Install Required Libraries

[1]:
```
#!pip install numpy
```

[2]:
```
#!pip install pandas opencv-python keras tensorflow
```

[3]:
```
#!pip install pandas opencv-python keras tensorflow
```

Task 1: Import libraries

[4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from keras.models import load_model
import cv2
```

Task 2: Import Dataset:

[19]:
```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

[20]:
```
x_train.shape
```

[20]:
```
(60000, 28, 28)
```

[21]:
```
print(x_test.shape, y_test.shape)
```

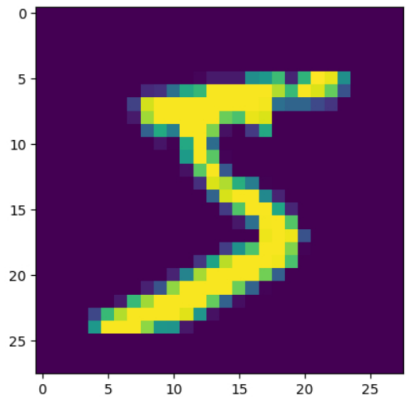
```
(10000, 28, 28) (10000,)
```

Let's test any data sample to check if it is corect data or not

[22]:
```
print("label=",y_train[0])
plt.imshow(x_train[0])
```

```
label= 5
```

[22]:
```
<matplotlib.image.AxesImage at 0x26c25b723b0>
```



Task 2: Data Preprocessing

[23]:
```
print(x_train.max(), x_train.min(), x_test.max(), x_test.min())
```

```
255 0 255 0
```

- The training data ranges between 0-255, now we will rescale the feature values to be in the range [0, 1]

[24]:
```
x_train_processed, x_test_processed = x_train / 255.0, x_test / 255.0
```

[25]:
```
print(x_train_processed.max(), x_train_processed.min(), x_test_processed.max(), x_test_processed.min())
```

```
1.0 0.0 1.0 0.0
```

Task 3: Build a Classifier using MLP (Multi Layer perceptron)

[13]:
```
from keras.models import Sequential
from keras.layers import Convolution2D, MaxPooling2D
from keras.layers import Flatten, Dense,Dropout
from keras.optimizers import Adam
```

[26]:
```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10, activation='softmax')
])
```

[28]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 784)               0

 dense_2 (Dense)             (None, 128)               100480
```

```
dropout_1 (Dropout)         (None, 128)               0

dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 101770 (397.54 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Task 4: Compile the Model

[32]:
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Task 5: Train and Test the model.

[34]:
```
history = model.fit(x_train_processed, y_train, epochs=5)
history
```

```
Epoch 1/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0660 - accuracy: 0.9792
Epoch 2/5
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0576 - accuracy: 0.9817
Epoch 3/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.0539 - accuracy: 0.9820
Epoch 4/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0475 - accuracy: 0.9847
Epoch 5/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0443 - accuracy: 0.9852
```

[34]:
```
<keras.src.callbacks.History at 0x26c21565d20>
```

Evaluation

[37]:
```
test_loss, test_acc = model.evaluate(x_test_processed, y_test)
```

```
313/313 [==============================] - 2s 5ms/step - loss: 0.0767 - accuracy: 0.9780
```

[38]:
```
print("Test Loss: ", test_loss)
print("Test Accuracy: ", test_acc)
```

```
Test Loss:  0.07668498158454895
Test Accuracy:  0.9779999852180481
```

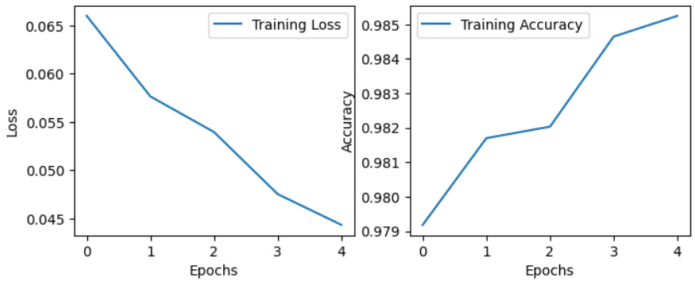
- Access Loss and Accuracy details from the training history

[41]:
```
training_loss = history.history['loss']
training_accuracy = history.history['accuracy']
```

[76]:
```
# Create subplots for loss and accuracy
plt.figure(figsize=(8, 3))
# Loss subplot
plt.subplot(1, 2, 1)
plt.plot(training_loss, label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Accuracy subplot
plt.subplot(1, 2, 2)
plt.plot(training_accuracy, label='Training Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



- As we can see in the graph, loss has been decreased with each epoch where accuracy has been increased

Let's make predictions on the test test and check whether those predictions are correct or not

[53]:
```
model.predict(x_test_processed)[0].argmax()
```

```
313/313 [==============================] - 2s 7ms/step
```

[53]:
```
7
```

[54]:
```
y_test[0]
```

[54]:
```
7
```

[71]:
```
predictions = model.predict(x_test_processed)
plt.figure(figsize=(12, 5))
for i in range(9):
    plt.subplot(1, 9, i+1)
    prediction = predictions[i].argmax()
    image =plt.imshow(x_test_processed[i])
    plt.xlabel('prediction: '+str(prediction))
    plt.xticks([])  # Hide the x-axis scale and ticks
    plt.yticks([])  # Hide the y-axis scale and ticks
```

```
313/313 [==============================] - 1s 4ms/step
```