This lab demonstrates how to load the MNIST dataset, normalize it, build a simple feedforward neural network, train it, evaluate its performance, and visualize predictions.

a simple feedforward neural network for classifying handwritten digits using the MNIST dataset.

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_openml

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

from sklearn.preprocessing import StandardScaler

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import accuracy\_score

# Load the MNIST dataset

mnist = fetch\_openml('mnist\_784', version=1)

X, y = mnist.data, mnist.target.astype(int)

# Normalize and split the dataset

X = X / 255.0

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize and train the neural network model

model = MLPClassifier(hidden\_layer\_sizes=(128,), activation='relu', solver='adam', max\_iter=10, random\_state=42)

model.fit(X\_train, y\_train)

# Evaluate the model

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'\nTest accuracy: {accuracy}')

# Function to plot image and prediction

def plot\_image(i, true\_label, img):

plt.grid(False)

plt.xticks([])

plt.yticks([])

plt.imshow(img.reshape(28, 28), cmap=plt.cm.binary)

plt.xlabel(f"Predicted: {y\_pred[i]}", color='blue')

# Display sample predictions

num\_rows, num\_cols = 3, 3

num\_images = num\_rows \* num\_cols

plt.figure(figsize=(2\*num\_cols, 2\*num\_rows))

for i in range(num\_images):

plt.subplot(num\_rows, num\_cols, i+1)

plot\_image(i, y\_test.iloc[i], X\_test.iloc[i])

plt.show()

## ****Generative Adversarial Network (GAN) for Image Generation (MNIST)****

A simple GAN generates handwritten digits.

import tensorflow as tf

from tensorflow import keras

import numpy as np

import matplotlib.pyplot as plt

# Load the MNIST dataset

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

x\_train = (x\_train.astype(np.float32) - 127.5) / 127.5 # Normalize to [-1,1]

x\_train = np.expand\_dims(x\_train, axis=-1) # Add channel dimension

# Hyperparameters

latent\_dim = 100

batch\_size = 128

epochs = 10000

# Generator Model

def build\_generator():

model = keras.Sequential([

keras.layers.Dense(256, activation="relu", input\_dim=latent\_dim),

keras.layers.BatchNormalization(),

keras.layers.Dense(512, activation="relu"),

keras.layers.BatchNormalization(),

keras.layers.Dense(1024, activation="relu"),

keras.layers.BatchNormalization(),

keras.layers.Dense(28 \* 28 \* 1, activation="tanh"),

keras.layers.Reshape((28, 28, 1))

])

return model

# Discriminator Model

def build\_discriminator():

model = keras.Sequential([

keras.layers.Flatten(input\_shape=(28, 28, 1)),

keras.layers.Dense(512, activation="relu"),

keras.layers.Dense(256, activation="relu"),

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

])

return model

# Build GAN

generator = build\_generator()

discriminator = build\_discriminator()

discriminator.compile(loss="binary\_crossentropy", optimizer=keras.optimizers.Adam(0.0002, 0.5), metrics=["accuracy"])

# Combine into GAN

discriminator.trainable = False

gan\_input = keras.Input(shape=(latent\_dim,))

gan\_output = discriminator(generator(gan\_input))

gan = keras.Model(gan\_input, gan\_output)

gan.compile(loss="binary\_crossentropy", optimizer=keras.optimizers.Adam(0.0002, 0.5))

# Training Loop

for epoch in range(epochs):

# Select a random batch of real images

idx = np.random.randint(0, x\_train.shape[0], batch\_size)

real\_images = x\_train[idx]

# Generate fake images

noise = np.random.normal(0, 1, (batch\_size, latent\_dim))

fake\_images = generator.predict(noise)

# Train Discriminator

d\_loss\_real = discriminator.train\_on\_batch(real\_images, np.ones((batch\_size, 1)))

d\_loss\_fake = discriminator.train\_on\_batch(fake\_images, np.zeros((batch\_size, 1)))

d\_loss = 0.5 \* np.add(d\_loss\_real, d\_loss\_fake)

# Train Generator

noise = np.random.normal(0, 1, (batch\_size, latent\_dim))

g\_loss = gan.train\_on\_batch(noise, np.ones((batch\_size, 1)))

# Print progress

if epoch % 1000 == 0:

print(f"{epoch} [D loss: {d\_loss[0]} | D accuracy: {100 \* d\_loss[1]}%] [G loss: {g\_loss}]")

print("GAN Training Complete!")

## ****Convolutional Neural Network (CNN) for Image Classification (CIFAR-10)****

A CNN classifies images from the CIFAR-10 dataset, which includes 10 categories like airplanes, cars, and animals.

import tensorflow as tf

from tensorflow import keras

import numpy as np

import matplotlib.pyplot as plt

# Load the CIFAR-10 dataset

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

# Normalize the dataset

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

# Define the CNN model

model = keras.Sequential([

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

keras.layers.MaxPooling2D((2, 2)),

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

keras.layers.MaxPooling2D((2, 2)),

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

keras.layers.Flatten(),

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

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

])

# Compile the model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Train the model

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

# Evaluate the model

test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=2)

print(f'\nTest accuracy: {test\_acc}')

## ****Recurrent Neural Network (RNN) for Text Classification (IMDB Movie Reviews)****

An RNN (using LSTM) predicts sentiment (positive or negative) for movie reviews.

import tensorflow as tf

from tensorflow import keras

import numpy as np

# Load the IMDB dataset

vocab\_size = 10000

max\_length = 200

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

# Pad sequences

x\_train = keras.preprocessing.sequence.pad\_sequences(x\_train, maxlen=max\_length)

x\_test = keras.preprocessing.sequence.pad\_sequences(x\_test, maxlen=max\_length)

# Define the RNN model using LSTM

model = keras.Sequential([

keras.layers.Embedding(vocab\_size, 32, input\_length=max\_length),

keras.layers.LSTM(64, return\_sequences=True),

keras.layers.LSTM(32),

keras.layers.Dense(1, activation='sigmoid')

])

# Compile the model

model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

# Train the model

model.fit(x\_train, y\_train, epochs=5, validation\_data=(x\_test, y\_test))

# Evaluate the model

test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=2)

print(f'\nTest accuracy: {test\_acc}')

**Transformer Model for Text Classification (IMDB Sentiment Analysis)**

A Transformer-based model classifies movie reviews.

import tensorflow as tf

from tensorflow import keras

import tensorflow\_hub as hub

import tensorflow\_text as text

# Load IMDB dataset

(x\_train, y\_train), (x\_test, y\_test) = keras.datasets.imdb.load\_data(num\_words=10000)

x\_train = keras.preprocessing.sequence.pad\_sequences(x\_train, maxlen=256)

x\_test = keras.preprocessing.sequence.pad\_sequences(x\_test, maxlen=256)

# Load Pretrained Transformer (BERT)

bert\_layer = hub.KerasLayer("https://tfhub.dev/tensorflow/bert\_en\_uncased\_preprocess/3", trainable=False)

# Build Model

inputs = keras.Input(shape=(256,), dtype=tf.int32)

x = bert\_layer(inputs)

x = keras.layers.Dense(128, activation="relu")(x)

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

model = keras.Model(inputs, x)

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

# Train Model

model.fit(x\_train, y\_train, epochs=3, validation\_data=(x\_test, y\_test), batch\_size=32)

# Evaluate Model

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

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