## Slip 1

Q.1) Create a simple chatbot using Python and natural language processing libraries

to engage in conversation with users. (pip install nltk)

a. Import all necessary libraries

b. Generate some patterns/prompts

c. Generate some Response

d. Make conversion

e. End conversion

import nltk

from nltk.chat.util import Chat, reflections

# Define pairs of patterns and responses

pairs = [

["hi|hello|hey", ["Hello!", "Hey there!", "Hi!"]],

["how are you?q", ["I'm good, thanks!", "Doing well, thank you!", "I'm fine, how

about you?"]],

["what's your name?", ["I'm a chatbot.", "You can call me Chatbot.", "I'm

Chatbot!"]],

["quit|exit|bye", ["Goodbye!", "Bye!", "See you later!"]],

]

# Create a chatbot

chatbot = Chat(pairs, reflections)

# Start conversation

print("Welcome to the chatbot!")

print("Type 'quit' to exit.")

while True:

user\_input = input("You: ")

response = chatbot.respond(user\_input)

print("Chatbot:", response)

if user\_input.lower() in ["quit", "exit", "bye"]:

break

## Slip 2

Build a single layer Perceptron,Train the network on a simple dataset, such as

the XOR problem, and analyse the model's performance.

a. Import all necessary libraries

b. Define Dataset x= ([[0, 0], [0, 1], [1, 0], [1, 1]]) y= ([[0], [1], [1], [0]])

c. Preprocessing Dataset and Splitting Dataset into train and test

d. Build a Model and Compile model

e. Display model summary and accuracy.

f. Make Predictions

import tensorflow as tf

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Define the dataset for the XOR problem

x\_train = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

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

# Define the Single Layer Perceptron model

model = Sequential([

Dense(units=1, input\_shape=(2,), activation='sigmoid')

])

# Compile the model

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

metrics=['accuracy'])

# Train the model

model.fit(x\_train, y\_train, epochs=1000, verbose=0)

# Evaluate the model

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

print("Loss:", loss)

print("Accuracy:", accuracy)

# Predictions

predictions = model.predict(X)

print("Predictions:")

print(predictions)

## Slip 3:

Build a Multi Layer Perceptron , Train the network on a simple dataset,

Such as the XOR problem, and analyse the model's performance.

a. Import all necessary libraries

b. Loading Dataset

c. Preprocessing Dataset and Splitting Dataset into train and test

d. Build a Model and Compile model

e. Display model summary and accuracy.

f. Make Predictions

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Define the dataset (XOR problem)

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

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

# Define the Multi Layer Perceptron model

model = Sequential([

Dense(2, input\_shape=(2,), activation='relu'),

Dense(1, activation='sigmoid')

])

# Compile the model

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

metrics=['accuracy'])

# Train the model

model.fit(X, y, epochs=1000, verbose=0)

# Evaluate the model

loss, accuracy = model.evaluate(X, y)

print("Model performance:")

print("Loss:", loss)

print("Accuracy:", accuracy)

# Predictions

predictions = model.predict(X)

print("Predictions:")

print(predictions)

## Slip 4

**Build a simple LSTM model using TensorFlow and the IMDB Movie Reviews dataset.**

a. Import all necessary libraries

b. Loading Dataset

c. Preprocessing Dataset

d. Splitting Dataset into train and test

e. Build a Model and Compile model

f. Display model summary and accuracy

import tensorflow as tf

from tensorflow.keras.datasets import imdb

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

# Load and preprocess the IMDB dataset

(X\_train, y\_train), (X\_test, y\_test) = imdb.load\_data(num\_words=10000)

X\_train = pad\_sequences(X\_train, maxlen=200)

X\_test = pad\_sequences(X\_test, maxlen=200)

# Define and compile the LSTM model

model = tf.keras.Sequential([

tf.keras.layers.Embedding(input\_dim=10000, output\_dim=128,

input\_length=200),

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

tf.keras.layers.LSTM(32),

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

])

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

metrics=['accuracy'])

# Train and evaluate the model

model.fit(X\_train, y\_train, batch\_size=128, epochs=5, validation\_split=0.2)

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

print("Test Loss:", loss)

print("Test Accuracy:", accuracy)

## Slip 5

**Implement Sentiment analysis using LSTM model and IMDB Movie Reviews dataset. Predict whether sentiment is positive or negative based on given text review.**

a. Import all necessary libraries

b. Loading Dataset

c. Preprocessing Dataset and Splitting Dataset into train and test d. Build a Model and Compile model

e. Display model summary and accuracy.

f. Make Predictions

import tensorflow as tf

import numpy as np

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing.sequence import

pad\_sequences

# Load and preprocess the IMDB dataset

(X\_train, y\_train), (X\_test, y\_test) =

imdb.load\_data(num\_words=10000)

X\_train = pad\_sequences(X\_train, maxlen=200)

X\_test = pad\_sequences(X\_test, maxlen=200)

# Define and compile the LSTM model

model = tf.keras.Sequential([

tf.keras.layers.Embedding(input\_dim=10000, output\_dim=128,

input\_length=200),

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

tf.keras.layers.LSTM(32),

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

])

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

metrics=['accuracy'])

# Train and evaluate the model

model.fit(X\_train, y\_train, batch\_size=128, epochs=10,

validation\_split=0.2)

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

print("Test Loss:", loss)

print("Test Accuracy:", accuracy)

# Make predictions

text="I hate this movie."

# Convert text to sequence

tokenizer = imdb.get\_word\_index()

sequence = [tokenizer[word] if word in tokenizer and

tokenizer[word] < 10000 else 0 for word in text.split()]

# Padding sequence to match input\_length

sequence = pad\_sequences([sequence], maxlen=200)

# Predict

prediction = model.predict(sequence)

if prediction >= 0.5:

print('Sentiment is positive')

else:

print('Sentiment is negative')

## Slip 6

**Build a simple CNN model using MNIST dataset and display model summary and test model Accuracy.**

a. Import all necessary libraries

b. Loading Dataset

c. Preprocessing Dataset

d. Splitting Dataset into train and test

e. Build a Model and Compile model

f. Display model summary and accuracy.

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

import numpy as np

import cv2

# Load MNIST dataset

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

# Normalize pixel values to be between 0 and 1

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

# Reshape data to add a channel dimension (required by Conv2D layers)

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

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

# Build the CNN model

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")

])

# Compile the model

model.compile(optimizer="adam",

loss="sparse\_categorical\_crossentropy",

metrics=["accuracy"])

# Display model summary

model.summary()

# Train the model

model.fit(x\_train, y\_train, epochs=2, batch\_size=64, validation\_data=(x\_test, y\_test))

# Evaluate the model

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

print("Test accuracy:", test\_acc)

## Slip 7

**Build a CNN model for handwritten digit recognition using MNIST dataset and predict the digit based on given image..**

a. Import all necessary libraries

b. Loading Dataset

c. Preprocessing Dataset and Splitting Dataset into train and test d. Build a Model and Compile model

e. Display model summary and accuracy.

f. Make Predictions

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

import numpy as np

import cv2

# Load MNIST dataset

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

# Normalize pixel values to be between 0 and 1

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

# Reshape data to add a channel dimension (required by Conv2D layers)

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

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

# Build the CNN model

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")

])

# Compile the model

model.compile(optimizer="adam",

loss="sparse\_categorical\_crossentropy",

metrics=["accuracy"])

# Display model summary

model.summary()

# Train the model

model.fit(x\_train, y\_train, epochs=2, batch\_size=64, validation\_data=(x\_test, y\_test))

# Evaluate the model

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

print("Test accuracy:", test\_acc)

# Make predictions

img = cv2.imread("3.png")

img = cv2.resize(img, (28, 28)) # Resize image to fit model input shape

img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) # Convert to grayscale

img = img / 255.0 # Normalize pixel values

digit\_image = np.expand\_dims(img, axis=0) # Add batch dimension

digit\_image = digit\_image.reshape((1, 28, 28, 1))

# Make predictions

predictions = model.predict(digit\_image)

predicted\_digit = np.argmax(predictions)

print("Predicted digit:", predicted\_digit)

# Display the image

cv2.imshow("Image", img)

cv2.waitKey(0)

cv2.destroyAllWindows()

## Slip 8

**Build a CNN model for fashion Dress recognition using Fashion MNIST dataset and predict theclass based on given image.**

class\_names = [‘T-shirt/top’, ‘Trouser’, ‘Pullover’, ‘Dress’, ‘Coat’, ‘Sandal’, ‘Shirt’, ‘Sneaker’, ‘Bag’,‘Ankle boot’].

a. Import all necessary libraries

b. Loading Dataset

c. Preprocessing Dataset and Splitting Dataset into train and test d. Build a Model and Compile model

e. Display model summary and accuracy.

f. Make Predictions

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import fashion\_mnist

import numpy as np

import cv2

# Load Fashion MNIST dataset

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

# Normalize pixel values to be between 0 and 1

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

# Reshape data to add a channel dimension (required by Conv2D layers)

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

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

# Build the CNN model

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")

])

# Compile the model

model.compile(optimizer="adam",

loss="sparse\_categorical\_crossentropy",

metrics=["accuracy"])

# Display model summary

model.summary()

# Train the model

model.fit(x\_train, y\_train, epochs=20, batch\_size=64, validation\_data=(x\_test, y\_test))

# Evaluate the model

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

print("Test accuracy:", test\_acc)

# Load and preprocess the image

img = cv2.imread("boot.png")

img = cv2.resize(img, (28, 28)) # Resize image to fit model input shape

img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) # Convert to grayscale

img = img / 255.0 # Normalize pixel values

digit\_image = np.expand\_dims(img, axis=0) # Add batch dimension

digit\_image = digit\_image.reshape((1, 28, 28, 1))

# Make predictions

predictions = model.predict(digit\_image)

predicted\_class = np.argmax(predictions)

class\_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt", "Sneaker", "Bag", "Ankle

boot"]

print("Predicted class:", class\_names[predicted\_class])

# Display the image

cv2.imshow("Image", img)

cv2.waitKey(0)

cv2.destroyAllWindows()

## Slip 9

**Build a simple CNN model using CIFAR -10 dataset and display model summary and test model Accuracy.**

a. Import all necessary libraries

b. Loading Dataset

c. Preprocessing Dataset

d. Splitting Dataset into train and test

e. Build a Model and Compile model

f. Display model summary and accuracy.

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import cifar10

import numpy as np

import cv2

# Load CIFAR-10 dataset

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

# Build the CNN model

model = models.Sequential([

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

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")

])

# Compile the model

model.compile(optimizer="adam",

loss="sparse\_categorical\_crossentropy",

metrics=["accuracy"])

# Display model summary

model.summary()

# Train the model

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

# Evaluate the model

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

print("Test accuracy:", test\_acc)

## Slip 10

**Implement a deep learning model CNN to classify images from the CIFAR-10 dataset and predict the class based on given image.**

class\_names = [‘airplane’, ‘automobile’, ‘bird’, ‘cat’, ‘deer’, ‘dog’, ‘frog’, ‘horse’, ‘ship’, ‘truck’]

a. Import all necessary libraries

b. Loading Dataset

c. Preprocessing Dataset and Splitting Dataset into train and test d. Build a Model and Compile model

e. Display model summary and accuracy.

f. Make Predictions

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import cifar10

import numpy as np

import cv2

# Load CIFAR-10 dataset

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

# Build the CNN model

model = models.Sequential([

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

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")

])

# Compile the model

model.compile(optimizer="adam",

loss="sparse\_categorical\_crossentropy",

metrics=["accuracy"])

# Display model summary

model.summary()

# Train the model

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

# Evaluate the model

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

print("Test accuracy:", test\_acc)

from tensorflow.keras.preprocessing import image

img\_path = "Desktop\DL Data\cat1.png"

# Load and preprocess the image

img = image.load\_img(img\_path, target\_size=(32, 32))

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

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

img\_array = img\_array / 255.0 # Normalize pixel values

# Make predictions

predictions = model.predict(img\_array)

predicted\_class = np.argmax(predictions)

class\_names = [‘airplane’, “automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck"]

print("Predicted class:", class\_names[predicted\_class])

## SLIP 11

Q.1) Build a Single Layer Neural Network Model and find Net input of Network

using different input and weight vectors including bias value.[ Use bias =0.45 ]

Ans: 0.94

a. Import all necessary libraries

b. Build a input vector & Accept values from user. X = [0.2 0.6 0.1]^T

c. Build a weight vector and Accept values from user .W=[0.3 0.7 0.1]^T

d. Build a Model and Compile model.

e. Calculate Net input of Network

f. Print result.

import numpy as np

x = []

w=[]

# number of elements as input

n = int(input("Enter number of elements : "))

# iterating till the range

print("enter input vector")

for i in range(0, n):

ele = float(input())

# adding the element

x.append(ele)

print("enter weight vector")

for i in range(0, n):

ele = float(input())

w.append(ele)

print("input vector=",x)

print("weight vector=",w)

bias= float(input("Enter Bias value: "))

X=np.array(x)

W=np.array(w)

y=X.dot(W)+bias

print("Net INput=",y)

## SLIP 12

Q.1) Build a Single Layer Neural Network Model and find output of different input

and weight variables including bias value using Activation function.

[ Use data: bias=0.53 ,binary=1/(1+np.exp(-y)) , bipolar = (np.exp(y)-1)/(np.exp(y)+1) ]

Ans: 0.53 ,0.629 & 0.259

a. Import all necessary libraries

b. Build a input vector & Accept values from user. X=[0.8, 0.6 , 0.4]

c. Build a weight vector and Accept values from user.W= [0.1, 0.3 -0.2]

d. Build a Model and Compile model.

e. Calculate Net output of Network

f. Print result.

import numpy as np

x = []

w=[]

# number of elements as input

n = int(input("Enter number of elements : "))

# iterating till the range

print("enter input vector")

for i in range(0, n):

ele = float(input())

# adding the element

x.append(ele)

print("enter weight vector")

for i in range(0, n):

ele = float(input())

w.append(ele)

print("input vector=",x)

print("weight vector=",w)

bias= float(input("Enter Bias value: "))

X=np.array(x)

W=np.array(w)

y=X.dot(W)+bias

print("Output=",y)

binary=1/(1+np.exp(-y))#binary sigmoid

print("binary sigmoid",binary)

bipolar = (np.exp(y)-1)/(np.exp(y)+1) #bipolar sigmoid

print("bipolar sigmoid",bipolar)

## SLIP 13

Q.1) Build a Single Layer Neural Network Model and find net input of different

input and weight vectors.

W=[0.2 0.1 -0.3]^T & X = [0.3 0.5 0.6]^T Ans:-0.07

a. Import all necessary libraries

b. Build a input vector & Accept values from user.

c. Build a weight vector and Accept values from user

d. Build a Model and Compile model.

e. Calculate Net input of Network

f. Print result.

import numpy as np

x = []

w=[]

# number of elements as input

n = int(input("Enter number of elements : "))

# iterating till the range

print("enter input vector")

for i in range(0, n):

ele = float(input())

# adding the element

x.append(ele)

print("enter weight vector")

for i in range(0, n):

ele = float(input())

w.append(ele)

print("input vector=",x)

print("weight vector=",w)

bias= float(input("Enter Bias value: "))

X=np.array(x)

W=np.array(w)

y=X.dot(W)

print("Net Input=",y)

## SLIP 14

Q1. Build a simple Autoencoder model for Image Compression using given image

and display model summary and test model Accuracy.

a. Import all necessary libraries

b. Loading & Pre-processing Image

c. Define the autoencoder architecture (Encoder &Decoder Layer)

d. Define model &Compile model

e. Train the autoencoder & Create the autoencoder model

f. Display the compressed image/Save image.

import numpy as np

import tensorflow as tf

from tensorflow.keras import layers, models

import cv2

# Load and preprocess the image

image = cv2.imread("dog.jpg")

#image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB) # Convert to RGB format

image = cv2.resize(image, (256, 256)) # Resize to target size

image = image.astype(np.float32) / 255.0 # Normalize pixel values

# Define the autoencoder architecture

def autoencoder\_model():

# Encoder

encoder\_input = layers.Input(shape=(256, 256, 3))

x = layers.Conv2D(32, (3, 3), activation=&#39;relu&#39;, padding=&#39;same&#39;)(encoder\_input)

x = layers.MaxPooling2D((2, 2), padding=&#39;same&#39;)(x)

x = layers.Conv2D(16, (3, 3), activation=&#39;relu&#39;, padding=&#39;same&#39;)(x)

x = layers.MaxPooling2D((2, 2), padding=&#39;same&#39;)(x)

# Decoder

x = layers.Conv2D(16, (3, 3), activation=&#39;relu&#39;, padding=&#39;same&#39;)(x)

x = layers.UpSampling2D((2, 2))(x)

x = layers.Conv2D(32, (3, 3), activation=&#39;relu&#39;, padding=&#39;same&#39;)(x)

x = layers.UpSampling2D((2, 2))(x)

decoder\_output = layers.Conv2D(3, (3, 3), activation=&#39;sigmoid&#39;, padding=&#39;same&#39;)(x)

# Define model

autoencoder = models.Model(encoder\_input, decoder\_output)

return autoencoder

# Create the autoencoder model

model = autoencoder\_model()

model.compile(optimizer=&#39;adam&#39;, loss=&#39;mse&#39;)

# Reshape the image for training

image = np.expand\_dims(image, axis=0) # Add batch dimension

# Train the autoencoder

model.fit(image, image, epochs=50, batch\_size=1, verbose=1)

# Encode and decode the image

encoded\_image = model.predict(image)

decoded\_image = encoded\_image[0]

# Display the decoded image

cv2.imshow("Decoded Image", decoded\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Save the decoded image

# Convert the decoded image back to BGR

#decoded\_image\_bgr = cv2.cvtColor(decoded\_image, cv2.COLOR\_RGB2BGR)

#decoded\_image = tf.keras.preprocessing.image.array\_to\_img(decoded\_image)

#decoded\_image.save("decoded\_image.jpg")

## SLIP 15

Q1. Build a simple Autoencoder model for Image Denosing using given image and

display model summary and test model Accuracy.

a. Import all necessary libraries

b. Loading & Pre-processing Image

c. Define the autoencoder architecture (Encoder &Decoder Layer)

d. Define model &Compile the Model

e. Train the autoencoder & Create the denoising autoencoder model

f. Display the Noisy image and Denoised image / Save image.

import numpy as np

import tensorflow as tf

from tensorflow.keras import layers, models

import cv2

# Load and preprocess the image

image = cv2.imread("dog.jpg")

image = cv2.resize(image, (256, 256)) # Resize to target size

image = image.astype(np.float32) / 255.0 # Normalize pixel values

# Adding Gaussian noise to the image

noise\_factor = 0.5

noisy\_image = image + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=image.shape)

noisy\_image = np.clip(noisy\_image, 0., 1.)

# Define the denoising autoencoder architecture

def denoising\_autoencoder\_model():

# Encoder

encoder\_input = layers.Input(shape=(256, 256, 3))

x = layers.Conv2D(32, (3, 3), activation=&#39;relu&#39;, padding=&#39;same&#39;)(encoder\_input)

x = layers.MaxPooling2D((2, 2), padding=&#39;same&#39;)(x)

x = layers.Conv2D(16, (3, 3), activation=&#39;relu&#39;, padding=&#39;same&#39;)(x)

x = layers.MaxPooling2D((2, 2), padding=&#39;same&#39;)(x)

# Decoder

x = layers.Conv2D(16, (3, 3), activation=&#39;relu&#39;, padding=&#39;same&#39;)(x)

x = layers.UpSampling2D((2, 2))(x)

x = layers.Conv2D(32, (3, 3), activation=&#39;relu&#39;, padding=&#39;same&#39;)(x)

x = layers.UpSampling2D((2, 2))(x)

decoder\_output = layers.Conv2D(3, (3, 3), activation=&#39;sigmoid&#39;, padding=&#39;same&#39;)(x)

# Define model

autoencoder = models.Model(encoder\_input, decoder\_output)

return autoencoder

# Create the denoising autoencoder model

model = denoising\_autoencoder\_model()

model.compile(optimizer=&#39;adam&#39;, loss=&#39;mse&#39;)

# Train the denoising autoencoder

model.fit(noisy\_image[np.newaxis, ...], image[np.newaxis, ...],

epochs=50, batch\_size=1, verbose=1)

# Display the denoised image

cv2.imshow("Noisy Image", noisy\_image)

# Denoise the noisy image

denoised\_image = model.predict(noisy\_image[np.newaxis, ...])[0]

# Display the denoised image

cv2.imshow("Denoised Image", denoised\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()