Open in app

Yashpreet Kaur

Lists About

Real-Time Sign Language Detection System using Deep Learning

 Yashpreet Kaur Dec 11, 2021 · 17 min read

Authors:

Yashpreet Kaur, Emilio Cabrera, Connor Gilmore, JT Flume, Junsu Kim

Abstract

In this blog post, we lay out a real-time language detection system that leverages real-time object detection methods in computer vision to classify American sign language signs (letters) into English text. This work is important to help close the gap between businesses that can't communicate with deaf customers, by helping to provide the technology to do so.

In our work, as explained below, we leverage the YOLOv4-tiny model to detect sign language inputs in image, video, and real-time formats. This work leverages transfer learning and has computational advantages in addition to accuracy and learning capabilities. Our results show that our model successfully works on images, videos, and real-time input forms. We then conclude by looking at current real-time sign language detection systems and the future of this work.

Introduction

Open in app

made, services of the like are not easily available for deaf people who sign.

This is an important problem because, according to the Communication Service for the Deaf (CSD) over 1 million people communicate using American Sign Language (ASL), and 93% of them prefer to communicate using ASL, including in business settings. They are economically very important, representing an $86 Billion dollar market (according to the CSD). However, businesses struggle to serve deaf people because they often don't have the people (who can sign) or the technological signs to communicate. Our research showed that there are less than 10 (not-widely-used) sign language detection systems available on the market. Another very important use case is for deaf children. The CSD states that 98% of deaf people do not receive an education in sign language, and 72% of families do not sign with their deaf children.

By building a real-time sign language detection system, we can help to bridge this gap and help businesses communicate with their deaf customers. Specifically, it is our goal to build this system that can take images, videos, and real-time sign language inputs and return an English alphabet letter detection (i.e translation).

Data. To solve this problem and to build our model, our team used 1,512 jpg images of ASL signs, as shown in the image below. Each image is a letter in ASL, complete with other objects in the background, and the letters appear in different orientations to help better train the model.

Open in app

In the data, we have 26 classes (alphabets) that we used to train our model. The images were pre-processed and converted into the format required for our model. We used YOLOv4 txt file format of these images in order to train our data. We used 1,512 pre-processed images for training our model, 144 files as the validation set and 72 files as the test set.

Object Detection Overview

Classification vs Detection. Prior to jumping into the technical material, we should first clarify the main difference between object classification vs detection. Object classification determines which category or class each object belongs to, where object localization determines where objects are located in a given image. Object Detection is essentially a combination of the two.

Two vs Single-stage Approach. There are currently two major object detection approaches; a two-stage object detector and a single-stage detector both of which consist of various object detection algorithms. Following are brief highlights comparing each approach:

Two-Stage

- **Accuracy:** Higher Accuracy
- **Network Architecture:**

1. Similar Backbone
2. Regional Proposal Network
3. RoI Pooling Layer

- **Computational Speed:** Slower Speed
- **Examples:** R-CNN & Fast(er) R-CNN

Single-Stage

- **Accuracy:** Slightly Less Accurate

Open in app

1. Similar Backbone
2. RoI Pooling Layer

- **Computational Speed:** Faster Speed
- **Examples:** You Only Look Once (YOLO) & Single-Shot Detection (SSD)

For the usage of real-time detection, single-stage detectors are the better approach.

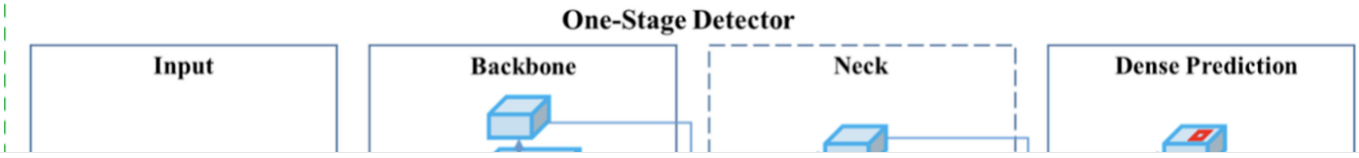
Object Detection Method: YOLO. For the purpose of this project, we selected YOLO as our object detection method due to its speed and performance. In 2016 Joseph Redmon published research comparing the performance and speed of fast detectors trained on Pascal Visual Object Classes training dataset. YOLO emerged as the clear choice as it operated with 10 mAP more accurately than its fast version while near real-time in speed. For reference, the real-time speed is ~60 frames per second (FPS) [Source 1]. Since then YOLO's algorithm has made several enhancements with progress in executing at very fast speeds with high-quality precision. According to research conducted by Alexey Bochkovskiy in 2020, YOLO version 4 improves YOLOv3's AP and FPS by 10% and 12%, respectively, outperforming other detectors which were all trained on Microsoft's COCO dataset [Source 2]. Overall, YOLO continues to make strides in speed and performance and remains one of the best real-time detectors.

YOLOv4-tiny

The basic structure/architecture of YOLOv4-tiny follows that of YOLOv4, albeit the former is designed to be a more concise version of the latter in order to make the processes faster and lightly used even in mobile configurations. In this section, we describe the YOLOv4 architecture and its comparison to YOLOv4-tiny.

Architecture.

(Backbone — Neck — Head)



Open in app

One-stage object detector (Source)

YOLO has two main parts, backbone and head with a neck between them. A network for detecting objects is trained at Backbone; features formed in the backbone are mixed and combined at Neck, and location and classification of objects are implemented at Head.

As YOLO was designed to work well in detecting real-time objects on a light GPU platform such as mobile devices, it has the option for the backbone of VGG, ResNet, CSPDarkent53, and so on. Among them, YOLO and its subsequent families adopt the Darknet as its framework to train a model. The reason behind this choice is that Darknet is written in C which operates much faster compared to python.

For the head part where the actual detection steps are conducted, we can have two broad options: two-stage detector and one-stage detector. The two-stage detectors can be represented by the R-CNN families including fast R-CNN, faster R-CNN, and R-FCN. Relatively recently developed one-stage detectors such as YOLO and SSD families are one-stage which aim faster detection rate, while not significantly reducing performance. The stages for models like R-CNN include: 1) Region Proposal Network to locate the region of interest and 2) object classification and bounding-box regression. The presence of the predetermined number of frames called grid cells makes the one-stage detection possible. Each grid cell contains information such as the likelihood of the presence of the object, class, and coordinate of the object. This allows one to build a faster object-detecting model.

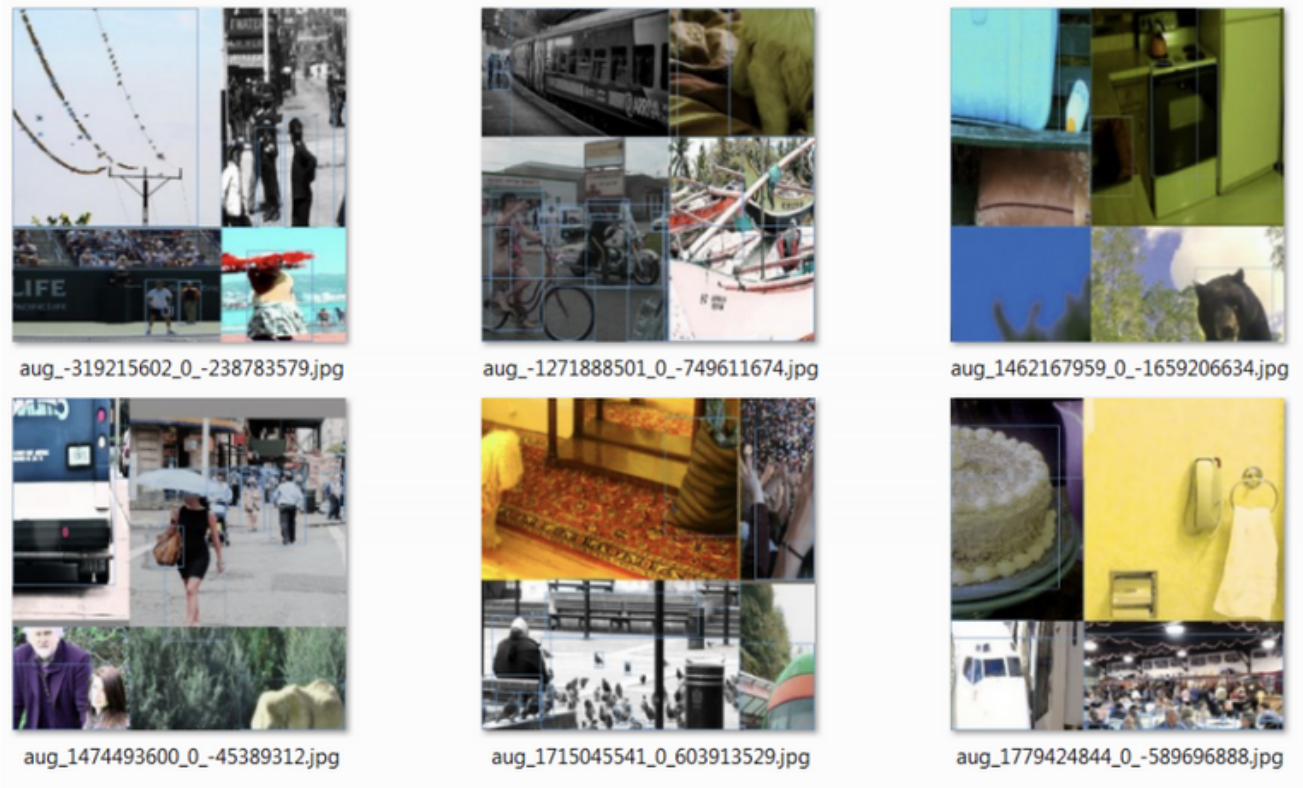
Bag of Freebies (BOF) and Bag of Specials (BOS). A series of methods that improve the performance of a model without adding to inferential time can be called Bag of Freebies (BoF). The methods that significantly increase performance with marginal costs in inferential time are called Bag of Specials (BoS). Both BoF and BoS can be applied to any part of the model architecture. Here we just mention some components of each method.

Open in app

from existing data:

1. Self-Adversarial Training is one of the data augmentation techniques. It represents a new data augmentation technique that operates in 2 forward — backward stages. In the 1st stage, the neural network alters the original image instead of the network weights. In this way the neural network executes an adversarial attack on itself, altering the original image to create the deception that there is no desired object on the image. In the 2nd stage, the neural network is trained to detect an object on this modified image in the normal way.

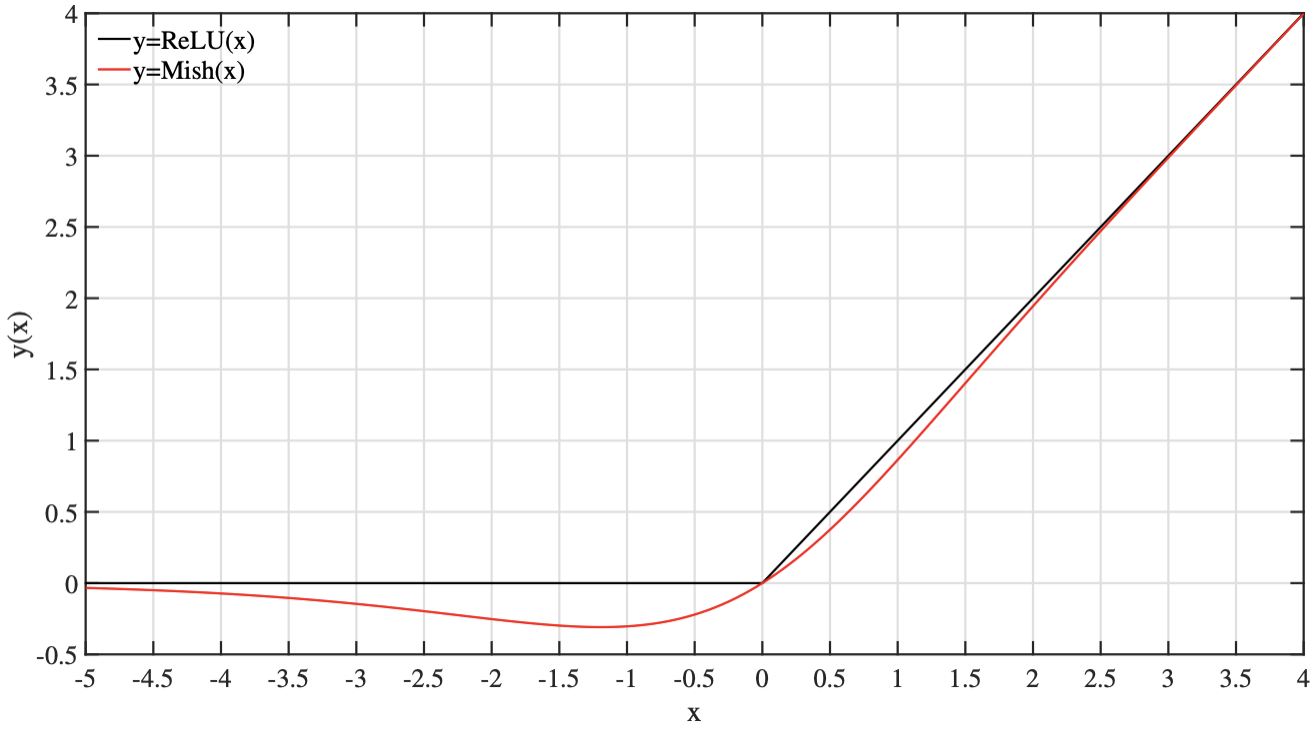
2. Mosaic data augmentation combines 4 training images into one in certain ratios. Mosaic [video] is the first new data augmentation technique introduced in YOLOv4. This allows for the model to learn how to identify objects at a smaller scale than normal.



Mosaic data augmentation (Source)

Open in app

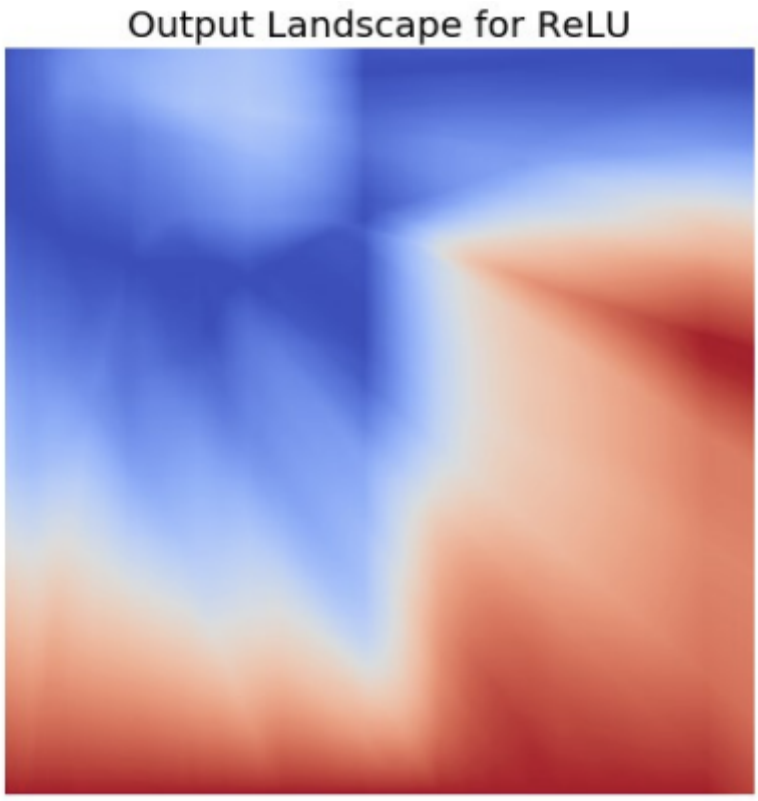
1. Mish activation function: A lot of networks choose ReLU(Rectified Linear Unit) as its activation function which adds more non-linearity and, thus, 'reality' to the model. We can observe from the graph below that ReLu ramps at 0 while Mish shows more smooth shape throughout the span. This property is closely related to the result from adopting each activation function. We can observe more radical transitions in the landscape from the model adopting ReLU, compared to that adopting Mish.



ReLU vs Mish activation function (Source)

Open in app

Mish activation function output (Source)



ReLU activation function output (Source)

2. Cross mini-Batch Normalization is a modified version of Cross Batch Normalization which collects statistics only between mini-batches within a single batch.

3. DIoU-NMS is a type of non-maximum suppression where Distance IoU is used rather than regular DIoU, in which the overlap area and the distance between two central points of bounding boxes are simultaneously considered when suppressing redundant boxes

YOLOv4 vs YOLOv4-tiny. As stated earlier, YOLOv4-tiny is a compressed form of YOLOv4. YOLOv4-tiny is trained on 29 pre-trained convolutional layers compared to

Open in app

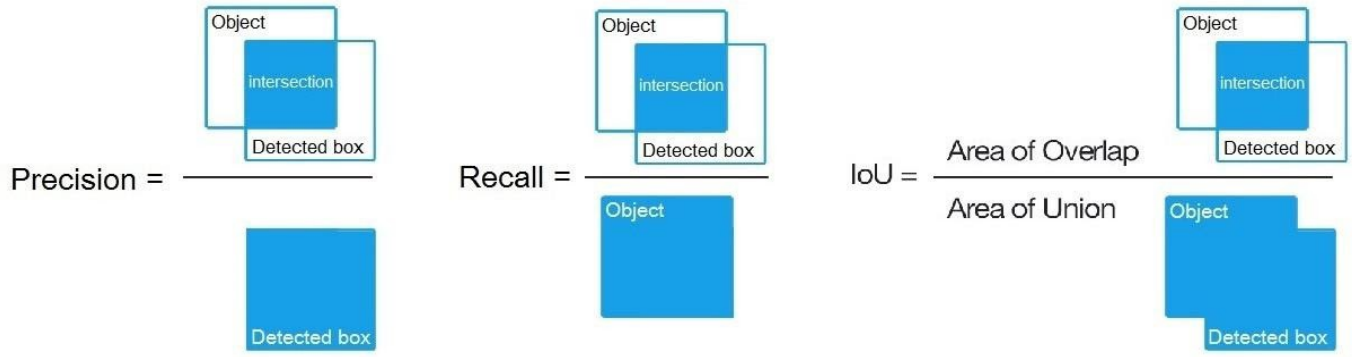
times that of YOLOv4. Since this project is focused on real-time object detection, we chose a faster inference time model over accuracy i.e. YOLOv4-tiny.

Scoring Metrics

IoU (Intersection over Union). Before we discuss how the CIoU contributes to the prediction process of models, going through what its precursors are will help us to get a better understanding of CIoU. IoU measures the extent of how well our model locates the bounding boxes for objects in question compared to the box for ground truth. 0.5 of threshold is often adopted. Assuming that there are two boxes having the same area of 4, this threshold can be met when the overlapped area is greater than ⅔ of the area of a box.

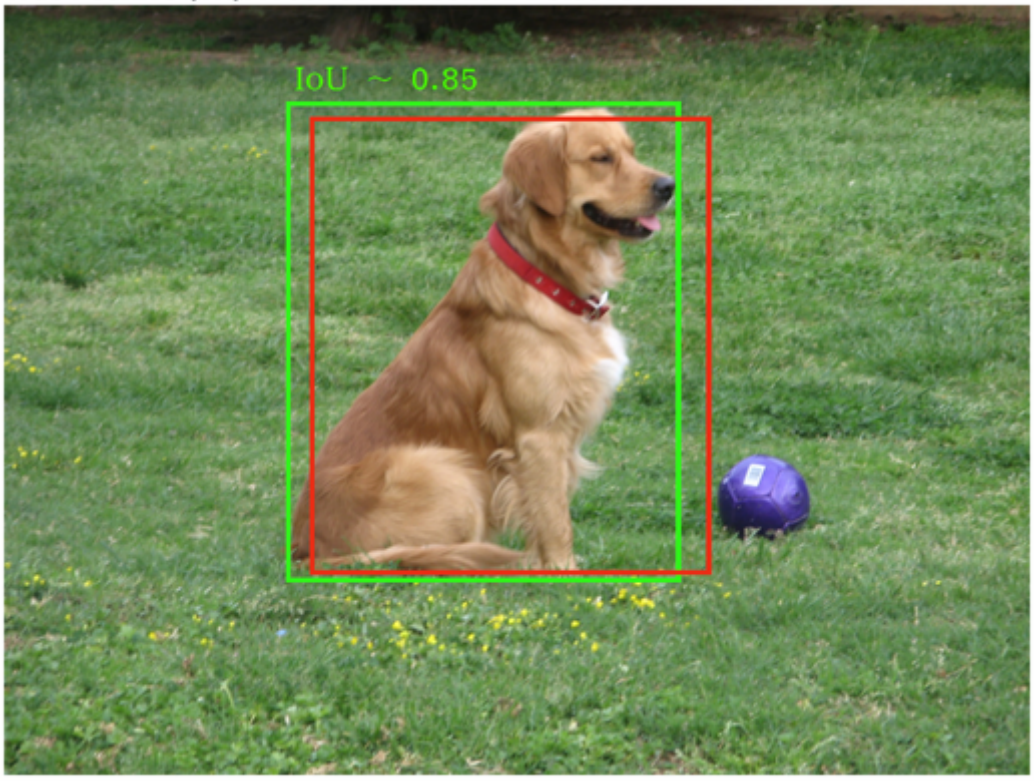
CIoU(Complete-Intersection over union). CIoU is used as loss metrics. CIOU is an improved version of IoU and Generalized IoU (GIoU), allowing bounding boxes to converge faster than IoU and GIoU and improving average precision and average recall. This is made possible with the blend of three geometric factors of IoU, the normalized distance between midpoints of bounding boxes, and consistency of aspect ratio. This measurement is adopted by Yolo-v3, Yolo-v4, SSD, and Faster RCNN.

Precision, Recall & mAP. While precision indicates how well an algorithm detects the true object among the whole detections which were made by the algorithm as positive, Recall illustrates how many truths have been detected among the overall truth. We can know the general performance of a model in question by looking at the PR-recall curve. If we want to quantify how this model performs compared to the other model, however, we need a different metric — Average Precision.

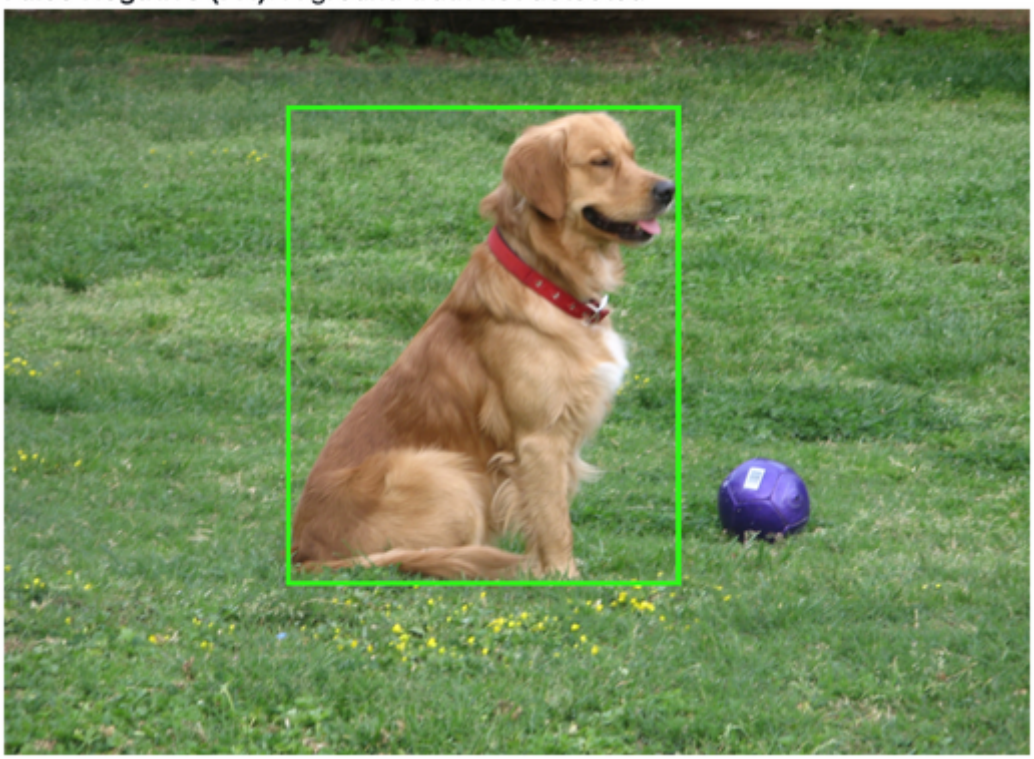


Open in app

True Positive (TP): A correct detection. Detection with IOU ≥ *threshold*

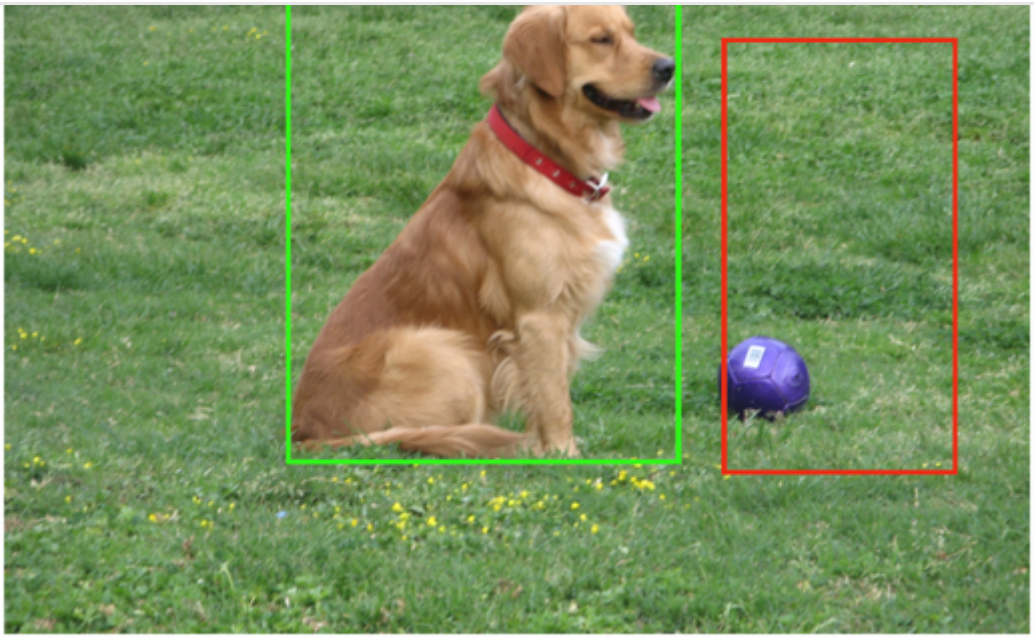


False Negative (FN): A ground truth not detected



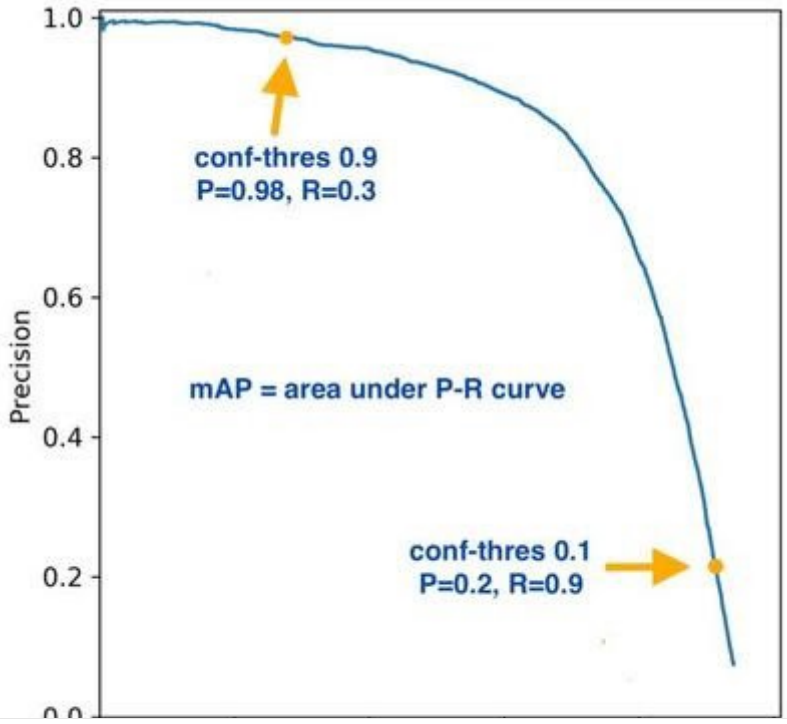
False Positive (FP): A wrong detection. Detection with IOU < *threshold*

Open in app



(Source)

Average Precision. Average Precision (AP) can be expressed as the area under the PR-Recall curve which allows us to compare the model performance. Before calculating the area under the curve, a step in which we convert the curve into a curve having monotonous decreasing shape. As we used to have a model classifying multiple classes, we need to average all the AP of each class. This is called mAP, mean-average-precision.



Open in app

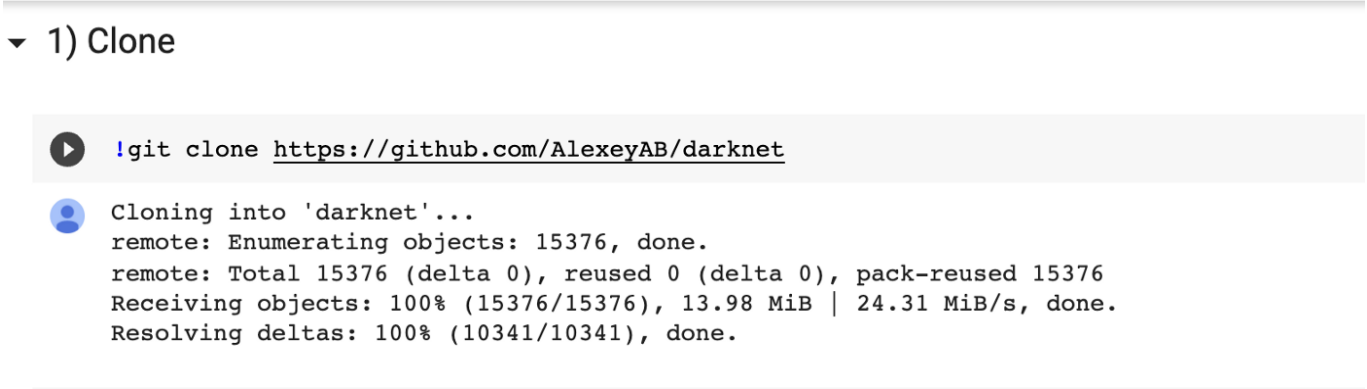
PR recall curve (Source)

Model training

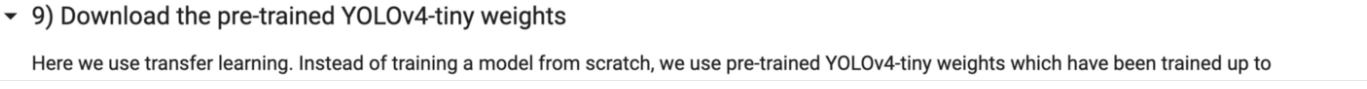
Transfer Learning. As mentioned earlier, considerable parts of the model performance — the accuracy of detecting objects and the cost it brings — are affected by how well a network is trained in the backbone. Building a model from scratch and tuning it to our purpose is deserving. However, we need to think if we need to train our model every time we are asked to conduct similar tasks. It would result in inefficient iterations without achieving impressive changes in model performance. The rise of transfer learning permits more efficient model training, leveraging pre-trained weights to some extent and then adjusting the pre-defined figures to what fits our object-detecting process. Consenting with this notion, we utilized the pre-trained weights from CSPDarknet53 and adjusted it to design a model that is able to detect American Sign Language.

Link to Google Colab notebook

The colab notebook above contains steps to build, train and test the model. We have attached a few snippets below of the important sections of the model. As seen below, we start by cloning the darknet repository. Then we download pre-trained yolov4-tiny weights.



Cloning the darknet repository



Open in app

```
--2021-12-08 23:40:50--  https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-tiny.conv.29
Resolving github.com (github.com)... 192.30.255.112
Connecting to github.com (github.com)|192.30.255.112|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/75388965/28807d00-3ea4-11eb-97b5-4c84
--2021-12-08 23:40:50--  https://objects.githubusercontent.com/github-production-release-asset-2e65be/75388965/28807d00-3ea4
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 19789716 (19M) [application/octet-stream]
Saving to: 'yolov4-tiny.conv.29'

yolov4-tiny.conv.29 100%[===================>]  18.87M  25.1MB/s    in 0.8s

2021-12-08 23:40:51 (25.1 MB/s) - 'yolov4-tiny.conv.29' saved [19789716/19789716]
```

Code snippet where the pre-trained YOLOv4-tiny weights are downloaded

Configurations. After this, the next main step is making changes to the configuration file as per our requirements. During the training process, 64 images went to the network and then 4 of them flowed through the GPU. As our purpose is to detect 26 alphabets, we have 26 classes to detect.

yolov4-tiny-custom.cfg

```
1 [net]
2 # Testing
3 #batch=1
4 #subdivisions=1
5 # Training
6 batch=64
7 subdivisions=16
8 width=416
```

Open in app

```
10 channels=3
```

```
208 [convolutional]
209 size=1
210 stride=1
211 pad=1
212 filters=93
213 activation=linear
214
215
216
217 [yolo]
218 mask = 3,4,5
219 anchors = 10,14,  23,2
220 classes=26
```

Code snippets for yolov4-tiny model settings

Open in app

10) Training

Train your custom detector

For best results, you should stop the training when the average loss is less than 0.05 if possible or at least constantly below 0.3, else train the model until the average loss does not show any significant change for a while.

```
!./darknet detector train data/obj.data cfg/yolov4-tiny-custom.cfg yolov4-tiny.conv.29 -dont_show -map
```

To restart your training (In case the training does not finish and you get disconnected)

If you get disconnected or lose your session, you don't have to start training your model from scratch again. You can restart training from where you left off. Use the weights that were saved last. The weights are saved every 100 iterations as yolov4-tiny-custom_last.weights in the yolov4-tiny/training folder on your drive. (The path we gave as backup in "obj.data" file).

Run the following command to restart training.

```
[ ] !./darknet detector train data/obj.data cfg/yolov4-tiny-custom.cfg /mydrive/yolov4-tiny/training/yolov4-tiny-custom_last.weights -dont_show -map
```

Code snippet showing the model training command

Training. During the training, we can observe that the model improves as the number of steps increases initially and then the loss begins to remain unchanged even if the number of steps is increased. In the training snippets below, we can observe the model training state at iteration 22 and 2800. We observe that at iteration 2800, the precision and recall are 0.97 each and the average IOU is 76.26%. In general, we stop training the model if the error is at 0.05 or continuously under 0.3.

```
+ 코드  + 텍스트                                                    RAM  디스크   수정 가능
!./darknet detector train data/obj.data cfg/yolov4-tiny-custom.cfg yolov4-tiny.conv.29 -dont_show -map
 (next mAP calculation at 1000 iterations)
 22: 358.882721, 360.083252 avg loss, 0.000000 rate, 0.528081 seconds, 1408 images, 4.571872 hours left
Loaded: 0.000060 seconds
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 30 Avg (IOU: 0.431788), count: 4, class_loss = 111.244377, iou_loss = 0.043274,
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 37 Avg (IOU: 0.000000), count: 1, class_loss = 607.742676, iou_loss = 0.000000,
 total_bbox = 1393, rewritten_bbox = 0.000000 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 30 Avg (IOU: 0.378957), count: 4, class_loss = 112.769005, iou_loss = 0.045959,
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 37 Avg (IOU: 0.000000), count: 1, class_loss = 604.979370, iou_loss = 0.000000,
 total_bbox = 1397, rewritten_bbox = 0.000000 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 30 Avg (IOU: 0.252846), count: 4, class_loss = 110.010147, iou_loss = 0.014381,
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 37 Avg (IOU: 0.000000), count: 1, class_loss = 605.386414, iou_loss = 0.000000,
 total_bbox = 1401, rewritten_bbox = 0.000000 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 30 Avg (IOU: 0.377667), count: 4, class_loss = 112.787743, iou_loss = 0.034821,
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 37 Avg (IOU: 0.000000), count: 1, class_loss = 605.502869, iou_loss = 0.000000,
 total_bbox = 1404, rewritten_bbox = 0.000000 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 30 Avg (IOU: 0.255493), count: 4, class_loss = 114.515427, iou_loss = 0.037231,
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 37 Avg (IOU: 0.000000), count: 1, class_loss = 605.999939, iou_loss = 0.000000,
 total_bbox = 1408, rewritten_bbox = 0.000000 %
```

```
2800: 0.101343, 0.147063 avg loss, 0.002610 rate, 0.524939 seconds, 179200 images, 2.552545 hours left

 calculation mAP (mean average precision)...
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
 detections_count = 348, unique_truth_count = 151
class_id = 0, name = A, ap = 100.00%      (TP = 9, FP = 1)
class_id = 1, name = B, ap = 100.00%      (TP = 5, FP = 0)
class_id = 2, name = C, ap = 100.00%      (TP = 2, FP = 0)
class_id = 3, name = D, ap = 100.00%      (TP = 8, FP = 0)
class_id = 4, name = E, ap = 100.00%      (TP = 6, FP = 0)
```

Open in app

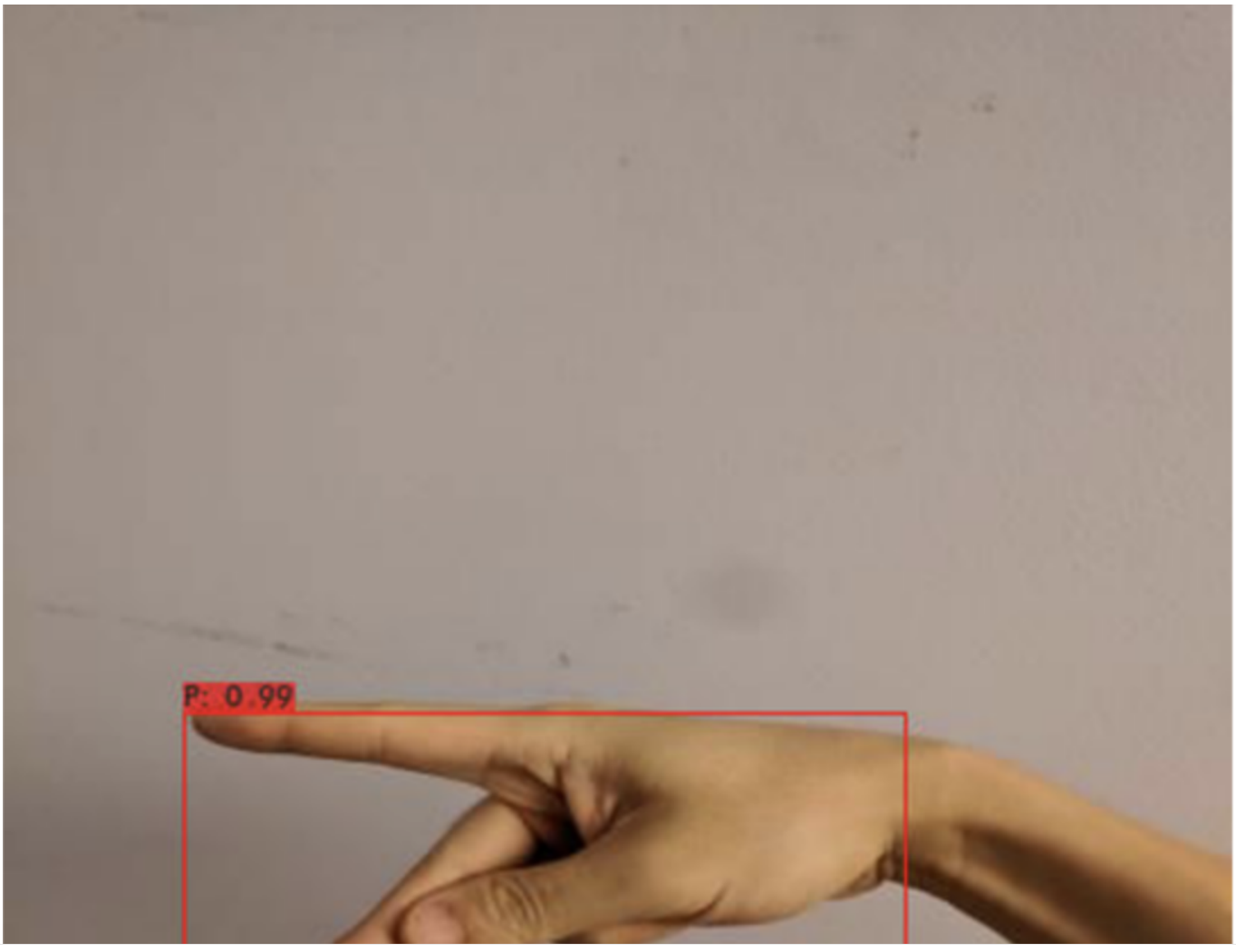
```
class_id = 10, name = K, ap = 100.00%           (TP = 6, FP = 0)
class_id = 11, name = L, ap = 100.00%           (TP = 4, FP = 0)
class_id = 12, name = M, ap = 100.00%           (TP = 8, FP = 1)
class_id = 13, name = N, ap = 100.00%           (TP = 2, FP = 0)
class_id = 14, name = O, ap = 100.00%           (TP = 1, FP = 0)
class_id = 15, name = P, ap = 100.00%           (TP = 6, FP = 0)
class_id = 16, name = Q, ap = 93.06%      (TP = 6, FP = 1)
class_id = 17, name = R, ap = 100.00%           (TP = 6, FP = 0)
class_id = 18, name = S, ap = 100.00%           (TP = 8, FP = 0)
class_id = 19, name = T, ap = 100.00%           (TP = 4, FP = 0)
class_id = 20, name = U, ap = 100.00%           (TP = 4, FP = 0)
class_id = 21, name = V, ap = 100.00%           (TP = 4, FP = 0)
class_id = 22, name = W, ap = 100.00%           (TP = 5, FP = 0)
class_id = 23, name = X, ap = 100.00%           (TP = 9, FP = 0)
class_id = 24, name = Y, ap = 100.00%           (TP = 4, FP = 0)
class_id = 25, name = Z, ap = 95.24%      (TP = 5, FP = 1)

 for conf_thresh = 0.25, precision = 0.97, recall = 0.97, F1-score = 0.97
 for conf_thresh = 0.25, TP = 146, FP = 4, FN = 5, average IoU = 76.26 %
```

Code snippets for training results

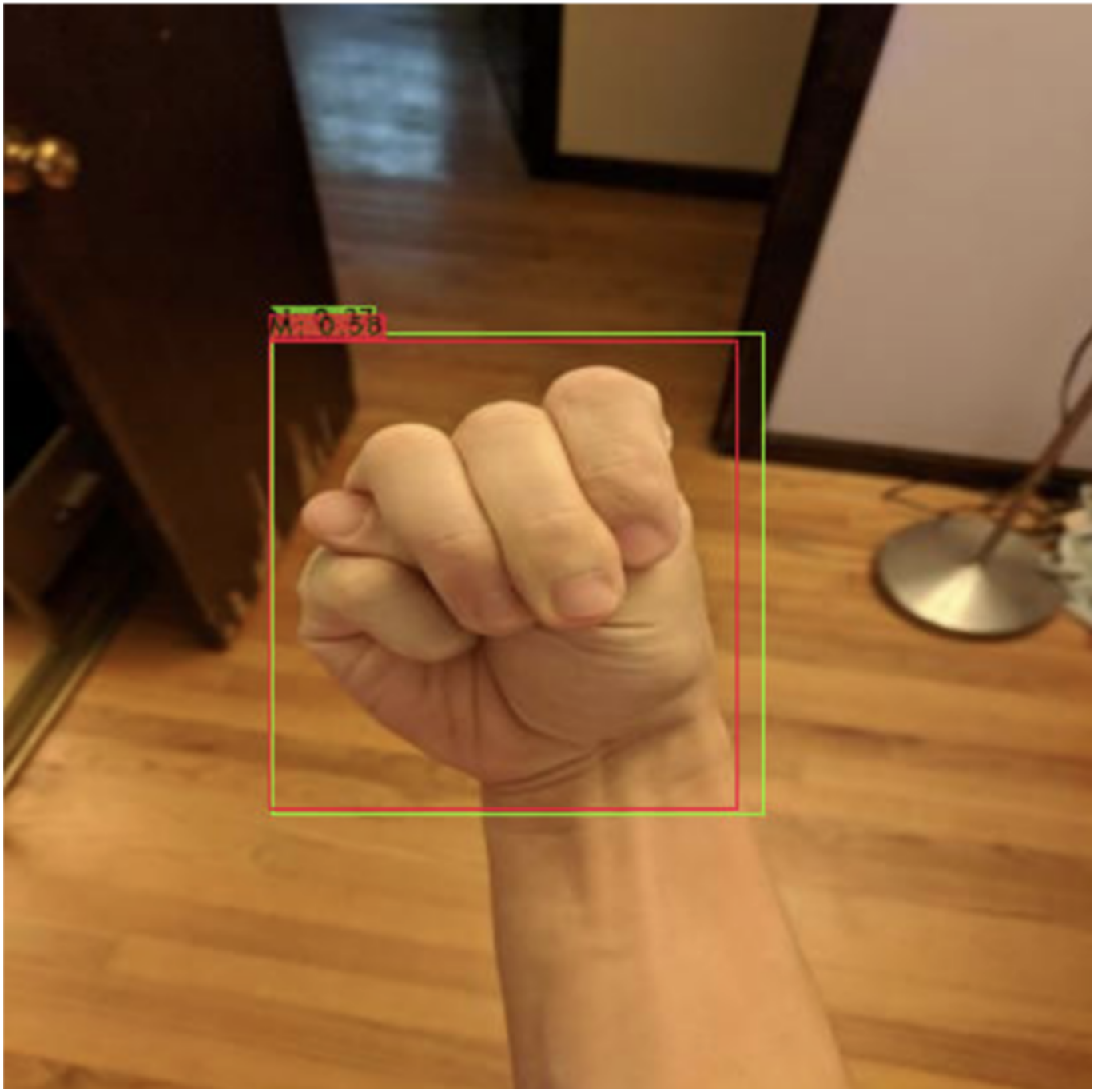
Results

Detection of Image and Video inputs. After a point when the model accuracy and cost were saturated, we tested the model on unseen data. We started by checking output for test images. We observed that the results for pictures with messy landscapes were not satisfactory whereas pictures with clear backgrounds resulted in a good confidence score. Even though the results for the video showed decent success, it was also not very impressive as the model could detect only 4 out of 6 letters in the video correctly. This can be attributed to the limitation of YOLOv4-tiny or smaller size of training dataset which consisted of around 1500 unique pictures. It is known that detecting static types of data such as pictures and pre-recorded video can be implemented in a better waywith heavier algorithms — YOLOv4 or SSD.

Open in app



Open in app



Open in app

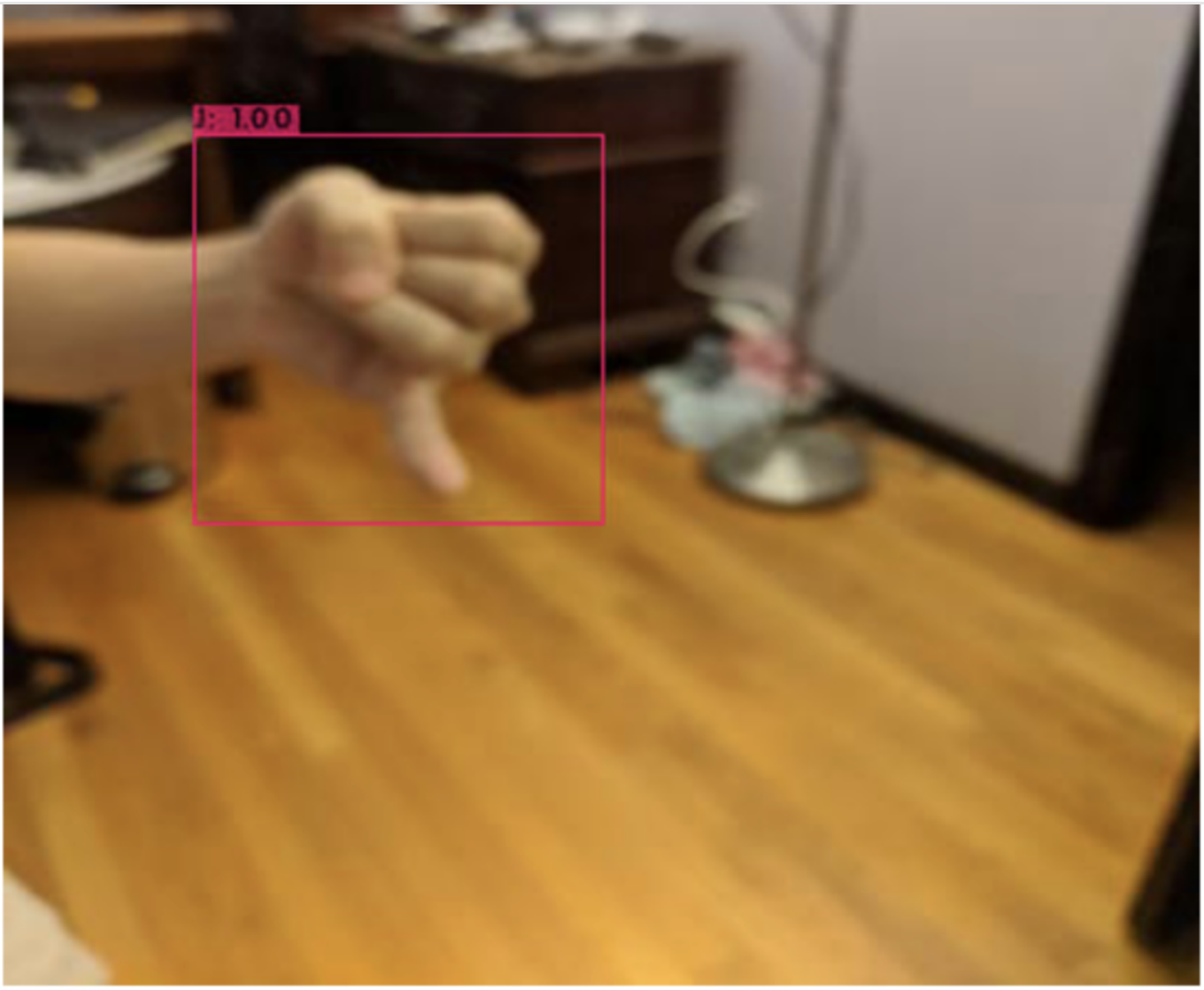


Image input object detection results

Link for video input object detection results

Detection On Real-Time Data. In the link below, we can observe how YOLOv4-tiny detects the sign alphabets made by our team members! It decently captured what the alphabets were. Even though the shapes of G and H are not so much different, our model differentiated them with some extent of confidence. In addition, it discerned the O from C. From morphological perspective, C and O share many properties like how we grip our hands, but our model successfully concluded to give the alphabet the sign of O, not C. These results, of course, could have varied depending upon what constitutes the frames such as brightness, how many objects are there with the targeted object, and/or how the objects moved. Even the number of dataset and pre-trained weights from darknet should

Open in app

that properly leveraging pre-train and selecting dataset might bring us significantly improved results.

Link for real-time object detection results

Applications

Business Applications. Now that we've seen the results of our model we wanted to make comparisons to real business applications that currently exist in the market to demonstrate the capabilities image detection has to tackle real-world problems. To start with we learned about Fingerspelling.xyz, which is a website designed to help teach people sign language in real-time. Just by using a webcam one can get live instruction on where to position their hand for certain letters and words and real-time feedback in the form of a precision score on how well they are doing. This is an early-stage example on how well image detection can help in the educational field of sign language. This can replace the need to be taught sign language by another person and is surely an improvement from using mere illustrations when learning sign language.

Next, we looked into a sign language detection system in the works from Google's AI research [Source 5]. Though it has yet to launch, Google has made strides towards creating a solution for sign language users who wish to interact with users through Zoom, Microsoft Teams, Google Meet, or other video conferencing software that is so prevalent in today's business world. The key to the model is that it looks for the "when" aspect of a person starting to use sign language to speak during a meeting, as opposed to the "what" of what is actually being said. Instead of translating the software, it detects when someone has gone from their base body language to actually using signs. By implementing a high pitch sound that is silent to humans, the model can trick applications like Zoom into lighting up the user's thumbnail (or video) such that it brings attention to the user signing as if they were talking. This brings a much-needed resource and accessibility to an area that thus far has been lacking for deaf users [Source 5].

Lastly, we researched SignAll, a relatively small company that is taking on the immeasurable task of real-time sign language detection and translation that other, larger companies seem to be forgoing [Source 6]. SignAll's goal is one that most relates

Open in app

translation of full phrases and sentences with the idea that the context of reading a sentence as a whole can have a huge impact on the correct interpretation of its meaning and, thus, a successful translation. SignAll is currently working on the launch of their pilot. However, because there currently isn't a complete 3D repository of sign language (and they doubt whether that can even be fully accomplished) they are experiencing limitations. In order to overcome this, they rely on customizing each pilot to the environment in which it is deployed using "domain-specific gestures" added to a database on a rolling basis [Source 6].

Future Applications. Next, we wanted to discuss beyond the current business applications and look at where image detection is heading in the context of assisting sign language users. In particular, it is the hope of many data scientists that sign language detection moves into the stages of translation experienced by verbal languages working with the same level of ease as speaking into a mobile device and having the phrase repeated to another person in a different language. Sign language seems to be making its way towards this goal of being converted into text and audio, though at a much slower pace [Source 7]. Eventually, we could see mobile devices being used as sign language dictionaries on hand, as well as being able to convert sign language into text and audio in real-time, such as by using Siri. Since there are also many other types of sign language in addition to American sign language, such as Indian and Thai sign language, it is also speculated that sign language will move into being able to convert and translate into other languages and dialects using detection models [Source 7]. We may also see sign language detection play a role in the future of sign language education, such as in the form of games directed at deaf children learning sign language for the first time [Source 7].

Sign language will also benefit greatly from the stored knowledge gained while solving a problem and using this knowledge to solve a similar problem in the application of transfer learning. In this context, models were able to learn on animated avatars of people and transfer this knowledge to spot words produced by human signers, which may open up pathways to how models are trained and utilized in the future. They were also able to use knowledge of certain angles of signs and apply this knowledge to determine the same signs at various other angles, which can help models deal with

Open in app

Lastly, we discovered data scientists are having success in creating AI proficient in human lip reading. For example, the University of Oxford's Department of Computer Science developed an AI called LipNet, which had an accuracy of 93.4% at identifying words when given a training set of three-second sentences. This was compared to an accuracy of only 52.3 percent for humans attempting to do the same. To pull this off, LipNet attempts to identify variations in mouth shape over time taking in the full context of a sentence to try and explain what is being said. The Oxford group also teamed up with Google's DeepMind using a much more difficult data set with a greater variety of words, head positions, and lighting. They were able to accomplish an accuracy of 46.8%, which is still impressive considering humans only managed an accuracy of 12.4%. Lip reading may be able to extend some assistance to sign language detection by translating silent videos into text or audio form just by seeing people's faces. Facial recognition may also prove to be a useful tool for sign language detection. Just as SignAll attempted to get the full context for their models, facial recognition can help add the context of how a person is delivering a sentence, which may change the meaning of a sentence entirely. Thus, the future of these detection models may incorporate a mix of lip reading and facial recognition to get the full picture and a sophisticated system that can be applied to real-world use [Source 9].

Conclusion

To summarize, we built a real-time sign language detection system, leveraging the YOLOv4-tiny object detection model that is capable of classifying sign language signs (into English text) from image, video, and real-time input. This model, which used transfer learning, was trained on 1,512 images. Most of that pre-processing occurred in the modeling stage, including data augmentation. Taking advantage of Bag of Freebies and Bag of Specials, our model has strong performance in our object detection. Going forward, future work can include building a model using YOLOv4 or YOLOv5, adding sign-language phrases, and even include adding novel behaviors such as more complex angles and lip-reading to enhance detection. Overall, our work shows how we can leverage real-time object detection to help serve an underserved community.

Sources

Open in app

2 YOLOv4: Optimal Speed and Accuracy of Object Detection

3 DFFAN: Dual Function Feature Aggregation Network for Semantic Segmentation of Land Cover

4 Mish: A Self Regularized Non-Monotonic Neural Activation Function

5 How Google Is Making Sign Language More Accessible In Video Calls

6 SignAll is slowly but surely building a sign language translation platform

7 Artificial Technologies for Sign Language

8 Transfer Learning in Sign Language

9 AI Has Beaten Humans at Lip-reading

10 https://blog.roboflow.com/a-thorough-breakdown-of-yolov4/

11 https://www.kdnuggets.com/2020/08/metrics-evaluate-deep-learning-object-detectors.html

Disclaimer: This is not a research / recommendation report and is solely for educational purposes

About Write Help Legal

Get the Medium app

Download on the App Store GET IT ON Google Play