import zipfile

import os

from google.colab import drive

# Authenticate and mount Google Drive

drive.mount('/content/drive')

# Specify the zip file paths

zip\_file\_path1 = '/content/drive/MyDrive/Ground\_imgs.zip'  # Replace with your own first zip file path in Google Colab

zip\_file\_path2 = '/content/drive/MyDrive/Hazy\_imgs.zip'  # Replace with your own second zip file path in Google Colab

# Specify the destination folder names for extracted images

extracted\_folder\_name1 = 'Clear Images'  # Replace with your desired first folder name

extracted\_folder\_name2 = 'Hazy Images'  # Replace with your desired second folder name

# Function to create a folder in Google Drive

def create\_folder(folder\_name):

    drive.mount('/content/drive')

    try:

        os.makedirs(f'/content/drive/MyDrive/{folder\_name}')

        print(f"Folder '{folder\_name}' created in Google Drive")

        return f'/content/drive/MyDrive/{folder\_name}'

    except FileExistsError:

        print(f"Folder '{folder\_name}' already exists in Google Drive")

        return f'/content/drive/MyDrive/{folder\_name}'

# Function to extract .tif images from a zip file to a folder in Google Drive

def extract\_images\_to\_folder(zip\_file\_path, folder\_path):

    with zipfile.ZipFile(zip\_file\_path, 'r') as zip\_ref:

        for file in zip\_ref.namelist():

            if file.endswith('.tif'):

                zip\_ref.extract(file, folder\_path)

                print(f"File '{file}' extracted to folder '{folder\_path}'")

# Create the first destination folder in Google Drive

folder\_path1 = create\_folder(extracted\_folder\_name1)

# Extract .tif images from the first zip file to the first destination folder in Google Drive

if folder\_path1 is not None:

    extract\_images\_to\_folder(zip\_file\_path1, folder\_path1)

# Create the second destination folder in Google Drive

folder\_path2 = create\_folder(extracted\_folder\_name2)

# Extract .tif images from the second zip file to the second destination folder in Google Drive

if folder\_path2 is not None:

    extract\_images\_to\_folder(zip\_file\_path2, folder\_path2)

#2

import os

import cv2

import glob

# Define paths for input and output folders

clear\_folder = "/content/drive/MyDrive/Clear Images/Ground\_imgs"

hazy\_folder = "/content/drive/MyDrive/Hazy Images/Hazy\_imgs"

output\_clear\_folder = "/content/drive/MyDrive/Preprocessed Images/Clear Images"

output\_hazy\_folder = "/content/drive/MyDrive/Preprocessed Images/Hazy Images"

# Preprocessing function

def preprocess\_image(image):

    denoised\_image = cv2.fastNlMeansDenoisingColored(image, None, 10, 10, 7, 21)

    # Adjust contrast

    alpha = 1.5  # Contrast control (1.0-3.0)

    beta = 0  # Brightness control (0-100)

    contrast\_adjusted\_image = cv2.convertScaleAbs(denoised\_image, alpha=alpha, beta=beta)

    return contrast\_adjusted\_image

# Process images in clear folder

for clear\_image\_path in glob.glob(os.path.join(clear\_folder, "\*.tif")):

    # Load image

    clear\_image = cv2.imread(clear\_image\_path)

    # Perform preprocessing

    clear\_image = preprocess\_image(clear\_image)

    # Save preprocessed image to output folder

    output\_clear\_image\_path = os.path.join(output\_clear\_folder, os.path.basename(clear\_image\_path))

    cv2.imwrite(output\_clear\_image\_path, clear\_image)

# Process images in hazy folder

for hazy\_image\_path in glob.glob(os.path.join(hazy\_folder, "\*.tif")):

    # Load image

    hazy\_image = cv2.imread(hazy\_image\_path)

    # Perform preprocessing

    hazy\_image = preprocess\_image(hazy\_image)

    # Save preprocessed image to output folder

    output\_hazy\_image\_path = os.path.join(output\_hazy\_folder, os.path.basename(hazy\_image\_path))

    cv2.imwrite(output\_hazy\_image\_path, hazy\_image)

#3

import os

import numpy as np

from PIL import Image

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

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, Activation, Add

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import ModelCheckpoint

#Define the ResNet block

def resnet\_block(inputs, filters, kernel\_size):

  x = Conv2D(filters, kernel\_size, padding='same')(inputs)

  x = BatchNormalization()(x)

  x = Activation('relu')(x)

  x = Conv2D(filters, kernel\_size, padding='same')(x)

  x = BatchNormalization()(x)

  x = Add()([x, inputs])

  x = Activation('relu')(x)

  return x

#4

#Define the ResNet model

def resnet\_model(input\_shape):

  inputs = Input(shape=input\_shape)

  x = Conv2D(32, 3, padding='same')(inputs)

  x = BatchNormalization()(x)

  x = Activation('relu')(x)

  for i in range(5):

    x = resnet\_block(x, 32, 3)

  x = Conv2D(3, 3, padding='same')(x)

  outputs = Activation('sigmoid')(x)

  model = Model(inputs=inputs, outputs=outputs)

  return model

#5

import os

import numpy as np

from PIL import Image

def data\_generator(hazy\_dir, clear\_dir, batch\_size):

    while True:

        hazy\_images = []

        clear\_images = []

        for i in range(241, 439):

            filename = str(i) + '.tif'

            img = Image.open(os.path.join(hazy\_dir, filename))

            hazy\_images.append(np.array(img))

            clear\_filename = str(i) + '.tif'

            clear\_img = Image.open(os.path.join(clear\_dir, clear\_filename))

            clear\_images.append(np.array(clear\_img))

            if len(hazy\_images) == batch\_size:

                X = np.array(hazy\_images) / 255.0

                y = np.array(clear\_images) / 255.0

                yield X, y

                hazy\_images = []

                clear\_images = []

#6

# Set the directories and batch size

hazy\_dir = '/content/drive/MyDrive/Preprocessed Images/Hazy Images'

clear\_dir = '/content/drive/MyDrive/Preprocessed Images/Clear Images'

batch\_size = 8

# Get the list of file names in the directories

hazy\_filenames = os.listdir(hazy\_dir)

clear\_filenames = [os.path.splitext(filename)[0] + '\_clear.tif' for filename in hazy\_filenames]

# Split the file names into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    hazy\_filenames, clear\_filenames, test\_size=0.2, random\_state=42)

#7

import os

folder\_path1 = "/content/drive/MyDrive/Preprocessed Images/Clear Images" # Replace with the path to your folder

num\_images1 = len(os.listdir(folder\_path1))

print("Number of images in folder:", num\_images1)

folder\_path2 = "/content/drive/MyDrive/Preprocessed Images/Hazy Images" # Replace with the path to your folder

num\_images2 = len(os.listdir(folder\_path2))

print("Number of images in folder:", num\_images2)

#8

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

#Split the file names into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    hazy\_filenames, clear\_filenames, test\_size=0.2, random\_state=42

)

#9

# Define data generators

train\_generator = data\_generator(hazy\_dir, clear\_dir, batch\_size=batch\_size)

test\_generator = data\_generator(hazy\_dir, clear\_dir, batch\_size=batch\_size)

#10

# Build the model

model = resnet\_model(input\_shape=(None, None, 3))

#11

# Compile the model

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

#12

import tensorflow as tf

print("TensorFlow version:", tf.\_\_version\_\_)

gpu\_devices = tf.config.list\_physical\_devices('GPU')

if gpu\_devices:

    print("GPU available:", True)

    print("CUDA version:", gpu\_devices[0].name)

else:

    print("GPU available:", False)

#13

import matplotlib.pyplot as plt

# Train the model and collect the loss and accuracy data

history = model.fit(train\_generator, epochs=71, steps\_per\_epoch=len(X\_train) // batch\_size,

          validation\_data=test\_generator, validation\_steps=len(X\_test) // batch\_size)

# Create subplots to visualize the loss and accuracy data

fig, (ax1) = plt.subplots(1, figsize=(10, 5))

# Plot the training and validation loss data

ax1.plot(history.history['loss'], label='Training Loss')

ax1.plot(history.history['val\_loss'], label='Validation Loss')

ax1.set\_title('Model Loss')

ax1.set\_xlabel('Epochs')

ax1.set\_ylabel('Loss')

ax1.legend()

# Show the plot

plt.show()

# Create subplots to visualize the loss and accuracy data

fig, (ax1) = plt.subplots(1, figsize=(10, 5))

# Plot the training and validation loss data

ax1.plot(history.history['loss'], label='Training Loss')

ax1.plot(history.history['val\_loss'], label='Validation Loss')

ax1.set\_title('Model Loss')

ax1.set\_xlabel('Epochs')

ax1.set\_ylabel('Loss')

ax1.legend()

# Show the plot

plt.show()

#14

# Save the model

model.save('dehaze\_model.h5')

#15

train\_loss = model.evaluate(train\_generator, steps=len(X\_train) // batch\_size)

print('Train loss:', train\_loss)

#16

# Evaluate the model on the test set

mse = model.evaluate(test\_generator, steps=len(X\_test)//batch\_size)

print("Mean Squared Error on Test Set:", mse)

#17

from matplotlib import pyplot as plt

from PIL import Image

# Select an image index to evaluate

idx = 0

# Load the hazy and clear images

hazy\_image = Image.open(os.path.join(hazy\_dir, '/content/drive/MyDrive/Hazy Images/Hazy\_imgs/108.tif'))

clear\_image = Image.open(os.path.join(clear\_dir, '/content/drive/MyDrive/Clear Images/Ground\_imgs/108.tif'))

# Generate a prediction for the hazy image

hazy\_array = np.array(hazy\_image) / 255.0

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

predicted\_array = model.predict(hazy\_array)[0]

predicted\_image = Image.fromarray(np.uint8(predicted\_array\*255))

clear\_array = np.array(clear\_image) / 255.0

loss = np.mean(np.square(clear\_array - predicted\_array))

fig, ax = plt.subplots(ncols=3, figsize=(12, 4))

ax[0].imshow(hazy\_image)

ax[0].set\_title('Hazy Image')

ax[1].imshow(clear\_image)

ax[1].set\_title('Clear Image')

ax[2].imshow(predicted\_image)

ax[2].set\_title(f'Predicted Image\nLoss: {loss:.4f}')

plt.show()

#18

import cv2

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.models import load\_model

# Load pre-trained dehazing model

model = load\_model('dehaze\_model.h5')

# Load hazy image

hazy\_img = cv2.imread('/content/drive/MyDrive/Preprocessed Images/HAzy/99.tif')

# Preprocess the hazy image

hazy\_img = hazy\_img / 255.0

hazy\_img = np.expand\_dims(hazy\_img, axis=0)

# Apply the dehazing model to the hazy image

dehazed\_img = model.predict(hazy\_img)[0]

# Display the dehazed image using matplotlib

plt.imshow(dehazed\_img)

plt.show()

import tkinter as tk

from tkinter import filedialog

from PIL import Image, ImageTk

import numpy as np

import tensorflow as tf

import os

# Load the saved ML model

model = tf.keras.models.load\_model('dehaze\_model.h5')

# Create the main application window

window = tk.Tk()

window.title("Image Processing with ML")

window.geometry("600x600") # Set the window size to 600x600 pixels

# Function to process the image

def process\_image():

# Open file dialog to select an image

file\_path = filedialog.askopenfilename()

if file\_path:

# Load and preprocess the image

image = Image.open(file\_path)

image = image.resize((400, 400)) # Adjust the image size as needed

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

# Perform prediction using the ML model

prediction = model.predict(np.expand\_dims(image, axis=0))

# Display the prediction result

label.config(text="Prediction: " + str(prediction)) # Modify as per your model's output format

# Convert the predicted array to an image

predicted\_array = np.squeeze(prediction) \* 255

predicted\_image = Image.fromarray(predicted\_array.astype(np.uint8))

# Display the predicted image

photo = ImageTk.PhotoImage(predicted\_image)

image\_label.config(image=photo)

image\_label.image = photo

# Function to open the image file

def open\_image():

file\_path = filedialog.askopenfilename()

if file\_path:

image = Image.open(file\_path)

image.thumbnail((512, 512)) # Adjust the thumbnail size as needed

photo = ImageTk.PhotoImage(image)

image\_label.config(image=photo)

image\_label.image = photo

# Create the image display label

image\_label = tk.Label(window)

image\_label.pack()

# Create the buttons

open\_button = tk.Button(window, text="Open Image", command=open\_image)

open\_button.pack()

process\_button = tk.Button(window, text="Process Image", command=process\_image)

process\_button.pack()

# Create the label for displaying prediction result

label = tk.Label(window, text="Final Image")

label.pack()

# Run the GUI application

window.mainloop()

#final window side by side

import tkinter as tk

from tkinter import filedialog

from PIL import Image, ImageTk

import numpy as np

import tensorflow as tf

import os

# Load the saved ML model

model = tf.keras.models.load\_model('dehaze\_model.h5')

# Create the main application window

window = tk.Tk()

window.title("Image Processing with ML")

window.geometry("800x400") # Set the window size to 800x400 pixels

# Function to process the image

def process\_image():

# Open file dialog to select an image

file\_path = filedialog.askopenfilename()

if file\_path:

# Load and preprocess the image

image = Image.open(file\_path)

image = image.resize((400, 400)) # Adjust the image size as needed

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

# Perform prediction using the ML model

prediction = model.predict(np.expand\_dims(image, axis=0))

# Display the prediction result

label.config(text="Prediction: " + str(prediction)) # Modify as per your model's output format

# Convert the predicted array to an image

predicted\_array = np.squeeze(prediction) \* 255

predicted\_image = Image.fromarray(predicted\_array.astype(np.uint8))

# Display the predicted image

photo = ImageTk.PhotoImage(predicted\_image)

predicted\_image\_label.config(image=photo)

predicted\_image\_label.image = photo

# Function to open the image file

def open\_image():

file\_path = filedialog.askopenfilename()

if file\_path:

image = Image.open(file\_path)

image.thumbnail((400, 400)) # Adjust the thumbnail size as needed

photo = ImageTk.PhotoImage(image)

original\_image\_label.config(image=photo)

original\_image\_label.image = photo

# Create the original image display label

original\_image\_label = tk.Label(window)

original\_image\_label.grid(row=0, column=0, padx=10, pady=10)

# Create the predicted image display label

predicted\_image\_label = tk.Label(window)

predicted\_image\_label.grid(row=0, column=1, padx=10, pady=10)

# Create the buttons

open\_button = tk.Button(window, text="Open Image", command=open\_image)

open\_button.grid(row=1, column=0, padx=10, pady=10)

process\_button = tk.Button(window, text="Process Image", command=process\_image)

process\_button.grid(row=1, column=1, padx=10, pady=10)

# Create the label for displaying prediction result

label = tk.Label(window, text="Prediction: ")

label.grid(row=2, column=0, columnspan=2)

# Run the GUI application

window.mainloop()

#Code3

import tkinter as tk

from tkinter import filedialog

from PIL import Image, ImageTk

import numpy as np

import tensorflow as tf

import os

# Load the saved ML model

model = tf.keras.models.load\_model('dehaze\_model.h5')

# Create the main application window

window = tk.Tk()

window.title("Image Processing with ML")

window.geometry("600x600")

# Set the background color

window.configure(bg="white")

# Create a header label

header\_label = tk.Label(window, text="Welcome to the Dehaze World", font=("Helvetica", 20, "bold"),fg='Green', bg='white')

header\_label.pack(pady=20)

# Create a label for the open button

open\_label = tk.Label(window, text="Open an image:", bg='white', fg='red')

open\_label.pack()

# Function to open the image file

def open\_image():

file\_path = filedialog.askopenfilename()

image = Image.open(file\_path)

image.thumbnail((256, 256)) # Adjust the thumbnail size as needed

photo = ImageTk.PhotoImage(image)

open\_image\_label.config(image=photo)

open\_image\_label.image = photo

# Create the open image label

open\_image\_label = tk.Label(window,bg='white')

open\_image\_label.pack()

# Create a label for the process button

process\_label = tk.Label(window, text="Process the image:",bg='white', fg='red')

process\_label.pack()

# Function to process the image

def process\_image():

# Open file dialog to select an image

file\_path = filedialog.askopenfilename()

# Load and preprocess the image

image = Image.open(file\_path)

image = image.resize((256, 256)) # Adjust the image size as needed

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

# Perform prediction using the ML model

prediction = model.predict(np.expand\_dims(image, axis=0))

# Display the prediction result

label.config(text="Prediction: " + str(prediction)) # Modify as per your model's output format

# Convert the predicted array to an image

predicted\_array = np.squeeze(prediction) \* 255

predicted\_image = Image.fromarray(predicted\_array.astype(np.uint8))

# Display the predicted image

photo = ImageTk.PhotoImage(predicted\_image)

processed\_image\_label.config(image=photo)

processed\_image\_label.image = photo

# Create the process image label

processed\_image\_label = tk.Label(window,bg='white')

processed\_image\_label.pack()

# Set the button color

button\_color = "light blue"

# Create the open and process buttons

open\_button = tk.Button(window, text="Open Image", command=open\_image, bg=button\_color)

open\_button.pack(pady=10)

process\_button = tk.Button(window, text="Process Image", command=process\_image, bg=button\_color)

process\_button.pack(pady=10)

# Create the label for displaying prediction result

label = tk.Label(window, text="Final Image", bg='white', fg='Grey')

label.pack(pady=10)

# Run the GUI application

window.mainloop()