Hand Sign Language Recognition Using Machine Learning And Computer Vision

Guided by – Dr. Subrata Kumar Mandal

(Professor)

Submited by – Shreetama Mukherjee

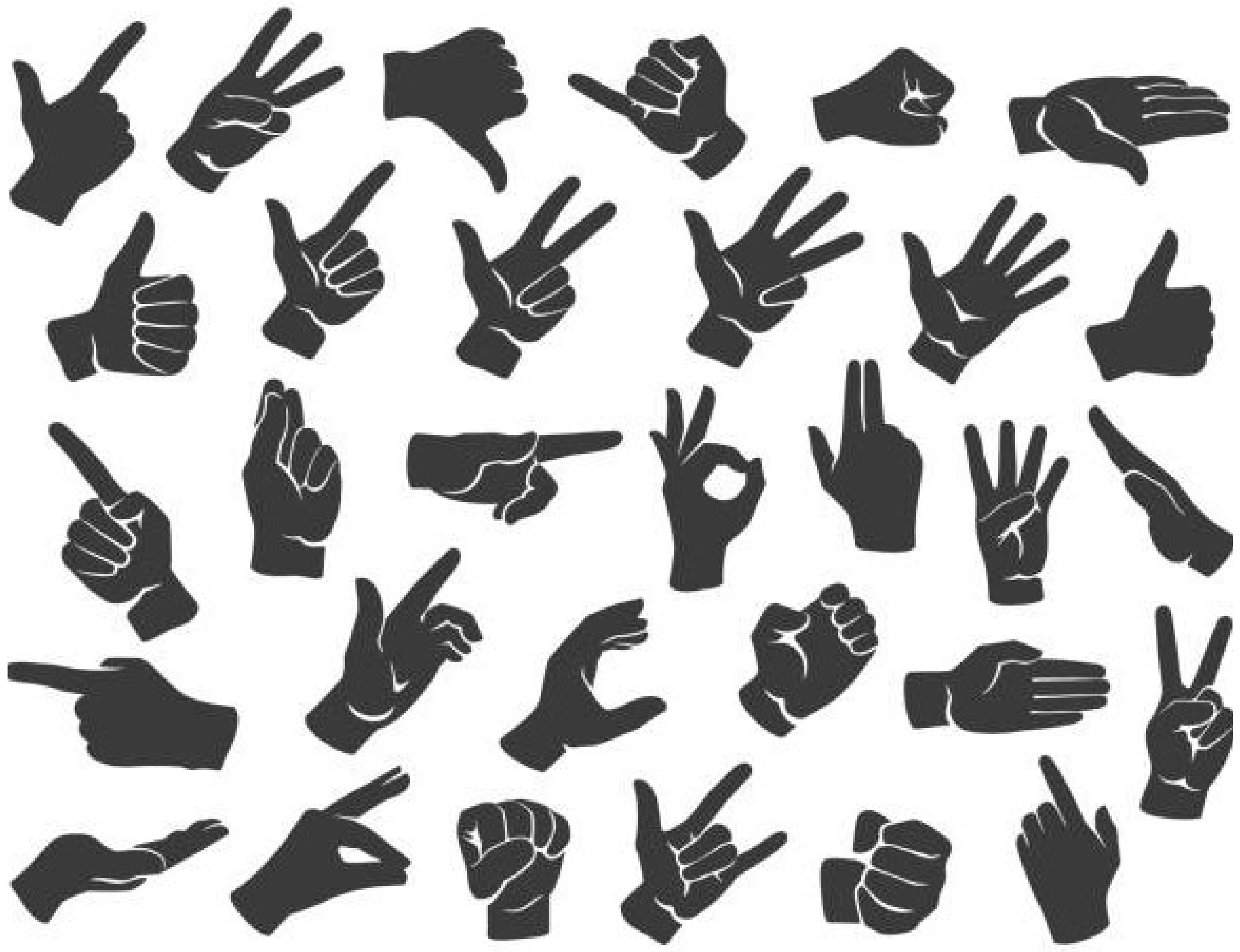
Akash Kumar Shaw

Abhishek Srivastav

Shouvik De

Department of Information Technology

(2019-23)



CONTENTS

INTRODUCTION

- Hand Sign Language Translation using Computer Vision is a technology that aims to recognize and translate hand signs into spoken or written language without relying on neural networks. This technology uses computer vision techniques to detect and track hand movements in real-time video streams, and then applies image processing algorithms to recognize the hand signs. This approach typically involves detecting the hand region, segmenting the hand from the background, and extracting relevant features, such as hand shape and orientation. These features are then matched with a pre-defined dictionary of hand signs to recognize the sign and translate it into spoken or written language.

PROBLEM STATEMENT

- The problem addressed in this report is the communication barrier faced by the deaf and hard-of-hearing community due to the lack of understanding of sign language by the general public. Sign language is an essential tool for communication for these individuals, and the inability of others to comprehend it can lead to social exclusion and communication barriers in various aspects of life, such as education, employment, and social interactions.

PROPOSED WORK

1. Feature extraction: Developing more robust and efficient feature extraction methods for recognizing hand signs, such as using shape analysis, texture analysis, or motion analysis.
2. Sign dictionary: Developing a comprehensive sign dictionary that includes a wider range of hand signs, including more complex signs and phrases.
3. Real-time tracking: Developing more advanced real-time tracking methods that can track hand movements in 3D space, which could help to improve the recognition accuracy and robustness of the system.
4. User interface: Developing a more user-friendly and accessible interface for the system, such as a mobile application or a web-based platform.
5. Multilingual support: Expanding the system to recognize and translate other sign languages used around the world, which could improve communication access for the global deaf and hard-of-hearing community.

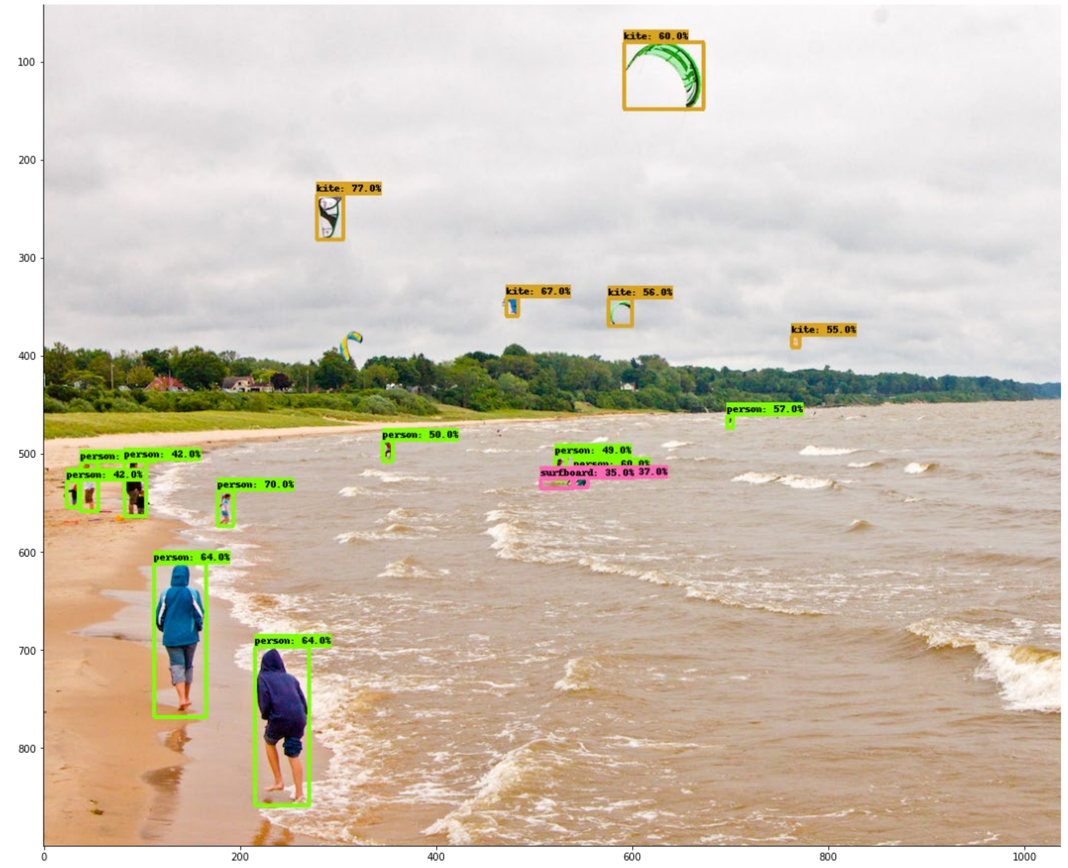
ONE STAGE/PROPOSAL FREE DETECTORS

- Where object detection is a simple regression problem that takes input and learns probability classes and bounding box coordinates. YOLO, YOLO v2, SSD, RetinaNet, etc., fall under one phase detector. Object detection is an advanced form of imaging classification where a neural network predicts objects in an image and draws attention to them in the form of bounding boxes.

SSD (Single Shot Detector)

The term SSD stands for Single Shot Detector. The SSD technique is based on a forward convolutional network that generates a collection of fixed-size bounding boxes and a score for the presence of object class instances in those boxes. The main features of SSD are speed, high accuracy, and learning ability.

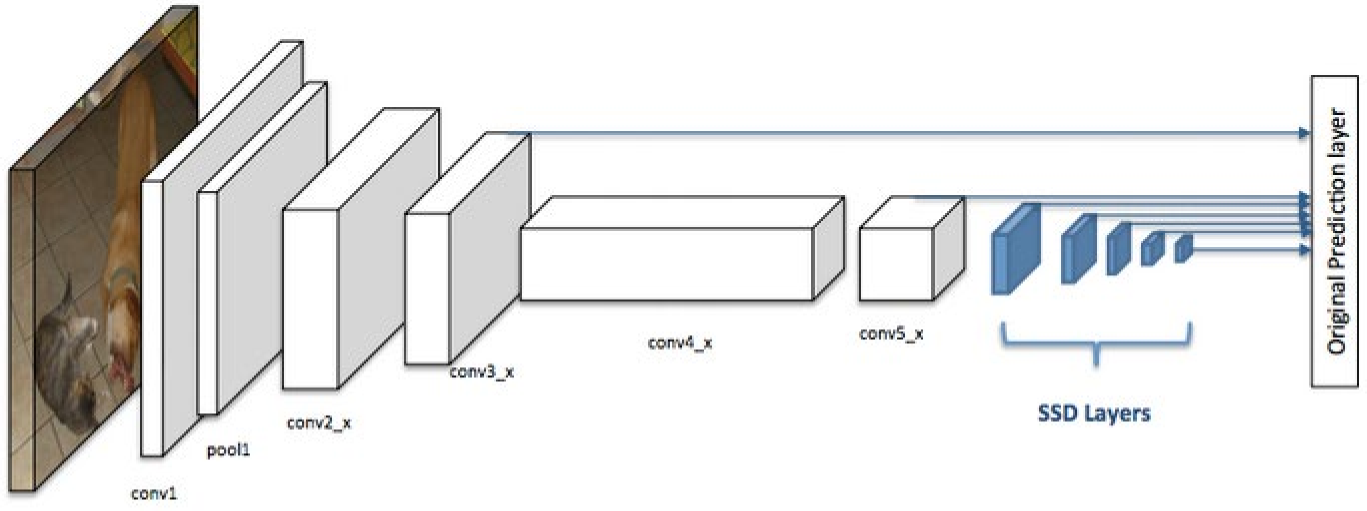
Speed - This algorithm has improved speed for real-time object detection.
High Accuracy - This technique gives accurate outcomes with minimal background errors.
Learning Ability - The excellent learning ability of this algorithm allows it to learn object representations and apply them in object detection.



MOBILENET SSD

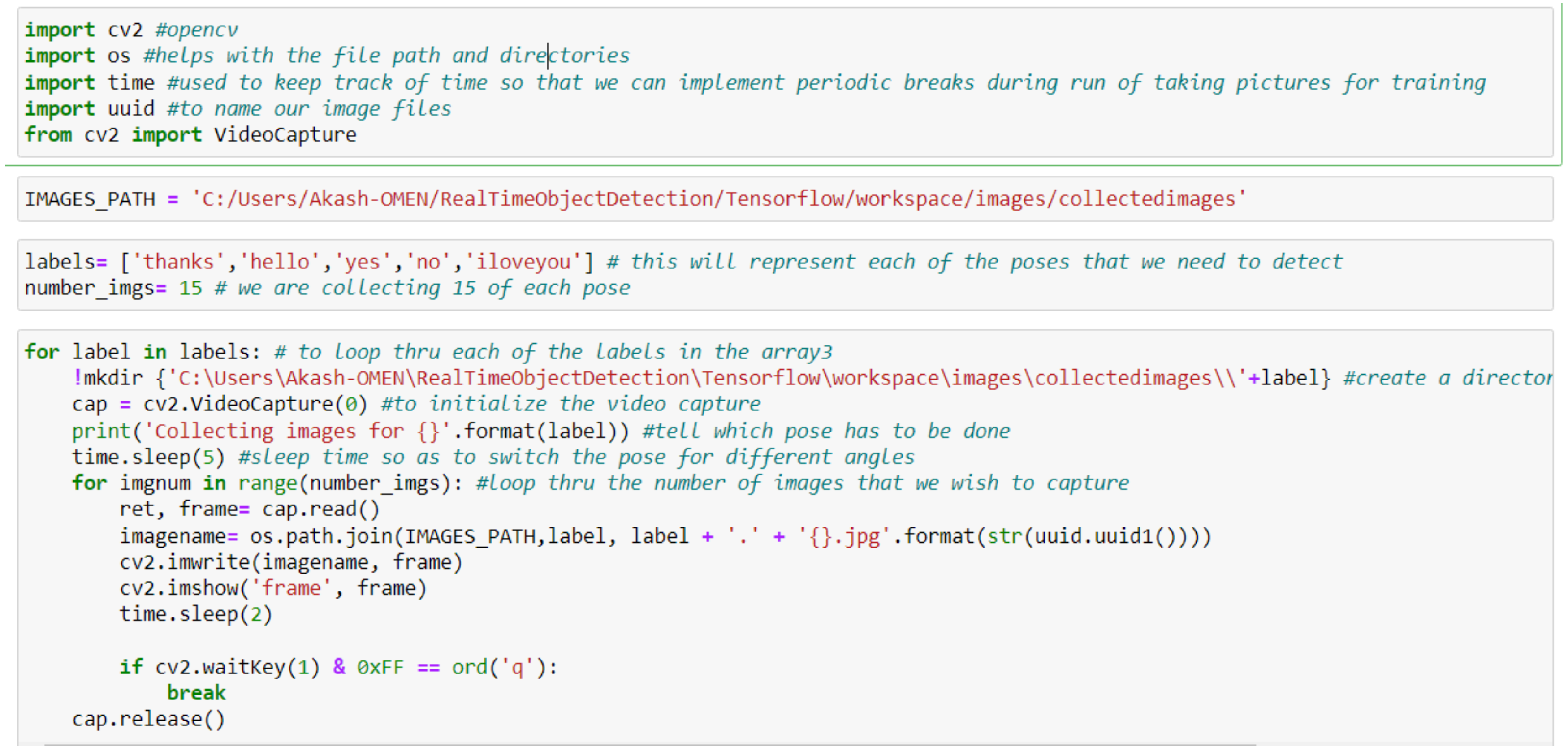
- Mobilenet SSD is an object detection model that computes the output bounding box and object class from the input image. This Single Shot Detector (SSD) object detection model uses Mobilenet as a backbone and can achieve fast object detection optimized for mobile devices.
- Convolutional neural networks are used to develop a model that consists of multiple layers for classifying given objects into one of the defined classes. These objects are detected using higher resolution feature maps made possible by recent advances in deep learning with image processing.

MOBILENET SSD



DATASET & IMAGE TAGGING

- Data Acquisition Data gathering is the first step in the process of training an object detection model. Despite the fact that there are numerous ways to automate the process, in this particular training case, data acquisition was done manually in order to remove noisy images and assure the level of quality of the training.



```
import cv2 #opencv
import os #helps with the file path and directories
import time #used to keep track of time so that we can implement periodic breaks during run of taking pictures for training
import uuid #to name our image files
from cv2 import VideoCapture
```

```
IMAGES_PATH = 'C:/Users/Akash-OMEN/RealTimeObjectDetection/Tensorflow/workspace/images/collectedimages'
```

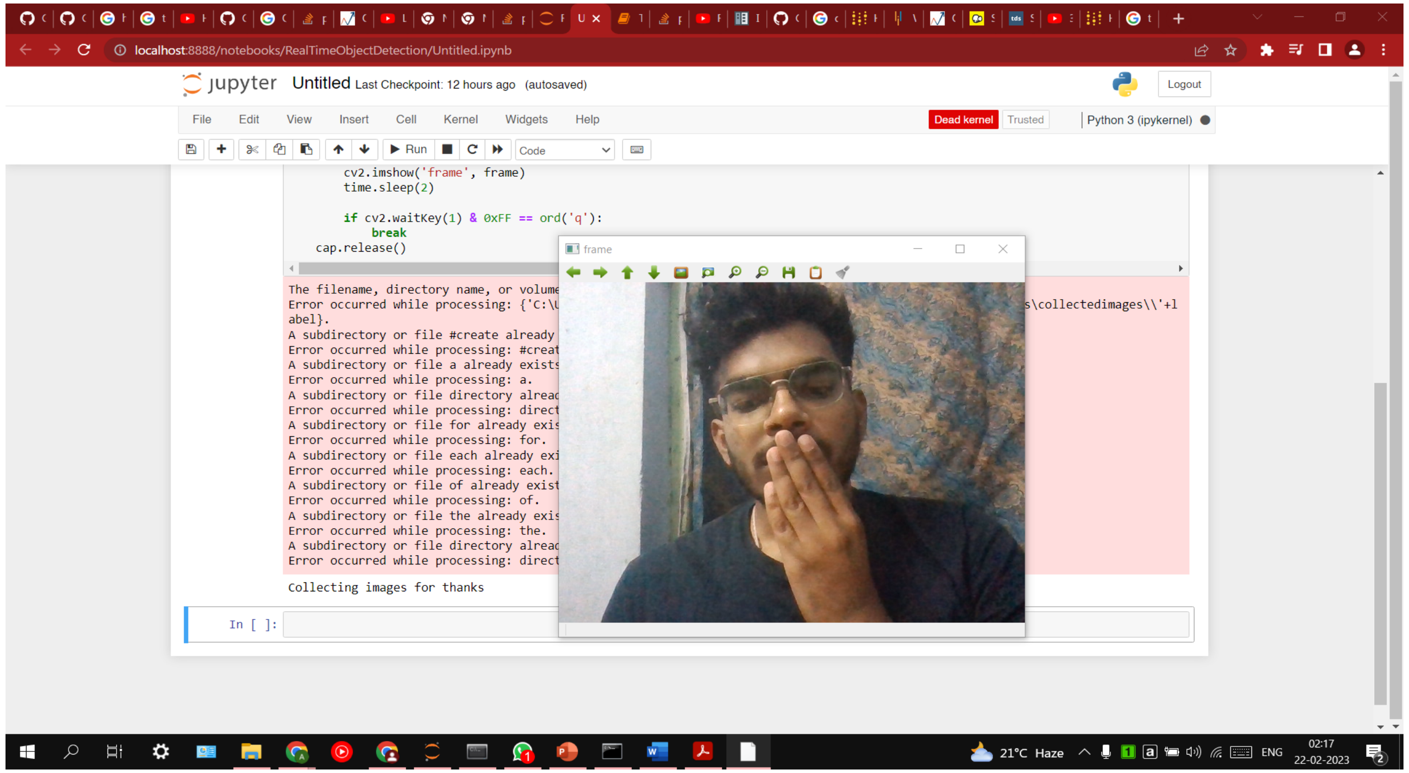
```
labels= ['thanks','hello','yes','no','iloveyou'] # this will represent each of the poses that we need to detect
number_imgs= 15 # we are collecting 15 of each pose
```

```
for label in labels: # to loop thru each of the labels in the array3
    !mkdir {'C:\Users\Akash-OMEN\RealTimeObjectDetection\Tensorflow\workspace\images\collectedimages\\'+label} #create a director
    cap = cv2.VideoCapture(0) #to initialize the video capture
    print('Collecting images for {}'.format(label)) #tell which pose has to be done
    time.sleep(5) #sleep time so as to switch the pose for different angles
    for imgnum in range(number_imgs): #loop thru the number of images that we wish to capture
        ret, frame= cap.read()
        imagename= os.path.join(IMAGES_PATH,label, label + '.' + '{}.jpg'.format(str(uuid.uuid1())))
        cv2.imwrite(imagename, frame)
        cv2.imshow('frame', frame)
        time.sleep(2)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
```

DATASET & IMAGE TAGGING

- Once the data is acquired, the images in the dataset need to be tagged. Image tagging was completed using the **LabelImg** software, which is an open-source python-based implementation of a system that tags the bounding boxes and records them in an XML file.



localhost:8888/notebooks/RealTimeObjectDetection/Untitled.ipynb
jupyter Untitled Last Checkpoint: 12 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Dead kernel
Trusted
Python 3 (ipykernel)
Run
Code
frame
21°C Haze
ENG
02:17
22-02-2023

```
        cv2.imshow('frame', frame)
        time.sleep(2)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
```

```
The filename, directory name, or volume
Error occurred while processing: {'C:\U                         s\collectedimages\\'+l
abel}.
A subdirectory or file #create already
Error occurred while processing: #creat
A subdirectory or file a already exists
Error occurred while processing: a.
A subdirectory or file directory alread
Error occurred while processing: direct
A subdirectory or file for already exis
Error occurred while processing: for.
A subdirectory or file each already exi
Error occurred while processing: each.
A subdirectory or file of already exist
Error occurred while processing: of.
A subdirectory or file the already exis
Error occurred while processing: the.
A subdirectory or file directory alread
Error occurred while processing: direct
Collecting images for thanks
```

```
In [ ]:
```

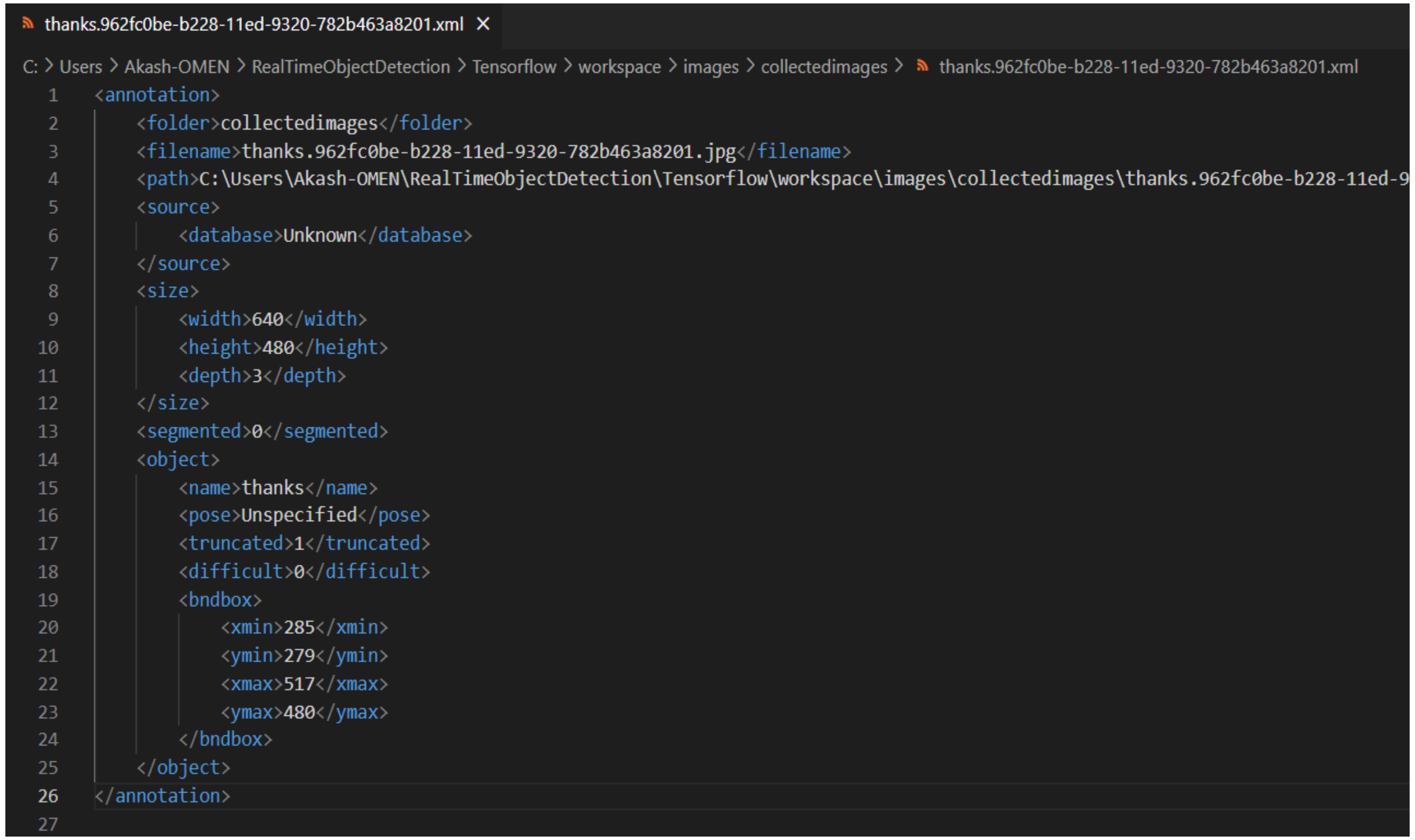
DATASET & IMAGE TAGGING

Capturing image for ‘hello’

Capturing image for ‘no’

ANNOTATION OF THE IMAGES AND CREATION OF XML FILES

- The generated result of the image tagging efforts is an XML file that encompasses the bounding boxes of the tagged images. A sample XML file



thanks.962fc0be-b228-11ed-9320-782b463a8201.xml ×

C: › Users › Akash-OMEN › RealTimeObjectDetection › Tensorflow › workspace › images › collectedimages › thanks.962fc0be-b228-11ed-9320-782b463a8201.xml

```
<annotation>
    <folder>collectedimages</folder>
    <filename>thanks.962fc0be-b228-11ed-9320-782b463a8201.jpg</filename>
    <path>C:\Users\Akash-OMEN\RealTimeObjectDetection\Tensorflow\workspace\images\collectedimages\thanks.962fc0be-b228-11ed-9
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>640</width>
        <height>480</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>thanks</name>
        <pose>Unspecified</pose>
        <truncated>1</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>285</xmin>
            <ymin>279</ymin>
            <xmax>517</xmax>
            <ymax>480</ymax>
        </bndbox>
    </object>
</annotation>

```

NEXT STEPS

- Download TF Models Pretrained Models from Tensorflow Model Zoo.
- **SSD mobilenet v2 fpnlite 320x320**
- **Copy Model Config to Training Folder**
- **Train the model**
- **Detect in Real-Time**

SSD Mobilenet V2 is a one-stage object detection model which has gained popularity for its lean network and novel depth wise separable convolutions. It is a model commonly deployed on low compute devices such as mobile (hence the name Mobilenet) with high accuracy performance.

The SSD object detection composes of 2 parts:

- Extract feature maps, and
- Apply convolution filters to detect objects.

Each prediction composes of a boundary box and 21 scores for each class (one extra class for no object), and we pick the highest score as the class for the bounded object. Conv4_3 makes a total of 38 × 38 × 4 predictions: four predictions per cell regardless of the depth of the feature maps. As expected, many predictions contain no object. SSD reserves a class "0" to indicate it has no objects.
Making multiple predictions containing boundary boxes and confidence scores is called multibox

Here are some key observations:

- SSD performs worse than Faster R-CNN for small-scale objects. In SSD, small objects can only be detected in higher resolution layers (leftmost layers). But those layers contain low-level features, like edges or color patches, that are less informative for classification.
- Accuracy increases with the number of default boundary boxes at the cost of speed.
- Multi-scale feature maps improve the detection of objects at a different scale.
- Design better default boundary boxes will help accuracy.
- SSD has lower localization error comparing with R-CNN but more classification error dealing with similar categories. The higher classification errors are likely because we use the same boundary box to make multiple class predictions.
- SSD512 has better accuracy (2.5%) than SSD300 but run at 22 FPS instead of 59.

RESULTS

- The performance of Hand Sign Language Translation using Computer Vision was evaluated using a dataset of hand sign images and video streams. The dataset consisted of five different hand signs of common phrases.

FUTURE WORK

- Future work could involve further improving the recognition accuracy of the proposed method, particularly for more complex hand signs. This could involve exploring more advanced computer vision techniques, such as deep learning approaches or 3D tracking of hand movements.
- The system will be able to accurately recognize and translate the hand signs in real-time, with a very low latency.

CONCLUSION

- In conclusion, hand sign recognition using computer vision is a complex and challenging task, but one with great potential for impact. With continuous testing and evaluation, attention to important factors affecting the system's performance, and consideration of potential applications, it is possible to create a robust and useful hand sign recognition system.