In [ ]:

Name : Dipali Chavan
Roll No: I4139

In [ ]:

```
!pip install tensorflow
```

In [1]:

```
#importing necessary libraries
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
%matplotlib inline
```

In [2]:

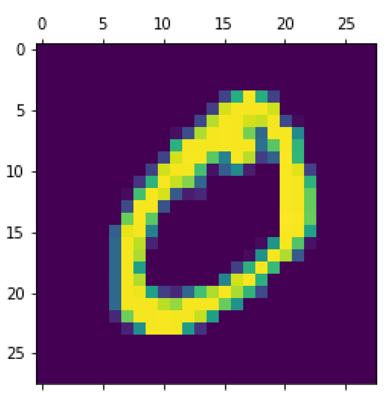
```
#import dataset and split into train and test data
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

In [3]:

```
plt.matshow(x_train[1])
```

Out[3]:

```
<matplotlib.image.AxesImage at 0x216d2a9c9d0>
```

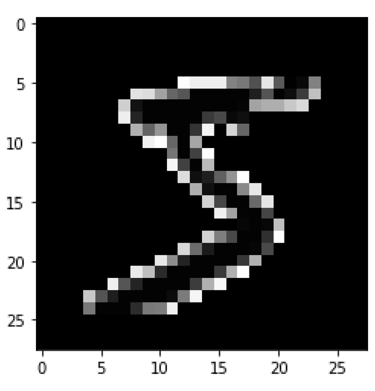


In [4]:

```
plt.imshow(-x_train[0], cmap="gray")
```

Out[4]:

```
<matplotlib.image.AxesImage at 0x216d321cac0>
```



In [5]:

```
x_train = x_train / 255
x_test = x_test / 255
```

In [6]:

```
model = keras.Sequential([
keras.layers.Flatten(input_shape=(28, 28)),
keras.layers.Dense(128, activation="relu"),
keras.layers.Dense(10, activation="softmax")
])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

In [7]:

```
model.compile(optimizer="sgd",
loss="sparse_categorical_crossentropy",
metrics=['accuracy'])
```

In [8]:

```
history=model.fit(x_train,
y_train,validation_data=(x_test,y_test),epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6614 - accuracy: 0.8326 - val_loss: 0.3559 - val_accuracy: 0.9034
Epoch 2/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3358 - accuracy: 0.9069 - val_loss: 0.2906 - val_accuracy: 0.9198
Epoch 3/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2860 - accuracy: 0.9197 - val_loss: 0.2582 - val_accuracy: 0.9290
Epoch 4/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2551 - accuracy: 0.9286 - val_loss: 0.2334 - val_accuracy: 0.9348
Epoch 5/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2325 - accuracy: 0.9351 - val_loss: 0.2150 - val_accuracy: 0.9390
Epoch 6/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2140 - accuracy: 0.9397 - val_loss: 0.2011 - val_accuracy: 0.9437
Epoch 7/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1991 - accuracy: 0.9441 - val_loss: 0.1890 - val_accuracy: 0.9472
Epoch 8/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1861 - accuracy: 0.9479 - val_loss: 0.1787 - val_accuracy: 0.9484
Epoch 9/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1747 - accuracy: 0.9509 - val_loss: 0.1690 - val_accuracy: 0.9505
Epoch 10/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1649 - accuracy: 0.9537 - val_loss: 0.1604 - val_accuracy: 0.9537
```

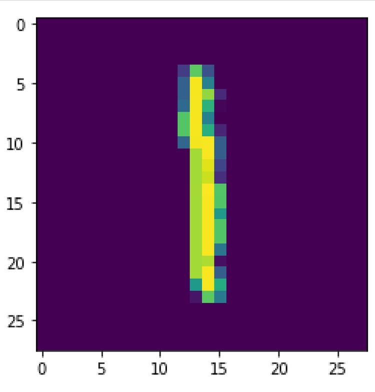
In [9]:

```
test_loss,test_acc=model.evaluate(x_test,y_test)
print("Loss=%.3f" %test_loss)
print("Accuracy=%.3f" %test_acc)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.1604 - accuracy: 0.9537
Loss=0.160
Accuracy=0.954
```

In [10]:

```
n=random.randint(0,9999)
plt.imshow(x_test[n])
plt.show()
```



In [11]:

```
x_train
```

Out[11]:

```
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.]]])
```

In [12]:

```
x_test
```

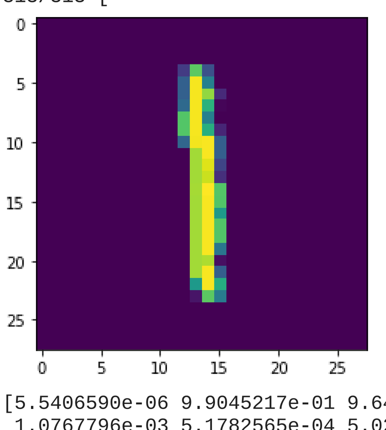
Out[12]:

```
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.]]])
```

In [13]:

```
predicted_value=model.predict(x_test)
plt.imshow(x_test[n])
plt.show()

print(predicted_value[n])
```

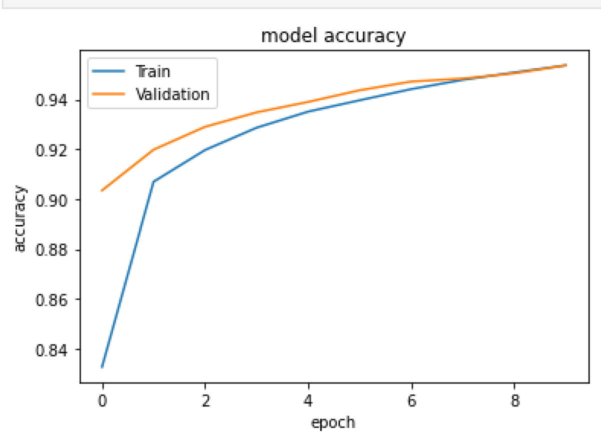
```
313/313 [==============================] - 0s 1ms/step
```



```
[5.5406590e-06 9.9045217e-01 9.6445804e-04 1.1881288e-03 5.5083026e-05
 1.0767796e-03 5.1782565e-04 5.0224096e-04 4.8863539e-03 3.5130000e-04]
```

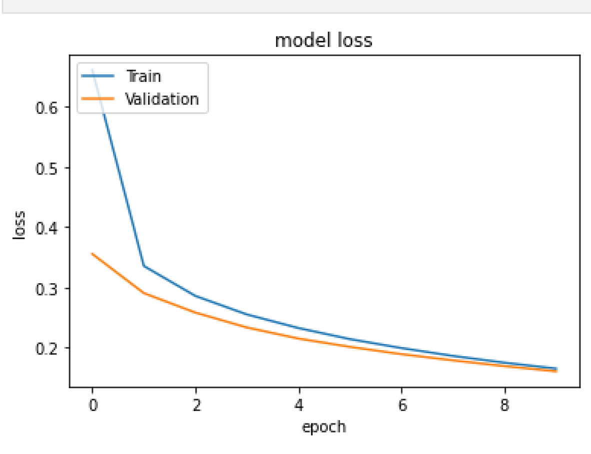
In [14]:

```
# history.history()
history.history.keys()
# dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



In [15]:

```
# history.history()
history.history.keys()
# dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



In [ ]:

In [ ]:

Name:Dipali Chavan
Roll No: I4139

In [1]:
```python
import numpy as np
import pandas as pd
import random
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Conv2D, Dense, MaxPooling2D
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist
```

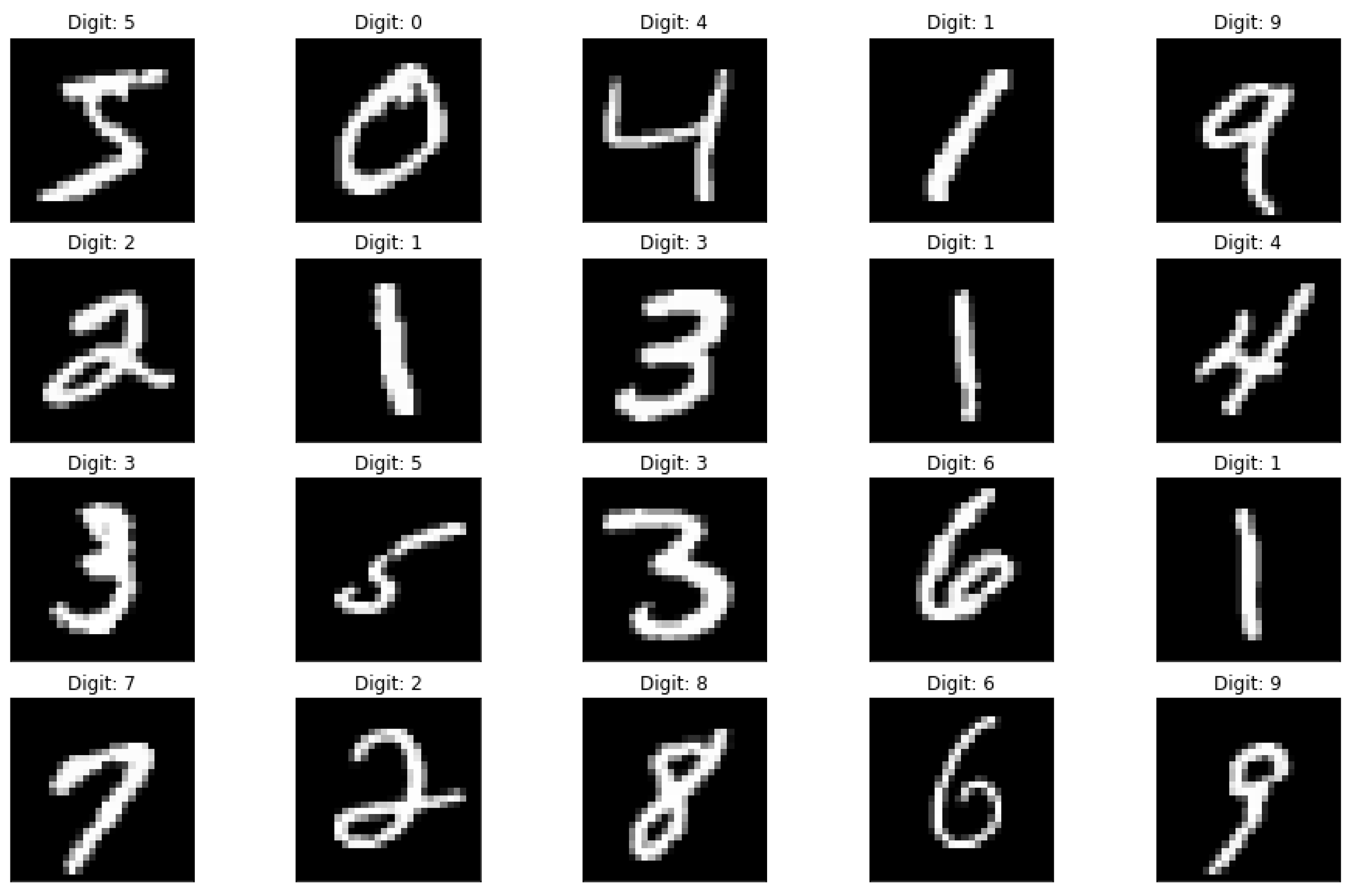
In [2]:
```python
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [3]:
```python
print(X_train.shape)
```
```
(60000, 28, 28)
```

In [4]:
```python
X_train[0].min(), X_train[0].max()
```
Out[4]:
```
(0, 255)
```

In [5]:
```python
X_train = (X_train - 0.0) / (255.0 - 0.0)
X_test = (X_test - 0.0) / (255.0 - 0.0)
X_train[0].min(), X_train[0].max()
```
Out[5]:
```
(0.0, 1.0)
```

In [6]:
```python
def plot_digit(image, digit, plt, i):
    plt.subplot(4, 5, i + 1)
    plt.imshow(image, cmap=plt.get_cmap('gray'))
    plt.title(f"Digit: {digit}")
    plt.xticks([])
    plt.yticks([])
plt.figure(figsize=(16, 10))
for i in range(20):
    plot_digit(X_train[i], y_train[i], plt, i)
plt.show()
```



In [7]:
```python
X_train = X_train.reshape((X_train.shape + (1,)))
X_test = X_test.reshape((X_test.shape + (1,)))
```

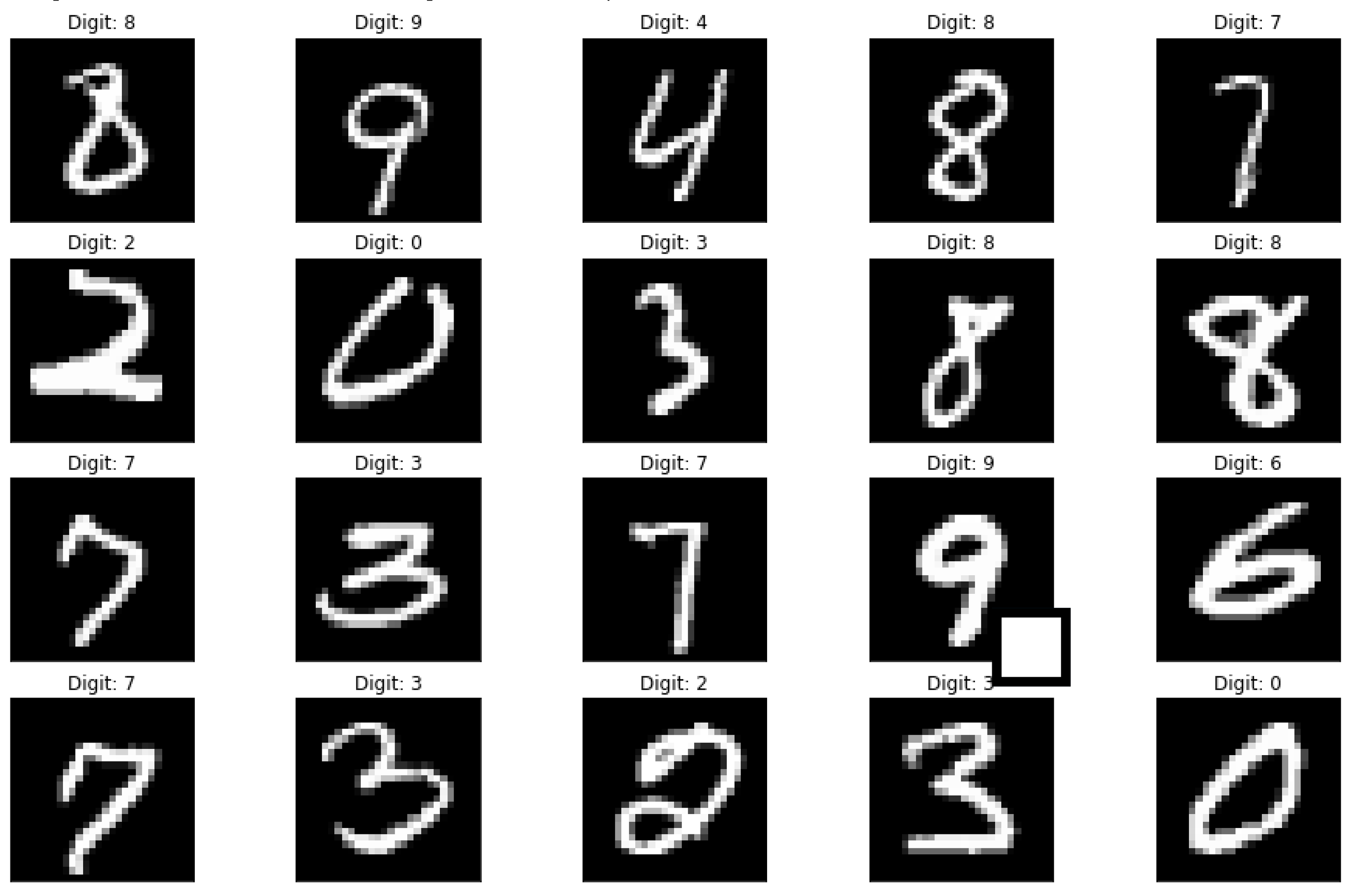
In [8]:
```python
y_train[0:20]
```
Out[8]:
```
array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4, 3, 5, 3, 6, 1, 7, 2, 8, 6, 9],
      dtype=uint8)
```

In [9]:
```python
model = Sequential([
    Conv2D(32, (3, 3), activation="relu", input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(100, activation="relu"),
    Dense(10, activation="softmax")
])
```

In [10]:
```python
optimizer = SGD(learning_rate=0.01, momentum=0.9)
model.compile(
    optimizer=optimizer,
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)
model.summary()
```
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 flatten (Flatten)           (None, 5408)              0

 dense (Dense)               (None, 100)               540900

 dense_1 (Dense)             (None, 10)                1010

=================================================================
Total params: 542,230
Trainable params: 542,230
Non-trainable params: 0
_________________________________________________________________
```

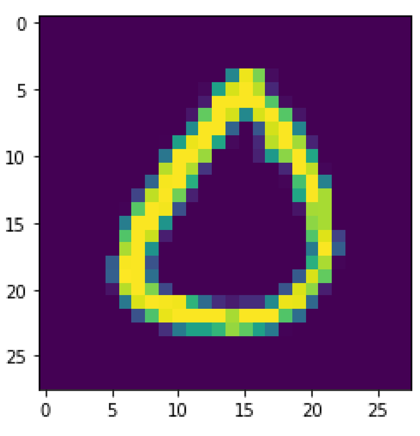
In [11]:
```python
model.fit(X_train, y_train, epochs=10, batch_size=32)
```
```
Epoch 1/10
1875/1875 [==============================] - 19s 10ms/step - loss: 0.2392 - accuracy: 0.9275
Epoch 2/10
1875/1875 [==============================] - 18s 9ms/step - loss: 0.0801 - accuracy: 0.9757
Epoch 3/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.0511 - accuracy: 0.9846
Epoch 4/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.0352 - accuracy: 0.9893
Epoch 5/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.0268 - accuracy: 0.9917
Epoch 6/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.0203 - accuracy: 0.9938
Epoch 7/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.0145 - accuracy: 0.9955
Epoch 8/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.0107 - accuracy: 0.9969
Epoch 9/10
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0092 - accuracy: 0.9975
Epoch 10/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.0066 - accuracy: 0.9983
```
Out[11]:
```
<keras.callbacks.History at 0x27b921bdfa0>
```

In [12]:
```python
plt.figure(figsize=(16, 10))
for i in range(20):
    image = random.choice(X_test).squeeze()
    digit = np.argmax(model.predict(image.reshape((1, 28, 28, 1)))[0], axis=-1)
    plot_digit(image, digit, plt, i)
plt.show()
```
```
1/1 [==============================] - 0s 105ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
```



In [13]:
```python
predictions = np.argmax(model.predict(X_test), axis=-1)
accuracy_score(y_test, predictions)
```
```
313/313 [==============================] - 1s 3ms/step
```
Out[13]:
```
0.9881
```

In [14]:
```python
n=random.randint(0,9999)
plt.imshow(X_test[n])
plt.show()
```



In [15]:
```python
predicted_value=model.predict(X_test)
print("Handwritten number in the image is= %d" %np.argmax(predicted_value[n]))
```
```
313/313 [==============================] - 1s 3ms/step
Handwritten number in the image is= 0
```

In [16]:
```python
score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0]) #Test loss: 0.0296396646054
print('Test accuracy:', score[1])
```
```
Test loss: 0.03916610777378082
Test accuracy: 0.988099992275238
```

In [17]:
```python
#The implemented CNN model is giving Loss=0.04624301567673683  and
#accuracy: 0.98720002174377744 for test mnist dataset
```

In [ ]:

Name:Dipali Chavan
Roll No:I4139

In [1]:

```python
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, recall_score, accuracy_score, precision_score

RANDOM_SEED = 2021
TEST_PCT = 0.3
LABELS = ["Normal","Fraud"]
```

In [2]:

```python
dataset = pd.read_csv("creditcard.csv")
```

In [3]:

```python
#check for any null values
print("Any nulls in the dataset",dataset.isnull().values.any())
print('-------')
print("No. of unique labels",len(dataset['Class'].unique()))
print("Label values",dataset.Class.unique())

#0 is for normal credit card transcation
#1 is for fraudulent credit card transcation
print('-------')
print("Break down of Normal and Fraud Transcations")
print(pd.value_counts(dataset['Class'],sort=True))
```

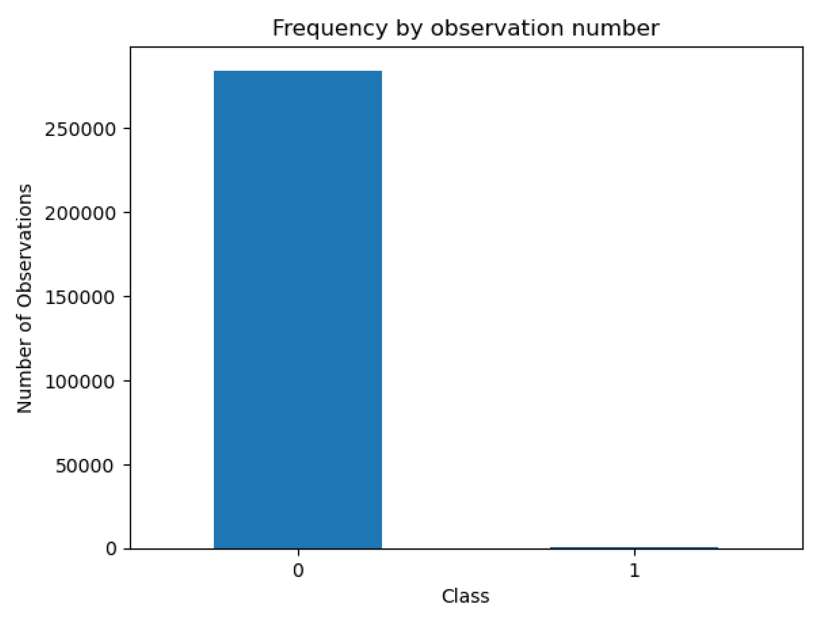
```
Any nulls in the dataset False
-------
No. of unique labels 2
Label values [0 1]
-------
Break down of Normal and Fraud Transcations
0    284315
1       492
Name: Class, dtype: int64
```

In [4]:

```python
#visualizing the imbalanced dataset
count_classes = pd.value_counts(dataset['Class'],sort=True)
count_classes.plot(kind='bar',rot=0)
plt.xticks(range(len(dataset['Class'].unique())),dataset.Class.unique())
plt.title("Frequency by observation number")
plt.xlabel("Class")
plt.ylabel("Number of Observations")
```

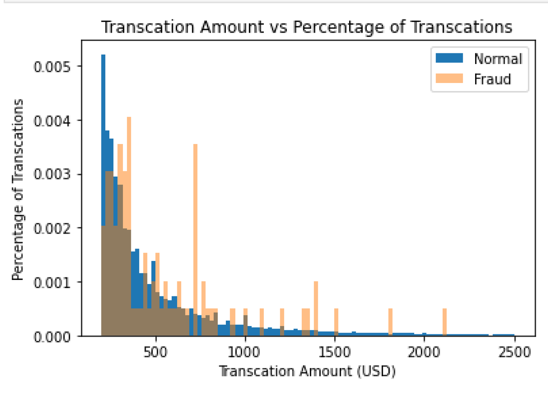
Out[4]:

```
Text(0, 0.5, 'Number of Observations')
```



In [4]:

```python
#Save the normal and fradulent transcations in seperate dataframe
normal_dataset = dataset[dataset.Class == 0]
fraud_dataset = dataset[dataset.Class == 1]

#Visualize transcation amounts for normal and fraudulent transcations
bins = np.linspace(200,2500,100)
plt.hist(normal_dataset.Amount,bins=bins,alpha=1,density=True,label='Normal')
plt.hist(fraud_dataset.Amount,bins=bins,alpha=0.5,density=True,label='Fraud')
plt.legend(loc='upper right')
plt.title("Transcation Amount vs Percentage of Transcations")
plt.xlabel("Transcation Amount (USD)")
plt.ylabel("Percentage of Transcations")
plt.show()
```



In [5]:

```python
dataset
```

Out[5]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651	0.823731	0.77	0
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.068472	-0.053527	24.79	0
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.026561	67.88	0
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.108821	0.104533	10.00	0
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.013649	217.00	0

284807 rows × 31 columns

In [6]:

```python
sc = StandardScaler()
dataset['Time'] = sc.fit_transform(dataset['Time'].values.reshape(-1,1))
dataset['Amount'] = sc.fit_transform(dataset['Amount'].values.reshape(-1,1))
```

In [7]:

```python
raw_data = dataset.values
#The last element contains if the transcation is normal which is represented by 0 and if fraud then 1
labels = raw_data[:,-1]

#The other data points are the electrocadriogram data
data = raw_data[:,0:-1]

train_data,test_data,train_labels,test_labels = train_test_split(data,labels,test_size = 0.2,random_state =2021)
```

In [8]:

```python
min_val = tf.reduce_min(train_data)
max_val = tf.reduce_max(train_data)

train_data = (train_data - min_val) / (max_val - min_val)
test_data = (test_data - min_val) / (max_val - min_val)

train_data = tf.cast(train_data,tf.float32)
test_data = tf.cast(test_data,tf.float32)
```

In [9]:

```python
train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)

#Creating normal and fraud datasets
normal_train_data = train_data[~train_labels]
normal_test_data = test_data[~test_labels]

fraud_train_data = train_data[train_labels]
fraud_test_data = test_data[test_labels]
print("No. of records in Fraud Train Data=",len(fraud_train_data))
print("No. of records in Normal Train Data=",len(normal_train_data))
print("No. of records in Fraud Test Data=",len(fraud_test_data))
print("No. of records in Normal Test Data=",len(normal_test_data))
```

```
No. of records in Fraud Train Data= 389
No. of records in Normal Train Data= 227456
No. of records in Fraud Test Data= 103
No. of records in Normal Test Data= 56859
```

In [10]:

```python
nb_epoch = 50
batch_size = 64
input_dim = normal_train_data.shape[1]
#num of columns,30
encoding_dim = 14
hidden_dim1 = int(encoding_dim / 2)
hidden_dim2 = 4
learning_rate = 1e-7
```

In [11]:

```python
#input layer
input_layer = tf.keras.layers.Input(shape=(input_dim,))

#Encoder
encoder = tf.keras.layers.Dense(encoding_dim,activation="tanh",activity_regularizer = tf.keras.regularizers.l2(learning_rate))(input_layer)
encoder = tf.keras.layers.Dropout(0.2)(encoder)
encoder = tf.keras.layers.Dense(hidden_dim1,activation='relu')(encoder)
encoder = tf.keras.layers.Dense(hidden_dim2,activation=tf.nn.leaky_relu)(encoder)

#Decoder
decoder = tf.keras.layers.Dense(hidden_dim1,activation='relu')(encoder)
decoder = tf.keras.layers.Dropout(0.2)(decoder)
decoder = tf.keras.layers.Dense(encoding_dim,activation='relu')(decoder)
decoder = tf.keras.layers.Dense(input_dim,activation='tanh')(decoder)

#Autoencoder
autoencoder = tf.keras.Model(inputs = input_layer,outputs = decoder)
autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 30)]              0

 dense (Dense)               (None, 14)                434

 dropout (Dropout)           (None, 14)                0

 dense_1 (Dense)             (None, 7)                 105

 dense_2 (Dense)             (None, 4)                 32

 dense_3 (Dense)             (None, 7)                 35

 dropout_1 (Dropout)         (None, 7)                 0

 dense_4 (Dense)             (None, 14)                112

 dense_5 (Dense)             (None, 30)                450

=================================================================
Total params: 1,168
Trainable params: 1,168
Non-trainable params: 0
_________________________________________________________________
```

In [12]:

```python
cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.h5",mode='min',monitor='val_loss',verbose=2,save_best_only=True)
#Define our early stopping
early_stop = tf.keras.callbacks.EarlyStopping(
                monitor='val_loss',
                min_delta=0.0001,
                patience=10,
                verbose=11,
                mode='min',
                restore_best_weights=True
)
```

In [13]:

```python
autoencoder.compile(metrics=['accuracy'],loss= 'mean_squared_error',optimizer='adam')
```

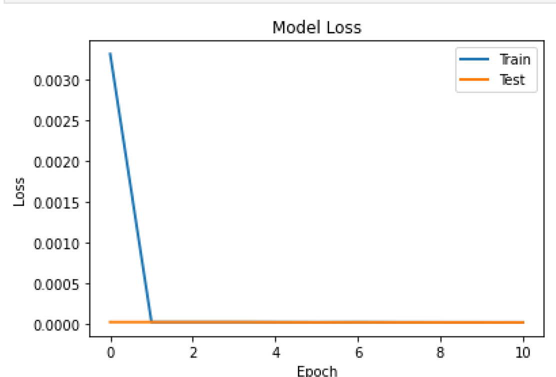
In [14]:

```python
history = autoencoder.fit(normal_train_data,normal_train_data,epochs = nb_epoch,
                          batch_size = batch_size,shuffle = True,
                          validation_data = (test_data,test_data),
                          verbose=1,
                          callbacks = [cp,early_stop]).history
```

```
Epoch 1/50
3543/3554 [============================>.] - ETA: 0s - loss: 0.0033 - accuracy: 0.0372
Epoch 1: val_loss improved from inf to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 11s 2ms/step - loss: 0.0033 - accuracy: 0.0372 - val_loss: 2.0179e-05 - val_accuracy: 0.0343
Epoch 2/50
3545/3554 [============================>.] - ETA: 0s - loss: 1.9583e-05 - accuracy: 0.0609
Epoch 2: val_loss did not improve from 0.00002
3554/3554 [==============================] - 6s 2ms/step - loss: 1.9599e-05 - accuracy: 0.0609 - val_loss: 2.0190e-05 - val_accuracy: 0.0078
Epoch 3/50
3517/3554 [============================>.] - ETA: 0s - loss: 1.9562e-05 - accuracy: 0.0619
Epoch 3: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 7s 2ms/step - loss: 1.9580e-05 - accuracy: 0.0619 - val_loss: 2.0025e-05 - val_accuracy: 0.0420
Epoch 4/50
3527/3554 [============================>.] - ETA: 0s - loss: 1.9545e-05 - accuracy: 0.0599
Epoch 4: val_loss did not improve from 0.00002
3554/3554 [==============================] - 6s 2ms/step - loss: 1.9530e-05 - accuracy: 0.0601 - val_loss: 2.0277e-05 - val_accuracy: 0.2168
Epoch 5/50
3549/3554 [============================>.] - ETA: 0s - loss: 1.8826e-05 - accuracy: 0.1758
Epoch 5: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 6s 2ms/step - loss: 1.8831e-05 - accuracy: 0.1759 - val_loss: 1.8344e-05 - val_accuracy: 0.2184
Epoch 6/50
3539/3554 [============================>.] - ETA: 0s - loss: 1.7526e-05 - accuracy: 0.2362
Epoch 6: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 8s 2ms/step - loss: 1.7518e-05 - accuracy: 0.2363 - val_loss: 1.7095e-05 - val_accuracy: 0.3538
Epoch 7/50
3516/3554 [============================>.] - ETA: 0s - loss: 1.8826e-05 - accuracy: 0.1125
Epoch 7: val_loss did not improve from 0.00002
3554/3554 [==============================] - 5s 2ms/step - loss: 1.8813e-05 - accuracy: 0.1137 - val_loss: 1.7990e-05 - val_accuracy: 0.2041
Epoch 8/50
3529/3554 [============================>.] - ETA: 0s - loss: 1.7206e-05 - accuracy: 0.2141
Epoch 8: val_loss did not improve from 0.00002
3554/3554 [==============================] - 5s 1ms/step - loss: 1.7328e-05 - accuracy: 0.2143 - val_loss: 1.7153e-05 - val_accuracy: 0.2781
Epoch 9/50
3518/3554 [============================>.] - ETA: 0s - loss: 1.6837e-05 - accuracy: 0.2481
Epoch 9: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 6s 2ms/step - loss: 1.6825e-05 - accuracy: 0.2482 - val_loss: 1.6811e-05 - val_accuracy: 0.3518
Epoch 10/50
3516/3554 [============================>.] - ETA: 0s - loss: 1.6692e-05 - accuracy: 0.2509
Epoch 10: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 6s 2ms/step - loss: 1.6682e-05 - accuracy: 0.2510 - val_loss: 1.6469e-05 - val_accuracy: 0.3492
Epoch 11/50
3526/3554 [============================>.] - ETA: 0s - loss: 1.6569e-05 - accuracy: 0.2484
Epoch 11: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5
Restoring model weights from the end of the best epoch: 1.
3554/3554 [==============================] - 7s 2ms/step - loss: 1.6561e-05 - accuracy: 0.2484 - val_loss: 1.6237e-05 - val_accuracy: 0.2865
Epoch 11: early stopping
```

In [15]:

```python
plt.plot(history['loss'],linewidth = 2,label = 'Train')
plt.plot(history['val_loss'],linewidth = 2,label = 'Test')
plt.legend(loc='upper right')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')

#plt.ylim(ymin=0.70,ymax=1)

plt.show()
```



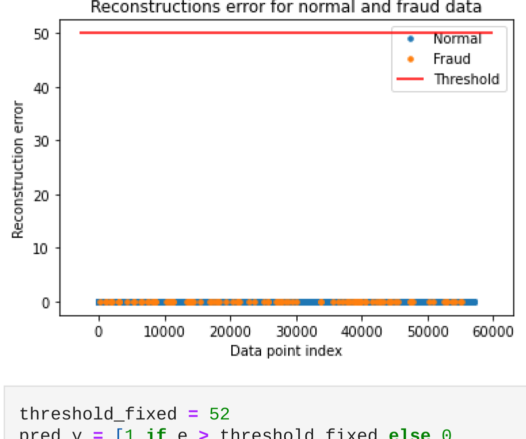
In [16]:

```python
test_x_predictions = autoencoder.predict(test_data)
mse = np.mean(np.power(test_data - test_x_predictions, 2),axis = 1)
error_df = pd.DataFrame({'Reconstruction_error':mse,
                         'True_class':test_labels})
```

```
1781/1781 [==============================] - 2s 865us/step
```

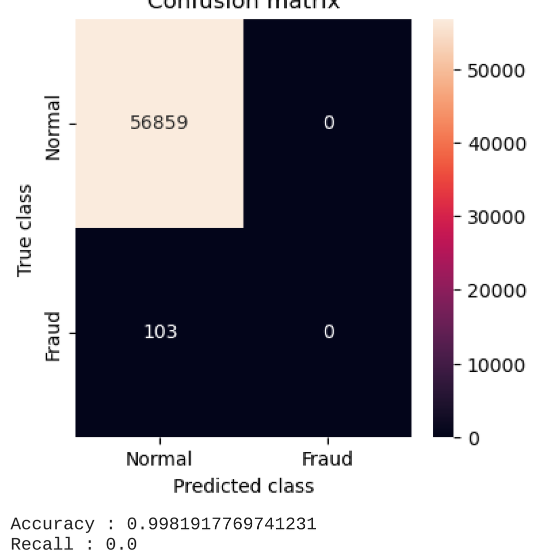
In [17]:

```python
threshold_fixed = 50
groups = error_df.groupby('True_class')
fig,ax = plt.subplots()

for name,group in groups:
        ax.plot(group.index,group.Reconstruction_error,marker='o',ms = 3.5,linestyle='',
                label = "Fraud" if  name==1 else "Normal")
ax.hlines(threshold_fixed,ax.get_xlim()[0],ax.get_xlim()[1],colors="r",zorder=100,label="Threshold")
ax.legend()
plt.title("Reconstructions error for normal and fraud data")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show()
```



In [19]:

```python
threshold_fixed = 52
pred_y = [1 if e > threshold_fixed else 0
          for e in
        error_df.Reconstruction_error.values]
error_df['pred'] = pred_y
conf_matrix = confusion_matrix(error_df.True_class,pred_y)

plt.figure(figsize = (4,4))
sns.heatmap(conf_matrix,xticklabels = LABELS,yticklabels = LABELS,annot = True,fmt="d")
plt.title("Confusion matrix")
plt.ylabel("True class")
plt.xlabel("Predicted class")
plt.show()

#Print Accuracy,Precision and Recall
print("Accuracy :",accuracy_score(error_df['True_class'],error_df['pred']))
print("Recall :",recall_score(error_df['True_class'],error_df['pred']))
print("Precision :",precision_score(error_df['True_class'],error_df['pred']))
```




```
Accuracy : 0.9981917769741231
Recall : 0.0
Precision : 0.0
C:\Users\Manish\.conda\envs\tensorflow\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no pr
edicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [ ]:

Name:Dipali Chavan
Roll No : I4139

In [1]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib as mpl
import matplotlib.pylab as pylab
import numpy as np
%matplotlib inline
```

In [2]:

```
#Data Prepration
import re
```

In [3]:

```
sentences = """We are about to study the idea of a computational process.
Computational processes are abstract beings that inhabit computers.
As they evolve, processes manipulate other abstract things called data.
The evolution of a process is directed by a pattern of rules
called a program. People create programs to direct processes. In effect,
we conjure the spirits of the computer with our spells."""
```

Clean Data

In [4]:

```
# remove special characters
sentences = re.sub('[^A-Za-z0-9]+', ' ', sentences)

# remove 1 letter words
sentences = re.sub(r'(?:^| )\w(?:$| )', ' ', sentences).strip()

# lower all characters
sentences = sentences.lower()
```

Vocabulary

In [5]:

```
words = sentences.split()
vocab = set(words)
```

In [6]:

```
vocab_size = len(vocab)
embed_dim = 10
context_size = 2
```

Implementation

In [7]:

```
word_to_ix = {word: i for i, word in enumerate(vocab)}
ix_to_word = {i: word for i, word in enumerate(vocab)}
```

Data bags

In [8]:

```
# data - [(context), target]

data = []
for i in range(2, len(words) - 2):
    [words[i - 2], words[i - 1], words[i + 1], words[i + 2]]
```

Loading [MathJax]/extensions/Safe.js

```
    target = words[i]
    data.append((context, target))
print(data[:5])
```

```
[(['we', 'are', 'to', 'study'], 'about'), (['are', 'about', 'study', 'the'], 'to'), (['abo
ut', 'to', 'the', 'idea'], 'study'), (['to', 'study', 'idea', 'of'], 'the'), (['study', 't
he', 'of', 'computational'], 'idea')]
```

Embeddings

In [9]:

```
embeddings =  np.random.random_sample((vocab_size, embed_dim))
```

Linear Model

In [10]:

```
def linear(m, theta):
    w = theta
    return m.dot(w)
```

Log softmax + NLLloss = Cross Entropy

In [11]:

```
def log_softmax(x):
    e_x = np.exp(x - np.max(x))
    return np.log(e_x / e_x.sum())
```

In [12]:

```
def NLLLoss(logs, targets):
    out = logs[range(len(targets)), targets]
    return -out.sum()/len(out)
```

In [13]:

```
def log_softmax_crossentropy_with_logits(logits,target):

    out = np.zeros_like(logits)
    out[np.arange(len(logits)),target] = 1

    softmax = np.exp(logits) / np.exp(logits).sum(axis=-1,keepdims=True)

    return (- out + softmax) / logits.shape[0]
```

Forward function

In [14]:

```
def forward(context_idxs, theta):
    m = embeddings[context_idxs].reshape(1, -1)
    n = linear(m, theta)
    o = log_softmax(n)

    return m, n, o
```

Backward function

In [15]:

```
def backward(preds, theta, target_idxs):
    m, n, o = preds

    dlog = log_softmax_crossentropy_with_logits(n, target_idxs)
    dw = m.T.dot(dlog)

    return dw
```

Optimize function

In [16]:
```
def optimize(theta, grad, lr=0.03):
    theta -= grad * lr
    return theta
```

Training

In [17]:
```
#Genrate training data

theta = np.random.uniform(-1, 1, (2 * context_size * embed_dim, vocab_size))
```

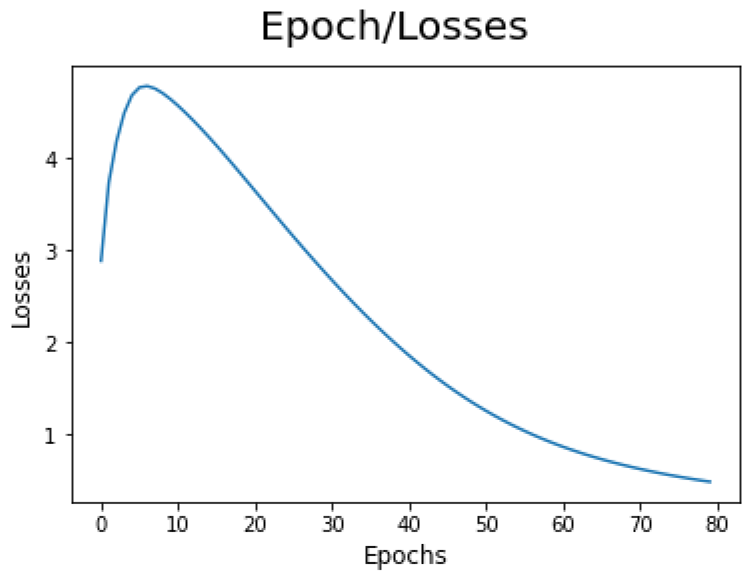
In [18]:
```
epoch_losses = {}

for epoch in range(80):

    losses =  []

    for context, target in data:
        context_idxs = np.array([word_to_ix[w] for w in context])
        preds = forward(context_idxs, theta)

        target_idxs = np.array([word_to_ix[target]])
        loss = NLLLoss(preds[-1], target_idxs)

        losses.append(loss)

        grad = backward(preds, theta, target_idxs)
        theta = optimize(theta, grad, lr=0.03)


    epoch_losses[epoch] = losses
```

Analyze

Plot loss/epoch

In [19]:
```
ix = np.arange(0,80)

fig = plt.figure()
fig.suptitle('Epoch/Losses', fontsize=20)
plt.plot(ix,[epoch_losses[i][0] for i in ix])
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Losses', fontsize=12)
```

Out[19]:
```
Text(0, 0.5, 'Losses')
```



Predict function

In [20]:
```
def predict(words):
    context_idxs = np.array([word_to_ix[w] for w in words])
    preds = forward(context_idxs, theta)
    word = ix_to_word[np.argmax(preds[-1])]

    return word
```

In [21]:
```
# (['we', 'are', 'to', 'study'], 'about')
predict(['we', 'are', 'to', 'study'])
```

Out[21]: 'about'

Accuracy

In [22]:
```
def accuracy():
    wrong = 0

    for context, target in data:
        if(predict(context) != target):
            wrong += 1

    return (1 - (wrong / len(data)))
```

In [23]:
```
accuracy()
```

Out[23]: 1.0

In [24]:
```
predict(['processes', 'manipulate', 'things', 'study'])
```

Out[24]: 'effect'

In [ ]:

In [ ]:

Name : Dipali Chavan
Roll No : I4139

In [1]:

```
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
from keras.applications.vgg16 import VGG16
# load an image from file
image = load_img('download.jpg', target_size=(224, 224))
# convert the image pixels to a numpy array
image = img_to_array(image)
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# prepare the image for the VGG model
image = preprocess_input(image)
# load the model
model = VGG16()
# predict the probability across all output classes
yhat = model.predict(image)
# convert the probabilities to class labels
label = decode_predictions(yhat)
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/v
gg16_weights_tf_dim_ordering_tf_kernels.h5
553467096/553467096 [==============================] - 151s 0us/step
1/1 [==============================] - 2s 2s/step
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet
_class_index.json
35363/35363 [==============================] - 0s 0us/step
castle (34.03%)
```

In [2]:

```
# load an image from file
image = load_img('download2.png', target_size=(224, 224))
# convert the image pixels to a numpy array
image = img_to_array(image)
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# prepare the image for the VGG model
image = preprocess_input(image)
# load the model
model = VGG16()
# predict the probability across all output classes
yhat = model.predict(image)
# convert the probabilities to class labels
label = decode_predictions(yhat)
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

```
1/1 [==============================] - 1s 672ms/step
valley (44.85%)
```

In [3]:

```
# load an image from file
img('download3.jpg', target_size=(224, 224))
```

```
# convert the image pixels to a numpy array
image = img_to_array(image)
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# prepare the image for the VGG model
image = preprocess_input(image)
# load the model
model = VGG16()
# predict the probability across all output classes
yhat = model.predict(image)
# convert the probabilities to class labels
label = decode_predictions(yhat)
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

```
1/1 [==============================] - 2s 2s/step
golden_retriever (84.78%)
```

In [ ]:

```

```

Loading [MathJax]/extensions/Safe.js