1- ANN CODE----------

a) Regression:

1. Import Libraries and Load Dataset

```
import tensorflow as tf
from tensorflow.keras.datasets import boston_housing

(X_train, y_train), (X_test, y_test) = boston_housing.load_data()
```

2. Build the Neural Network Model

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer='sgd', loss='mse', metrics=['mae'])
model.summary()
```

3. Training the Model

```
history = model.fit(X_train, y_train, epochs=10)
```

4. Evaluating the Model

```
loss, mae = model.evaluate(X_test, y_test)
print(f"\nMean Absolute Error  : {mae}")
print(f"Loss                 : {loss}")
```

5. Predicting on Test Data

```
y_preds = model.predict(X_test)
```

6. Re-Evaluate to Verify

```
model.evaluate(X_test, y_test)
```

b)Classification

1. Import Libraries and Load Dataset

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np

(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
plt.imshow(X_train[0], cmap="gray")
plt.show()
```

2. Build the Neural Network Model

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(28,28)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(28, activation='relu'),
    tf.keras.layers.Dense(28, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])

model.summary()
```

3. Training the Model

```
history = model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)
```

4. Evaluating the Model

```
loss, acc = model.evaluate(X_test, y_test)
print(f"\nAccuracy : {acc}")
print(f"Loss     : {loss}")
```

5. Predicting on Test Data

```
pred = model.predict(X_test[0].reshape(-1,28,28))
print("\nPrediction Vector:\n", pred)

print("\nPredicted Digit:", np.argmax(pred))
```

6. Display Test Image

```
plt.imshow(X_test[0])
plt.show()
```

2- CNN CODE----------

1. Import Libraries + Mount Drive + Extract

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import os, zipfile, glob, cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout

zip_path     = "/content/drive/MyDrive/FDL 5th Sem Practical Codes/MRI DATASET.zip"
extract_path = "/content/mri_dataset"

if not os.path.exists(extract_path):
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_path)
```

2. Load & Preprocess Data

```
yes_path = os.path.join(extract_path, "yes/*")
no_path  = os.path.join(extract_path, "no/*")

has_tumor, no_tumor = [], []

# Images with tumor
for file in glob.iglob(yes_path):
    img = cv2.imread(file)
    if img is not None:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        img = cv2.resize(img, (128, 128))
        has_tumor.append((img, 1))

# Images without tumor
for file in glob.iglob(no_path):
    img = cv2.imread(file)
    if img is not None:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        img = cv2.resize(img, (128, 128))
        no_tumor.append((img, 0))

all_data = has_tumor + no_tumor
np.random.shuffle(all_data)

data   = np.array([item[0] for item in all_data])
labels = np.array([item[1] for item in all_data])

x_train, x_test, y_train, y_test = train_test_split(
    data, labels, test_size=0.2, random_state=42
)
```

```
x_train = x_train.reshape(x_train.shape[0], 128, 128, 1) / 255.0
x_test  = x_test.reshape(x_test.shape[0], 128, 128, 1) / 255.0
```

3. Build the CNN Model

```
model = Sequential([
    Conv2D(64, kernel_size=3, activation='relu', input_shape=(128, 128, 1)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(32, kernel_size=3, activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
```

4. Train the Model

```
history = model.fit(
    x_train, y_train,
    batch_size=32,
    epochs=10,
    validation_data=(x_test, y_test),
    verbose=1
)
```

5. Evaluate the Model

```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print(f"\nTest Accuracy : {test_acc:.4f}")
print(f"Test Loss     : {test_loss:.4f}")
```

6. Plot Accuracy & Loss Curves

```
plt.figure(figsize=(12, 4))

# Accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Acc')
```

```
plt.plot(history.history['val_accuracy'], label='Validation Acc')
plt.legend()
plt.title("Model Accuracy")

# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title("Model Loss")

plt.show()
```

7. Predictions & Plot Some

```
predictions = model.predict(x_test)
predicted_classes = (predictions > 0.5).astype(int).flatten()

plt.figure(figsize=(15, 10))
for i in range(12):
    plt.subplot(3, 4, i + 1)
    plt.imshow(x_test[i].reshape(128, 128), cmap="gray")
    plt.title(f"True: {y_test[i]}, Pred: {predicted_classes[i]}")
    plt.axis("off")

plt.tight_layout()
plt.show()
```