**1.Aim: Introduction to Keras and Tensorflow (Optional – Pytorch). Configure and use google colab and kaggle GPU Objectives:**

1. To configure anaconda and google colab, kaggle environment

2. To Explore TF/Keras/Pytorch libraries

3. To learn to use GPU/TPU

4. To learn and understand Git

**Code:**

import pandas as pd

import numpy as np

import tensorflow as tf

from tensorflow.keras import layers, models

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

from sklearn.preprocessing import StandardScaler

from tensorflow.keras.callbacks import EarlyStopping

import matplotlib.pyplot as plt

# Load dataset

df = pd.read\_csv("/content/sample\_data/diabetes.csv")

# Prepare features and target

X = df.iloc[:, 0:8]

y = df["Outcome"]

# Standardize the data

scaler = StandardScaler()

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

# Split data into training and test sets

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_scaled, y, test\_size=0.1, random\_state=42)

# Build the neural network model with dropout layers

model = models.Sequential([

layers.Dense(100, activation="relu", input\_shape=(X\_train.shape[1],)),

layers.Dropout(0.3),

layers.Dense(75, activation="relu"),

layers.Dropout(0.3),

layers.Dense(50, activation="relu"),

layers.Dropout(0.3),

layers.Dense(25, activation="relu"),

layers.Dropout(0.2),

layers.Dense(12, activation="relu"),

layers.Dense(1, activation="sigmoid")

])

# Compile the model

model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=0.001),

loss="binary\_crossentropy", metrics=["accuracy"])

# Early stopping

early\_stop = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)

# Train the model

history = model.fit(X\_train, Y\_train, epochs=150, validation\_data=(X\_test, Y\_test), callbacks=[early\_stop])

# Evaluate the model

test\_loss, test\_acc = model.evaluate(X\_test, Y\_test, verbose=2)

print(f"Test Loss: {test\_loss:.4f}")

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

# Plot Training Loss and Accuracy

plt.figure(figsize=(12, 5))

# Plot Loss

plt.subplot(1, 2, 1)

plt.plot(history.history['loss'], label='Training Loss', color='b')

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

plt.title('Loss over Epochs')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend(loc='upper right')

plt.ylim([0, 1.0])

# Plot Accuracy

plt.subplot(1, 2, 2)

plt.plot(history.history['accuracy'], label='Training Accuracy', color='g')

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

plt.title('Accuracy over Epochs')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend(loc='lower right')

plt.ylim([0.6, 1.0])

# Show both plots

plt.tight\_layout()

plt.show()

**2.Aim: Develop multi class classifier using deep multilayer perceptron (Keras/tensorflow/pytorch) for MNIST hand recognition dataset and CIFAR10. Fine the parameters for better accuracy.**

**• Develop application with GUI to upload input to the system**

**• Test the model Objectives:**

**• Learn Deep Neural Network modeling**

**• Learn to develop and deploy models**

**Code: minist dataset**

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten, Dropout

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.datasets import mnist

import numpy as np

import tkinter as tk

from tkinter import filedialog

from PIL import Image, ImageEnhance, ImageOps, ImageTk

# Load and preprocess MNIST data

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

x\_train = x\_train.astype("float32") / 255.0

x\_test = x\_test.astype("float32") / 255.0

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

# Define the MNIST model

mnist\_model = Sequential([

Flatten(input\_shape=(28, 28, 1)), # Adding channel dimension

Dense(128, activation='relu'),

Dropout(0.2),

Dense(64, activation='relu'),

Dropout(0.2),

Dense(10, activation='softmax')

])

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

mnist\_model.fit(x\_train[..., np.newaxis], y\_train, validation\_data=(x\_test[..., np.newaxis], y\_test), epochs=10, batch\_size=128)

# GUI function to upload and predict on MNIST model

def upload\_and\_predict\_mnist():

file\_path = filedialog.askopenfilename()

if file\_path:

# Load image and ensure it's in grayscale

img = Image.open(file\_path).convert("L") # Convert to grayscale

# Enhance contrast to ensure the digit stands out

enhancer = ImageEnhance.Contrast(img)

img = enhancer.enhance(2.0)

# Resize to 28x28 and invert colors if necessary

img\_resized = img.resize((28, 28))

img\_resized = ImageOps.invert(img\_resized) if np.array(img\_resized).mean() > 128 else img\_resized

# Display uploaded image on the screen

img\_display = ImageTk.PhotoImage(img\_resized.resize((140, 140))) # Resize for better display

image\_label.config(image=img\_display)

image\_label.image = img\_display

# Normalize and reshape for model input

img\_array = np.array(img\_resized).astype("float32") / 255.0

img\_array = img\_array.reshape(1, 28, 28, 1) # Add channel dimension for grayscale

# Make prediction and display result

prediction = np.argmax(mnist\_model.predict(img\_array), axis=1)

result\_label.config(text=f"Predicted Digit: {prediction[0]}")

# Tkinter GUI setup

root = tk.Tk()

root.title("Digit Recognition - MNIST")

# GUI elements

upload\_button = tk.Button(root, text="Upload Digit Image", command=upload\_and\_predict\_mnist)

upload\_button.pack()

# Label for displaying uploaded image

image\_label = tk.Label(root)

image\_label.pack()

# Label for displaying prediction result

result\_label = tk.Label(root, text="Prediction will appear here")

result\_label.pack()

root.mainloop()

**cifar 10 dataset:**

from tensorflow.keras.layers import Conv2D, MaxPooling2D

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten, Dropout

from tensorflow.keras.utils import to\_categorical # Add this import line

import numpy as np

import tkinter as tk

from tkinter import filedialog

from PIL import Image , ImageTk

# Load and preprocess CIFAR-10 data

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

x\_train = x\_train.astype("float32") / 255.0

x\_test = x\_test.astype("float32") / 255.0

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

# Define the CIFAR-10 model

cifar\_model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)),

MaxPooling2D((2, 2)),

Dropout(0.2),

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

MaxPooling2D((2, 2)),

Dropout(0.2),

Flatten(),

Dense(128, activation='relu'),

Dropout(0.3),

Dense(10, activation='softmax')

])

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

cifar\_model.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=5, batch\_size=64)

cifar\_labels = ["airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck"]

# GUI for uploading an image and testing the CIFAR-10 model

def upload\_and\_predict\_cifar():

file\_path = filedialog.askopenfilename()

if file\_path:

# Load image and resize to 32x32

img = Image.open(file\_path).convert("RGB") # CIFAR-10 expects RGB images

img\_resized = img.resize((32, 32))

# Display uploaded image on the screen

img\_display = ImageTk.PhotoImage(img\_resized.resize((140, 140))) # Resize for better display in GUI

image\_label.config(image=img\_display)

image\_label.image = img\_display

# Normalize and reshape for model input

img\_array = np.array(img\_resized).astype("float32") / 255.0

img\_array = img\_array.reshape(1, 32, 32, 3) # CIFAR-10 input shape

# Make prediction and display result

prediction = np.argmax(cifar\_model.predict(img\_array), axis=1)

result\_label.config(text=f"Predicted Class: {cifar\_labels[prediction[0]]}")

# Tkinter GUI setup

root = tk.Tk()

root.title("Image Recognition - CIFAR-10")

# GUI elements

upload\_button = tk.Button(root, text="Upload Image", command=upload\_and\_predict\_cifar)

upload\_button.pack()

# Label for displaying uploaded image

image\_label = tk.Label(root)

image\_label.pack()

# Label for displaying prediction result

result\_label = tk.Label(root, text="Prediction will appear here")

result\_label.pack()

root.mainloop()

**3.Aim: Develop classification model for cat-dogs dataset using CNN model. Analyze the model accuracy and generate classification report.**

**• Analyze the result with and without regularization/dropout**

**• Develop an application and test the user given inputs**

**Code:**

#2-CIFAR-10 && 3

from tensorflow.keras.layers import Conv2D, MaxPooling2D

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten, Dropout

from tensorflow.keras.utils import to\_categorical # Add this import line

import numpy as np

import tkinter as tk

from tkinter import filedialog

from PIL import Image , ImageTk

# Load and preprocess CIFAR-10 data

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

x\_train = x\_train.astype("float32") / 255.0

x\_test = x\_test.astype("float32") / 255.0

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

# Define the CIFAR-10 model

cifar\_model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)),

MaxPooling2D((2, 2)),

Dropout(0.2),

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

MaxPooling2D((2, 2)),

Dropout(0.2),

Flatten(),

Dense(128, activation='relu'),

Dropout(0.3),

Dense(10, activation='softmax')

])

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

cifar\_model.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=5, batch\_size=64)

cifar\_labels = ["airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck"]

# GUI for uploading an image and testing the CIFAR-10 model

def upload\_and\_predict\_cifar():

file\_path = filedialog.askopenfilename()

if file\_path:

# Load image and resize to 32x32

img = Image.open(file\_path).convert("RGB") # CIFAR-10 expects RGB images

img\_resized = img.resize((32, 32))

# Display uploaded image on the screen

img\_display = ImageTk.PhotoImage(img\_resized.resize((140, 140))) # Resize for better display in GUI

image\_label.config(image=img\_display)

image\_label.image = img\_display

# Normalize and reshape for model input

img\_array = np.array(img\_resized).astype("float32") / 255.0

img\_array = img\_array.reshape(1, 32, 32, 3) # CIFAR-10 input shape

# Make prediction and display result

prediction = np.argmax(cifar\_model.predict(img\_array), axis=1)

result\_label.config(text=f"Predicted Class: {cifar\_labels[prediction[0]]}")

# Tkinter GUI setup

root = tk.Tk()

root.title("Image Recognition - CIFAR-10")

# GUI elements

upload\_button = tk.Button(root, text="Upload Image", command=upload\_and\_predict\_cifar)

upload\_button.pack()

# Label for displaying uploaded image

image\_label = tk.Label(root)

image\_label.pack()

# Label for displaying prediction result

result\_label = tk.Label(root, text="Prediction will appear here")

result\_label.pack()

root.mainloop()

**4.Aim: Develop face recognition system using CNN. Create a dataset of minimum 20 students from your class. Check and validate the accuracy of the model.**

**Apply dimensionality reduction on input image and plot the change in accuracy of system**

**Code:**

# Import necessary libraries

import pandas as pd

import tensorflow as tf

from tensorflow.keras import models, Sequential, layers, preprocessing

from tensorflow.keras.preprocessing.image import ImageDataGenerator, load\_img

from tensorflow.keras.applications.vgg16 import VGG16

import os

import mtcnn

# Load file names from the FaceRecog directory and create NameArray with labels

file\_names = os.listdir("Assets/FaceRecog")

NameArray = []

# Categorize files based on name and append corresponding labels to NameArray

for name in file\_names:

category = name.split('.')[0]

if category == 'gourishankar':

NameArray.append('Gourishankar')

elif category == 'Aditya\_Panchwagh':

NameArray.append("Aditya")

elif category == "Dhananjay\_Jha":

NameArray.append("Dhananjay")

elif category == 'Habil\_Bhagat':

NameArray.append("Habil")

elif category == "Karan\_Mahajan":

NameArray.append("Karan")

elif category == 'Kartik\_Jawanjal':

NameArray.append("Kartik")

elif category == "Krish\_Shah":

NameArray.append("Krish")

elif category == 'Manas\_Oswal':

NameArray.append("Manas")

elif category == "Mayank\_Modi":

NameArray.append("Mayank")

elif category == 'Shubham\_Pagare':

NameArray.append("Shubham")

elif category == "Vishal\_Kasa":

NameArray.append("Kasa")

# Create a DataFrame to hold filenames and corresponding categories

train = pd.DataFrame({

'filename': file\_names,

'category': NameArray

})

# Split data into training and validation sets

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

train\_df, validate\_df = train\_test\_split(train, test\_size=0.2, random\_state=0)

train\_df = train\_df.reset\_index(drop=True)

validate\_df = validate\_df.reset\_index(drop=True)

# Set up ImageDataGenerator for training and validation data augmentation

training\_gen = ImageDataGenerator(

rotation\_range=5,

rescale=1./255,

shear\_range=0.1,

zoom\_range=0.2,

horizontal\_flip=True,

width\_shift\_range=0.1,

height\_shift\_range=0.1

)

trainingdata = training\_gen.flow\_from\_dataframe(

train\_df,

"Assets/FaceRecog",

x\_col='filename',

y\_col='category',

target\_size=(224, 224),

class\_mode='categorical'

)

validation\_gen = ImageDataGenerator(

rotation\_range=5,

rescale=1./255,

shear\_range=0.1,

zoom\_range=0.2,

horizontal\_flip=True,

width\_shift\_range=0.1,

height\_shift\_range=0.1

)

validationdata = validation\_gen.flow\_from\_dataframe(

validate\_df,

"Assets/FaceRecog",

x\_col='filename',

y\_col='category',

target\_size=(224, 224),

class\_mode='categorical'

)

# Build the model using VGG16 as the base

base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

base\_model.trainable = False

model = models.Sequential([

base\_model,

layers.Flatten(),

layers.Dense(400, activation='relu'),

layers.Dense(6, activation='softmax')

])

# Compile the model

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

# Train the model

model.fit(trainingdata, validation\_data=validationdata, epochs=30)

# Evaluate the model on validation data

\_, validation\_acc = model.evaluate(validationdata, verbose=0)

print("Validation Accuracy:", validation\_acc)

**5.Write a program to demonstrate the change in accuracy/loss/convergence time with change in optimizers like stochastic gradient descent, adam, adagrad, RMSprop and Nadam for any suitable application Objectives:**

**1. To learn optimization algorithms**

**2. To learn and understand hyperparameter**

**Code:**

# Aditya Kulkarni

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten, Dropout

from tensorflow.keras.optimizers import SGD, Adam, Adagrad, RMSprop, Nadam

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import to\_categorical

import matplotlib.pyplot as plt

import time

# Load and preprocess MNIST data

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

x\_train = x\_train.astype("float32") / 255.0

x\_test = x\_test.astype("float32") / 255.0

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

# Define the neural network model

def create\_model(optimizer):

model = Sequential([

Flatten(input\_shape=(28, 28)),

Dense(128, activation='relu'),

Dropout(0.2),

Dense(64, activation='relu'),

Dropout(0.2),

Dense(10, activation='softmax')

])

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

return model

# Optimizers to be tested

optimizers = {

'SGD': SGD(),

'Adam': Adam(),

'Adagrad': Adagrad(),

'RMSprop': RMSprop(),

'Nadam': Nadam()

}

# Store results

results = {}

# Train the model with different optimizers and record time, accuracy, and loss

for name, optimizer in optimizers.items():

print(f"Training with {name} optimizer...")

start\_time = time.time()

model = create\_model(optimizer)

history = model.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=10, batch\_size=128, verbose=0)

end\_time = time.time()

training\_time = end\_time - start\_time

results[name] = {

'history': history,

'training\_time': training\_time

}

print(f"{name} optimizer took {training\_time:.2f} seconds to converge.\n")

# Plot the results (Accuracy and Loss)

fig, axes = plt.subplots(2, 1, figsize=(10, 12))

# Plot Accuracy

for name, result in results.items():

axes[0].plot(result['history'].history['accuracy'], label=f"{name} Train Accuracy")

axes[0].plot(result['history'].history['val\_accuracy'], label=f"{name} Validation Accuracy")

axes[0].set\_title('Accuracy Comparison')

axes[0].set\_xlabel('Epochs')

axes[0].set\_ylabel('Accuracy')

axes[0].legend()

# Plot Loss

for name, result in results.items():

axes[1].plot(result['history'].history['loss'], label=f"{name} Train Loss")

axes[1].plot(result['history'].history['val\_loss'], label=f"{name} Validation Loss")

axes[1].set\_title('Loss Comparison')

axes[1].set\_xlabel('Epochs')

axes[1].set\_ylabel('Loss')

axes[1].legend()

plt.tight\_layout()

plt.show()

# Display training time for each optimizer

print("Training Time for each Optimizer:")

for name, result in results.items():

print(f"{name}: {result['training\_time']:.2f} seconds")

**6.Aim: Apply transfer learning with pre-trained VGG16 model on assignment 3 and analyze the result.**

**Code:**

# Aditya Kulkarni

import tensorflow as tf

from tensorflow.keras.applications import VGG16

from tensorflow.keras.layers import Dense, Flatten, Dropout

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.utils import to\_categorical

# Load and preprocess CIFAR-10 dataset

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

# Normalize pixel values to range 0-1

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

# Convert labels to one-hot encoding

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

# Load the VGG16 model without the top fully connected layers

base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(32, 32, 3))

# Freeze all the layers in the base model

for layer in base\_model.layers:

layer.trainable = False

# Add custom layers on top of the base model

x = Flatten()(base\_model.output)

x = Dense(256, activation='relu')(x)

x = Dropout(0.5)(x) # Add dropout for regularization

x = Dense(128, activation='relu')(x)

x = Dropout(0.5)(x)

output = Dense(10, activation='softmax')(x)

# Create the new model

model = Model(inputs=base\_model.input, outputs=output)

# Compile the model

model.compile(optimizer=Adam(learning\_rate=0.0001), loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(x\_train, y\_train, epochs=10, batch\_size=64, validation\_data=(x\_test, y\_test))

# Evaluate the model on test set

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

print(f"Test accuracy: {test\_acc \* 100:.2f}%")

# Plot training history

import matplotlib.pyplot as plt

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

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

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

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

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

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

**7.Aim: Develop RNN model for Cryptocurrency pricing prediction or text sentiment analysis Objectives:**

**1. To learn RNN**

**2. To learn and implement LSTM**

**Code:**

# Aditya Kulkarni

# Importing necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, LSTM, GRU, Bidirectional, Dropout

# Load dataset

df = pd.read\_csv("Assets/BTC\_USD.csv", parse\_dates=['date'])

df = df.sort\_values('date')

df.reset\_index(drop=True, inplace=True)

# Data Preprocessing

# Using MinMaxScaler to scale data between 0 and 1

scaler = MinMaxScaler()

close\_price = df[['close']].values

scaled\_close = scaler.fit\_transform(close\_price)

# Define sequence length and prepare data sequences for RNN

SEQ\_LEN = 60 # sequence length (60 days)

def to\_sequences(data, seq\_len=SEQ\_LEN):

X, y = [], []

for i in range(len(data) - seq\_len):

X.append(data[i:i + seq\_len])

y.append(data[i + seq\_len])

return np.array(X), np.array(y)

X, y = to\_sequences(scaled\_close)

# Split data into training and testing sets (80% train, 20% test)

train\_size = int(0.8 \* len(X))

X\_train, X\_test = X[:train\_size], X[train\_size:]

y\_train, y\_test = y[:train\_size], y[train\_size:]

# Model Architecture

model = Sequential()

# Adding Bidirectional LSTM layer

model.add(Bidirectional(LSTM(64, return\_sequences=True), input\_shape=(X\_train.shape[1], 1)))

model.add(Dropout(0.2))

# Adding GRU layer

model.add(GRU(32, return\_sequences=False))

model.add(Dropout(0.2))

# Dense layer for output

model.add(Dense(1))

# Compiling the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Training the model

history = model.fit(X\_train, y\_train, epochs=20, batch\_size=32, validation\_split=0.1)

# Model evaluation

model.evaluate(X\_test, y\_test)

# Plotting training and validation loss

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

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

plt.title('Model Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

# Making Predictions

predictions = model.predict(X\_test)

predicted\_price = scaler.inverse\_transform(predictions)

actual\_price = scaler.inverse\_transform(y\_test)

# Plotting Actual vs Predicted Prices

plt.plot(actual\_price, color='blue', label='Actual Price')

plt.plot(predicted\_price, color='red', label='Predicted Price')

plt.title('Cryptocurrency Price Prediction')

plt.xlabel('Time')

plt.ylabel('Price')

plt.legend()

plt.show()

**8.Aim: Develop an autoencoder to encode and decode the image. Analyze the results.**

**a) Develop AE for MNIST dataset**

**b) Use output of AE as input to CNN**

**CODE:**

# Aditya Kulkarni

import tensorflow as tf

from tensorflow.keras.datasets import mnist

from tensorflow.keras.layers import Input, Dense, Flatten, Reshape

from tensorflow.keras.models import Model

import numpy as np

import matplotlib.pyplot as plt

# Load the MNIST dataset

(x\_train, \_), (x\_test, \_) = mnist.load\_data()

# Preprocess the data

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

x\_train = x\_train.reshape(len(x\_train), 28, 28, 1)

x\_test = x\_test.reshape(len(x\_test), 28, 28, 1)

# Define the encoder and decoder

input\_img = Input(shape=(28, 28, 1))

x = Flatten()(input\_img)

encoded = Dense(128, activation='relu')(x)

decoded = Dense(784, activation='sigmoid')(encoded)

decoded = Reshape((28, 28, 1))(decoded)

# Create the autoencoder model

autoencoder = Model(input\_img, decoded)

# Create the encoder model

encoder = Model(input\_img, encoded)

# Compile the model

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

# Train the model

history = autoencoder.fit(

x\_train, x\_train,

epochs=50,

batch\_size=256,

shuffle=True,

validation\_data=(x\_test, x\_test),

verbose=2

)

# Encode and decode some digits

encoded\_imgs = encoder.predict(x\_test)

decoded\_imgs = autoencoder.predict(x\_test)

# Display original and decoded images

n = 10 # Number of digits to display

plt.figure(figsize=(20, 4))

for i in range(n):

# Display original

ax = plt.subplot(2, n, i + 1)

plt.imshow(x\_test[i].reshape(28, 28), cmap='gray')

ax.axis('off')

# Display reconstruction

ax = plt.subplot(2, n, i + 1 + n)

plt.imshow(decoded\_imgs[i].reshape(28, 28), cmap='gray')

ax.axis('off')

plt.show()

# Calculate reconstruction accuracy

threshold = 0.5 # Binary threshold for pixel values

x\_test\_binary = (x\_test > threshold).astype(np.float32)

decoded\_imgs\_binary = (decoded\_imgs > threshold).astype(np.float32)

# Calculate accuracy as the proportion of matching pixels

accuracy = np.mean(np.equal(x\_test\_binary, decoded\_imgs\_binary))

print(f"Reconstruction Accuracy: {accuracy \* 100:.2f}%")

# Plot training history

plt.figure(figsize=(10, 5))

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

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

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()