Main.py

"""

These are the necessary libraries to be imported to run the code:

(1)Keras for the model

(2)OpenCV for image processing

(3)NumPy for mathematical operations on arrays

(4)OS for file system operations such as reading and writing files.

(5)Gradio for building the interface:

It allows users to interact with the model by inputting data and receiving predictions or outputs.

It is useful for building prototypes,testing models, and sharing models with others.

"""

import keras

from keras.models import load\_model

import cv2

import numpy as np

import os

import gradio as gr

"""

Load a Keras model that has been previously trained

and saved to the file path "model/model.h5".

The model is loaded into memory and stored in the variable model.

"""

model = load\_model(r'model/model.h5')

"""

This line of code specifies the file path of the image

that we want to classify.

The image is located at "images/non\_fire.10.png"

"""

img =  r'images/non\_fire.10.png'

"""

This is the function that will be called when

the user wants to use the image from the local machine

This function takes the file path of an image as input and returns

the predicted probabilities of it being a fire or non-fire image.

using a pre-trained model.

"""

def predict\_input\_image(img):

    #(1)Load the image

    # read the image using OpenCV

    image = cv2.imread(img)

    #(2)Preprocess the image

    # resize the image to the required size

    image = cv2.resize(image, (224, 224))

    # the class names for the labels

    class\_names = ['fire\_images', 'non\_fire\_images']

    # Convert the image to a 4D tensor:

    # In computer vision, images are typically represented as arrays of numbers.

    # The dimensions of the array correspond to the height, width, and color

# channels of the image.

    # In this code, we are using a 224x224 pixel RGB image, which means that it

# has dimensions of 224x224x3.

    #

# However, in order to input this image into the Keras model for prediction,

    # we need to reshape it into a 4D tensor, with dimensions of (batch\_size,

# height, width, channels).

    # The reshape function is used to perform this operation.

    #

# The -1 in the reshape function is a placeholder for the size of the first

# dimension (batch\_size).

    # Since we are only processing one image at a time, the batch size is 1.

    # Therefore, the -1 tells the reshape function to automatically

    # calculate the size of the first dimension based on

    # the total number of elements in the input image and the other three

# dimensions.

    img\_4d = image.reshape(-1, 224, 224, 3)

    # Make a prediction on the image:

    # `model.predict(img\_4d)` takes the 4D tensor of the input image and

    # returns a 2D array with two elements. Each element represents the

    # probability of the input image belonging to one of the two classes that the

# model was trained on (fire or non-fire).

    #

# Since we only have one image to classify,

    # we take the first element of the 2D array by indexing with [0].

    # This gives us the predicted probability of the input image belonging

    # to the first class (fire).

    prediction = model.predict(img\_4d)[0]

   # If the prediction value is greater than 0.5, it is considered as fire

    if prediction > 0.5:

        # probability of fire and non-fire

        pred = [1-prediction, prediction]

    else:

         # probability of non-fire and fire

         pred = [1-prediction, prediction]

    # Create a dictionary of class names and their probabilities

    # The code iterates through the indices of class\_names (which is 0 for "fire"

    # and 1 for "non-fire"),and sets the value of each class name in confidences

# to be the corresponding probability from pred converted to a float. The

# resulting confidences dictionary will have the form:

    # {'fire': <probability of fire>, 'non-fire': <probability of non-fire>}

    confidences = {class\_names[i]: float(pred[i]) for i in range(2)}

    # return the dictionary of class names and their probabilities

    return confidences

# that's how we call the function

result = predict\_input\_image(img)

# print the result

print(result)

"""

This is the function that will be called when

the user uploads an image

This function takes an image as input and returns

the predicted probabilities of it being a fire or non-fire image.

using a pre-trained model.

"""

def predict\_input\_image\_gr(img):

# the class names for the labels

    class\_names = ['fire\_images', 'non\_fire\_images'

    # Convert the image to a 4D tensor:

    # In computer vision, images are typically represented as arrays of numbers.

    # The dimensions of the array correspond to the height, width, and color

# channels of the image.

    # In this code, we are using a 224x224 pixel RGB image, which means that it

# has dimensions of 224x224x3.

#

    # However, in order to input this image into the Keras model for prediction,

    # we need to reshape it into a 4D tensor, with dimensions of (batch\_size,

# height, width, channels).

    # The reshape function is used to perform this operation.

#

    # The -1 in the reshape function is a placeholder for the size of the first

# dimension (batch\_size).

    # Since we are only processing one image at a time, the batch size is 1.

    # Therefore, the -1 tells the reshape function to automatically

    # calculate the size of the first dimension based on

    # the total number of elements in the input image and the other three

# dimensions.

    img\_4d = img.reshape(-1, 224, 224, 3)

    # Make a prediction on the image:

    # ‘model.predict(img\_4d)’ takes the 4D tensor of the input image and

    # returns a 2D array with two elements. Each element represents the

# Probability of the input image belonging to one of the two classes that the

    # model was trained on (fire or non-fire).

#

    # Since we only have one image to classify,

    # we take the first element of the 2D array by indexing with [0].

    # This gives us the predicted probability of the input image belonging

    # to the first class (fire).

    prediction = model.predict(img\_4d)[0]

    # If the prediction value is greater than 0.5, it is considered as fire

    if prediction > 0.5:

        # Probability of fire and non-fire

        pred = [1-prediction, prediction]

    else:

        # Probability of non-fire and fire

        pred = [1-prediction, prediction]

    # Create a dictionary of class names and their probabilities

    # The code iterates through the indices of class\_names (which is 0 for "fire"

# and 1 for "non-fire"),and sets the value of each class name in confidences

# to be the corresponding probability pred converted to a float. The

# resulting confidences dictionary will have the form:

    # {'fire': <probability of fire>, 'non-fire': <probability of non-fire>}

    confidences = {class\_names[i]: float(pred[i]) for i in range(2)}

    # Return the dictionary of class names and their probabilities

    return confidences

# Define the input for the Gradio interface as an image with a shape of 224x224

image = gr.inputs.Image(shape=(224, 224))

# Define the output for the Gradio interface as a label with one class

label = gr.outputs.Label(num\_top\_classes=1)

"""

Launch the Gradio interface with the predict\_input\_image\_gr function as the prediction function

The inputs are defined as the 'image' input, and the outputs are defined as the 'label' output

The interpretation parameter is set to default to allow for image upload and display

The launch function is called to start the interface

Debug is set to 'True' to display debug information, and share is set to 'True' to allow for sharing the link to the interface

"""

gr.Interface(fn=predict\_input\_image\_gr, inputs=image, outputs=label,

interpretation='default').launch(debug='True', share='True')

With cam.py

"""

These are the necessary libraries to be imported to run the code:

(1)Keras for the model

(2)OpenCV for image processing

(3)NumPy for mathematical operations on arrays

(4)OS for file system operations such as reading and writing files.

(5)Gradio for building the interface:

It allows users to interact with the model by inputting

data and receiving predictions or outputs. It is useful for building

prototypes,testing models, and sharing models with others.

(6) pyplot for ploting histogram

"""

import numpy as np

import cv2

import os

import keras

from keras.models import load\_model

import matplotlib.pyplot as plt

# Define the path of the input image and load the pre-trained model

img = r'images/fire.90.png'

model = load\_model(r'model/model.h5')

# Define the path of the input image and load the pre-trained model

def predict\_input\_image(img):

    class\_names = ['fire\_images', 'non\_fire\_images']

    # Reshape the input image into a 4D array

    img\_4d = img.reshape(-1, 224, 224, 3)

    # Use the loaded model to predict the class of the input image

    prediction = model.predict(img\_4d)[0]

   # If the prediction value is greater than 0.5, it is considered as fire

    if prediction > 0.5:

        # probability of fire and non-fire

        pred = [1-prediction, prediction]

    else:

         # probability of non-fire and fire

         pred = [1-prediction, prediction]

 # Create a dictionary containing the confidence scores for each class

    confidences = {class\_names[i]: float(pred[i]) for i in range(2)}

    print(confidences)

    return confidences

# Open a video by camera

cam = cv2.VideoCapture(0)

if not cam.isOpened():

    print('Camera not found')

    exit()

# Loop through each frame of the video

while True:

    # Loop through each frame of the video

    ret, frame = cam.read()

     # If there are no more frames, exit the loop

    if not ret:

        print('Can not receive frame (stream end?). Exiting ...')

        break

    # Resize the frame to a desired size for model function

    resized\_frame = cv2.resize(frame, (224, 224))

    # Use the loaded model to predict the class of the resized frame

    pred = predict\_input\_image(resized\_frame)

    # Determine prediction probability for fire if there is a fire

# and Determine prediction probability for non fire if there is a no fire

# store result

    if pred['fire\_images'] > pred['non\_fire\_images'] and

round(pred['fire\_images']\*100, 2) >55 :

        res = f"Fire {round(pred['fire\_images']\*100, 2)} %"

    else:

       # res = f"No Fire {round(pred['non\_fire\_images']\*100, 2)} %"

        res = "No Fire"

    # Resize the original frame and overlay the result string on top of it

    frame = cv2.resize(frame, (640, 480))

    # Put text and prediction of detected fire on displaying video

    frame = cv2.putText(frame, str(res), (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 1,

(0, 0, 255), 2, cv2.LINE\_AA)

    cv2.imshow('frame', frame)

    # Create a mask for the image by filtering it based on a certain color range

    blurred = cv2.GaussianBlur(frame, (15, 15), 0)

    hsv = cv2.cvtColor(blurred, cv2.COLOR\_BGR2HSV)

    lower\_bound = np.array([0, 50, 50], dtype='uint8')

    upper\_bound = np.array([35, 255, 255], dtype='uint8')

    mask = cv2.inRange(hsv, lower\_bound, upper\_bound)

    masked\_frame = cv2.bitwise\_and(frame, frame, mask=mask)

    # Calculate the normalized histogram of the filtered image and display it

    fire\_hist = cv2.calcHist([masked\_frame], [0], mask, [256], [0, 256])

    # This line normalizes the fire\_hist using cv2.normalize() function.

    # The resulting values are scaled between 0 and 255,

    # which are the minimum and maximum values of an 8-bit image.

    fire\_hist\_norm = cv2.normalize(fire\_hist, None, alpha=0, beta=255,

norm\_type=cv2.NORM\_MINMAX)

    # This block of code plots the normalized histogram

    # using Matplotlib's plot() function.

    # It also sets the x and y limits of the plot

    # and adds a title and labels for the axes.

 if pred['fire\_images'] > pred['non\_fire\_images'] and

round(pred['fire\_images']\*100, 2) >60 :

      plt.plot(fire\_hist\_norm)

    else:

    plt.xlim([0, 256])

    plt.ylim([0, 1000])

    plt.title("Fire Histogram")

    plt.xlabel("Pixel Intensity")

    plt.ylabel("Frequency")

    plt.show(block=False)

    # This line pauses the plot for 0.1 seconds to allow time

    # for the plot to be displayed.

    plt.pause(0.1)

    # Clears the plot for the next iteration.

    plt.clf()

    # This line waits for the 'q' key to be pressed to break the while loop

    # and exit the program. The waitKey() function waits

    # for a specified delay (in milliseconds)

    # for a key event to occur. If the pressed key's ASCII

    # value matches that of 'q', then the loop is broken.

    if cv2.waitKey(1) == ord('q'):

        break