**Application Code:**

import tkinter as tk

from tkinter import filedialog, messagebox, ttk

from PIL import Image, ImageTk

from keras.models import load\_model

import os

from keras.preprocessing.image import load\_img, img\_to\_array

from sklearn.preprocessing import LabelEncoder

import numpy as np

from keras.layers import Conv2D, MaxPooling2D

from keras.models import Model

import tkinter as tk

from tkinter import ttk

import threading

import time

class HomeScreen:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Home Screen")

self.root.state('zoomed')

self.current\_screen = None

self.intro\_screen = None

self.show\_intro\_screen()

def show\_intro\_screen(self):

self.current\_screen = self.intro\_screen = tk.Frame(self.root)

self.intro\_screen.pack(fill="both", expand=True)

image\_path = "C:\\Users\\Asus\\OneDrive\\Desktop\\6th Sem Project\\Final Dependencies\\Home.jpg"

image = Image.open(image\_path)

image = image.resize((1500, 820))

image = ImageTk.PhotoImage(image)

image\_label = tk.Label(self.intro\_screen, image=image)

image\_label.image = image

image\_label.place(x=0, y=0, relwidth=1, relheight=1)

button\_frame = tk.Frame(self.intro\_screen, bg="#1f497d")

button\_frame.place(relx=0.73, rely=0.85, anchor=tk.CENTER)

button2 = tk.Button(button\_frame, text="Abstract", command=self.run\_abstract\_screen, width=15, height=3, bg="#e3fef7")

button3 = tk.Button(button\_frame, text="Help", command=self.run\_help\_screen, width=15, height=3, bg="#e3fef7")

button4 = tk.Button(button\_frame, text="Start", command=self.run\_model\_screen, width=15, height=3, bg="#e3fef7")

button2.grid(row=0, column=0, padx=(20, 20))

button3.grid(row=0, column=1, padx=(20, 20))

button4.grid(row=0, column=2, padx=(20, 20))

def run\_abstract\_screen(self):

self.current\_screen.destroy()

self.current\_screen = tk.Toplevel(self.root)

self.current\_screen.title("Abstract Screen")

self.current\_screen.state('zoomed')

abstract\_image\_path = "C:\\Users\\Asus\\OneDrive\\Desktop\\6th Sem Project\\Final Dependencies\\Abstract.jpg"

abstract\_image = Image.open(abstract\_image\_path)

abstract\_image = abstract\_image.resize((1500, 820))

abstract\_image = ImageTk.PhotoImage(abstract\_image)

abstract\_image\_label = tk.Label(self.current\_screen, image=abstract\_image)

abstract\_image\_label.image = abstract\_image

abstract\_image\_label.place(x=0, y=0, relwidth=1, relheight=1)

home\_button = tk.Button(

self.current\_screen,

text="Home",

command=self.back\_to\_home,

width=10,

height=2,

bg="#e3fef7"

)

home\_button.place(relx=0.02, rely=0.02)

def run\_help\_screen(self):

self.current\_screen.destroy()

self.current\_screen = tk.Toplevel(self.root)

self.current\_screen.title("Help Screen")

self.current\_screen.state('zoomed')

help\_image\_path = "C:\\Users\\Asus\\OneDrive\\Desktop\\6th Sem Project\\Final Dependencies\\Help.jpg"

help\_image = Image.open(help\_image\_path)

help\_image = help\_image.resize((1500, 820))

help\_image = ImageTk.PhotoImage(help\_image)

help\_image\_label = tk.Label(self.current\_screen, image=help\_image)

help\_image\_label.image = help\_image

help\_image\_label.place(x=0, y=0, relwidth=1, relheight=1)

home\_button = tk.Button(

self.current\_screen,

text="Home",

command=self.back\_to\_home,

width=10,

height=2,

bg="#e3fef7"

)

home\_button.place(relx=0.02, rely=0.02)

def run\_model\_screen(self):

self.current\_screen.destroy()

ModelGUI(self.root)

def back\_to\_home(self):

self.current\_screen.destroy()

self.show\_intro\_screen()

class ModelGUI:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Load Keras Model")

self.root.state('zoomed')

self.root.configure(bg='#1f497d') # Set background color of root window

# Add a label for the welcome message

self.welcome\_label = tk.Label(self.root, text="Welcome To The Disaster Detection System", font=("Times New Roman", 25), bg='#1f497d', fg='white')

self.welcome\_label.pack(pady=(20, 10)) # Adjusted pady here

# Frame to hold back and next buttons

self.navigation\_frame = tk.Frame(self.root, bg='#1f497d') # Set background color of frame

self.navigation\_frame.pack(pady=10)

self.back\_button = tk.Button(self.navigation\_frame, text="Back", command=self.back\_to\_home\_screen, font=("Arial", 14), height=3, width=20, bg='#e3fef7', fg='black')

self.back\_button.pack(side=tk.LEFT, padx=10)

self.next\_button = tk.Button(self.navigation\_frame, text="Next", command=self.open\_select\_image\_screen, font=("Arial", 14), height=3, width=20, bg='#e3fef7', fg='black')

self.next\_button.pack(side=tk.LEFT, padx=10)

self.load\_model\_button = tk.Button(self.root, text="Load Model", command=self.load\_model, font=("Arial", 14), height=3, width=20, bg='#e3fef7', fg='black')

self.load\_model\_button.pack(pady=10)

# Frame for model summary

self.model\_summary\_frame = tk.Frame(self.root, bg='#1f497d')

self.model\_summary\_frame.pack(expand=False, fill='both', padx=20, pady=(0, 2)) # Adjusted pady here

# Frame for the default screen color

self.default\_screen\_frame = tk.Frame(self.root, bg='#1f497d')

self.default\_screen\_frame.pack(expand=True, fill='both')

# Create model summary inside self.model\_summary\_frame

self.summary\_frame = ttk.Frame(self.model\_summary\_frame, style="Custom.TFrame")

self.summary\_frame.pack(expand=False, fill='both', padx=5, pady=3)

self.loaded\_model = None # Attribute to store the loaded model

def load\_model(self):

try:

model\_path = filedialog.askopenfilename(title="Select Model", filetypes=[("Keras Model", "\*.keras")])

with open('C:\\Users\\Asus\\OneDrive\\Desktop\\6th Sem Project\\Finally Final\\modelpath.txt', 'w') as model\_file:

model\_file.write(model\_path) # Write the model path to the modelpath.txt file

self.loaded\_model = load\_model(model\_path)

self.show\_model\_summary(self.loaded\_model)

messagebox.showinfo("Model Loaded", "Model loaded successfully!")

# Configure the background color of any additional space

self.root.configure(bg='#1f497d')

except Exception as e:

messagebox.showerror("Error", f"Error loading model: {e}")

def show\_model\_summary(self, model):

for widget in self.summary\_frame.winfo\_children():

widget.destroy()

tree = ttk.Treeview(self.summary\_frame, style="Custom.Treeview")

tree.pack(expand=True, fill='both')

tree["columns"] = ("Layer", "Output Shape")

tree.column("#0", width=0, stretch=tk.NO)

tree.heading("#1", text="Layer")

tree.heading("#2", text="Output Shape")

for i, layer in enumerate(model.layers):

output\_shape = layer.output\_shape[1:] if layer.output\_shape else "-"

tree.insert("", "end", text=f"Layer {i}", values=(layer.name, output\_shape))

for col in tree["columns"]:

tree.column(col, width=150, stretch=tk.YES)

def back\_to\_home\_screen(self):

self.root.destroy()

root = tk.Tk()

app = HomeScreen(root)

root.mainloop()

def open\_select\_image\_screen(self):

self.root.destroy()

root = tk.Tk()

app = ImagePreviewApp(root)

root.mainloop()

class LoadingScreen:

def \_\_init\_\_(self):

self.window = tk.Tk()

self.window.title("Loading")

self.window.state('zoomed') # Maximize window

self.window.configure(bg='#1f497d') # Set background color to #1f497d

self.label = ttk.Label(self.window, text="Loading, please wait...", font=("Arial", 45), foreground='white', background='#1f497d') # White text color

self.label.pack(pady=5)

def start(self):

self.thread = threading.Thread(target=self.\_update\_progress)

self.thread.start()

self.window.mainloop()

def \_update\_progress(self):

time.sleep(8) # Wait for 5 seconds

self.label.config(text="Loading completed") # Update the label to indicate task completion

self.window.after(2000, self.window.destroy) # Destroy the window after 2 seconds

class ImagePreviewApp:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Image Preview")

self.root.state('zoomed')

self.root.configure(bg='#1f497d') # Set background color of root window

# Add a label for the welcome message

self.welcome\_label = tk.Label(self.root, text="Welcome To The Disaster Detection System", font=("Times New Roman", 25), bg='#1f497d', fg='white')

self.welcome\_label.pack(pady=(20, 10)) # Adjusted pady here

# Frame for buttons

self.button\_frame = tk.Frame(self.root, bg='#1f497d')

self.button\_frame.pack(side=tk.TOP, pady=10)

# Back button

self.back\_button = tk.Button(self.button\_frame, text="Back", command=self.back\_to\_model\_from\_image, font=("Arial", 16), width=15, height=2, bg='#e3fef7', fg='#1f497d')

self.back\_button.pack(side=tk.LEFT, padx=20)

# Next button

self.next\_button = tk.Button(self.button\_frame, text="Next", command=self.imagetoprocess, font=("Arial", 16), width=15, height=2, bg='#e3fef7', fg='#1f497d')

self.next\_button.pack(side=tk.LEFT, padx=20)

# Select image button

self.select\_button = tk.Button(self.root, text="Upload Image", command=self.select\_image, font=("Arial", 16), bg='#e3fef7', fg='#1f497d')

self.select\_button.pack(pady=20)

# Image and details labels

self.image\_label = tk.Label(self.root, bg='#1f497d')

self.image\_label.pack(pady=10)

self.details\_label = tk.Label(self.root, justify='left', bg='#1f497d', fg='white', font=("Arial", 14)) # Increased font size

self.details\_label.pack(pady=10)

def select\_image(self):

file\_path = filedialog.askopenfilename(

title="Upload Image",

filetypes=[("Image files", "\*.png;\*.jpg;\*.jpeg;\*.gif;\*.bmp")]

)

if file\_path:

if self.is\_image\_compatible(file\_path):

# Write the image path to the imagepath.txt file

with open('C:\\Users\\Asus\\OneDrive\\Desktop\\6th Sem Project\\Finally Final\\imagepath.txt', 'w') as path\_file:

path\_file.write(file\_path)

self.display\_image\_preview(file\_path)

self.display\_image\_details(file\_path)

else:

messagebox.showwarning("Incompatible Image", "The selected image must have dimensions of at least 150x150 pixels.")

def is\_image\_compatible(self, file\_path):

img = Image.open(file\_path)

width, height = img.size

return width >= 150 and height >= 150

def display\_image\_preview(self, file\_path):

img = Image.open(file\_path)

img = img.resize((450, 450))

photo = ImageTk.PhotoImage(img)

self.image\_label.config(image=photo)

self.image\_label.image = photo

def display\_image\_details(self, file\_path):

file\_name = os.path.basename(file\_path)

file\_type = os.path.splitext(file\_path)[1]

img = Image.open(file\_path)

width, height = img.size

details\_text = f"File Name: {file\_name}\nFile Path: {file\_path}\nFile Type: {file\_type}\nDimensions: {width} x {height}"

self.details\_label.config(text=details\_text)

def imagetoprocess(self):

# Create an Event to signal when the ImageProcessingApp is ready

self.app\_ready\_event = threading.Event()

# Start the loading screen in a separate thread

loading\_thread = threading.Thread(target=self.\_start\_loading\_screen)

loading\_thread.start()

# Start the ImageProcessingApp in the main thread

self.\_create\_image\_processing\_app()

def \_start\_loading\_screen(self):

# Start the loading screen

loading\_screen = LoadingScreen()

loading\_screen.start()

def \_create\_image\_processing\_app(self):

# Create the ImageProcessingApp

self.root.destroy() # Assuming self.root is initialized elsewhere

root = tk.Tk()

app = ImageProcessingApp(root)

# Set the flag to signal that the ImageProcessingApp is ready

self.app\_ready\_event.set()

root.mainloop()

def back\_to\_model\_from\_image(self):

self.root.destroy()

root = tk.Tk()

app = ModelGUI(root)

root.mainloop()

class ImageProcessingApp:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Image Processing App")

self.root.state('zoomed') # Maximize the window

self.original\_image = None

self.processed\_image = None

# Create a frame for the buttons

self.button\_frame = tk.Frame(self.root)

self.button\_frame.pack(side=tk.TOP, fill=tk.X, pady=10)

# Back button

self.back\_button = tk.Button(self.button\_frame, text="Back", command=self.run\_back\_script, width=10)

self.back\_button.pack(side=tk.LEFT, padx=(10, 5))

# Next button

self.next\_button = tk.Button(self.button\_frame, text="Next", command=self.run\_next\_script, width=10)

self.next\_button.pack(side=tk.LEFT, padx=5)

# Process button

self.process\_button = tk.Button(self.button\_frame, text="Process Image", command=self.visualize\_feature\_maps, width=15)

self.process\_button.pack(side=tk.LEFT, padx=5)

# Canvas and scrollbar

self.canvas = tk.Canvas(self.root)

self.canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

self.frame = tk.Frame(self.canvas)

self.frame.bind("<Configure>", self.on\_frame\_configure)

self.scrollbar = tk.Scrollbar(self.root, orient="vertical", command=self.canvas.yview)

self.scrollbar.pack(side=tk.RIGHT, fill="y")

self.canvas.configure(yscrollcommand=self.scrollbar.set)

self.canvas.create\_window((0, 0), window=self.frame, anchor="nw")

self.process\_steps = []

self.current\_step = 0

# Initialize model attribute

self.model = None

# Automatically load the image and model

self.load\_model\_and\_image()

def load\_model\_and\_image(self):

try:

with open('C:\\Users\\Asus\\OneDrive\\Desktop\\6th Sem Project\\Finally Final\\imagepath.txt', 'r') as f:

image\_path = f.read().strip()

if image\_path:

self.original\_image = Image.open(image\_path)

# Resize the image to 700x700 pixels

resized\_image = self.original\_image.resize((700, 700))

self.display\_image(resized\_image)

else:

messagebox.showwarning("Image Not Found", "Please specify the image path.")

except FileNotFoundError:

messagebox.showwarning("Image Not Found", "Please specify the image path.")

try:

with open('C:\\Users\\Asus\\OneDrive\\Desktop\\6th Sem Project\\Finally Final\\modelpath.txt', 'r') as f:

model\_path = f.read().strip()

if model\_path:

self.model = load\_model(model\_path)

self.process\_button.config(state=tk.NORMAL)

else:

messagebox.showwarning("Model Not Found", "Please specify the model path.")

except FileNotFoundError:

messagebox.showwarning("Model Not Found", "Please specify the model path.")

def visualize\_feature\_maps(self):

if self.original\_image is not None and self.model is not None:

self.process\_button.config(state=tk.DISABLED)

self.\_visualize\_feature\_maps()

else:

messagebox.showwarning("Image or Model not found", "Please make sure both image and model are loaded.")

def \_visualize\_feature\_maps(self):

# Resize the image to match the required input size of the CNN model

resized\_image = self.original\_image.resize((250, 250))

self.display\_image(resized\_image, "Resized Image")

# Get the convolutional and max pooling layers

conv\_pool\_layers = [layer for layer in self.model.layers if isinstance(layer, (Conv2D, MaxPooling2D))]

# Create a feature map model

feature\_map\_model = Model(inputs=self.model.inputs, outputs=[layer.output for layer in conv\_pool\_layers])

# Convert the image to an array

img\_array = img\_to\_array(resized\_image)

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

# Get the feature maps for the image

feature\_maps = feature\_map\_model.predict(img\_array)

print("Number of feature maps:", len(feature\_maps))

# Plot feature maps of convolutional and max pooling layers

for layer, feature\_map in zip(conv\_pool\_layers, feature\_maps):

print("Layer:", layer.name)

print("Feature map shape:", feature\_map.shape)

self.\_plot\_feature\_maps(layer.name, feature\_map)

def \_plot\_feature\_maps(self, layer\_name, feature\_map):

num\_features = feature\_map.shape[-1]

size = feature\_map.shape[1]

images\_per\_row = 4 # Set the number of images to display per row

row\_count = 0

# Create a frame to contain the row of images

row\_frame = tk.Frame(self.frame)

row\_frame.pack()

for i in range(num\_features):

feature\_image = feature\_map[0, :, :, i]

feature\_image\_mean = feature\_image.mean()

feature\_image\_std = feature\_image.std()

if feature\_image\_std != 0: # Avoid division by zero

feature\_image -= feature\_image\_mean

feature\_image /= feature\_image\_std

else:

feature\_image -= feature\_image\_mean

feature\_image \*= 64

feature\_image += 128

feature\_image = np.clip(feature\_image, 0, 255).astype('uint8')

image = Image.fromarray(feature\_image)

# Display image in the row frame

label = tk.Label(row\_frame, text=f"{layer\_name}\_{i}")

label.pack(side='left')

photo = ImageTk.PhotoImage(image)

label = tk.Label(row\_frame, image=photo)

label.image = photo

label.pack(side='left') # Display images side by side

row\_count += 1

# If the number of images in the row equals images\_per\_row or if this is the last image

if row\_count == images\_per\_row or i == num\_features - 1:

row\_count = 0 # Reset row count

# Create a new row frame

row\_frame = tk.Frame(self.frame)

row\_frame.pack()

def \_add\_new\_line(self):

label = tk.Label(self.frame, text="") # Empty label to add new line

label.pack()

def display\_image\_in\_frame(self, image, layer\_name):

label = tk.Label(self.frame, text=layer\_name)

label.pack()

photo = ImageTk.PhotoImage(image)

label = tk.Label(self.frame, image=photo)

label.image = photo

label.pack(side='left') # Display images side by side

def display\_image(self, image, layer\_name=None):

# Clear existing images from the frame

for widget in self.frame.winfo\_children():

widget.destroy()

# Display the new image

image = ImageTk.PhotoImage(image)

self.image\_label = tk.Label(self.frame, image=image)

self.image\_label.image = image

self.image\_label.pack(side='top', fill='both', expand=True) # Center the image horizontally

# Display the layer name if provided

if layer\_name:

label = tk.Label(self.frame, text=layer\_name)

label.pack()

def on\_frame\_configure(self, event):

self.canvas.configure(scrollregion=self.canvas.bbox("all"))

def run\_back\_script(self):

# Create the ImageProcessingApp

self.root.destroy() # Assuming self.root is initialized elsewhere

root = tk.Tk()

app = ImagePreviewApp(root)

def run\_next\_script(self):

# Create an Event to signal when the ImageProcessingApp is ready

self.app\_ready\_event = threading.Event()

# Start the loading screen in a separate thread

loading\_thread = threading.Thread(target=self.\_start\_loading\_screen)

loading\_thread.start()

# Start the ImageProcessingApp in the main thread

self.resultscreen()

def \_start\_loading\_screen(self):

# Start the loading screen

loading\_screen = LoadingScreen()

loading\_screen.start()

def resultscreen(self):

# Create the ImageProcessingApp

self.root.destroy() # Assuming self.root is initialized elsewhere

root = tk.Tk()

app = ResultScreen(root)

# Set the flag to signal that the ImageProcessingApp is ready

self.app\_ready\_event.set()

root.mainloop()

class ResultScreen:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Disaster Classifier")

self.root.configure(bg="#1f497d")

self.root.state('zoomed')

frame = tk.Frame(self.root, bg="#1f497d")

frame.pack(pady=10)

self.image\_label = tk.Label(frame, bg="#1f497d")

self.image\_label.grid(row=0, column=0, padx=20, pady=10)

self.result\_label = tk.Label(frame, text="", font=("Helvetica", 16), bg="#1f497d")

self.result\_label.grid(row=0, column=1, padx=20, pady=10)

button\_frame = tk.Frame(self.root, bg="#1f497d")

button\_frame.pack(side=tk.BOTTOM, pady=(50, 10))

back\_button = tk.Button(button\_frame, text="Back", command=self.run\_back\_script, bg="#e3fef7", fg="#1f497d", width=25, height=3)

back\_button.grid(row=0, column=0, padx=(10, 5))

home\_button = tk.Button(button\_frame, text="Home", command=self.run\_home\_script, bg="#e3fef7", fg="#1f497d", width=25, height=3)

home\_button.grid(row=0, column=1, padx=(5, 10))

self.load\_and\_process\_image()

@staticmethod

def preprocess\_image\_for\_app(file\_path):

try:

img = load\_img(file\_path, target\_size=(250, 250))

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

return img\_array

except Exception as e:

print(f"Error preprocessing image: {e}")

return None

def load\_and\_process\_image(self):

with open('C:\\Users\\Asus\\OneDrive\\Desktop\\6th Sem Project\\Finally Final\\modelpath.txt', 'r') as f:

model\_path = f.read().strip()

with open('C:\\Users\\Asus\\OneDrive\\Desktop\\6th Sem Project\\Finally Final\\imagepath.txt', 'r') as f:

image\_path = f.read().strip()

self.classify\_and\_display\_result(image\_path, model\_path)

def classify\_and\_display\_result(self, image\_path, model\_path):

model = load\_model(model\_path)

target\_labels = ['Fire Disaster', 'Land Disaster', 'Human', 'Nature', 'Sea', 'Rain', 'Water Disaster']

label\_encoder = LabelEncoder()

label\_encoder.fit(target\_labels)

image\_array = self.preprocess\_image\_for\_app(image\_path)

if image\_array is not None:

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

class\_index = np.argmax(predictions[0])

class\_label = label\_encoder.inverse\_transform([class\_index])[0]

if class\_label in ['Human', 'Nature', 'Sea', 'Rain']:

result\_text = f"Disaster Not Detected\n\n{class\_label}\n"

else:

result\_text = f"Disaster Type Prediction:\n\n{class\_label}"

self.result\_label.config(text=result\_text, fg="white")

img = Image.open(image\_path)

img = img.resize((650, 650))

img = ImageTk.PhotoImage(img)

self.image\_label.config(image=img)

self.image\_label.image = img

def run\_back\_script(self):

# Create an Event to signal when the ImageProcessingApp is ready

self.app\_ready\_event = threading.Event()

# Start the loading screen in a separate thread

loading\_thread = threading.Thread(target=self.load1)

loading\_thread.start()

# Start the ImageProcessingApp in the main thread

self.pscreen()

def load1(self):

# Start the loading screen

loading\_screen = LoadingScreen()

loading\_screen.start()

def pscreen(self):

# Create the ImageProcessingApp

self.root.destroy() # Assuming self.root is initialized elsewhere

root = tk.Tk()

app = ImageProcessingApp(root)

# Set the flag to signal that the ImageProcessingApp is ready

self.app\_ready\_event.set()

root.mainloop()

def run\_home\_script(self):

self.root.destroy() # Close the current screen

root = tk.Tk() # Create a new Tkinter window

app = HomeScreen(root) # Open the home screen

root.mainloop()

def main():

root = tk.Tk()

app = HomeScreen(root)

root.mainloop()

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Model Code:**

**Code cell 1:**

import numpy as np

import pandas as pd

from keras.preprocessing.image import load\_img, img\_to\_array

from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from keras.callbacks import EarlyStopping, ModelCheckpoint

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

from sklearn.preprocessing import LabelEncoder

from PIL import Image

import os

# Set the width and height for loading images

w, h = 250, 250 # You can adjust these values based on your requirements

# List of folders with their respective label1 and label2 values

folder\_data = [

{'folder': 'C:\\Users\\Asus\\OneDrive\\Desktop\\6th Sem Project\\archive\\Comprehensive Disaster Dataset(CDD)\\Fire\_Disaster\\Wild\_Fire', 'label1': 'Fire Disaster', 'label2': 'Wild Fire'},

{'folder': 'C:\\Users\\Asus\\OneDrive\\Desktop\\6th Sem Project\\archive\\Comprehensive Disaster Dataset(CDD)\\Land\_Disaster\\Drought', 'label1': 'Land Disaster', 'label2': 'Drought'},

{'folder': 'C:\\Users\\Asus\\OneDrive\\Desktop\\6th Sem Project\\archive\\Comprehensive Disaster Dataset(CDD)\\Non\_Damage\\human', 'label1': 'Human', 'label2': 'Human'},

{'folder': 'C:\\Users\\Asus\\OneDrive\\Desktop\\6th Sem Project\\archive\\Comprehensive Disaster Dataset(CDD)\\Non\_Damage\\Non\_Damage\_Wildlife\_Forest', 'label1': 'Nature', 'label2': 'Undamaged Forest'},

{'folder': "C:\\Users\\Asus\\OneDrive\\Desktop\\6th Sem Project\\archive\\Sea", 'label1': 'Sea', 'label2': 'Sea'},

{'folder': "C:\\Users\\Asus\\OneDrive\\Desktop\\6th Sem Project\\archive\\Comprehensive Disaster Dataset(CDD)\\Non\_Damage\\Rain", 'label1': 'Rain', 'label2': 'Water Disaster'},

{'folder': 'C:\\Users\\Asus\\OneDrive\\Desktop\\6th Sem Project\\archive\\Comprehensive Disaster Dataset(CDD)\\Water\_Disaster', 'label1': 'Water Disaster', 'label2': 'Water Disaster'}

# Add the remaining folders and label values

]

# Combine file paths and label values into a list of dictionaries

data\_list = []

for folder\_info in folder\_data:

folder\_path = folder\_info['folder']

if os.path.exists(folder\_path):

for root, dirs, files in os.walk(folder\_path):

for file in files:

if file.endswith('.jpg') or file.endswith('.png'):

image\_path = os.path.join(root, file)

data\_list.append({'image\_path': image\_path, 'label1': folder\_info['label1'], 'label2': folder\_info['label2']})

else:

print(f"Folder not found: {folder\_info['folder']}")

# Convert the list of dictionaries to a DataFrame

df = pd.DataFrame(data\_list)

# Split data into training and testing sets with stratification

train\_data, test\_data = train\_test\_split(df, test\_size=0.2, random\_state=42, stratify=df['label1'])

# Load images and labels into arrays

def load\_images\_and\_labels(data):

images = []

labels1 = []

labels2 = []

for index, row in data.iterrows():

try:

img = load\_img(row['image\_path'], target\_size=(w, h))

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

images.append(img\_array)

labels1.append(row['label1'])

labels2.append(row['label2'])

except Exception as e:

print(f"Error loading image {row['image\_path']}: {e}")

return np.array(images), np.array(labels1), np.array(labels2)

# Load images and labels for training and testing sets

train\_images, train\_labels1, train\_labels2 = load\_images\_and\_labels(train\_data)

test\_images, test\_labels1, test\_labels2 = load\_images\_and\_labels(test\_data)

**Code Cell 2:**

import pandas as pd

# Assuming df is your DataFrame

# df['label'] is the column for which you want to get the value counts

value\_counts = df['label1'].value\_counts()

print(value\_counts)

**Output Cell 2:**

label1

Nature 4900

Water Disaster 627

Fire Disaster 361

Human 299

Land Disaster 201

Sea 90

Rain 69

Name: count, dtype: int64

**Code Cell 3:**

from sklearn.preprocessing import LabelEncoder

from keras.utils import to\_categorical

# Determine the number of unique classes for label1 and label2

num\_classes1 = len(df['label1'].unique())

num\_classes2 = len(df['label2'].unique())

# Create a LabelEncoder for each category

label\_encoder1 = LabelEncoder()

label\_encoder2 = LabelEncoder()

# Fit the LabelEncoder on the combined set of training and test labels

combined\_labels1 = np.concatenate([train\_labels1, test\_labels1])

combined\_labels2 = np.concatenate([train\_labels2, test\_labels2])

label\_encoder1.fit(combined\_labels1)

label\_encoder2.fit(combined\_labels2)

# Transform string labels to numerical indices

train\_labels1\_indices = label\_encoder1.transform(train\_labels1)

train\_labels2\_indices = label\_encoder2.transform(train\_labels2)

test\_labels1\_indices = label\_encoder1.transform(test\_labels1)

test\_labels2\_indices = label\_encoder2.transform(test\_labels2)

# Apply to\_categorical on numerical indices

train\_labels1\_one\_hot = to\_categorical(train\_labels1\_indices, num\_classes=num\_classes1)

train\_labels2\_one\_hot = to\_categorical(train\_labels2\_indices, num\_classes=num\_classes2)

test\_labels1\_one\_hot = to\_categorical(test\_labels1\_indices, num\_classes=num\_classes1)

test\_labels2\_one\_hot = to\_categorical(test\_labels2\_indices, num\_classes=num\_classes2)

**Code Cell 4:**

from keras.models import Model

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

# Define input shape

input\_shape = (250, 250, 3) # Adjust based on your image size and channels

# Define the input layer

input\_layer = Input(shape=input\_shape, name='input\_layer')

# Convolutional block

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)

# First output branch for label1

dense1 = Dense(128, activation='relu')(flatten)

output1 = Dense(7, activation='softmax', name='output1')(dense1)

# Combine both output branches

model = Model(inputs=input\_layer, outputs=[output1])

# Compile the model

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

**Code Cell 5:**

from keras.models import Model

# Create model1 with output1 branch

model1 = Model(inputs=model.input, outputs=model.get\_layer('output1').output)

model1.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

**Code Cell 6:**

# Define callbacks

es1 = EarlyStopping(monitor='val\_accuracy', min\_delta=0.01, patience=2, verbose=1, restore\_best\_weights=True)

mc1 = ModelCheckpoint("Finally\_Final.keras", monitor="val\_accuracy", verbose=1, save\_best\_only=True)

**Code Cell 7:**

# Fit method for training model1

#model1.fit(

# x=train\_images,

# y=train\_labels1\_one\_hot,

# epochs=10,

#validation\_data=(test\_images, test\_labels1\_one\_hot)

#)

**Code Cell 8:**

# Fit method for training model2

model1.fit(

x=train\_images,

y=train\_labels1\_one\_hot,

epochs=3,

validation\_data=(test\_images, test\_labels1\_one\_hot),

callbacks=[es1, mc1]

)

**Output Cell 8:**

164/164 [==============================] - ETA: 0s - loss: 0.7262 - accuracy: 0.8385

Epoch 1: val\_accuracy improved from -inf to 0.91985, saving model to Finally\_Final.keras

164/164 [==============================] - 108s 644ms/step - loss: 0.7262 - accuracy: 0.8385 - val\_loss: 0.2729 - val\_accuracy: 0.9198

Epoch 2/3

164/164 [==============================] - ETA: 0s - loss: 0.1913 - accuracy: 0.9395

Epoch 2: val\_accuracy improved from 0.91985 to 0.93588, saving model to Finally\_Final.keras

164/164 [==============================] - 90s 551ms/step - loss: 0.1913 - accuracy: 0.9395 - val\_loss: 0.1945 - val\_accuracy: 0.9359

Epoch 3/3

164/164 [==============================] - ETA: 0s - loss: 0.0816 - accuracy: 0.9761

Epoch 3: val\_accuracy improved from 0.93588 to 0.94885, saving model to Finally\_Final.keras

164/164 [==============================] - 89s 541ms/step - loss: 0.0816 - accuracy: 0.9761 - val\_loss: 0.1898 - val\_accuracy: 0.9489

<keras.src.callbacks.History at 0x1f8ecdd9010>

**Code Cell 11:**

# Evaluate model1

loss1, accuracy1 = model1.evaluate(test\_images, test\_labels1\_one\_hot, verbose=1)

print(f"Model1 Accuracy: {accuracy1 \* 100:.2f}%")

**Output Cell 11:**

41/41 [==============================] - 4s 95ms/step - loss: 0.1898 - accuracy: 0.9489

Model1 Accuracy: 94.89%