Graphical user interface, text

Description automatically generated

Baseline Pipeline

The following code was used in Python to prepare the initial dataset used. When the code is run, it opens the webcam, and I would then signal different ASL letters with my hand and saving images through the “s” key. A challenge I faced was since the ASL hand signs had different proportions (some being taller, some wider etc.) I had to come up with a solution to make all the images uniform somehow. Thus, the images saved were saves as a 300x300 image no matter the different proportions. This was done through adding a white background to and resizing images to become more uniform. I also added a counter to help in knowing the amount of images saved.

For the baseline implementation, around 100 images were taken for each of the letters: A, B, C and D.

import cv2  
from cvzone.HandTrackingModule import HandDetector  
import numpy as np  
import math  
import time  
  
  
videoCapture = cv2.VideoCapture(0)  
detect = HandDetector(maxHands=1)  
offsetValue = 25  
imageSize = 300  
counter = 0  
  
#Change path for different letters!  
folder = "Data/A"  
  
while True:  
 #find and capture hand  
 successful, image = videoCapture.read()  
 hands, image = detect.findHands(image)  
  
 #crop image to uniform shape  
 if hands:  
 capturedHand = hands[0]  
 x, y, w, h = capturedHand['bbox']  
  
 imageWithBackground = np.ones((imageSize, imageSize, 3), np.uint8) \* 255  
 imageCropped = image[y - offsetValue:y + h+offsetValue, x-offsetValue:x + w+offsetValue]  
 imageCroppedShape = imageCropped.shape  
 heightOverWidth = (h/w)  
  
 #if statement to keep proportion of images  
 if heightOverWidth > 1:  
 constant = imageSize / h  
 calculatedWidth = math.ceil(constant \* w)  
 resizedImage = cv2.resize(imageCropped, (calculatedWidth, imageSize))  
 resizedImageShape = resizedImage.shape  
 widthDiff = math.ceil((300 - calculatedWidth) \* 0.5)  
 imageWithBackground[:resizedImageShape[0], widthDiff:calculatedWidth+widthDiff] = resizedImage  
 else:  
 constant = imageSize / w  
 calculatedHeight = math.ceil(constant \* h)  
 resizedImage = cv2.resize(imageCropped, (imageSize, calculatedHeight))  
 resizedImageShape = resizedImage.shape  
 heightDiff = math.ceil((imageSize - calculatedHeight) \* 0.5)  
 imageWithBackground[heightDiff:calculatedHeight + heightDiff, :] = resizedImage  
  
 cv2.imshow("ImageCrop", imageCropped)  
 cv2.imshow("ImageWithBackground", imageWithBackground)  
  
  
 #output  
 cv2.imshow("Image", image)  
 key = cv2.waitKey(1)  
  
 #saving images  
 if key == ord("s"):  
 counter = counter + 1  
 cv2.imwrite(f'{folder}/Image\_{time.time()}.jpg', imageWithBackground)  
 print(counter)

Baseline Solution:

Classifier Code:

import cv2  
from cvzone.HandTrackingModule import HandDetector  
from cvzone.ClassificationModule import Classifier  
import numpy as np  
import math  
  
  
def main():  
 classify()  
  
   
def classify():  
 videoCapture = cv2.VideoCapture(0)  
 detect = HandDetector(maxHands=1)  
 offsetValue = 25  
 imageSize = 300  
 classifier = Classifier("Model/keras\_model.h5", "Model/labels.txt")  
 labels = ["A", "B", "C", "D"]  
  
 while True:  
 # To check if ESC key is pressed  
 key = cv2.waitKey(1) & 0xFF  
 if key == 27: # If ESC key is pressed  
 break # Exit the loop  
  
 success, image = videoCapture.read()  
 imageOutput = image.copy()  
 hands = detect.findHands(image, draw=False)  
  
 if hands:  
 hand = hands[0]  
 x, y, w, h = hand["bbox"]  
  
 imageWithBackground = np.ones((imageSize, imageSize, 3), np.uint8) \* 255  
 imageCropped = image[  
 y - offsetValue : y + h + offsetValue,  
 x - offsetValue : x + w + offsetValue,  
 ]  
 heightOverWidth = h / w # aspect ratio  
  
 # if statement to keep proportion of images, depending on aspect ratio  
 if heightOverWidth > 1: # if image height is larger than width  
 constant = imageSize / h  
 calculatedWidth = math.ceil(constant \* w)  
 resizedImage = cv2.resize(  
 imageCropped, (calculatedWidth, imageSize)  
 ) # resize the image  
 resizedImageShape = resizedImage.shape  
 widthDiff = math.ceil((300 - calculatedWidth) \* 0.5)  
 imageWithBackground[  
 : resizedImageShape[0], widthDiff : calculatedWidth + widthDiff  
 ] = resizedImage  
 prediction, index = classifier.getPrediction(  
 imageWithBackground  
 ) # get prediction and index value  
  
 else: # if image width is larger than height  
 constant = imageSize / w  
 calculatedHeight = math.ceil(constant \* h)  
 resizedImage = cv2.resize(  
 imageCropped, (imageSize, calculatedHeight)  
 ) # resize the image  
 heightDiff = math.ceil((imageSize - calculatedHeight) \* 0.5)  
 imageWithBackground[  
 heightDiff : calculatedHeight + heightDiff, :  
 ] = resizedImage  
 prediction, index = classifier.getPrediction(  
 imageWithBackground, draw=False  
 )  
  
 # styling output  
 cv2.rectangle(  
 imageOutput,  
 (x - offsetValue, y - offsetValue - 50),  
 (x - offsetValue + 90, y - offsetValue),  
 ((0, 255, 0)),  
 cv2.FILLED,  
 )  
 cv2.putText(  
 imageOutput,  
 labels[index],  
 (x, y - 30),  
 cv2.FONT\_HERSHEY\_DUPLEX,  
 1.5,  
 (0, 0, 0),  
 2,  
 )  
 cv2.rectangle(  
 imageOutput,  
 (x - offsetValue, y - offsetValue),  
 (x + w + offsetValue, y + h + offsetValue),  
 ((0, 255, 0)),  
 3,  
 )  
  
 cv2.imshow("Image", imageOutput)  
 cv2.waitKey(1)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Interface Code (NEEDS CHANGING):

import customtkinter as ctk  
from PIL import Image  
from Classifier\_in\_functions import classify  
  
class MainWindow:  
 def \_\_init\_\_(self, master):   
   
 # Creating window  
 master.geometry("320x190")  
 master.resizable(False, False)  
 master.wm\_title("Sign Language Convertor")  
  
 # Creating grid  
 master.grid\_rowconfigure(0, weight=1)  
 master.grid\_columnconfigure(0, weight=1)  
  
 my\_font = ("Helvetica", 14)  
  
 # create buttons  
 master.start = ctk.CTkButton(master, text="Start", font=my\_font, width=150, height=30, command=lambda: start())  
 master.exit = ctk.CTkButton(master, text="Exit", font=my\_font, width=150, height=30, command=lambda: close())  
  
 # add buttons to the grid  
 master.start.grid(row=0, column=0, columnspan=1, padx=70, pady=(60,5), sticky="nsew")  
 master.exit.grid(row=1, column=0, columnspan=1, padx=70, pady=(5,60), sticky="nsew")  
   
   
 def start():  
 classify()  
  
 def close():  
 master.quit()  
  
  
  
def main():  
 root = ctk.CTk()  
 window = MainWindow(root)  
 root.mainloop()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Model Code:

To create the model, I used Google’s teachablemachine website, where I inputted the training data I created and set the epoch value to 50. With a learning rate of 0.01 and a batch size of 16. After the training was done I received a keras model as well as a label text file.

from keras.models import load\_model # TensorFlow is required for Keras to work  
from PIL import Image, ImageOps # Install pillow instead of PIL  
import numpy as np  
  
# Disable scientific notation for clarity  
np.set\_printoptions(suppress=True)  
  
# Load the model  
model = load\_model("keras\_Model.h5", compile=False)  
  
# Load the labels  
class\_names = open("labels.txt", "r").readlines()  
  
# Create the array of the right shape to feed into the keras model  
# The 'length' or number of images you can put into the array is  
# determined by the first position in the shape tuple, in this case 1  
data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)  
  
# Replace this with the path to your image  
image = Image.open("<IMAGE\_PATH>").convert("RGB")  
  
# resizing the image to be at least 224x224 and then cropping from the center  
size = (224, 224)  
image = ImageOps.fit(image, size, Image.Resampling.LANCZOS)  
  
# turn the image into a numpy array  
image\_array = np.asarray(image)  
  
# Normalize the image  
normalized\_image\_array = (image\_array.astype(np.float32) / 127.5) - 1  
  
# Load the image into the array  
data[0] = normalized\_image\_array  
  
# Predicts the model  
prediction = model.predict(data)  
index = np.argmax(prediction)  
class\_name = class\_names[index]  
confidence\_score = prediction[0][index]  
  
# Print prediction and confidence score  
print("Class:", class\_name[2:], end="")  
print("Confidence Score:", confidence\_score)

Labels File:

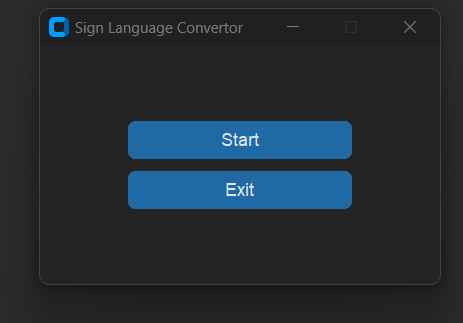
0 A  
1 B  
2 C  
3 D

Baseline Evaluation:

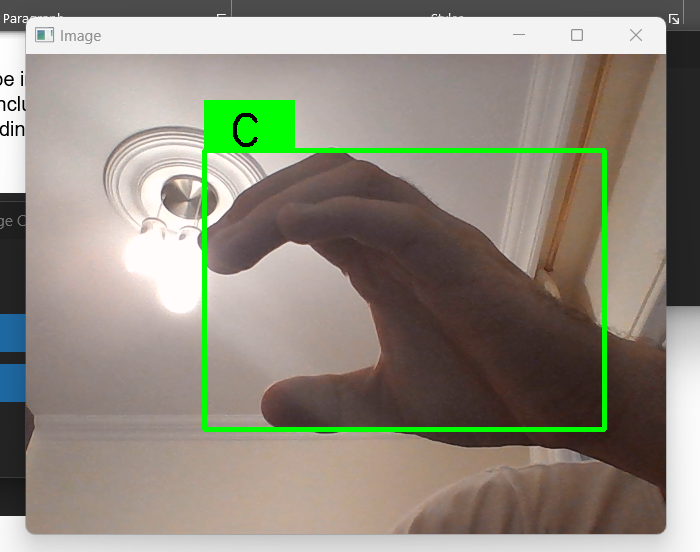
The developed baseline solution works without any errors and functions as desired; this includes the data collection code in which data is collected successfully and saved in the designated folder, the classifier code in which the program is able to classify certain ASL letters, and the interface where the start and exit buttons function as needed.

On the other hand, there are some various accuracy issues in the model recognition. This leads us to knowing that a better dataset should be used for the iterative solution and the number of epochs used in training should be increased to facilitate better accuracy.

The solution can be improved in the iterative development process by increasing the model accuracy, including support for the rest of the ASL alphabet and the additional data we will be adding, and attempt to make the interface code run faster.

Interface Image:

Classifier In Action:



Environment Setup:

To be able to run the code successfully, python 3.8 needs to be installed and set in the interpreter. Furthermore, the following libraries need to be installed.

* Tensorflow
* Mediapipe
* Cvzone
* Cv2
* Numpy
* Pandas
* Keras
* Customtkinter
* Pillow (PIL)
* Math
* Time

Additionally, the files need to be organized in the following manner. With the model file and the labels file in a folder named Model.

**Graphical user interface, text, application

Description automatically generated**