**PROGRAM:**

**#1.a Feed Forward Neural Network**

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.losses import SparseCategoricalCrossentropy

from tensorflow.keras.metrics import SparseCategoricalAccuracy

mnist = tf.keras.datasets.mnist

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

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

print(x\_train.shape)

print(x\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

plt.imshow(x\_train[59999], cmap="gray")

plt.show()

model = Sequential([

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

Dense(128, activation='relu'),

Dense(10, activation='softmax')

])

model.compile(optimizer=Adam(),

loss=SparseCategoricalCrossentropy(),

metrics=[SparseCategoricalAccuracy()])

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

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

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

X\_test = np.array([[0.2], [0.5], [0.8]])

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

print(f'Predictions for test data:\n{y\_pred\_test}')

**OUTPUT:**

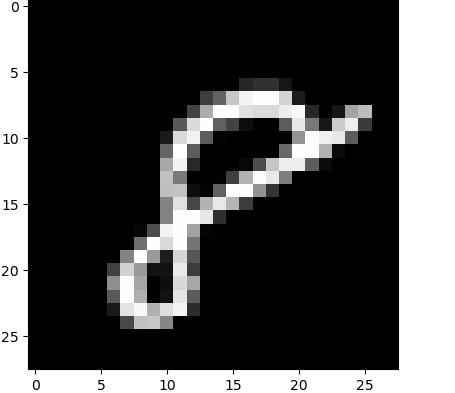
**11490434/11490434** ━━━━━━━━━━━━━━━━━━━━ **2s** 0us/step

(60000, 28, 28)

(10000, 28, 28)

(60000,)

(10000,)



Epoch 1/5

**1875/1875** ━━━━━━━━━ **4s** 2ms/step - loss: 0.4416 - sparse\_categorical\_accuracy: 0.8764

Epoch 2/5

**1875/1875** ━━━━━━━━━**3s** 2ms/step - loss: 0.1241 - sparse\_categorical\_accuracy: 0.9643

Epoch 3/5

**1875/1875** ━━━━━━━━━ **3s** 2ms/step - loss: 0.0801 - sparse\_categorical\_accuracy: 0.9761

Epoch 4/5

**1875/1875** ━━━━━━━━━ **4s** 2ms/step - loss: 0.0561 - sparse\_categorical\_accuracy: 0.9837

Epoch 5/5

**1875/1875** ━━━━━━━━━ **4s** 2ms/step - loss: 0.0434 - sparse\_categorical\_accuracy: 0.9864

**313/313** ━━━━━━━━━**1s** 2ms/step - loss: 0.0829 - sparse\_categorical\_accuracy: 0.9735

Test accuracy: 0.9776999950408936

**PROGRAM:**

**#1.b Back Propagation Using IRIS Dataset**

import tensorflow as tf

import numpy as np

from sklearn import datasets

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

iris = datasets.load\_iris()

X, y = iris.data, iris.target

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

print(f"X\_train shape: {X\_train.shape}")

print(f"X\_test shape: {X\_test.shape}")

print(f"y\_train shape: {y\_train.shape}")

print(f"y\_test shape: {y\_test.shape}")

hidden\_layer\_size = 10

model = tf.keras.Sequential([

tf.keras.layers.Dense(hidden\_layer\_size, activation='relu', input\_shape=(X\_train.shape[1],)),

tf.keras.layers.Dense(3, activation='softmax')

])

model.summary()

learning\_rate = 0.01

epochs = 1000

loss\_fn = tf.keras.losses.SparseCategoricalCrossentropy()

optimizer = tf.keras.optimizers.SGD(learning\_rate=learning\_rate)

train\_dataset = tf.data.Dataset.from\_tensor\_slices((X\_train, y\_train)).shuffle(100).batch(16)

test\_dataset = tf.data.Dataset.from\_tensor\_slices((X\_test, y\_test)).batch(16)

for epoch in range(epochs):

for step, (batch\_X, batch\_y) in enumerate(train\_dataset):

with tf.GradientTape() as tape:

logits = model(batch\_X, training=True)

loss\_value = loss\_fn(batch\_y, logits)

grads = tape.gradient(loss\_value, model.trainable\_variables)

optimizer.apply\_gradients(zip(grads, model.trainable\_variables))

if (epoch + 1) % 100 == 0:

print(f"Epoch {epoch + 1}/{epochs}, Loss: {loss\_value.numpy():.4f}")

test\_loss = loss\_fn(y\_test, model(X\_test, training=False)).numpy()

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

new\_samples = np.array([[6.7, 3.3, 5.7, 2.5]])

predictions = model(new\_samples)

predicted\_class = np.argmax(predictions.numpy(), axis=1)

print(f"Predicted Class: {predicted\_class}")

**OUTPUT:**

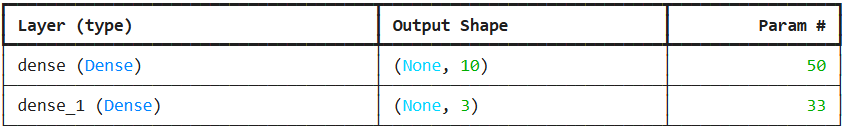
X\_train shape: (120, 4)

X\_test shape: (30, 4)

y\_train shape: (120,)

y\_test shape: (30,)

**Model: "sequential"**



**Total params:** 83 (332.00 B)

**Trainable params:** 83 (332.00 B)

**Non-trainable params:** 0 (0.00 B)

Epoch 100/1000, Loss: 0.0633

Epoch 200/1000, Loss: 0.1987

Epoch 300/1000, Loss: 0.0340

Epoch 400/1000, Loss: 0.0880

Epoch 500/1000, Loss: 0.0422

Epoch 600/1000, Loss: 0.0134

Epoch 700/1000, Loss: 0.1528

Epoch 800/1000, Loss: 0.1556

Epoch 900/1000, Loss: 0.0049

Epoch 1000/1000, Loss: 0.0123

Test Loss: 0.0809

Predicted Class: [2]

**PROGRAM:**

**#2.Naive Bayes Classifier**

import tensorflow as tf

import tensorflow\_probability as tfp

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import accuracy\_score

from sklearn import datasets, model\_selection

iris = datasets.load\_iris()

data = iris.data[:, :2]

targets = iris.target

x\_train, x\_test, y\_train, y\_test = model\_selection.train\_test\_split(data, targets, test\_size=0.2)

labels = {0: 'Setosa', 1: 'Versicolour', 2: 'Virginica'}

label\_colours = ['blue', 'red', 'green']

def plot\_data(x, y, labels, colours):

for y\_class in np.unique(y):

index = np.where(y == y\_class)

plt.scatter(x[index, 0], x[index, 1], label=labels[y\_class], c=colours[y\_class])

plt.title("Training set")

plt.xlabel("Sepal length (cm)")

plt.ylabel("Sepal width (cm)")

plt.legend()

plt.figure(figsize=(8, 5))

plot\_data(x\_train, y\_train, labels, label\_colours)

plt.show()

def learn\_parameters(x, y, mus, scales, optimiser, epochs):

@tf.function

def nll(dist, x\_train, y\_train):

log\_probs = dist.log\_prob(x\_train)

L = len(tf.unique(y\_train)[0])

y\_train = tf.one\_hot(indices=y\_train, depth=L)

return -tf.reduce\_mean(log\_probs \* y\_train)

@tf.function

def get\_loss\_and\_grads(dist, x\_train, y\_train):

with tf.GradientTape() as tape:

tape.watch(dist.trainable\_variables)

loss = nll(dist, x\_train, y\_train)

grads = tape.gradient(loss, dist.trainable\_variables)

return loss, grads

nll\_loss = []

mu\_values = []

scales\_values = []

x = tf.cast(np.expand\_dims(x, axis=1), tf.float32)

dist = tfd.MultivariateNormalDiag(loc=mus, scale\_diag=scales)

for epoch in range(epochs):

loss, grads = get\_loss\_and\_grads(dist, x, y)

optimiser.apply\_gradients(zip(grads, dist.trainable\_variables))

nll\_loss.append(loss)

mu\_values.append(mus.numpy())

scales\_values.append(scales.numpy())

nll\_loss, mu\_values, scales\_values = np.array(nll\_loss), np.array(mu\_values), np.array(scales\_values)

return (nll\_loss, mu\_values, scales\_values, dist)

mus = tf.Variable([[1., 1.], [1., 1.], [1., 1.]])

scales = tf.Variable([[1., 1.], [1., 1.], [1., 1.]])

opt = tf.keras.optimizers.Adam(learning\_rate=0.005)

epochs = 10000

nlls, mu\_arr, scales\_arr, class\_conditionals = learn\_parameters(x\_train, y\_train, mus, scales, opt, epochs)

fig, ax = plt.subplots(1, 3, figsize=(15, 4))

ax[0].plot(nlls)

ax[0].set\_title("Loss vs. epoch")

ax[0].set\_xlabel("Epoch")

ax[0].set\_ylabel("Negative log-likelihood")

for k in [0, 1, 2]:

ax[1].plot(mu\_arr[:, k, 0])

ax[1].plot(mu\_arr[:, k, 1])

ax[1].set\_title("ML estimates for model's\nmeans vs. epoch")

ax[1].set\_xlabel("Epoch")

ax[1].set\_ylabel("Means")

for k in [0, 1, 2]:

ax[2].plot(scales\_arr[:, k, 0])

ax[2].plot(scales\_arr[:, k, 1])

ax[2].set\_title("ML estimates for model's\nscales vs. epoch")

ax[2].set\_xlabel("Epoch")

ax[2].set\_ylabel("Scales")

plt.show()

print("Class conditional means:")

print(class\_conditionals.loc.numpy())

print("\nClass conditional standard deviations:")

print(class\_conditionals.stddev().numpy())

def get\_prior(y):

counts = np.bincount(y)

dist = tfd.Categorical(probs=counts/len(y))

return dist

prior = get\_prior(y\_train)

def predict\_class(prior, class\_conditionals, x):

def predict\_fn(myx):

class\_probs = class\_conditionals.prob(tf.cast(myx, dtype=tf.float32))

prior\_probs = tf.cast(prior.probs, dtype=tf.float32)

class\_times\_prior\_probs = class\_probs \* prior\_probs

Q = tf.reduce\_sum(class\_times\_prior\_probs)

P = tf.math.divide(class\_times\_prior\_probs, Q)

Y = tf.cast(tf.argmax(P), dtype=tf.float64)

return Y

y = tf.map\_fn(predict\_fn, x)

return y

predictions = predict\_class(prior, class\_conditionals, x\_test)

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

print("Test accuracy: {:.4f}".format(accuracy))

def contour\_plot(x0\_range, x1\_range, prob\_fn, batch\_shape, levels=None, n\_points=100):

X0, X1 = get\_meshgrid(x0\_range, x1\_range, n\_points=n\_points)

X\_values = np.expand\_dims(np.array([X0.ravel(), X1.ravel()]).T, axis=1)

Z = prob\_fn(X\_values)

Z = np.array(Z).T.reshape(batch\_shape, \*X0.shape)

for batch in np.arange(batch\_shape):

plt.contourf(X0, X1, Z[batch], alpha=0.3, levels=levels)

**OUTPUT:**

Class conditional means: [[5.0550003 3.4325001]

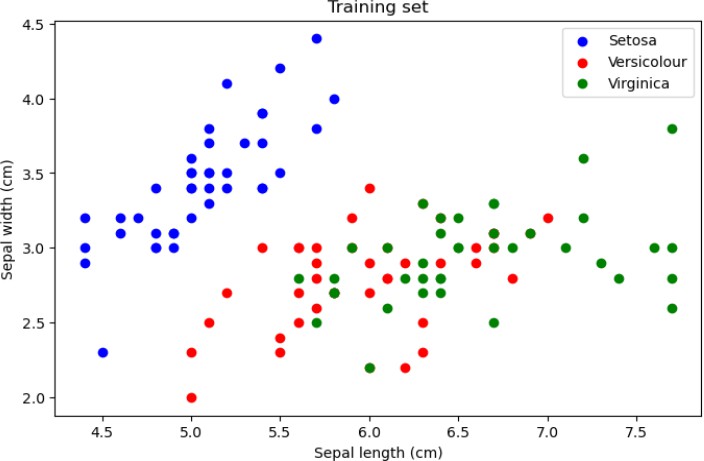
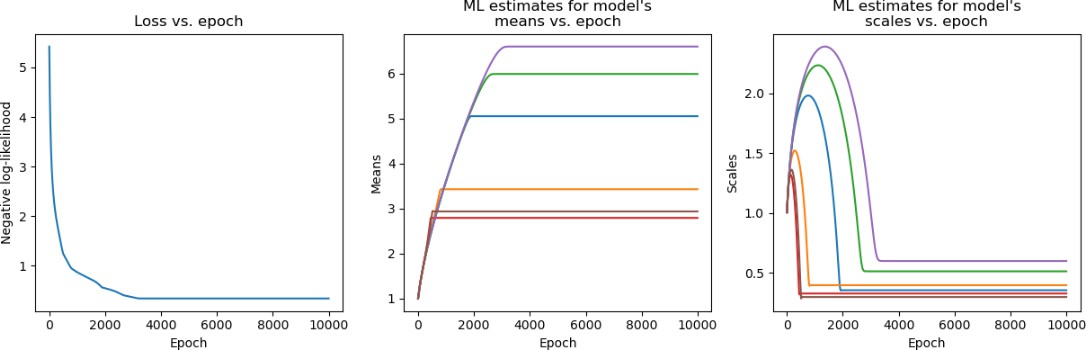
[5.992683 2.795122 ]

[6.602564 2.9384618]]

Class conditional standard deviations: [[0.35351804 0.39584568]

[0.51199764 0.32606354]

[0.5980683 0.2966611 ]]

Test accuracy: 0.8667

**PROGRAM:**

**#3.Skip Gram Model Using NLP**

import tensorflow as tf

import numpy as np

corpus = ["I like playing football with my friends",

"I enjoy playing tennis",

"I hate swimming",

"I love basketball"]

window\_size = 3

embedding\_dim = 50

batch\_size = 16

epochs = 100

learning\_rate = 0.01

tokenizer = tf.keras.preprocessing.text.Tokenizer()

tokenizer.fit\_on\_texts(corpus)

vocab\_size = len(tokenizer.word\_index) + 1

sequences = tokenizer.texts\_to\_sequences(corpus)

data = []

for seq in sequences:

for i in range(len(seq)):

for j in range(max(0, i - window\_size), min(len(seq), i + window\_size + 1)):

if i != j:

data.append([seq[i], seq[j]])

data = np.array(data)

x\_train = data[:, 0]

y\_train = data[:, 1]

inputs = tf.keras.layers.Input(shape=(1,))

embeddings = tf.keras.layers.Embedding(vocab\_size, embedding\_dim)(inputs)

flatten = tf.keras.layers.Flatten()(embeddings)

output = tf.keras.layers.Dense(vocab\_size, activation='softmax')(flatten)

model = tf.keras.models.Model(inputs=inputs, outputs=output)

model.compile(loss='sparse\_categorical\_crossentropy',

optimizer=tf.keras.optimizers.Adam(learning\_rate=learning\_rate))

model.fit(x\_train, y\_train, batch\_size=batch\_size, epochs=epochs)

word\_embeddings = model.get\_layer(index=1).get\_weights()[0]

def get\_vector(word):

idx = tokenizer.word\_index.get(word)

    if idx is not None:

        return word\_embeddings[idx]

    else:

        return None

word = "football"

vector = get\_vector(word)

print(f"Vector representation of '{word}': {vector}")

def get\_context\_words(word):

idx = tokenizer.word\_index[word]

context\_indices = list(range(max(0, idx - window\_size), min(vocab\_size, idx + window\_size + 1)))

context\_words = [word for word, index in tokenizer.word\_index.items() if index in context\_indices]

return context\_words

focus\_word = "playing"

context\_words = get\_context\_words(focus\_word)

print(f"Context words for '{focus\_word}': {context\_words}")

**OUTPUT:**

Epoch 1/100

4/4 [==============================] - 1s 5ms/step - loss: 2.6431

Epoch 2/100

4/4 [==============================] - 0s 4ms/step - loss: 2.5489

Epoch 3/100

4/4 [==============================] - 0s 3ms/step - loss: 2.4655

Epoch 4/100

4/4 [==============================] - 0s 4ms/step - loss: 2.3813

Epoch 5/100

4/4 [==============================] - 0s 4ms/step - loss: 2.2917

Vector representation of 'football': [ 0.17141712 0.30853176 0.22219557 0.17984286 -0.24223702 0.3068344

0.03928155 -0.6306034 -0.08273036 0.24466318 -0.08293969 -0.17579114

0.14115772 0.00276356 -0.11821245 -0.09002695 -0.34314537 0.45394325

0.16790457 0.04163431 0.07040803 -0.29762593 -0.1315126 0.717115

0.17201905 -0.08250948 0.23961464 0.09098674 0.3917652 -0.20037314

0.47015923 -0.26363885 -0.36365074 -0.23005496 -0.572872 -0.11484142

0.25438645 -0.20655242 0.00347094 -0.20453684 -0.19304553 -0.22932066

0.32481378 -0.15313971 0.24397568 -0.6262668 0.09316769 -0.5698216

0.06133474 0.37273604]

Context words for 'playing': ['i', 'playing', 'like', 'football', 'with']

**PROGRAM:**

**#4.a Object Detection**

import cv2

import numpy as np

import os

yolov3\_config = r'C:\Users\ygoku\Desktop\object detection\yolov3.cfg'

yolov3\_weights = r'C:\Users\ygoku\Desktop\object detection\yolov3.weights'

coco\_names\_path = r'C:\Users\ygoku\Desktop\object detection\coco.names'

net = cv2.dnn.readNet(yolov3\_weights, yolov3\_config)

classes = []

with open(coco\_names\_path, "r") as f:

classes = f.read().strip().split('\n')

image\_filename = "signs-landing.jpg"

desktop\_folder = os.path.expanduser("~/Desktop")

image\_path = os.path.join(desktop\_folder, image\_filename)

if not os.path.isfile(image\_path):

print(f"Image file '{image\_path}' not found.")

else:

image = cv2.imread(image\_path)

height, width, \_ = image.shape

blob = cv2.dnn.blobFromImage(image, 1/255.0, (416, 416), swapRB=True, crop=False)

net.setInput(blob)

layer\_names = net.getUnconnectedOutLayersNames()

outs = net.forward(layer\_names)

class\_ids = []

confidences = []

boxes = []

conf\_threshold = 0.5

nms\_threshold = 0.4

for out in outs:

for detection in out:

scores = detection[5:]

class\_id = np.argmax(scores)

confidence = scores[class\_id]

if confidence > conf\_threshold:

center\_x = int(detection[0] \* width)

center\_y = int(detection[1] \* height)

w = int(detection[2] \* width)

h = int(detection[3] \* height)

x = int(center\_x - w / 2)

y = int(center\_y - h / 2)

boxes.append([x, y, w, h])

confidences.append(float(confidence))

class\_ids.append(class\_id)

indices = cv2.dnn.NMSBoxes(boxes, confidences, conf\_threshold, nms\_threshold)

for i in indices:

box = boxes[i]

x, y, w, h = box

label = str(classes[class\_ids[i]])

confidence = confidences[i]

color = (0, 255, 0)

cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)

cv2.putText(image, f"{label} {confidence:.2f}", (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, color, 2)

cv2.imshow("Object Detection", image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**OUTPUT:**

**PROGRAM:**

**#4.b Object Detection(Video)**

import cv2

import numpy as np

net = cv2.dnn.readNet("yolov4.weights", "yolov4.cfg")

layer\_names = net.getLayerNames()

output\_layers = [layer\_names[i - 1] for i in net.getUnconnectedOutLayers()]

with open("coco.names", "r") as f:

classes = [line.strip() for line in f.readlines()]

cap = cv2.VideoCapture("traffic\_video.mp4")

while True:

ret, frame = cap.read()

if not ret:

break

blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

net.setInput(blob)

outs = net.forward(output\_layers)

class\_ids = []

confidences = []

boxes = []

for out in outs:

for detection in out:

scores = detection[5:]

class\_id = np.argmax(scores)

confidence = scores[class\_id]

if confidence > 0.5:

center\_x = int(detection[0] \* frame.shape[1])

center\_y = int(detection[1] \* frame.shape[0])

w = int(detection[2] \* frame.shape[1])

h = int(detection[3] \* frame.shape[0])

x = center\_x - w // 2

y = center\_y - h // 2

boxes.append([x, y, w, h])

confidences.append(float(confidence))

class\_ids.append(class\_id)

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

if len(indexes) > 0:

for i in indexes.flatten():

x, y, w, h = boxes[i]

label = str(classes[class\_ids[i]])

color = (0, 255, 0)

cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)

cv2.putText(frame, label, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, color, 2)

cv2.imshow("Traffic Analysis", frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

**OUTPUT:**

**PROGRAM:**

**#5.Sentiments Analysis Using LSTM**

import os

import cv2

import numpy as np

import matplotlib.pyplot as plt

from deepface import DeepFace

os.environ["KMP\_DUPLICATE\_LIB\_OK"] = "TRUE"

image\_path = "D:\\DL\\images.jpg"

img = cv2.imread(image\_path)

if img is None:

raise ValueError(f"Error loading image at path: {image\_path}")

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

try:

result = DeepFace.analyze(img\_path=image\_path, actions=['emotion'], enforce\_detection=False)

dominant\_emotion = result[0]['dominant\_emotion']

except Exception as e:

raise ValueError(f"Error in DeepFace analysis: {e}")

emotion\_to\_sentiment = {

"happy": "Positive",

"surprise": "Positive",

"neutral": "Neutral",

"sad": "Negative",

"angry": "Negative",

"fear": "Negative",

"disgust": "Negative"

}

final\_sentiment = emotion\_to\_sentiment.get(dominant\_emotion, "Neutral")

font = cv2.FONT\_HERSHEY\_SIMPLEX

cv2.putText(img, f"Emotion: {dominant\_emotion}", (10, 50), font, 1, (0, 255, 0), 2, cv2.LINE\_AA)

cv2.putText(img, f"Sentiment: {final\_sentiment}", (10, 90), font, 1, (255, 0, 0), 2, cv2.LINE\_AA)

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

plt.imshow(img\_rgb\_overlay)

plt.title(f"Emotion: {dominant\_emotion}, Sentiment: {final\_sentiment}")

plt.axis("off")

plt.show()

print("Emotion Analysis Result:", result)

print("Predicted Sentiment:", final\_sentiment)

**OUTPUT:**

0%| | 0.00/5.98M [00:00<?, ?B/s]

9%|8 | 524k/5.98M [00:00<00:01, 3.62MB/s]

26%|##6 | 1.57M/5.98M [00:00<00:00, 5.71MB/s]

53%|#####2 | 3.15M/5.98M [00:00<00:00, 8.77MB/s]

88%|########7 | 5.24M/5.98M [00:00<00:00, 11.9MB/s]

100%|##########| 5.98M/5.98M [00:00<00:00, 10.4MB/s]

**PROGRAM:**

**#6.a Mnist Dataset Autoencoders**

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

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

x\_train = x\_train.astype('float32') / 255.

x\_test = x\_test.astype('float32') / 255.

x\_train = x\_train.reshape((len(x\_train), np.prod(x\_train.shape[1:])))

x\_test = x\_test.reshape((len(x\_test), np.prod(x\_test.shape[1:])))

input\_dim = x\_train.shape[1]

encoding\_dim = 32

input\_img = tf.keras.Input(shape=(input\_dim,))

encoded = tf.keras.layers.Dense(encoding\_dim, activation='relu')(input\_img)

decoded = tf.keras.layers.Dense(input\_dim, activation='sigmoid')(encoded)

autoencoder = tf.keras.Model(input\_img, decoded)

autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

autoencoder.fit(x\_train, x\_train, epochs=50, batch\_size=256, shuffle=True, validation\_data=(x\_test, x\_test))

reconstructed\_images = autoencoder.predict(x\_test)

n = 10

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

for i in range(n):

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

plt.imshow(x\_test[i].reshape(28, 28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

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

plt.imshow(reconstructed\_images[i].reshape(28, 28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

plt.show()

**OUTPUT:**

Epoch 1/50

235/235 [==============================] - 1s 4ms/step - loss: 0.2802 - val\_loss: 0.1958

Epoch 2/50

235/235 [==============================] - 1s 4ms/step - loss: 0.1749 - val\_loss: 0.1560

Epoch 3/50

235/235 [==============================] - 1s 4ms/step - loss: 0.1461 - val\_loss: 0.1348

Epoch 4/50

235/235 [==============================] - 1s 4ms/step - loss: 0.1296 - val\_loss: 0.1221

Epoch 5/50

235/235 [==============================] - 1s 4ms/step - loss: 0.1190 - val\_loss: 0.1136

Epoch 6/50

235/235 [==============================] - 1s 4ms/step - loss: 0.1117 - val\_loss: 0.1075

Epoch 7/50

235/235 [==============================] - 1s 4ms/step - loss: 0.1066 - val\_loss: 0.1033

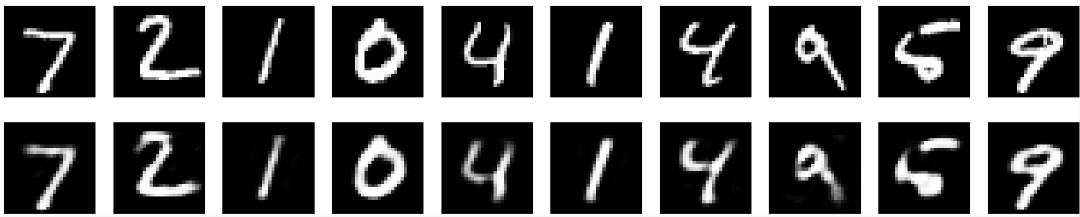
Epoch 8/50

235/235 [==============================] - 1s 4ms/step - loss: 0.1028 - val\_loss: 0.1000

Epoch 9/50

235/235 [==============================] - 1s 4ms/step - loss: 0.1000 - val\_loss: 0.0975

Epoch 10/50

235/235 [==============================] - 1s 4ms/step - loss: 0.0979 - val\_loss: 0.0959

**PROGRAM:**

**#6.b Medical X-ray Image**

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train\_path = "D:\\DL\\chest\_xray\\train"

test\_path = "D:\\DL\\chest\_xray\\test"

valid\_path = "D:\\DL\\chest\_xray\\val"

batch\_size = 16

image\_gen = ImageDataGenerator(

rescale=1./255,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True

)

test\_data\_gen = ImageDataGenerator(rescale=1./255)

train = image\_gen.flow\_from\_directory(

train\_path,

target\_size=(99, 128),

color\_mode='grayscale',

class\_mode='binary',

batch\_size=batch\_size

)

test = test\_data\_gen.flow\_from\_directory(

test\_path,

target\_size=(99, 128),

color\_mode='grayscale',

shuffle=False,

class\_mode='binary',

batch\_size=batch\_size

)

valid = test\_data\_gen.flow\_from\_directory(

valid\_path,

target\_size=(99, 128),

color\_mode='grayscale',

class\_mode='binary',

batch\_size=batch\_size

)

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

for i in range(10):

plt.subplot(2, 5, i + 1)

for X\_batch, Y\_batch in train:

image = X\_batch[0]

dic = {0: 'NORMAL', 1: 'PNEUMONIA'}

plt.title(dic.get(Y\_batch[0]))

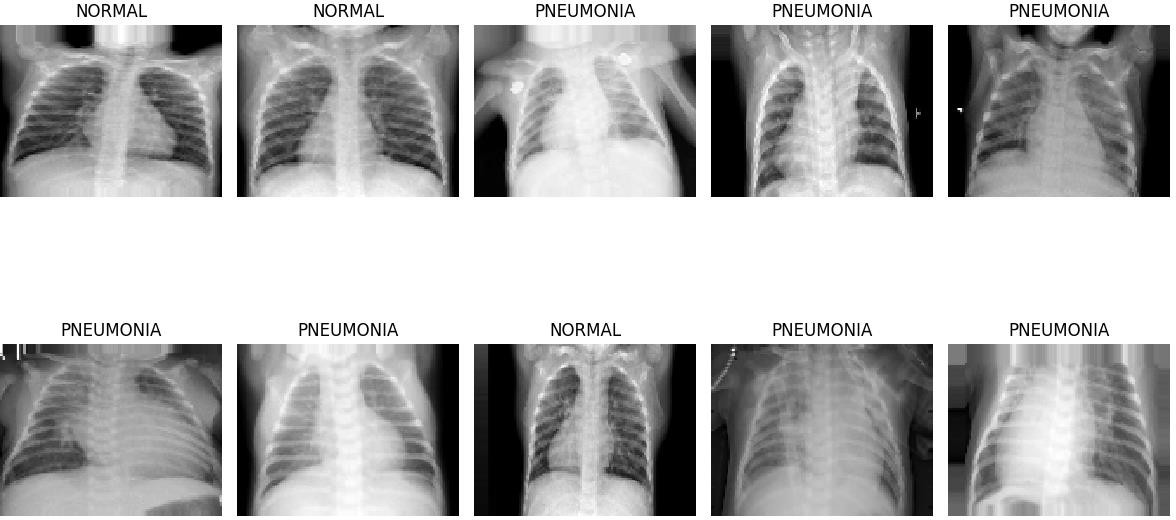
plt.axis('off')

plt.imshow(np.squeeze(image), cmap='gray', interpolation='nearest')

break

plt.tight\_layout()

plt.show()

**OUTPUT:**

**PROGRAM:**

**#7.Continuous Bag Of Words Using CNN**

import tensorflow as tf

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.preprocessing.text import Tokenizer

import numpy as np

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

corpus = [

"The quick brown fox jumps over the lazy dog.",

"A fast brown dog jumps over the lazy cat.",

"The speedy black cat jumps over the lazy dog."

]

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(corpus)

sequences = tokenizer.texts\_to\_sequences(corpus)

vocab\_size = len(tokenizer.word\_index) + 1

embedding\_size = 10

window\_size = 2

contexts, targets = [], []

for sequence in sequences:

for i in range(window\_size, len(sequence) - window\_size):

context = sequence[i - window\_size:i] + sequence[i + 1:i + window\_size + 1]

target = sequence[i]

contexts.append(context)

targets.append(target)

X = np.array(contexts)

model = Sequential()

model.add(Embedding(input\_dim=vocab\_size, output\_dim=embedding\_size, input\_length=2 \* window\_size))

model.add(Lambda(lambda x: tf.reduce\_mean(x, axis=1)))

model.add(Dense(units=vocab\_size, activation='softmax'))

model.save\_weights('cbow\_weights.h5')

model.load\_weights('cbow\_weights.h5')

embeddings = model.get\_weights()[0]

pca = PCA(n\_components=2)

reduced\_embeddings = pca.fit\_transform(embeddings)

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

for i, word in enumerate(tokenizer.word\_index.keys()):

x, y = reduced\_embeddings[i]

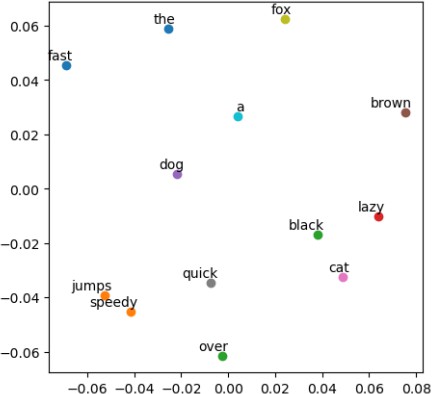
plt.scatter(x, y)

plt.annotate(word, xy=(x, y), xytext=(5, 2), textcoords='offset points', ha='right', va='bottom')

plt.show()

**OUTPUT:**

After converting our words in the corpus into a vector of integers:

 [[1, 8, 6, 9, 2, 3, 1, 4, 5],[10, 11, 6, 5, 2, 3, 1, 4, 7], [1, 12, 13, 7, 2, 3, 1, 4, 5]]

**PROGRAM:**

**#8.Cat or Dog**

import os

os.environ["KMP\_DUPLICATE\_LIB\_OK"] = "TRUE"

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.preprocessing.image import ImageDataGenerator, image

import numpy as np

import matplotlib.pyplot as plt

dataset\_path = 'Dataset'

img\_height, img\_width = 150, 150

batch\_size = 32

datagen = ImageDataGenerator(rescale=1.0/255, rotation\_range=20, width\_shift\_range=0.2, height\_shift\_range=0.2, shear\_range=0.2, zoom\_range=0.2, horizontal\_flip=True, validation\_split=0.2)

train\_generator = datagen.flow\_from\_directory(dataset\_path + '/train', target\_size=(img\_height, img\_width), batch\_size=batch\_size, class\_mode='binary', subset='training')

validation\_generator = datagen.flow\_from\_directory(dataset\_path + '/train', target\_size=(img\_height, img\_width), batch\_size=batch\_size, class\_mode='binary', subset='validation')

model = keras.Sequential([layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(img\_height, img\_width, 3)),

layers.MaxPooling2D(2, 2),

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

layers.MaxPooling2D(2, 2),

layers.Conv2D(128, (3, 3), activation='relu'),

layers.MaxPooling2D(2, 2),

layers.Flatten(),

layers.Dense(512, activation='relu'),

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

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

ehistory = model.fit(train\_generator, validation\_data=validation\_generator, epochs=10)

plt.plot(ehistory.history['accuracy'], label='train\_accuracy')

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

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

model.save('cat\_dog\_classifier.h5')

print("Model training complete. Saved as 'cat\_dog\_classifier.h5'")

model = tf.keras.models.load\_model('cat\_dog\_classifier.h5')

def prepare\_image(img\_path, img\_height=150, img\_width=150):

img = image.load\_img(img\_path, target\_size=(img\_height, img\_width))

img\_array = image.img\_to\_array(img) / 255.0

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

return img\_array

img\_path = 'dogtest.jpg'

img\_array = prepare\_image(img\_path)

prediction = model.predict(img\_array)

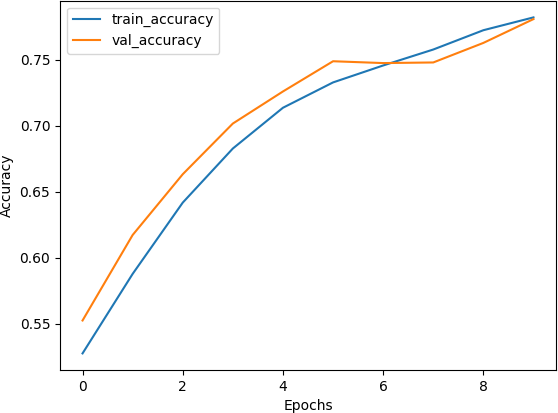
title = "Prediction: Dog" if prediction[0] > 0.5 else "Prediction: Cat"

img = image.load\_img(img\_path)

plt.imshow(img)

plt.title(title)

plt.show()

**OUTPUT:**

**PROGRAM:**

**#9.CIFAR-10**

import os

import numpy as np

import tensorflow as tf

import cv2

import matplotlib.pyplot as plt

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense

from tensorflow.keras.models import Model

from tensorflow.keras.utils import to\_categorical

os.environ["KMP\_DUPLICATE\_LIB\_OK"] = "TRUE"

(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

num\_classes = 10

y\_train = to\_categorical(y\_train, num\_classes)

y\_test = to\_categorical(y\_test, num\_classes)

class\_labels = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog', 'Horse', 'Ship', 'Truck']

input\_layer = Input(shape=(32, 32, 3))

conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(input\_layer)

pool1 = MaxPooling2D((2, 2))(conv1)

conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(pool1)

pool2 = MaxPooling2D((2, 2))(conv2)

flatten = Flatten()(pool2)

dense1 = Dense(64, activation='relu')(flatten)

output\_layer = Dense(num\_classes, activation='softmax')(dense1)

model = Model(inputs=input\_layer, outputs=output\_layer)

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

model.fit(x\_train, y\_train, epochs=10, batch\_size=32, validation\_data=(x\_test, y\_test))

def predict\_custom\_image(img\_path):

img = cv2.imread(img\_path)

if img is None:

raise FileNotFoundError(f"Image not found at path: {img\_path}")

img = cv2.resize(img, (32, 32))

img = img.astype('float32') / 255.0

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

prediction = model.predict(img)

predicted\_class\_idx = np.argmax(prediction)

predicted\_class = class\_labels[predicted\_class\_idx]

img\_display = cv2.cvtColor(cv2.imread(img\_path), cv2.COLOR\_BGR2RGB)

plt.imshow(img\_display)

plt.title(f'Predicted Class: {predicted\_class}')

plt.axis('off')

plt.show()

print(f'Predicted Class: {predicted\_class}')

predict\_custom\_image(r"Automobile.jpg")

**OUTPUT:**

Epoch 1/10

1563/1563 [==============================] - 14s 9ms/step - loss: 1.4275 -accuracy: 0.4898

- val\_loss: 1.1954 - val\_accuracy: 0.5832

Epoch 2/10

1563/1563 [==============================] - 14s 9ms/step - loss: 1.0617-accuracy: 0.6271

- val\_loss: 1.0365 - val\_accuracy: 0.6359

Epoch 3/10

1563/1563 [==============================] - 16s 10ms/step - loss:0.9271- accuracy: 0.6766

-val\_loss: 0.9413 - val\_accuracy: 0.6670

Epoch 4/10

1563/1563 [==============================] - 16s 10ms/step - loss:0.8355- accuracy: 0.7089

- val\_loss: 0.9285 - val\_accuracy: 0.6838

Predicted Class: Automobile

**PROGRAM:**

**#10.a Human Face Detection**

import os

import cv2

import matplotlib.pyplot as plt

os.environ["KMP\_DUPLICATE\_LIB\_OK"] = "TRUE"

image\_path = "group photo.jpg"

img = cv2.imread(image\_path)

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

face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade\_frontalface\_default.xml")

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

faces = face\_cascade.detectMultiScale(gray, 1.1, 4)

for (x, y, w, h) in faces:

cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)

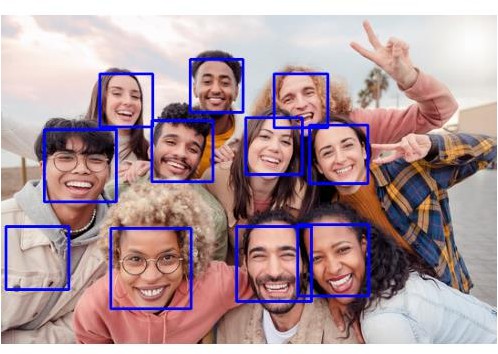
img\_rgb\_overlay = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

plt.imshow(img\_rgb\_overlay)

plt.axis("off")

plt.show()

print("Detected Faces:", faces)

**OUTPUT:**

**PROGRAM:**

**#10.b Human Face Detection**

import torch

from torchvision.models.detection import fasterrcnn\_resnet50\_fpn

from torchvision.transforms import functional as F

from PIL import Image

import matplotlib.pyplot as plt

import matplotlib.patches as patches

model = fasterrcnn\_resnet50\_fpn(pretrained=True)

model.eval()

image\_path = "h1.png"

image = Image.open(image\_path).convert("RGB")

image\_tensor = F.to\_tensor(image).unsqueeze(0)

with torch.no\_grad():

predictions = model(image\_tensor)

boxes = predictions[0]["boxes"]

labels = predictions[0]["labels"]

scores = predictions[0]["scores"]

threshold = 0.8

human\_boxes = [box for i, box in enumerate(boxes) if labels[i] == 1 and scores[i] > threshold]

fig, ax = plt.subplots(1, figsize=(10, 6))

ax.imshow(image)

for box in human\_boxes:

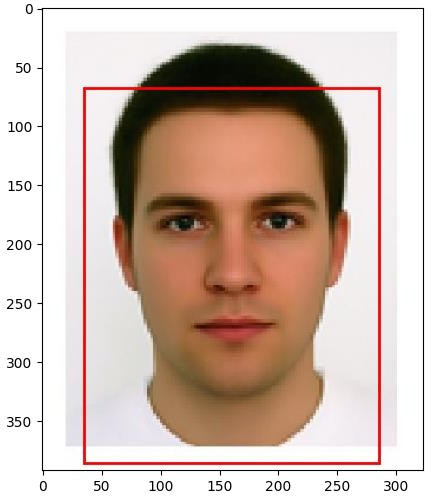
x1, y1, x2, y2 = box.tolist()

rect = patches.Rectangle((x1, y1), x2 - x1, y2 - y1, linewidth=2, edgecolor="red", facecolor="none")

ax.add\_patch(rect)

plt.axis("off")

plt.show()

**OUTPUT:**

**PROGRAM:**

**#11.Chat Bot**

import google.generativeai as genai

API\_KEY = 'Your-API-Key'

genai.configure(api\_key=API\_KEY)

def chatbot():

print("Chatbot: Hello! Type 'exit' to end the chat.")

model = genai.GenerativeModel("gemini-1.5-pro-latest")

while True:

user\_input = input("You: ")

if user\_input.lower() == "exit":

print("Chatbot: Goodbye!")

break

response = model.generate\_content(user\_input)

print("Chatbot:", response.text)

chatbot()

**OUTPUT:**

Chatbot: Hello! Type 'exit' to end the chat.

You: hi

Chatbot: Hi there! How can I help you today?

You: who is prime minister of india

Chatbot: The current Prime Minister of India is Narendra Modi.

You: what is weather in chennai

Chatbot: Chennai's weather is typically hot and humid.