**A-Z**

**DataCollection.py:**

import cv2

import numpy as np

from cvzone.HandTrackingModule import HandDetector

import math

import time

# Initialize the video capture object to use the default camera (usually the webcam)

cap = cv2.VideoCapture(0)

# Initialize the hand detector with a maximum of 2 hands detection

detector = HandDetector(maxHands=2)

# Constants for image processing

offset = 20

imgSize = 300

folder = "../Data3004/thankyou"

counter = 0

while True:

# Read a frame from the video capture

success, img = cap.read()

# Detect hands in the frame

hands, img = detector.findHands(img)

if hands:

# Loop through each detected hand

for hand in hands:

x, y, w, h = hand['bbox'] # Get bounding box coordinates and size

# Adjust the bounding box if it goes out of the frame boundaries

x = max(0, x)

y = max(0, y)

w = min(img.shape[1] - x, w)

h = min(img.shape[0] - y, h)

# Create a white image of size 300x300 pixels

imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) \* 255

# Crop the hand region with an offset

imgCrop = img[y - offset:y + h + offset, x - offset:x + w + offset]

if imgCrop.shape[0] > 0 and imgCrop.shape[1] > 0:

# Calculate the aspect ratio of the hand bounding box

aspectRatio = h / w

# Adjust the cropped image to fit into the white image based on the aspect ratio

if aspectRatio > 1:

k = imgSize / h

wCal = math.ceil(k \* w)

imgResize = cv2.resize(imgCrop, (wCal, imgSize))

wGap = math.ceil((imgSize - wCal) / 2)

imgWhite[:, wGap:wCal + wGap] = imgResize

else:

k = imgSize / w

hCal = math.ceil(k \* h)

imgResize = cv2.resize(imgCrop, (imgSize, hCal))

hGap = math.ceil((imgSize - hCal) / 2)

imgWhite[hGap:hCal + hGap, :] = imgResize

# Display the cropped and adjusted images

cv2.imshow("ImageCrop", imgCrop)

cv2.imshow("ImageWhite", imgWhite)

else:

print("Error: Cropped image dimensions are invalid")

# Display the original frame with detected hands

cv2.imshow("Image", img)

# Wait for a key press and check if it's the 's' key to save the image

key = cv2.waitKey(1)

if key == ord("s") and hands:

counter += 1

# Save the processed image to the specified folder with a timestamp

cv2.imwrite(f'{folder}/Image\_{time.time()}.jpg', imgWhite)

print(f"Saved Image\_{counter}.jpg")

**Train.py:**

import os

import numpy as np

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

from PIL import Image

import tensorflow as tf

from tensorflow.keras import layers, models, callbacks

# Define constants for image dimensions and dataset parameters

image\_height = 64

image\_width = 64

num\_classes = 26 # Number of classes (A-Z)

batch\_size = 32

epochs = 20 # Number of epochs for training

dataset\_dir = "../Data" # Directory containing the dataset

# Function to load the dataset from the given directory

def load\_dataset(dataset\_dir):

images = []

labels = []

label\_map = {} # Map class names to integer labels

for class\_index, class\_name in enumerate(sorted(os.listdir(dataset\_dir))):

class\_dir = os.path.join(dataset\_dir, class\_name)

if not os.path.isdir(class\_dir):

print(f"Skipping {class\_dir}. Not a directory.")

continue # Skip non-directory files

label\_map[class\_name] = class\_index

for image\_name in os.listdir(class\_dir):

image\_path = os.path.join(class\_dir, image\_name)

try:

# Load and resize image

image = Image.open(image\_path).resize((image\_height, image\_width))

images.append(np.array(image))

labels.append(class\_index) # Use class\_index as label

except Exception as e:

print(f"Error loading {image\_path}: {e}")

return np.array(images), np.array(labels), label\_map

# Load the dataset

try:

X, y, label\_map = load\_dataset(dataset\_dir)

except Exception as e:

print(f"Error loading dataset: {e}")

exit()

# Split the data into training, validation, and test sets

try:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.15, random\_state=42)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train, y\_train, test\_size=0.15, random\_state=42)

except Exception as e:

print(f"Error splitting dataset: {e}")

exit()

# Data augmentation to enhance the training dataset

data\_augmentation = tf.keras.Sequential([

layers.experimental.preprocessing.RandomRotation(0.2),

layers.experimental.preprocessing.RandomZoom(0.1),

layers.experimental.preprocessing.RandomFlip(mode="horizontal"),

])

# Define the Convolutional Neural Network (CNN) model

model = models.Sequential([

data\_augmentation, # Apply data augmentation

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(image\_height, image\_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.Dropout(0.5), # Add dropout for regularization

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

layers.Dense(num\_classes, activation='softmax') # Output layer for classification

])

# Compile the model with optimizer, loss function, and metrics

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Define early stopping callback to prevent overfitting

early\_stopping = callbacks.EarlyStopping(monitor='val\_loss', patience=3, restore\_best\_weights=True)

# Train the model

try:

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

validation\_data=(X\_val, y\_val), callbacks=[early\_stopping])

except Exception as e:

print(f"Error training the model: {e}")

exit()

# Evaluate the model on the test set

try:

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

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

except Exception as e:

print(f"Error evaluating the model: {e}")

# Save the trained model to a file

try:

model.save(keras\_model.h5')

print("Model saved successfully.")

except Exception as e:

print(f"Error saving the model: {e}")

**Test.py:**

import cv2

from cvzone.HandTrackingModule import HandDetector

from cvzone.ClassificationModule import Classifier

import numpy as np

import math

import time

# Initialize the webcam

cap = cv2.VideoCapture(0)

# Initialize the hand detector (detects a maximum of 1 hand) and the classifier

detector = HandDetector(maxHands=1)

classifier = Classifier("../model/keras\_model.h5", "../model/labels.txt")

# Define constants

offset = 20 # Offset for cropping the hand region

imgSize = 64 # Resize the image to 64x64 pixels for the classifier

labels = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"]

while True:

# Read a frame from the webcam

success, img = cap.read()

imgOutput = img.copy() # Create a copy of the frame for output display

# Use the hand detector to find hands in the frame

hands, img = detector.findHands(img)

# If a hand is found

if hands:

hand = hands[0] # Get the first detected hand

x, y, w, h = hand['bbox'] # Get bounding box coordinates and size

imgCrop = img[y - offset:y + h + offset, x - offset:x + w + offset] # Crop the hand region with an offset

# Check if imgCrop is empty

if imgCrop.size == 0:

print("Error: Empty region of interest (ROI)")

continue

# Resize the cropped image to match the expected input size of the model (64x64 pixels)

imgResize = cv2.resize(imgCrop, (imgSize, imgSize))

# Pass the resized image to the classifier for prediction

prediction, index = classifier.getPrediction(imgResize)

# Display the predicted label on the original image

cv2.putText(imgOutput, labels[index], (x, y - 20), cv2.FONT\_HERSHEY\_COMPLEX, 2, (255, 0, 255), 2)

# Show the cropped image

cv2.imshow("ImageCrop", imgCrop)

# Show the original image with annotations

cv2.imshow("Image", imgOutput)

# Wait for a key press, and break the loop if 'q' is pressed

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

break

# Release the webcam and close all OpenCV windows

cap.release()

cv2.destroyAllWindows()