Real-Time Sign Language Translation

Introduction

I have always wanted to experiment with AI and gesture recognition. I had some experience with python before, however, the development of this program took quite some time. Though the process was challenging, I was able to finish a completed program and I hope to further improve the speed and accuracy. This article describes the code in detail and explains the ups and downs as this program was created. The finished real-time program is designed to detect your hand gestures (American Sign Language) and display its English translation on-screen. With this program, I wish to help bridge the communication gap between deaf and hearing people.

Background

Gesture recognition has long been an extensively researched field in the Computer Vision community. In real time, hand segmentation can be a challenging problem. Humans can easily be taught to differentiate between different objects and textures, however for machines, images are essentially just 3-dimensional arrays.

Problem Motivation

Through the pandemic and working fully remotely, some have had more difficulty adapting than others. On video meeting apps, such as zoom and google meeting, those who are hearing impaired might have struggled communicating with others on video. This program allows easier communication while working remotely. The program can also be used to teach potential signers. The user can match each hand gesture to the letter and learn quickly using the program.

Overview

In order for machines to recognize images, we model the human brain using neural networks. A neural network is like a flowchart, outlining questions that eventually lead to a decision. By using neural networks and matrices, programmers are able to make a model that can detect, calculate, and solve. The accuracy and speed of AI is incomparable to traditional methods.

For this program, I referenced:
training and classifying- https://github.com/xuetsing/image-classification-tensorflow
hand detection and screen display - https://techvidvan.com/tutorials/hand-gesture-recognition-tensorflow-opencv/
dataset- https://www.kaggle.com/datasets/grassknoted/asl-alphabet?resource=download

Setup

I decided to use python as the main language. Python has access to large libraries and frameworks for machine learning. I also ran this program on a Macbook, my main development platform.

Development

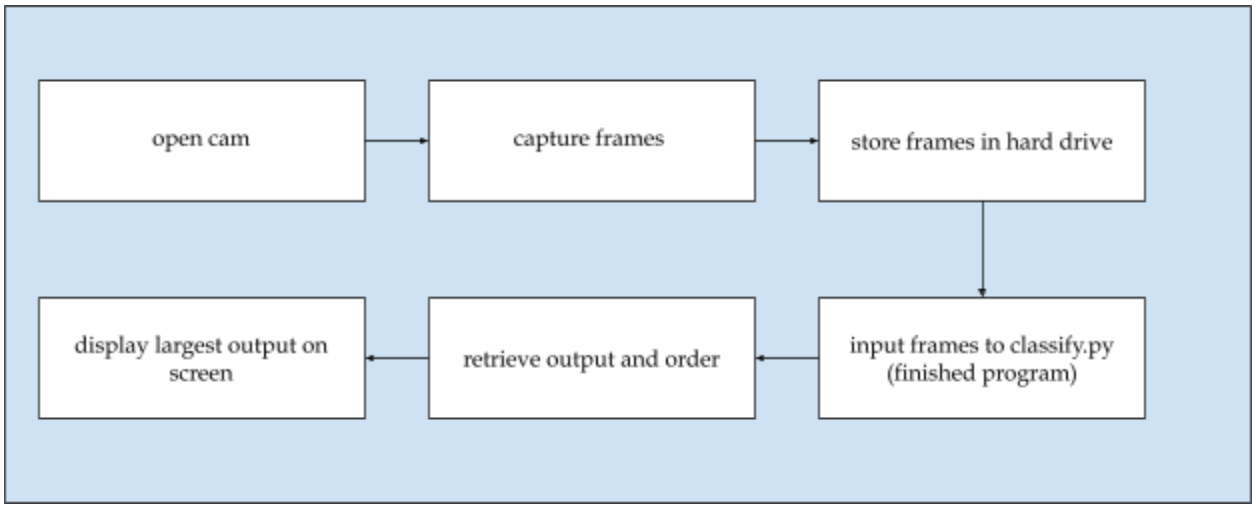
I used a pre-trained model that is able to classify images and output a corresponding letter that best matches the image. In order to make it real-time, I used cv2 to capture the frames and input the images into the model. An output is given and displayed on the screen.

The following python libraries are used:

- opencv
- numpy
- tensorflow

Design

This is the initial layout for the program. It splits the code into 6 sections, with each section performing a specific task.



Implementation

The following is an in-depth description of the main program (run.py). It explains the main blocks of the code: importation, initialization, reading/inputting, and display.

import packages-

```
import cv2
import numpy as np
import os
import io
import tensorflow as tf
import tensorflow.compat.v1 as tf
from numpy import array
from tensorflow.keras.models import load_model
from PIL import Image
```

import all necessary modules:

- cv2 used for capturing frames
- numpy used for ordering numbers
- os used for disabling warnings
- io used for reading files and creating a binary stream
- tensorflow used for loading detection/translation model
- PIL used for image conversion (see error #1)

initialize model-

```
# c:disable tensorflow compilation warnings
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'

# m:initialize the webcam
cap = cv2.VideoCapture(0)

# c:loads label file, strips off carriage return
label_lines = [line.rstrip() for line
               in tf.io.gfile.GFile("image-classification-tensorflow/logs/trained_labels.txt")]

# c:unpersists graph from file; load model
with tf.io.gfile.GFile("image-classification-tensorflow/logs/trained_graph.pb", 'rb') as f:
   graph_def = tf.compat.v1.GraphDef()
   graph_def.ParseFromString(f.read())
   _ = tf.import_graph_def(graph_def, name='')
```

fix any compilation warnings:

- ‘1’ means info messages are not printed
- ‘2’ means info and warning messages should not be printed
- ‘3’ means info, warning, and error messages are not printed.

initialize camera:

- 0 is the camera index/default camera

folder structure:

- the main directory contains a folder for training and classifying, (image-classification-tensorflow) and the main program
- ‘trained-labels.txt’ is in the training/classifying folder.
- ‘trained-labels.txt’ contains a list of possible outcomes (a-z, del, space, nothing)

load the trained tensorflow model:

- ‘trained_graph.pb’ is the name of pretrained model
- use tensorflow command ‘tf.io.gfile.Gfile’
- ‘rb’ means to read binary file

read and input frames-

```
while True:
    # m:read each frame from the webcam
    ret, frame = cap.read()

    x, y, c = frame.shape

    # m:flip the frame vertically
    frame = cv2.flip(frame, 1)
    framergb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    im = Image.fromarray(framergb)
    with io.BytesIO() as output:
        im.save(output, format="PNG")
        frames = output.getvalue()



    with tf.compat.v1.Session() as sess:
        # c:feed the image_data
        softmax_tensor = sess.graph.get_tensor_by_name('final_result:0')
        predictions = sess.run(softmax_tensor, \
                {'DecodeJpeg/contents:0': frames})
```

while loop:

- ‘while True’ keeps the program running until interrupted by pressing “q” key

read frames:

- ‘cap.read()’ used to read frames

display:

- ‘cv2.flip’ used to flip the frame (so camera is mirrored)
- ‘cv2.cvtColor’ changes BGR to RGB (opencv uses BGR)
- ‘im.save’ saves image as png (so image can be input into model)

input image data:

- ‘sess.run’ is tensorflow command that runs operations
- ‘DecodeJpeg’ is the name of input node

display-

```
# m:print(prediction)
   classID = np.argmax(predictions)
   className = label_lines[classID]

   # m:show the prediction on the frame
   cv2.putText(frame, className, (50,50), cv2.FONT_HERSHEY_SIMPLEX,
               1, (128,128,128), 2, cv2.LINE_AA)

   # m:show the final output
   cv2.imshow("gsproject", frame)

   if cv2.waitKey(1) == ord('q'):
      break

   print("finish frame")

# m: release the webcam and destroy all active windows
cap.release()

cv2.destroyAllWindows()
```

print prediction:

- numpy sorts prediction results (decimal form) from greatest to least
- ‘cv2.putText’ displays outcome on screen

final program:

- ‘cv2.imshow’ shows frame in OpenCV window
- “gsproject” is window name
- ‘cv2.waitkKey’ sets the condition that if “q” is pressed, you will exit the program
- “finish frame” is printed once a frame is classified and output is printed
- ‘cap.release’ destroys active windows and closes webcam

Problems Encountered

When I wrote the program, there were two main errors: a ValueError and AttributeError. For the first error, when the frame was read, it was saved with no shape, so I redefined it as a PNG file. For the second error, if you have TF2.0, the app function will not be supported, so you need to import tensorflow.compat.v1.

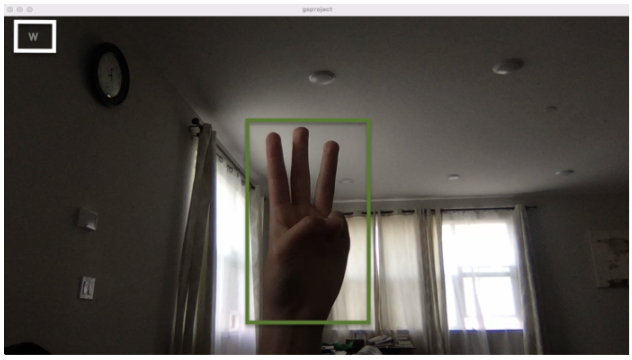
Error	*ValueError: Cannot feed value of shape (720, 1280, 3) for Tensor DecodeJpeg/contents:0, which has shape ()*
Problem	TensorFlow does not have “shape”
Solution	save image in specific format (png, jpg, etc.) *im.save(‘image’, format="PNG")*

Error	*AttributeError: module 'tensorflow' has no attribute 'app'*
Problem	TensorFlow does not have ‘app’ function
Solution	use *import tensorflow.compat.v1 as tf*

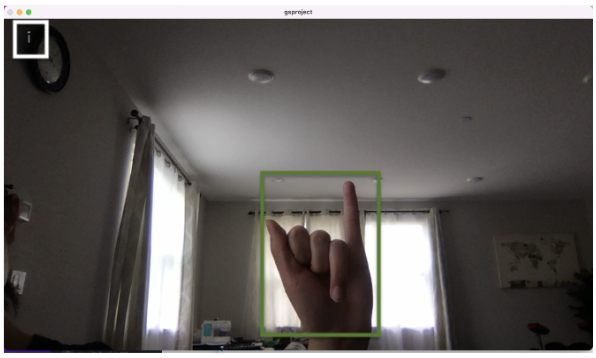
Result

Below are screenshots of the final product. The white box shows the letter translation, and the green box isolates the hand. 2 of the pictures are just the hand and the other two show the user in frame as well. DIfferent lighting is also varied in the pictures.

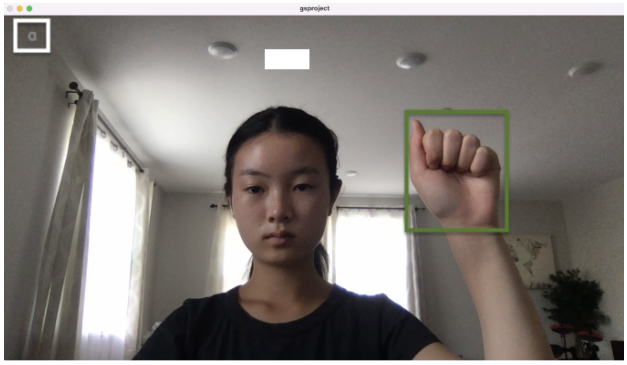
letter w:



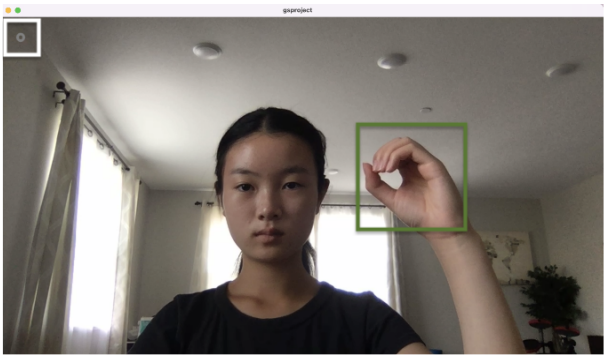
letter i:



letter a:



letter o:



Summary

In this gesture recognition project, I was able to build a program using opencv, numpy, and python. By using Tensorflow framework for detection and translation, we have learned about the basics of machine learning and AI.

Further Improvements

In the future, I want to improve performance speed. Currently, I have to input an image from hard drive into the model, so the speed of the program is slow. Once I gain more experience in machine learning, I want to optimize the model speed.

References

Akash. “Asl Alphabet.” *Kaggle*, 22 Apr. 2018, https://www.kaggle.com/datasets/grassknoted/asl-alphabet?resource=download.

Gogul Ilango. “Hand Gesture Recognition Using Python and Opencv - Part 1.” *Gogul Ilango*, 6 Apr. 2017, https://gogul.dev/software/hand-gesture-recognition-p1.

Team, TechVidvan. “Real-Time Hand Gesture Recognition Using Tensorflow & Opencv.” *TechVidvan*, 21 July 2021, https://techvidvan.com/tutorials/hand-gesture-recognition-tensorflow-opencv/.

Xuetsing. “Xuetsing/Image-Classification-Tensorflow: Simple Transfer Learning with an Inception V3 Architecture Model.” *GitHub*, https://github.com/xuetsing/image-classification-tensorflow.