Baseline Implementation Documentation

Team Bothal

Development of SignPal

Table of Contents

Baseline Pipeline

The following code was developed in Python and used to prepare the initial dataset. 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 white bars to either side of the images and resizing the images to become more uniform.

I also added a counter that increments each time an image is saved to help in knowing the number of images saved. Around 1000 images of each letter were taken.

Furthermore, there were slight hand movements when taking the images such as horizontal/vertical movement and slight rotation to allow for a more natural hand position to be recognized.

For the baseline implementation, the implementation was designed to detect the letters A, B, C, D and E.

Data Collection Code:

```
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:

Moving on, Python code was developed using OpenCV and Cvzone to be able to detect hand gestures. The code detects hands using the HandDetector library and uses the Keras model and the corresponding labels to classify the performed hand sign/gesture. When the sign is detected, a bounding box is outputted over the hand alongside the corresponding label. The “esc” key is used to exit the program.

To encapsulate the learning parts of the system, the code was split to separate and different files such as a classifier, an interface, and a model. The function classify() is imported from the classifier code into the interface. Furthermore, the model is loaded from a separate folder named Model which contains the keras model. This allows rapid iteration and testing of different models in the future where the file can easily be modified without any hassle.

Furthermore, a unit test was created using the pytest library to be able to test the developed system and helps catching errors and ensuring that the code is working as expected. There were 4 tests, 3 for detecting the letters A, B and C respectively and one for detecting that a hand is in the image.

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", "E"]

    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,
```

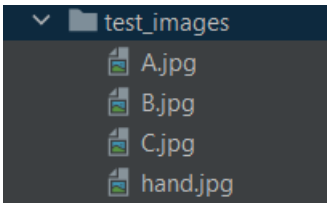
```
            )
            cv2.rectangle(
                imageOutput,
                (x - offsetValue, y - offsetValue),
                (x + w + offsetValue, y + h + offsetValue),
                ((0, 255, 0)),
                3,
            )

        cv2.imshow("Image", imageOutput)
        cv2.waitKey(1)

if __name__ == "__main__":
    main()
```

Interface Code:

```
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()
```

Unit Test Code:

To ensure that the unit test code will work properly, make sure that the test_images directory is available with the four images inside.




```python
import cv2
import pytest
import cvzone
from cvzone.ClassificationModule import Classifier
from cvzone.HandTrackingModule import HandDetector

@pytest.fixture(scope="module")
def classifier():
    model_path = "Model/keras_model.h5"
    classifier = cvzone.ClassificationModule.Classifier(model_path)
    return classifier


def test_classifier_a(classifier):
    image = cv2.imread("test_images/A.jpg")
    image = cv2.resize(image, (300, 300))
    prediction, index = classifier.getPrediction(image)
    classes = ["A", "B", "C", "D", "E"]
    assert classes[index] == "A"

def test_classifier_b(classifier):
    image = cv2.imread("test_images/B.jpg")
    image = cv2.resize(image, (300, 300))
    prediction, index = classifier.getPrediction(image)
    classes = ["A", "B", "C", "D", "E"]
    assert classes[index] == "B"

def test_classifier_c(classifier):
    image = cv2.imread("test_images/C.jpg")
    image = cv2.resize(image, (300, 300))
    prediction, index = classifier.getPrediction(image)
    classes = ["A", "B", "C", "D", "E"]
    assert classes[index] == "C"

@pytest.fixture(scope="module")
def hand_detector():
    return HandDetector(maxHands=1)


def test_hand_detector(hand_detector):
    image = cv2.imread("test_images/hand.jpg")
    image = cv2.resize(image, (640, 480))
    hands = hand_detector.findHands(image)
    assert len(hands) > 1
```

Model Code:

To create the model, I used Google's teachablemachine website, where I input 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 complete, I received a Keras model as well as a label text file which were placed in the Model folder in the working directory.

```
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
4 E
```

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 can classify certain ASL letters, and the interface where the start button functions as intended, there were several issues in the exit button as the classifier code would not close through the interface, this will be improved throughout the iterative solution process.

On the other hand, the model's accuracy was good but there were some various accuracy issues in the model recognition such as some confusion between the letters A and E. 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 since the full solution will have 26+ letters.

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.

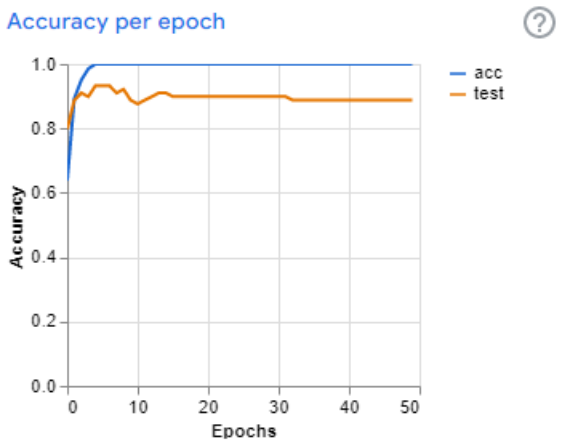
Model Accuracy:

Model Accuracy per Class

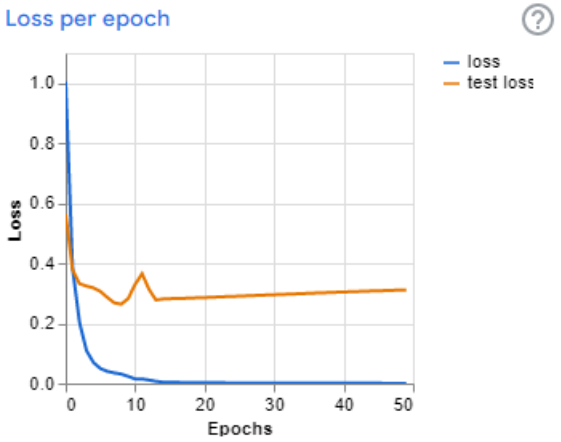
Class	Accuracy	Number Of Samples
A	0.83	18
B	0.89	19
C	0.94	17
D	0.89	19
E	0.86	17

15% of the input data was used to test the model while 85% was used to train the model. An average accuracy level of 0.88 was achieved which was significantly good.

Model Accuracy per Epoch

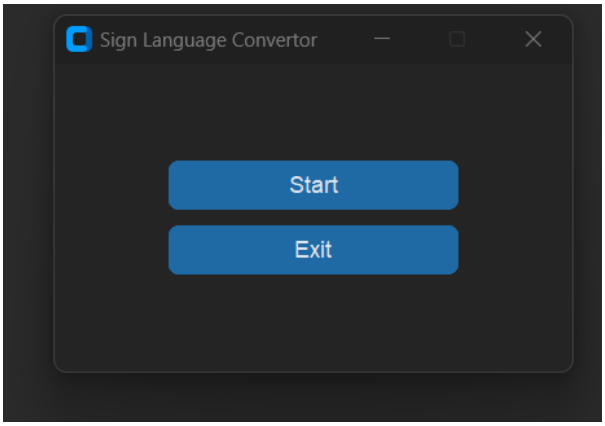


Model Loss per Epoch

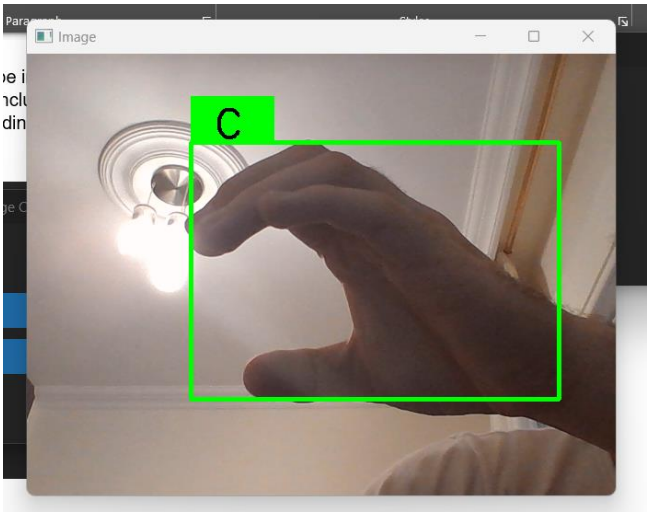


Images of Solution in Action

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
- Pytest

Additionally, the files need to be organized in the following manner with the same directories. With the Keras model file and the labels file in a folder named Model. Furthermore, it should be noted that the device running the code should have a webcam enabled for the code not to produce any errors.