[2]:
```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense , Conv2D ,Dropout,Flatten,MaxPooling2D
import matplotlib.pyplot as plt
import numpy as np
```

[3]:
```
mnist = tf.keras.datasets.mnist
(x_train,y_train),(x_test,y_test) = mnist.load_data()
input_shape = (28,28,1)
```

[4]:
```
x_train = x_train.reshape(x_train.shape[0],28,28,1)
x_test = x_test.reshape(x_test.shape[0],28,28,1)
```

[5]:
```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
```

[6]:
```
x_train = x_train / 255;
x_test = x_test / 255;
print("shape of Training :",x_train.shape)
print("shape of Testing :",x_test.shape)
```

```
shape of Training : (60000, 28, 28, 1)
shape of Testing : (10000, 28, 28, 1)
```

Defining the Model

[7]:
```
model = Sequential([
    Conv2D(28,kernel_size=(3,3), input_shape = input_shape),
    MaxPooling2D(pool_size = (2,2)),
    Flatten(),
    Dense(200,activation = "relu"),
    Dropout(0.3),
    Dense(10,activation = "softmax")
])
```

```
C:\Users\ASUS\AppData\Roaming\Python\Python312\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[8]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 28)	280
max_pooling2d (MaxPooling2D)	(None, 13, 13, 28)	0
flatten (Flatten)	(None, 4732)	0
dense (Dense)	(None, 200)	946,600
dropout (Dropout)	(None, 200)	0
dense_1 (Dense)	(None, 10)	2,010

Total params: 948,890 (3.62 MB)

Trainable params: 948,890 (3.62 MB)

Non-trainable params: 0 (0.00 B)

[9]:
```
model.compile(optimizer="adam",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
```

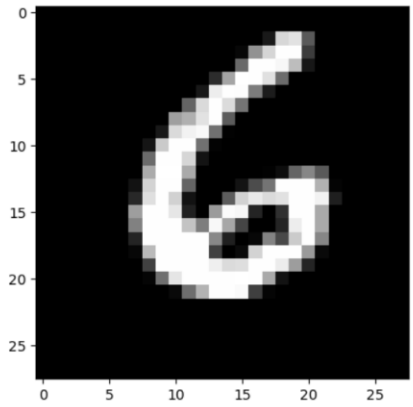
[15]:
```
history = model.fit(x_train,y_train,epochs=5)
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 14s 8ms/step - accuracy: 0.9897 - loss: 0.0304
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - accuracy: 0.9917 - loss: 0.0245
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - accuracy: 0.9920 - loss: 0.0236
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - accuracy: 0.9933 - loss: 0.0194
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - accuracy: 0.9937 - loss: 0.0194
```

[16]:
```
test_loss , test_acc = model.evaluate(x_test,y_test)
print("loss = %.3f"%test_loss)
print("accuracy = %.3f"%test_acc)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9789 - loss: 0.0897
loss = 0.070
accuracy = 0.984
```

[17]:
```
image = x_train[90]
plt.imshow(np.squeeze(image),cmap="gray")
plt.show()
```



[18]:
```
image = image.reshape(1,image.shape[0],image.shape[1],image.shape[2])
predict_model = model.predict([image])
print("predicted class : {} ".format(np.argmax(predict_model)))
```

```
## upto this is enough
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
predicted class : 6
```

[21]:
```
plt.plot(history.history['accuracy'])
# plt.plot(history.history['loss'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train','Validation'],loc = 'upper right')
plt.show()
```