by layer
'flatten_input'"), but it was called on an input with incompatible shape (None,
```

```
784).
1313/1313 [==============================] - 10s 6ms/step - loss: 1.9986 -
accuracy: 0.2394 - val_loss: 74.9380 - val_accuracy: 0.3326
Epoch 2/100
1313/1313 [==============================] - 3s 2ms/step - loss: 1.4268 -
accuracy: 0.4470 - val_loss: 70.2096 - val_accuracy: 0.5352
Epoch 3/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.9504 -
accuracy: 0.6764 - val_loss: 69.0205 - val_accuracy: 0.6538
Epoch 4/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.7310 -
accuracy: 0.7655 - val_loss: 83.3662 - val_accuracy: 0.6350
Epoch 5/100
1313/1313 [==============================] - 2s 2ms/step - loss: 0.6042 -
accuracy: 0.8176 - val_loss: 86.0261 - val_accuracy: 0.6967
Epoch 6/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.4934 -
accuracy: 0.8569 - val_loss: 87.9015 - val_accuracy: 0.7010
Epoch 7/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.4319 -
accuracy: 0.8757 - val_loss: 96.4963 - val_accuracy: 0.7004
Epoch 8/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.4032 -
accuracy: 0.8836 - val_loss: 95.8176 - val_accuracy: 0.7274
Epoch 9/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.3669 -
accuracy: 0.8948 - val_loss: 91.0459 - val_accuracy: 0.7166
Epoch 10/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.3415 -
accuracy: 0.9016 - val_loss: 84.1077 - val_accuracy: 0.7363
Epoch 11/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.3083 -
accuracy: 0.9127 - val_loss: 74.8968 - val_accuracy: 0.7575
Epoch 12/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.2952 -
accuracy: 0.9178 - val_loss: 79.3102 - val_accuracy: 0.7541
Epoch 13/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.2847 -
accuracy: 0.9169 - val_loss: 73.6239 - val_accuracy: 0.7657
Epoch 14/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.2731 -
accuracy: 0.9216 - val_loss: 65.4409 - val_accuracy: 0.7925
Epoch 15/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.2627 -
accuracy: 0.9258 - val_loss: 65.7465 - val_accuracy: 0.8148
Epoch 16/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.2530 -
accuracy: 0.9268 - val_loss: 61.1051 - val_accuracy: 0.8240
```

```
Epoch 17/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.2533 -
accuracy: 0.9312 - val_loss: 61.6479 - val_accuracy: 0.8279
Epoch 18/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.2344 -
accuracy: 0.9315 - val_loss: 71.9565 - val_accuracy: 0.8193
Epoch 19/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.2255 -
accuracy: 0.9358 - val_loss: 70.7585 - val_accuracy: 0.8177
Epoch 20/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.2278 -
accuracy: 0.9368 - val_loss: 61.4477 - val_accuracy: 0.8367
Epoch 21/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.2182 -
accuracy: 0.9388 - val_loss: 60.7149 - val_accuracy: 0.8427
Epoch 22/100
1313/1313 [==============================] - 2s 2ms/step - loss: 0.2182 -
accuracy: 0.9373 - val_loss: 56.8648 - val_accuracy: 0.8416
Epoch 23/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.2017 -
accuracy: 0.9414 - val_loss: 62.5120 - val_accuracy: 0.8338
Epoch 24/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.2048 -
accuracy: 0.9410 - val_loss: 56.3433 - val_accuracy: 0.8475
Epoch 25/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1995 -
accuracy: 0.9427 - val_loss: 63.5517 - val_accuracy: 0.8348
Epoch 26/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1969 -
accuracy: 0.9447 - val_loss: 70.7078 - val_accuracy: 0.8274
Epoch 27/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1924 -
accuracy: 0.9429 - val_loss: 79.6578 - val_accuracy: 0.8151
Epoch 28/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1874 -
accuracy: 0.9459 - val_loss: 65.0604 - val_accuracy: 0.8311
Epoch 29/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1900 -
accuracy: 0.9463 - val_loss: 70.3447 - val_accuracy: 0.8271
Epoch 30/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1823 -
accuracy: 0.9479 - val_loss: 76.4673 - val_accuracy: 0.8110
Epoch 31/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1836 -
accuracy: 0.9475 - val_loss: 74.3584 - val_accuracy: 0.8176
Epoch 32/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1841 -
accuracy: 0.9470 - val_loss: 66.5767 - val_accuracy: 0.8234
```

```
Epoch 33/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1797 -
accuracy: 0.9487 - val_loss: 66.0537 - val_accuracy: 0.8283
Epoch 34/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1666 -
accuracy: 0.9522 - val_loss: 72.6994 - val_accuracy: 0.8136
Epoch 35/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1717 -
accuracy: 0.9509 - val_loss: 77.8710 - val_accuracy: 0.8034
Epoch 36/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1708 -
accuracy: 0.9519 - val_loss: 70.4602 - val_accuracy: 0.8263
Epoch 37/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1666 -
accuracy: 0.9525 - val_loss: 83.8880 - val_accuracy: 0.8064
Epoch 38/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1645 -
accuracy: 0.9510 - val_loss: 81.3851 - val_accuracy: 0.8171
Epoch 39/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1574 -
accuracy: 0.9547 - val_loss: 74.1195 - val_accuracy: 0.8218
Epoch 40/100
1313/1313 [==============================] - 2s 2ms/step - loss: 0.1535 -
accuracy: 0.9563 - val_loss: 94.9540 - val_accuracy: 0.7918
Epoch 41/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1618 -
accuracy: 0.9512 - val_loss: 91.9671 - val_accuracy: 0.7950
Epoch 42/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1553 -
accuracy: 0.9548 - val_loss: 73.3441 - val_accuracy: 0.8119
Epoch 43/100
1313/1313 [==============================] - 2s 2ms/step - loss: 0.1508 -
accuracy: 0.9564 - val_loss: 84.6498 - val_accuracy: 0.8096
Epoch 44/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1562 -
accuracy: 0.9534 - val_loss: 85.1347 - val_accuracy: 0.8064
Epoch 45/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1503 -
accuracy: 0.9584 - val_loss: 81.8338 - val_accuracy: 0.8189
Epoch 46/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1479 -
accuracy: 0.9573 - val_loss: 72.5547 - val_accuracy: 0.8172
Epoch 47/100
1313/1313 [==============================] - 2s 2ms/step - loss: 0.1507 -
accuracy: 0.9546 - val_loss: 89.7208 - val_accuracy: 0.8044
Epoch 48/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1442 -
accuracy: 0.9595 - val_loss: 89.3821 - val_accuracy: 0.7996
```

```
Epoch 49/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1419 -
accuracy: 0.9586 - val_loss: 81.9983 - val_accuracy: 0.8125
Epoch 50/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1497 -
accuracy: 0.9577 - val_loss: 101.7287 - val_accuracy: 0.7758
Epoch 51/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1336 -
accuracy: 0.9607 - val_loss: 100.7438 - val_accuracy: 0.7894
Epoch 52/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1375 -
accuracy: 0.9612 - val_loss: 96.2939 - val_accuracy: 0.8043
Epoch 53/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1375 -
accuracy: 0.9599 - val_loss: 78.8169 - val_accuracy: 0.8119
Epoch 54/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1366 -
accuracy: 0.9604 - val_loss: 97.7821 - val_accuracy: 0.7979
Epoch 55/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1401 -
accuracy: 0.9583 - val_loss: 105.5465 - val_accuracy: 0.7778
Epoch 56/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1314 -
accuracy: 0.9621 - val_loss: 96.5253 - val_accuracy: 0.8035
Epoch 57/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1377 -
accuracy: 0.9581 - val_loss: 88.3585 - val_accuracy: 0.8080
Epoch 58/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1352 -
accuracy: 0.9609 - val_loss: 83.7529 - val_accuracy: 0.8161
Epoch 59/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1301 -
accuracy: 0.9625 - val_loss: 81.3904 - val_accuracy: 0.8189
Epoch 60/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1307 -
accuracy: 0.9624 - val_loss: 82.4090 - val_accuracy: 0.8096
Epoch 61/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1346 -
accuracy: 0.9599 - val_loss: 88.8635 - val_accuracy: 0.8095
Epoch 62/100
1313/1313 [==============================] - 2s 2ms/step - loss: 0.1259 -
accuracy: 0.9612 - val_loss: 87.2317 - val_accuracy: 0.8126
Epoch 63/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1338 -
accuracy: 0.9612 - val_loss: 100.6909 - val_accuracy: 0.7792
Epoch 64/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1279 -
accuracy: 0.9641 - val_loss: 92.6808 - val_accuracy: 0.8014
```

```
Epoch 65/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1276 -
accuracy: 0.9640 - val_loss: 95.6666 - val_accuracy: 0.7879
Epoch 66/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1311 -
accuracy: 0.9611 - val_loss: 85.1676 - val_accuracy: 0.8035
Epoch 67/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1233 -
accuracy: 0.9644 - val_loss: 88.5130 - val_accuracy: 0.7967
Epoch 68/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1191 -
accuracy: 0.9642 - val_loss: 89.1576 - val_accuracy: 0.7961
Epoch 69/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1259 -
accuracy: 0.9647 - val_loss: 90.4982 - val_accuracy: 0.8030
Epoch 70/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1271 -
accuracy: 0.9641 - val_loss: 88.2230 - val_accuracy: 0.7924
Epoch 71/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1203 -
accuracy: 0.9645 - val_loss: 73.0173 - val_accuracy: 0.8172
Epoch 72/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1249 -
accuracy: 0.9632 - val_loss: 87.8738 - val_accuracy: 0.8091
Epoch 73/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1183 -
accuracy: 0.9662 - val_loss: 95.9266 - val_accuracy: 0.7864
Epoch 74/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1199 -
accuracy: 0.9655 - val_loss: 80.7140 - val_accuracy: 0.8230
Epoch 75/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1185 -
accuracy: 0.9661 - val_loss: 77.3080 - val_accuracy: 0.8149
Epoch 76/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1119 -
accuracy: 0.9675 - val_loss: 73.4771 - val_accuracy: 0.8203
Epoch 77/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1172 -
accuracy: 0.9660 - val_loss: 73.8731 - val_accuracy: 0.8123
Epoch 78/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1163 -
accuracy: 0.9663 - val_loss: 64.3321 - val_accuracy: 0.8265
Epoch 79/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1151 -
accuracy: 0.9657 - val_loss: 69.4369 - val_accuracy: 0.8156
Epoch 80/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1164 -
accuracy: 0.9666 - val_loss: 78.9606 - val_accuracy: 0.8194
```

```
Epoch 81/100
1313/1313 [==============================] - 4s 3ms/step - loss: 0.1180 -
accuracy: 0.9653 - val_loss: 76.0332 - val_accuracy: 0.8227
Epoch 82/100
1313/1313 [==============================] - 4s 3ms/step - loss: 0.1077 -
accuracy: 0.9682 - val_loss: 71.9972 - val_accuracy: 0.8234
Epoch 83/100
1313/1313 [==============================] - 4s 3ms/step - loss: 0.1140 -
accuracy: 0.9667 - val_loss: 60.9060 - val_accuracy: 0.8390
Epoch 84/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1167 -
accuracy: 0.9664 - val_loss: 78.8322 - val_accuracy: 0.8086
Epoch 85/100
1313/1313 [==============================] - 2s 2ms/step - loss: 0.1106 -
accuracy: 0.9686 - val_loss: 73.9958 - val_accuracy: 0.8084
Epoch 86/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1119 -
accuracy: 0.9673 - val_loss: 64.0418 - val_accuracy: 0.8273
Epoch 87/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1130 -
accuracy: 0.9666 - val_loss: 68.6149 - val_accuracy: 0.8157
Epoch 88/100
1313/1313 [==============================] - 2s 2ms/step - loss: 0.1063 -
accuracy: 0.9682 - val_loss: 77.6607 - val_accuracy: 0.8202
Epoch 89/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1064 -
accuracy: 0.9689 - val_loss: 77.1301 - val_accuracy: 0.8140
Epoch 90/100
1313/1313 [==============================] - 2s 2ms/step - loss: 0.1107 -
accuracy: 0.9677 - val_loss: 67.4448 - val_accuracy: 0.8303
Epoch 91/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1095 -
accuracy: 0.9680 - val_loss: 61.3893 - val_accuracy: 0.8351
Epoch 92/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1046 -
accuracy: 0.9687 - val_loss: 70.1049 - val_accuracy: 0.8228
Epoch 93/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1036 -
accuracy: 0.9707 - val_loss: 63.7225 - val_accuracy: 0.8327
Epoch 94/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1103 -
accuracy: 0.9667 - val_loss: 72.0828 - val_accuracy: 0.8240
Epoch 95/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1111 -
accuracy: 0.9674 - val_loss: 73.2828 - val_accuracy: 0.8246
Epoch 96/100
1313/1313 [==============================] - 2s 2ms/step - loss: 0.0989 -
accuracy: 0.9712 - val_loss: 72.0921 - val_accuracy: 0.8289
```

```
Epoch 97/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1094 -
accuracy: 0.9684 - val_loss: 74.0238 - val_accuracy: 0.8256
Epoch 98/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.0986 -
accuracy: 0.9710 - val_loss: 67.3066 - val_accuracy: 0.8402
Epoch 99/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1065 -
accuracy: 0.9688 - val_loss: 73.3421 - val_accuracy: 0.8275
Epoch 100/100
1313/1313 [==============================] - 3s 2ms/step - loss: 0.1015 -
accuracy: 0.9702 - val_loss: 62.6121 - val_accuracy: 0.8372
```

[30]:

```
results = model.evaluate(X_test, Y_test)
print("test loss, test acc:", results)
predictions = model.predict(X_test)
print("predictions shape:", predictions.shape)
```

```
434/434 [==============================] - 1s 1ms/step - loss: 62.6121 -
accuracy: 0.8372
test loss, test acc: [62.61207580566406, 0.8371573090553284]
WARNING:tensorflow:Model was constructed with shape (None, 28, 28) for input
KerasTensor(type_spec=TensorSpec(shape=(None, 28, 28), dtype=tf.float32,
name='flatten_input'), name='flatten_input', description="created by layer
'flatten_input'"), but it was called on an input with incompatible shape (None,
784).
predictions shape: (13860, 10)
```

[31]:

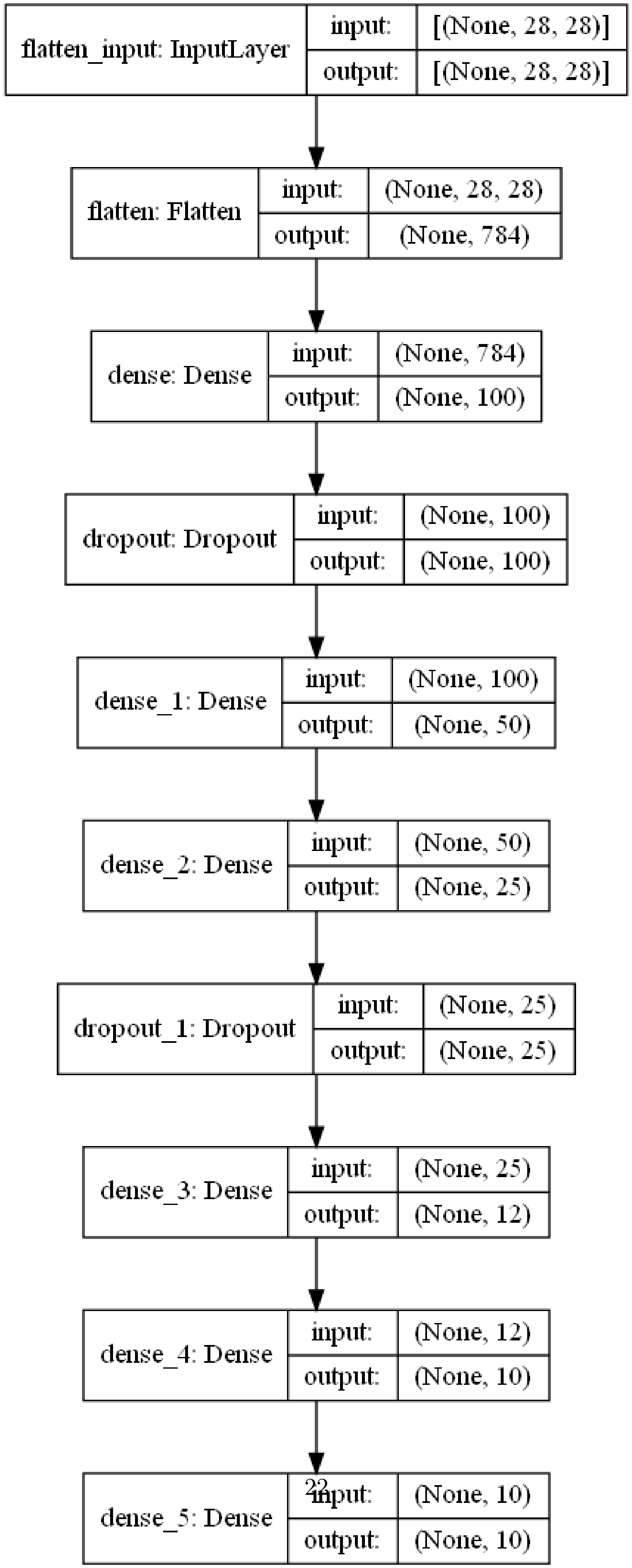
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 100)               78500
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_2 (Dense)              (None, 25)                1275
_________________________________________________________________
dropout_1 (Dropout)          (None, 25)                0
_________________________________________________________________
dense_3 (Dense)              (None, 12)                312
_________________________________________________________________
dense_4 (Dense)              (None, 10)                130
```

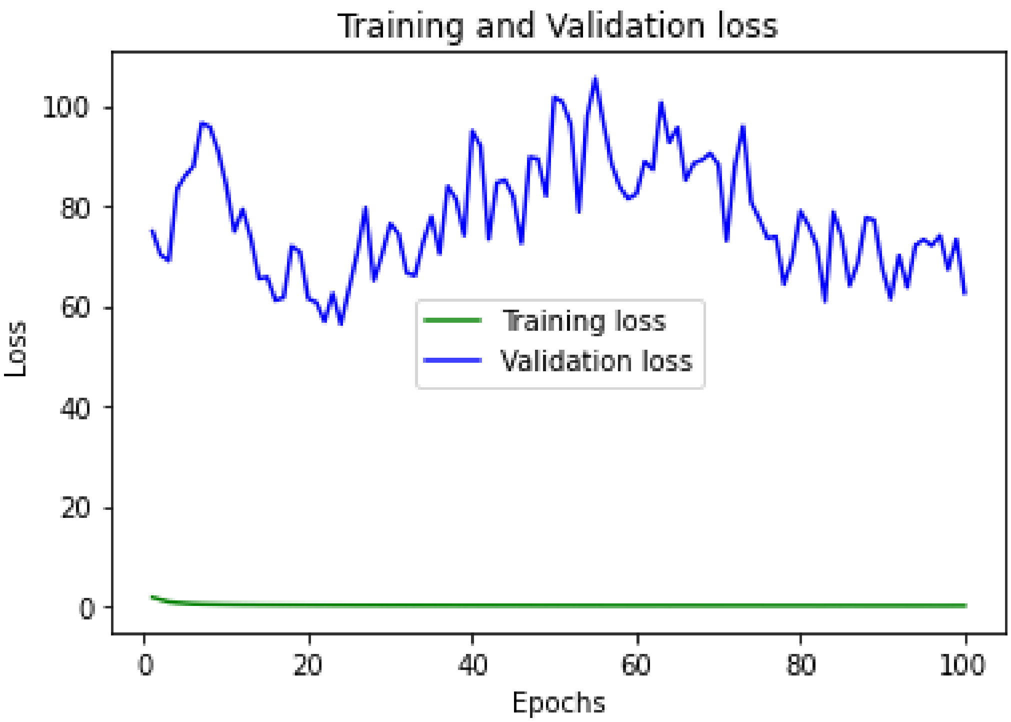
```
_________________________________________________________________
dense_5 (Dense)              (None, 10)                110
=================================================================
Total params: 85,377
Trainable params: 85,377
Non-trainable params: 0
_________________________________________________________________
```

[32]:
```
from keras.utils.vis_utils import plot_model
plot_model(model,to_file='model_plot.
 ↪png',show_shapes=True,show_layer_names=True)
```

[32]:

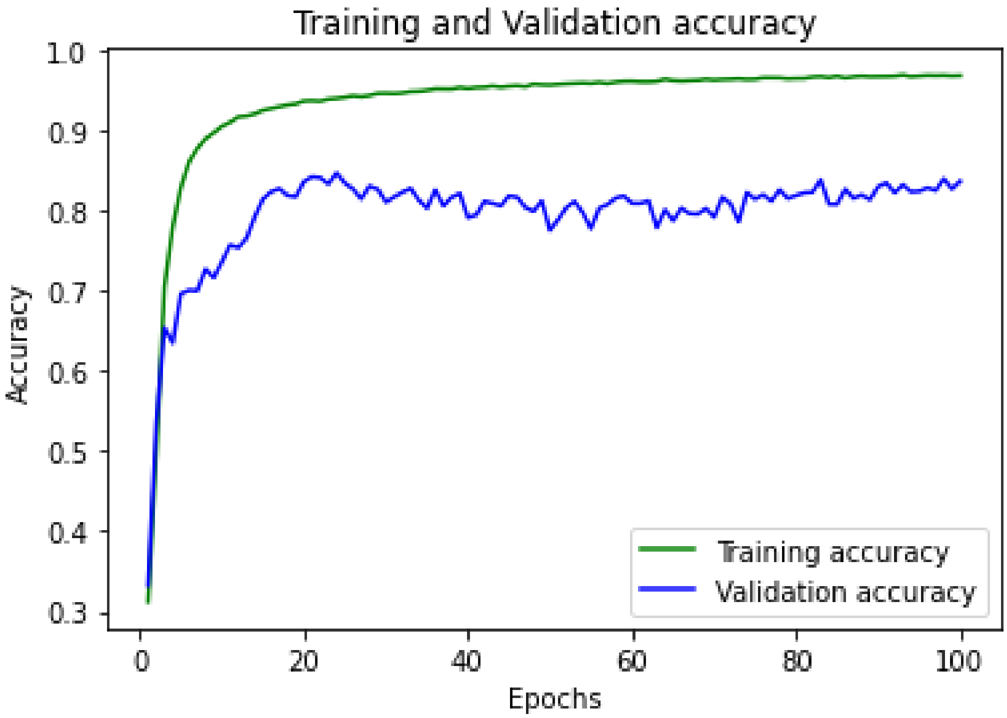

flatten_input: InputLayer
input: [(None, 28, 28)]
output: [(None, 28, 28)]
flatten: Flatten
input: (None, 28, 28)
output: (None, 784)
dense: Dense
input: (None, 784)
output: (None, 100)
dropout: Dropout
input: (None, 100)
output: (None, 100)
dense_1: Dense
input: (None, 100)
output: (None, 50)
dense_2: Dense
input: (None, 50)
output: (None, 25)
dropout_1: Dropout
input: (None, 25)
output: (None, 25)
dense_3: Dense
input: (None, 25)
output: (None, 12)
dense_4: Dense
input: (None, 12)
output: (None, 10)
dense_5: Dense
input: (None, 10)
output: (None, 10)

[33]:
```
loss_train=history.history['loss']
loss_val=history.history['val_loss']
epochs=range(1,101)
plt.plot(epochs,loss_train,'g',label='Training loss')
plt.plot(epochs,loss_val,'b',label='Validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
#plt.ylim(0,10.0)
plt.legend()
plt.show()
```



[34]:
```
loss_train=history.history['accuracy']
loss_val=history.history['val_accuracy']
epochs=range(1,101)
plt.plot(epochs,loss_train,'g',label='Training accuracy')
plt.plot(epochs,loss_val,'b',label='Validation accuracy')
plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
plt.show()
```

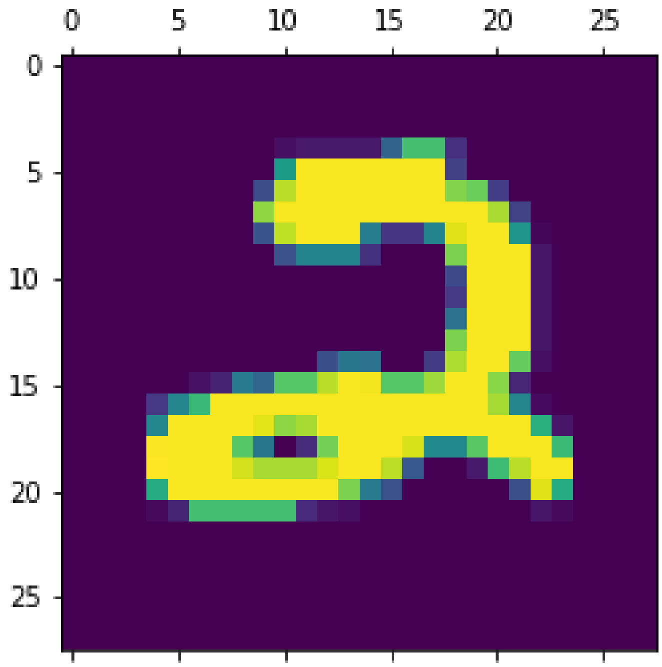


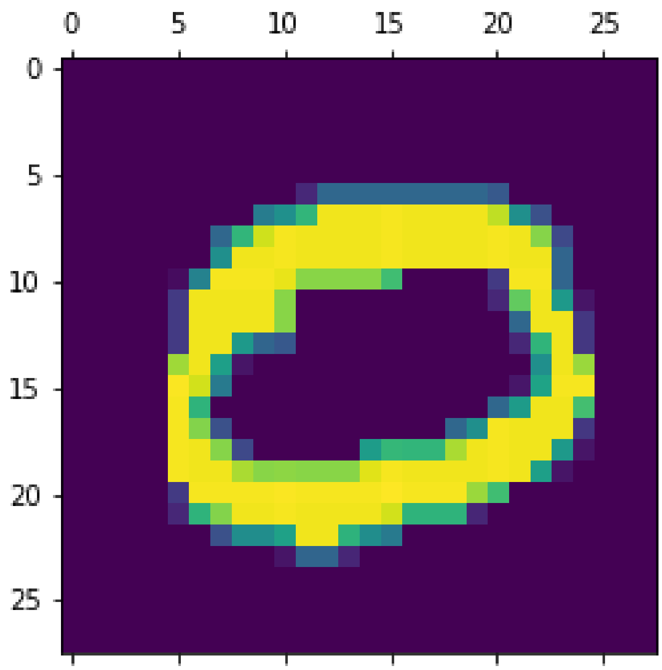
[35]:
```
Label=model.predict(df_test)
```

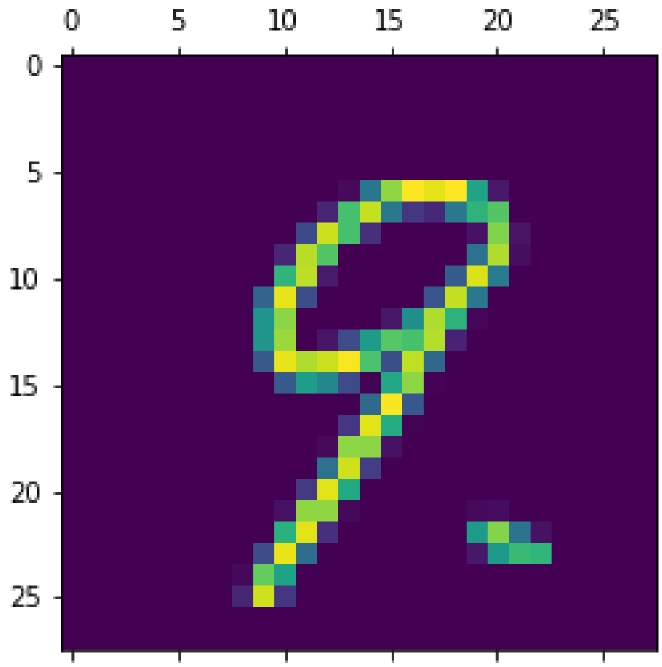
[36]:
```
Label[0]
```

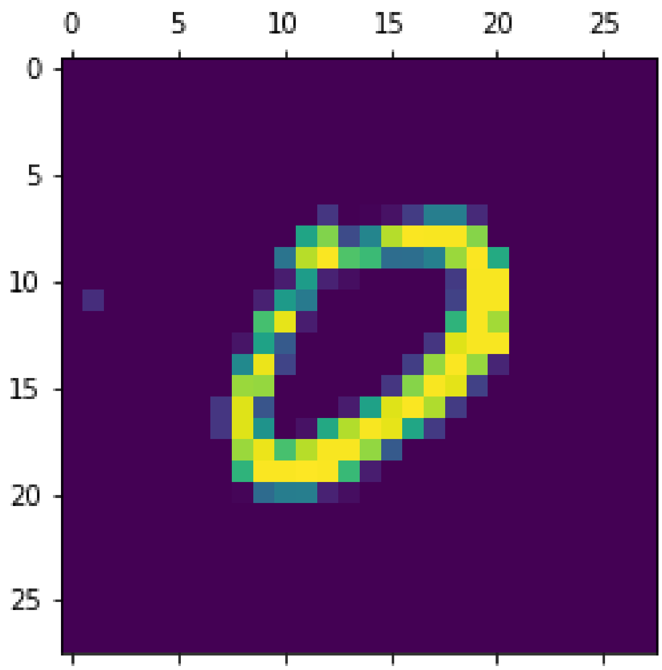
[36]:
```
array([3.4796378e-15, 3.0284216e-06, 9.9998760e-01, 9.4336083e-06,
       6.1116016e-27, 1.0965604e-13, 4.4143778e-21, 8.0405513e-09,
       1.6610380e-09, 1.0660506e-21], dtype=float32)
```

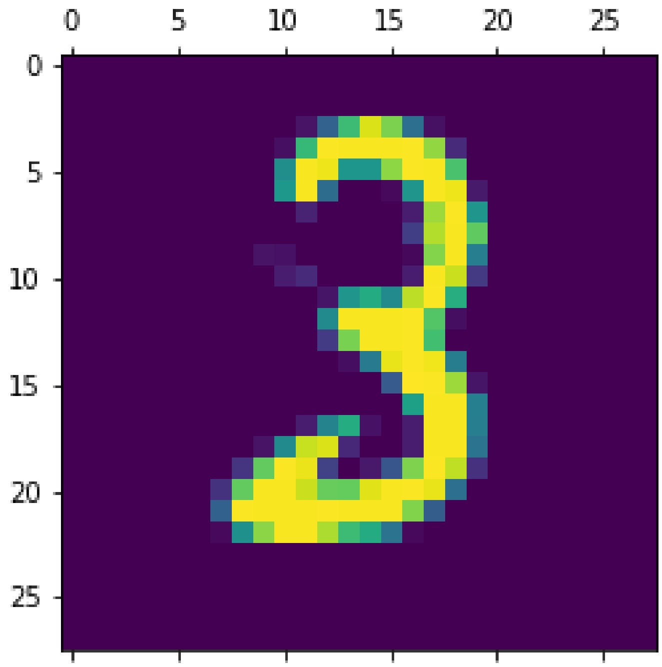
[37]:
```
for i in range(10):
    plt.matshow(df_test[i])
```

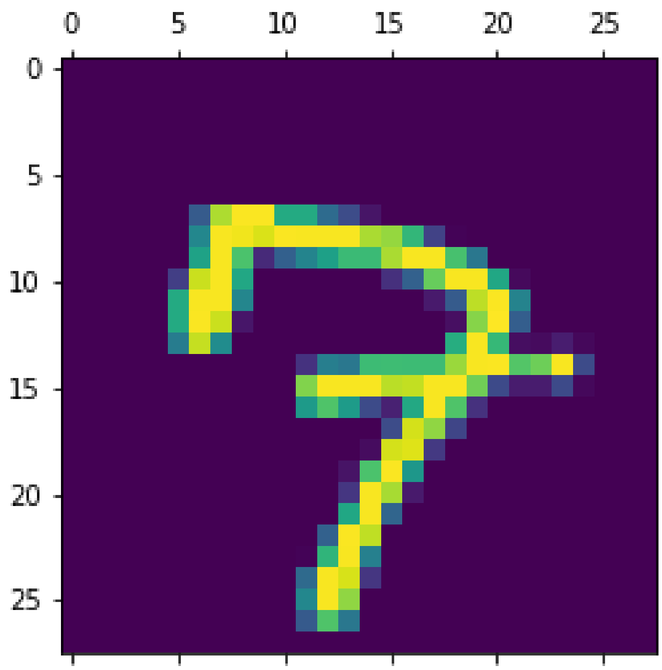

0
5
10
15
20
25

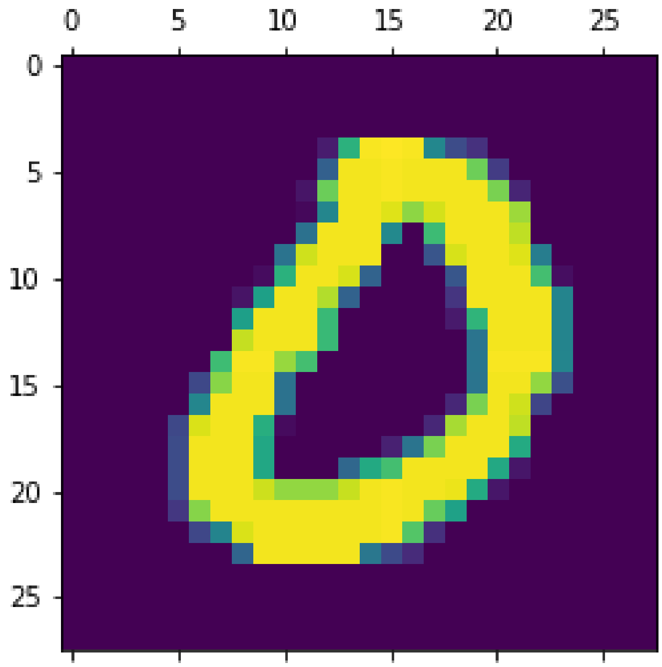

0
5
10
15
20
25

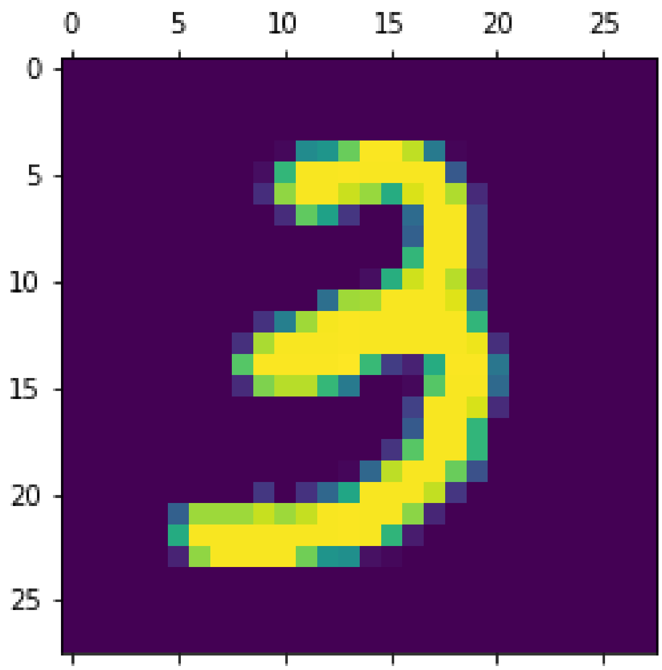

0
5
10
15
20
25
0
5
10
15
20
25

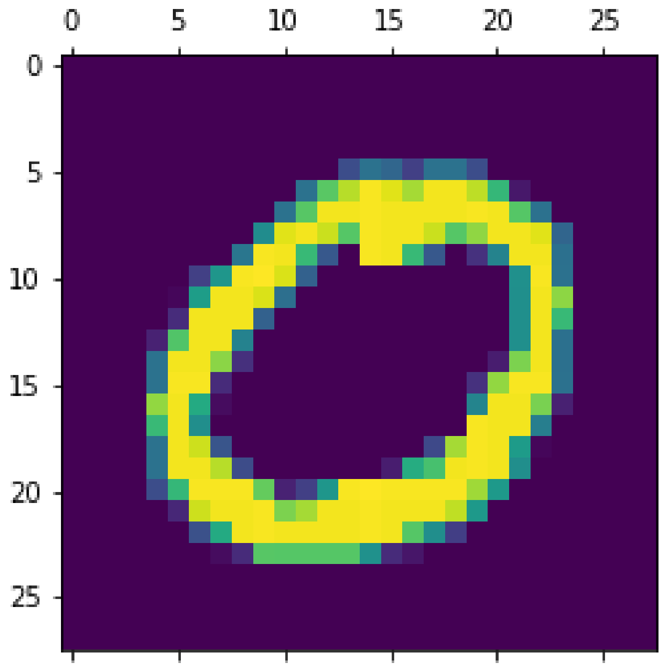

0
5
10
15
20
25
0
5
10
15
20
25

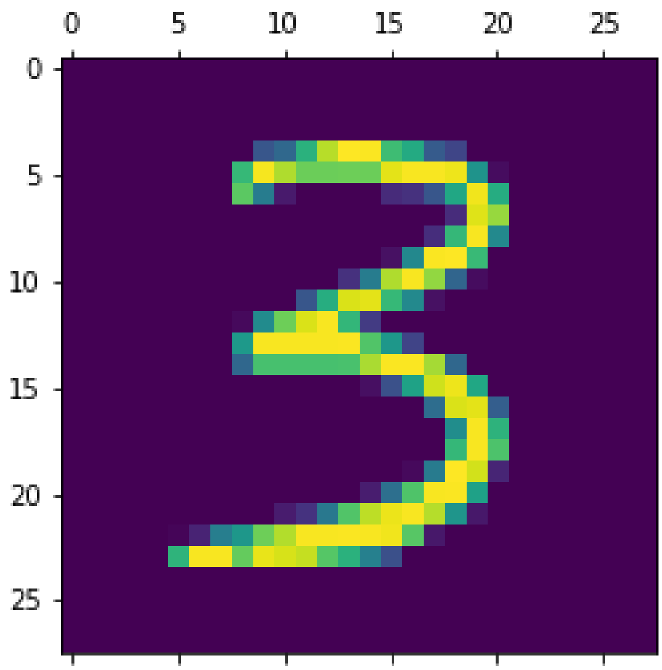

0 5 10 15 20 25
0 5 10 15 20 25


0 5 10 15 20 25
0 5 10 15 20 25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25





[38]:
```
np.argmax(Label[0])
```

[38]: 2

[39]:
```python
Label=[np.argmax(i) for i in Label]
Label[:10]
```

[39]: [2, 0, 9, 9, 3, 7, 0, 3, 0, 3]

[ ]:
```python
tsne = TSNE(n_components = 3, perplexity = 50)
projections = tsne.fit_transform(df_train.drop('label', axis = 1))
```

[ ]:
```python

```