Multimedia Retrieval

Web Application for Real-Time Sign Language Detection

Realized by
Amina Ben Salem
Supervised by
Prof. Ph.D Thomas Skopal

August, 2021

Contents

List of Figures

Introduction

According to the World Health Organization (WHO), over 5% of the world's population suffers from hearing loss. This number is estimated to grow even much higher, that by 2050 hearing disabilities will touch the lives of 1 in 4 people around the globe [1]. Although sign languages were created to help the deaf communicate better, these unfortunately are not universal languages. In the United States, for example, there are only 250,000 to 500,000 people capable of signing [2], which is a limiting factor for the deaf in today's highly connected society. Hence, a real-time sign language translator can bridge the expanding communication gap and facilitate the lives of the millions of deaf as well as their families and friends. Various studies have applied deep learning to sign language detect, however, few are those that address the issue of real-time detection and work with static images. Additionally, numerous efforts make use of 3D motion capture technology with special accessories such as gloves or Microsoft Kinect sensor.

In this project, we take advantage of the advances in the field of computer vision to build a sign language detector capable of detecting and translating 5 signs from the American Sign language (ASL). We apply transfer learning in our detection problem and use the pre-trained SSD MobileNet V2 FPNLite 320x320 architecture. We summarize the steps we undertook as follows:

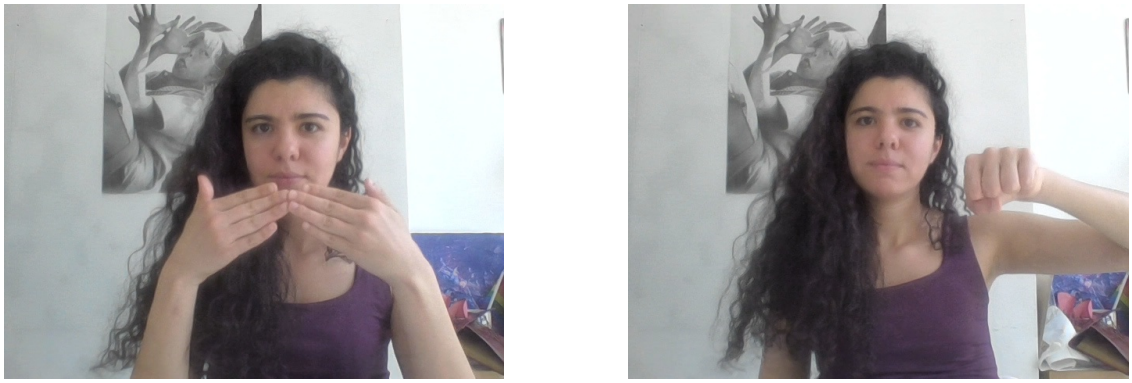
- We collect and label images for our task.
- We apply transfer learning and train our model.
- We then convert our model into tfjs format to use for our app.
- We host the generated model for deployment.
- Build a REACT.JS application for detection.

This work was conducted under the supervision of Prof. Ph. D. Tomas Skopal as the passing criteria of the course Multimedia Retrieval at the University of Passau.

1 Methodology

1.1 Dataset

For this project, we decided to generate the dataset from scratch. We wrote a python script allowing us to capture images using our own laptop's webcam (see Figure 1). Considering our technical resources, we opted for a dataset comprising of five basic ASL signs. For each of these classes, we collect 15 images which we then annotate using labelImg [3]. In total, we obtain 75 images which we label manually.



(a) Thank you in ASL (b) Yes in ASL

Figure 1: Samples from captured images for our dataset

1.2 Our Model

1.2.1 Transfer learning

Transfer learning is an optimization technique in machine learning where we utilize a model, that was pre-trained on a large data for a specific task, as the starting point for another task. Computer vision problems are usually extensive in terms of time and compute resources, however, using this approach enhances the performance and the training speed of the model. A typical example would be taking a neural network that was trained to recognize objects such as cats, re-initializing the weights of its last output layer to classify a completely different dataset (see Figure 2).

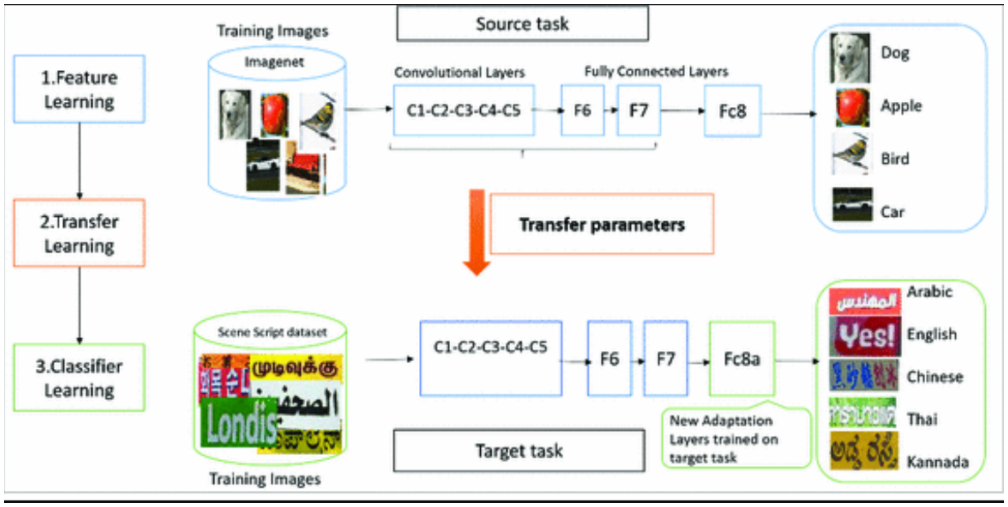


Figure 2: Transfer Learning Illustration

1.2.2 SSD Mobilenet V2

An alternative model for object detection is YOLO (You Only Look Once). It returns predictions for class probabilities of input images and as its name suggests, it conducts a

single forward propagation through Convolutional Neural Networks (CNN) to detect in real-time relatively fast. This, however, comes at the cost of performance, where YOLO usually yields worse the worst results. Considering this Accuracy/Speed trade-off, and our dataset's size, we chose to go with SSD MobileNet. This model is based on the RetinaNet detector in [4] and was trained on the COCO dataset [5]. It inherites from MobileNet V1, which replaces the computationally expensive convolutional layers with depthwise separable convolutions. Its layers are presented below in Figure 3.

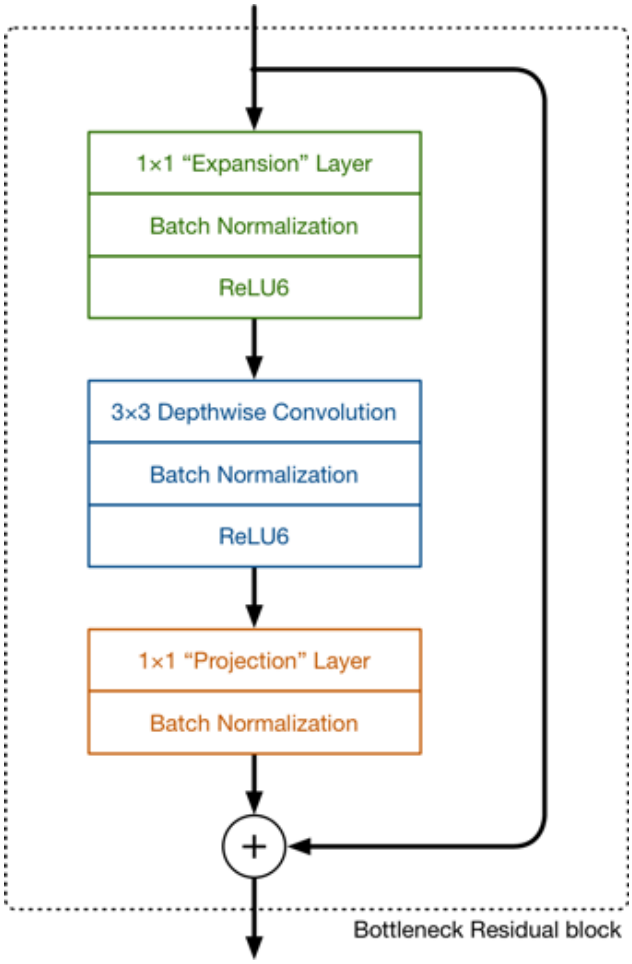


Figure 3: MobileNet V2

The Expansion layer is a 1x1 convolutions which expands the number of channels of data before being passed to the Depthwise convolution. The Projection layer ensures that the highly dimensional input is projected into a tensor with lower dimensions. The full architecture, is made out of sequence of 17 of these blocks in Figure 3

1.3 Training

Training was conducted locally using a CPU AMD Ryzen 7 4700U card with 16GB of RAM. As mentioned previously we applied transfer learning on MobileNet V2. Before we could start training our model, we first needed to satisfy the following steps:

1. Generate the label map file which contains our classes each mapped to a specific id.
2. Having split our images into train and test sets after the annotation phase, we now proceed to creating the TFRecords from each folder. This is a file format required by Tensorflow.
3. Install the TensorFlow Object Detection API and customize the configuration of our model. We use L2 Regulirization to add a penalty on the norm of the weights to the loss for the layers. The added cost on large weights helps battle against overfitting. We also use a fixed shape resizer of height and width equal to 320 to ensure that the input satisfies model's requirements.
4. Build the model using the predefined configuration
5. Restore the trained model to a checkpoint which we can then use to define a function for image pre-processing and detection generation.

2 Web Application

2.1 Hosting

The first step we needed to perform was to export the checkpoint from the previous step 1.3 which we then converted to a TFJS graph file. This file represents the operations of the model and allows it to execute on the client side of the browser. We chose to host our model on IBM Cloud which ensures that our application can access the model globally. However, in order for our app to work properly, we needed to ensure that Cross-Origin Resource Sharing (CORS) is enabled as this policy is disabled by default in browsers for security reasons. To achieve this, we created a JSON file defining the new CORS Rules to be applied (see Figure 5) and linked it to our bucket (see Figure 4).

Figure 4: Redefined CORS Rules

2.2 Deployment and Results

We used REACT.JS, a powerful JavaScript Libraby for building UIs and components. We utilized the shell template from Facebook on Github to create our own standalone application [6] accessible via "http://localhost:3000/" (see Figure 6)

```
PS C:\Windows\models\research\MR> ibmcloud cos bucket-cors-put --bucket mrmodel --cors-configuration file://cors.json
OK
Die CORS-Konfiguration f³r Bucket 'mrmodel' wurde erfolgreich festgelegt
PS C:\Windows\models\research\MR>
```

Figure 5: Enabling CORS for our bucket

Displaying the results of the model on the application was a particularly challenging task. Firstly, it seems the Tensorflowjs librabry suffers from a minor issue causing the model loaded via the URL in the app to return the detection objects in a different order than that initially returned before the conversion. This meant I had to restart the app multiple times while logging the returned arrays till I could map them to the correct constants.

Additionally, it seems that the boxes are not accurately displayed although the detections are correct, I have tried multiple approaches and came to the conclusion that is´t is probably related to the values returned within the object boxes.

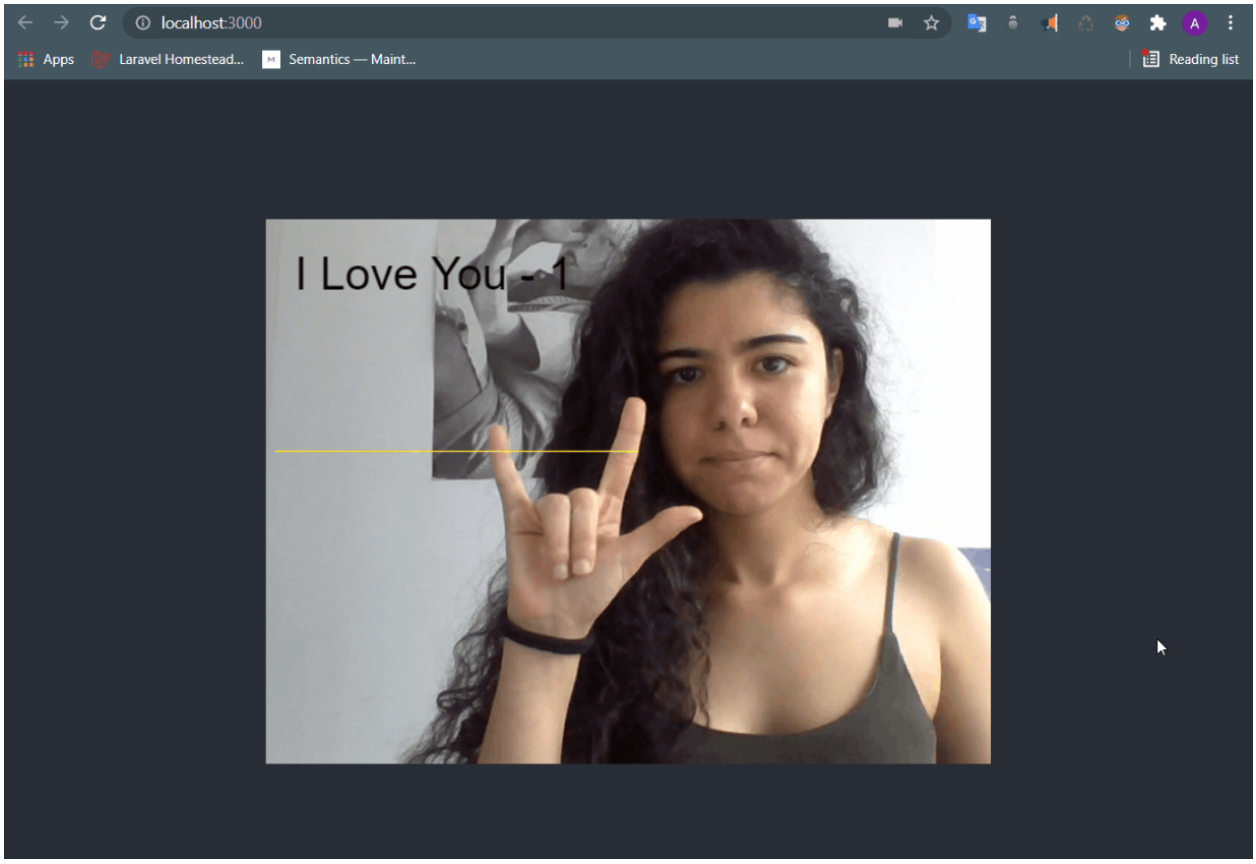


Figure 6: Application Interface and Results

3 Limitations

In addition to the typical challenges encountered in ML, working with computer vision in general, presents further aspects for consideration. Following our personal experience with this project, we cite the following:

- Surrounding factors can play a role in the result of detection process, we mention mainly: lighting, background, and hands position.
- Occlusion: this occurs when the hands are not completely in the scope of the visual field of the camera. Another possibility would be a hidden part due to the direction of the hand.
- Sign transition: the motion of hands might yield a sub-optimal performance which which doesn't take in consideration the states in between different signs.
- Adding to the previous point, the delay between the hand motion and its detection, and the display.

4 Conclusion and Future Work

In this project we created a web application with a ASL detection model capable of conducting translation in real time. We leveraged the technique of Transfer Learning and used the SSD MobileNet V2 pretrained model to optimize our training process which yielded satisfactory results. Nevertheless, many are the possibilities to extend this work. For instance, we might consider signs that are not static but comprised of multiple steps instead. However, this requires more resources.

References

[1] World health Organization. Deafness and hearing loss. In https://www.who.int, 2021.

[2] Mitchell, Ross & Young, Travas & Bachleda, Bellamie & Karchmer, Michael. (2006). How Many People Use ASL in the United States? Why Estimates Need Updating. Sign Language Studies. 6. 10.1353/sls.2006.0019.

[3] https://github.com/tzutalin/labelImg

[4] T. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, "Focal Loss for Dense Object Detection," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 2, pp. 318-327, 1 Feb. 2020, doi: 10.1109/TPAMI.2018.2858826.

[5] Common Objects in Context, https://cocodataset.org/, 2017

[6] https://github.com/facebook/create-react-app