**Experiment no 3: Loading datasets into PyTorch creating training and Testing Split**

import pprint as pp

from sklearn import datasets

import numpy as np

import torch

from torch.utils.data import Dataset, random\_split

# Generate Sample Data

total\_samples = 1800

X\_data, Y\_data = datasets.make\_blobs(n\_samples=total\_samples, n\_features=3,

centers=[(-2, 5), (3, -4)], random\_state=42)

import torch

from torch.utils.data import Dataset

class CustomDataset(Dataset):

    def \_\_init\_\_(self, x, y):

        self.x = x

        self.y = y

    def \_\_getitem\_\_(self, index):

        sample = {

            'features': torch.tensor(self.x[index], dtype=torch.float32),

            'label': torch.tensor(self.y[index], dtype=torch.long)

        }

        return sample

    def \_\_len\_\_(self):

        return len(self.x)

# Create the Dataset Instance

dataset = CustomDataset(X\_data, Y\_data)

print("Total number of samples in the dataset:", len(dataset))

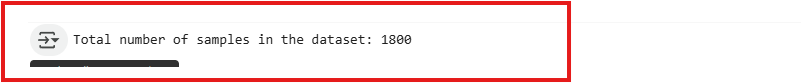
# Split the Dataset

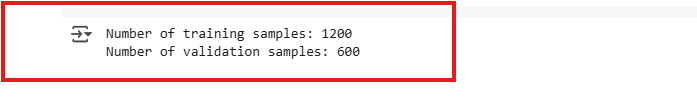
train\_data, val\_data = random\_split(dataset, [1200, 600])

print("Number of training samples:", len(train\_data))

print("Number of validation samples:", len(val\_data))

**OUTPUT:**





**Experiement No. 4: Creating functions to compute various losses.**

import numpy as np

def mean\_squared\_error(y\_true, y\_pred):

    return np.mean((y\_true - y\_pred) \*\* 2)

def mean\_absolute\_error(y\_true, y\_pred):

    return np.mean(np.abs(y\_true - y\_pred))

def binary\_cross\_entropy(y\_true, y\_pred):

    y\_pred = np.clip(y\_pred, 1e-9, 1 - 1e-9)

    return -np.mean(y\_true \* np.log(y\_pred) + (1 - y\_true) \* np.log(1 - y\_pred))

def categorical\_cross\_entropy(y\_true, y\_pred):

    y\_pred = np.clip(y\_pred, 1e-9, 1 - 1e-9)

    return -np.sum(y\_true \* np.log(y\_pred)) / y\_true.shape[0]

y\_true = np.array([1, 0, 1, 1])

y\_pred = np.array([0.9, 0.1, 0.8, 0.7])

print("Mean Squared Error:", mean\_squared\_error(y\_true, y\_pred))

print("Mean Absolute Error:", mean\_absolute\_error(y\_true, y\_pred))

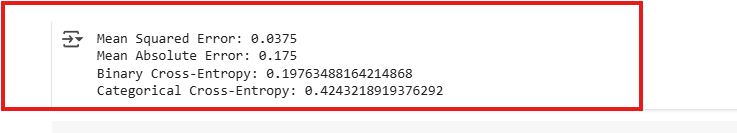
print("Binary Cross-Entropy:", binary\_cross\_entropy(y\_true, y\_pred))

y\_true\_cat = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])

y\_pred\_cat = np.array([[0.7, 0.2, 0.1], [0.1, 0.8, 0.1], [0.2, 0.3, 0.5]])

print("Categorical Cross-Entropy:", categorical\_cross\_entropy(y\_true\_cat, y\_pred\_cat))

**OUTPUT:**



**Experiement No.5: Feeding data to pretrained neural network and making predictions**.

import tensorflow as tf

from tensorflow.keras.applications import ResNet50

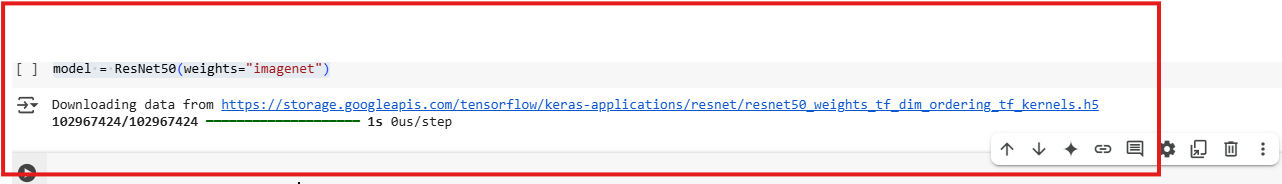
from tensorflow.keras.applications.resnet50 import preprocess\_input, decode\_predictions

import numpy as np

import cv2

from google.colab import files

model = ResNet50(weights="imagenet")



img\_path = '/content/images.jpg'

img = cv2.imread(img\_path)

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

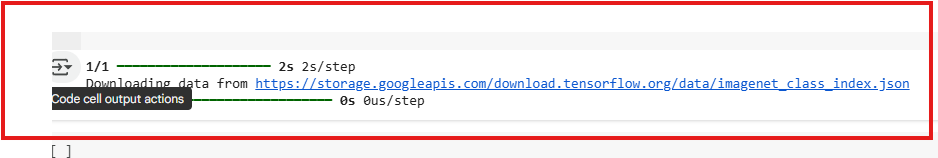
img = cv2.resize(img, (224, 224))

img = np.expand\_dims(img, axis=0)

img = preprocess\_input(img)

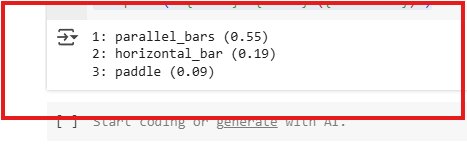
predictions = model.predict(img)

decoded\_predictions = decode\_predictions(predictions, top=3)[0]



for i, (imagenet\_id, label, score) in enumerate(decoded\_predictions):

    print(f"{i + 1}: {label} ({score:.2f})")



**Experiement No. 6: Implementing regression using deep neural network.**

# Step 1: Install and import everything

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.optimizers import Adam

# Step 2: Load dataset

data = fetch\_california\_housing()

X = data.data

y = data.target

# Step 3: Preprocess data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2,

random\_state=42)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Step 4: Build the model

model = Sequential([

Dense(64, activation='relu', input\_shape=(X\_train.shape[1],)),

Dense(64, activation='relu'),

Dense(1) # Linear output for regression

])

model.compile(optimizer=Adam(learning\_rate=0.001), loss='mse', metrics=['mae'])

# Step 5: Train the model

history = model.fit(X\_train\_scaled, y\_train, epochs=100, batch\_size=32,

validation\_split=0.1, verbose=1)

# Step 6: Evaluate

loss, mae = model.evaluate(X\_test\_scaled, y\_test)

print(f"Test MAE: {mae:.3f}")

# Step 7: Plot training history

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Val Loss')

plt.xlabel("Epochs")

plt.ylabel("MSE Loss")

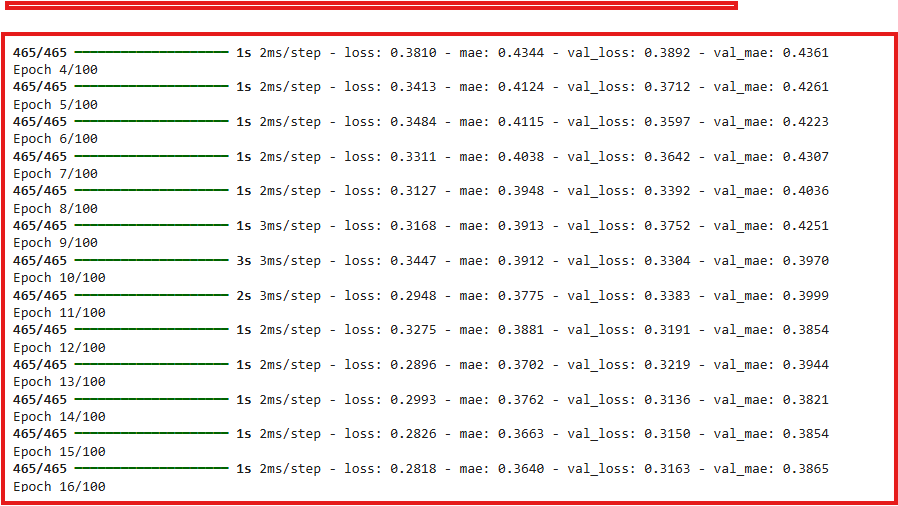
plt.title("Training vs Validation Loss")

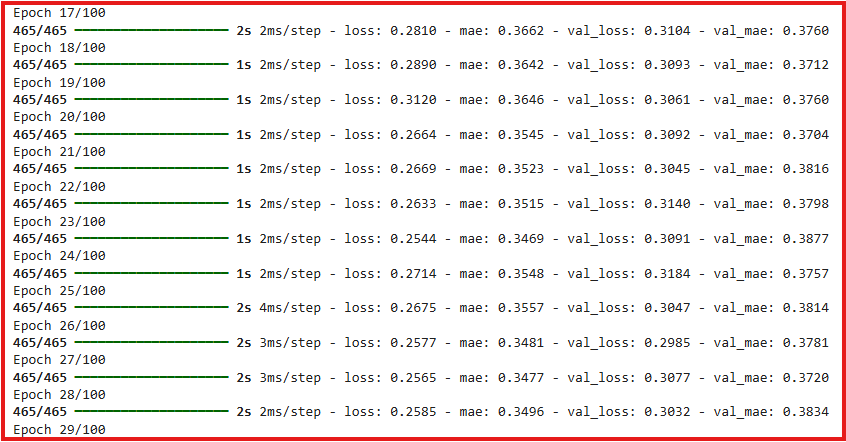
plt.legend()

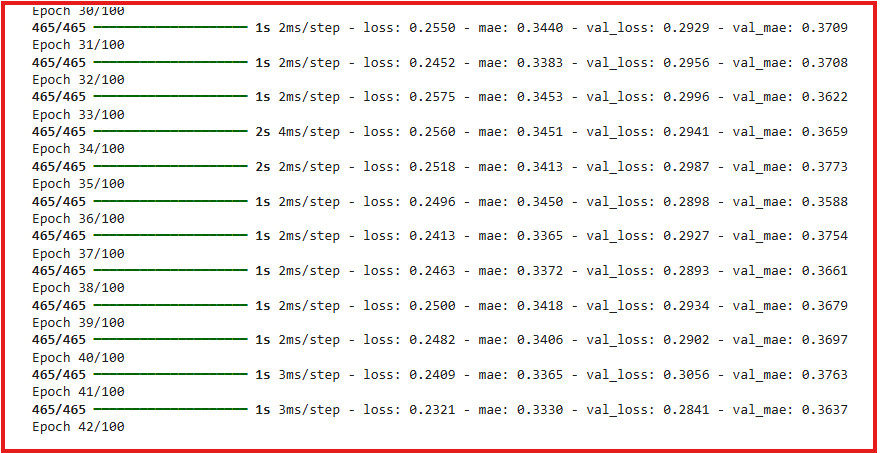
plt.grid(True)

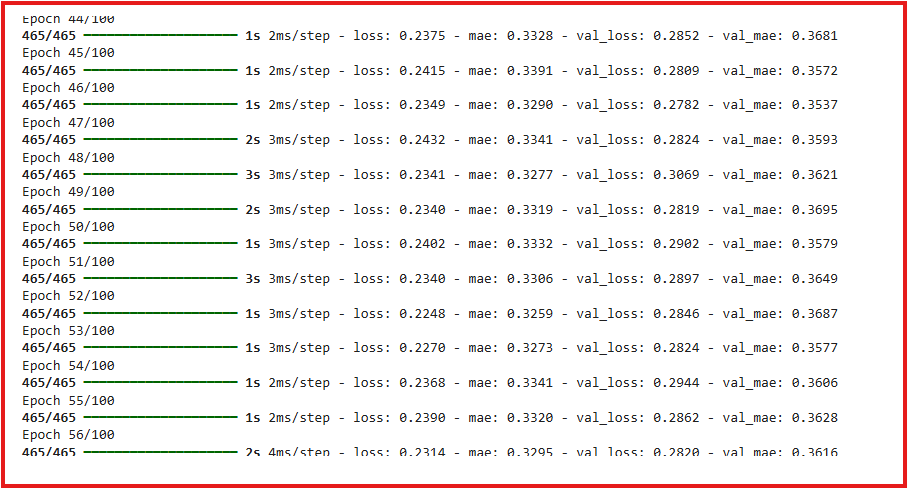
plt.show()

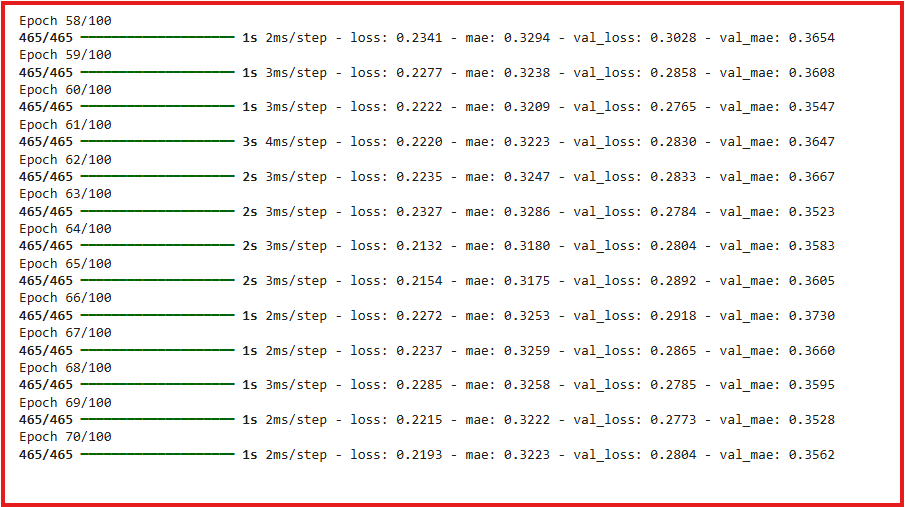
**OUTPUT:**

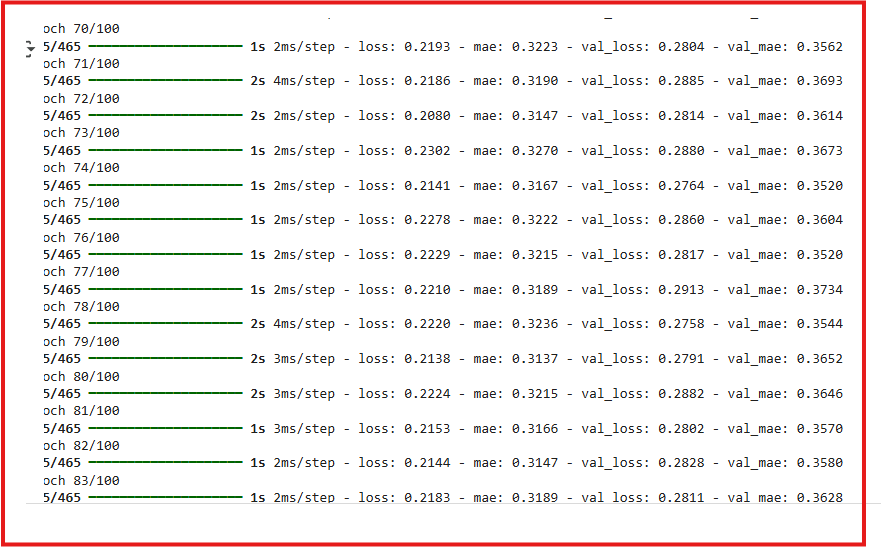


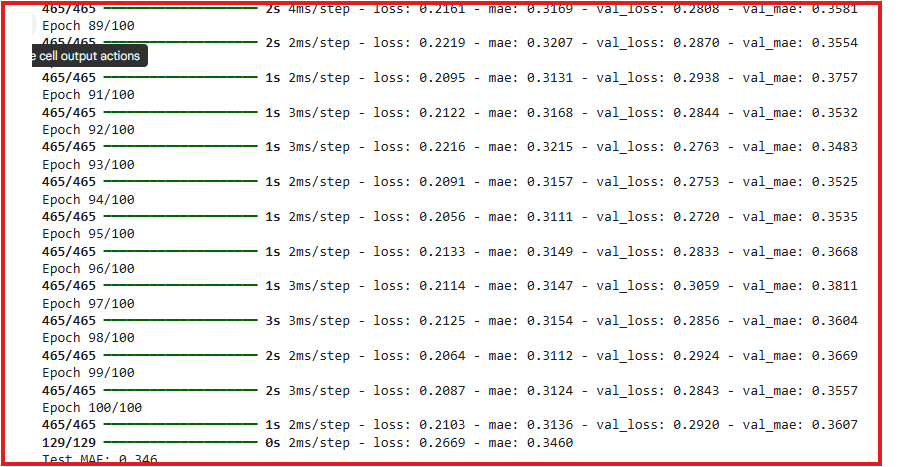


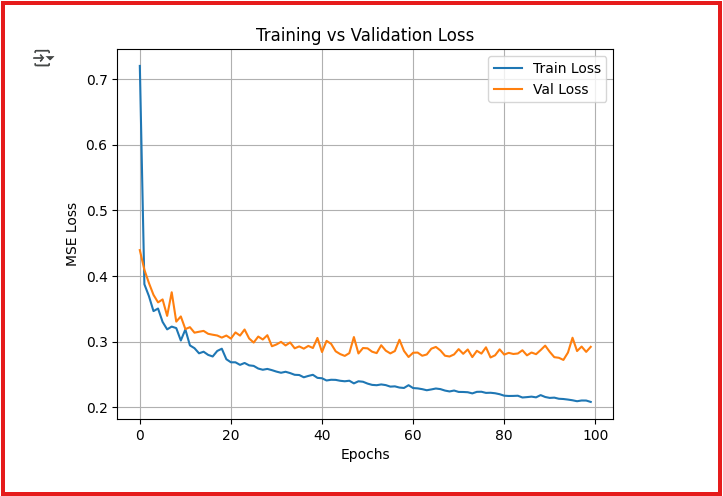












**Experiement No.7 Classifying IMDB movie review dataset using deep neural network-binary classification problem.**

import numpy as np

import matplotlib.pyplot as plt

from tensorflow import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Embedding, GlobalAveragePooling1D

from tensorflow.keras.preprocessing.sequence import pad\_sequences

vocab\_size = 10000

maxlen = 200

(x\_train, y\_train), (x\_test, y\_test) = keras.datasets.imdb.load\_data(num\_words=vocab\_size)

x\_train\_padded = pad\_sequences(x\_train, maxlen=maxlen)

x\_test\_padded = pad\_sequences(x\_test, maxlen=maxlen)

model = Sequential([

    Embedding(input\_dim=vocab\_size, output\_dim=32, input\_length=maxlen),

    GlobalAveragePooling1D(),

    Dense(64, activation='relu'),

    Dense(1, activation='sigmoid')

])

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

history = model.fit(x\_train\_padded, y\_train, epochs=10, batch\_size=512, validation\_split=0.2, verbose=1)

loss, accuracy = model.evaluate(x\_test\_padded, y\_test)

print(f"Test Accuracy: {accuracy:.3f}")

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Val Accuracy')

plt.xlabel("Epochs")

plt.ylabel("Accuracy")

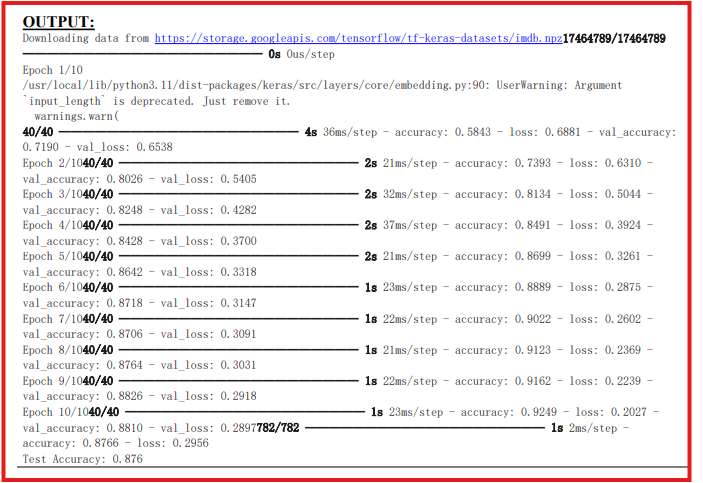
plt.title("Training vs Validation Accuracy")

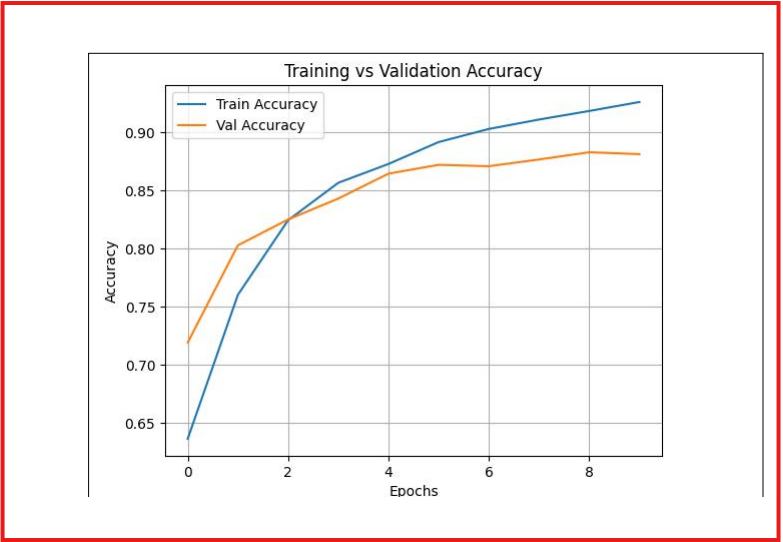
plt.legend()

plt.grid(True)

plt.show()

**OUTPUT:**





**Experiement No 8. Classifying Reuters dataset using deep neural network-multiclassclassification problem**

import numpy as np

import matplotlib.pyplot as plt

from tensorflow import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.datasets import reuters

# Step 2: Load the Reuters dataset

num\_words = 10000 # Use top 10,000 words

(x\_train, y\_train), (x\_test, y\_test) = reuters.load\_data(num\_words=num\_words)# Step 3: Vectorize the input data (convert sequences to one-hot encoded vectors)def vectorize\_sequences(sequences, dimension=10000):

results = np.zeros((len(sequences), dimension))

for i, sequence in enumerate(sequences):

results[i, sequence] = 1.0

return results

x\_train\_vec = vectorize\_sequences(x\_train)

x\_test\_vec = vectorize\_sequences(x\_test)

# Step 4: One-hot encode the labels

y\_train\_cat = to\_categorical(y\_train)

y\_test\_cat = to\_categorical(y\_test)

# Step 5: Build the model

model = Sequential([

Dense(128, activation='relu', input\_shape=(10000,)),

Dropout(0.5),

Dense(64, activation='relu'),

Dense(46, activation='softmax') # 46 output classes for Reuters dataset])

# Step 6: Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy',

metrics=['accuracy'])

# Step 7: Train the model

history = model.fit(

x\_train\_vec, y\_train\_cat,

epochs=10,

batch\_size=512,

validation\_split=0.2,

verbose=1

)

# Step 8: Evaluate on the test set

loss, accuracy = model.evaluate(x\_test\_vec, y\_test\_cat)

print(f"Test Accuracy: {accuracy:.3f}")

# Step 9: Plot training history

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title("Training vs Validation Accuracy")

plt.xlabel("Epochs")

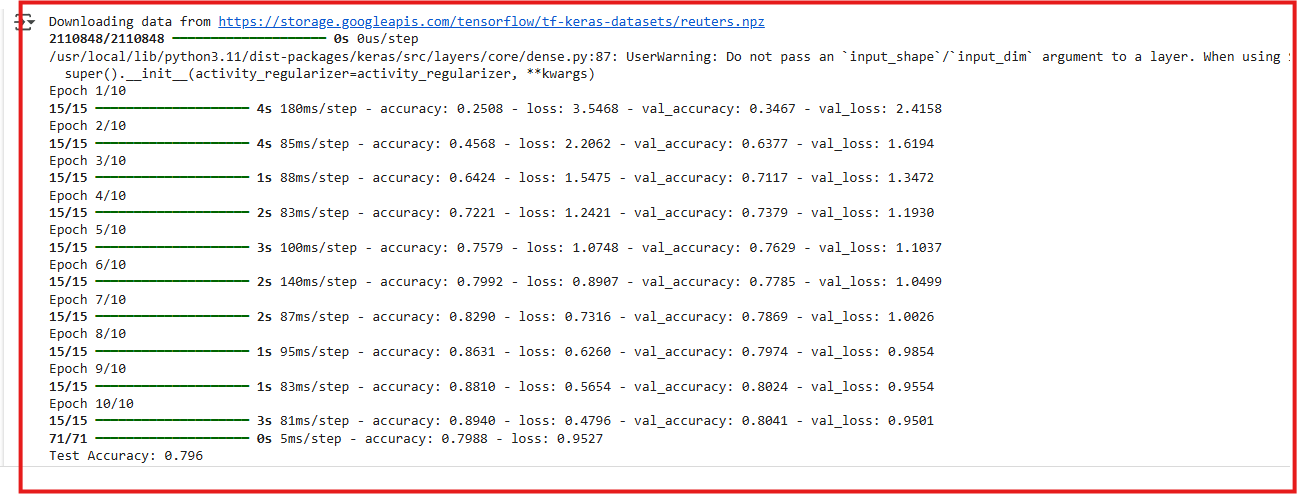
plt.ylabel("Accuracy")

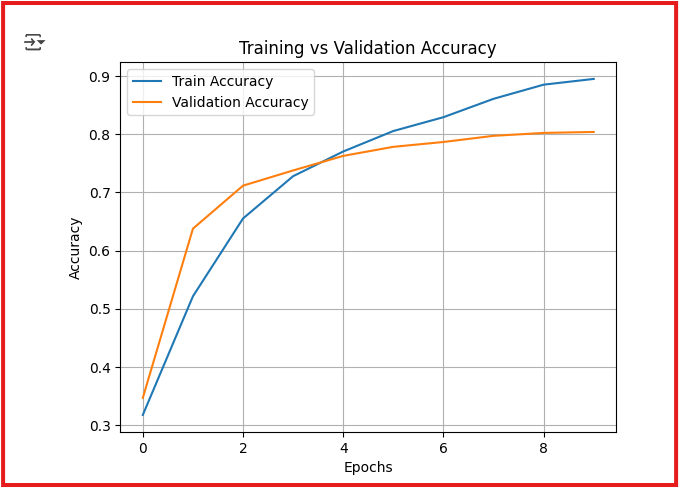
plt.legend()

plt.grid(True)

plt.show()

**OUTPUT:**





**Experiement No 9.Classifying MNIST Dataset using CNN.**

import tensorflow as tf

from tensorflow.keras import layers, models

import matplotlib.pyplot as plt

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.mnist.load\_data()

x\_train = x\_train.reshape((x\_train.shape[0], 28, 28, 1)).astype('float32') / 255

x\_test = x\_test.reshape((x\_test.shape[0], 28, 28, 1)).astype('float32') / 255

y\_train = tf.keras.utils.to\_categorical(y\_train, 10)

y\_test = tf.keras.utils.to\_categorical(y\_test, 10)

model = models.Sequential([

    layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.Flatten(),

    layers.Dense(64, activation='relu'),

    layers.Dense(10, activation='softmax')

])

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

history = model.fit(x\_train, y\_train, epochs=5, batch\_size=64, validation\_split=0.2, verbose=1)

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print(f"Test accuracy: {test\_acc:.4f}")

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.grid(True)

plt.show()

**OUTPUT:**



