Project: The classification contest

LIS4042 Artificial Vision, fall 2023 Professor: Zobeida Guzmán

Selection of the model:

- Recognition of sign language can have multiple applications, on this Project we fine tuned different object detection models to detect different letters of the ASL sign language. We could use this for human computer interaction where users could interact with computers without touching them, this could also be the base for future full sign language recognition that could allow people to that uses ASL to communicate more easily.
- The main problem, lies on choosing the right model for our task and to train it on the right dataset with the correct hyperparameters. On top of that, circumstances must be considered, on this case, I only have a computer with a low quality camera and no GPU, so my priority Will be to get an efficient model that does not make my video go to -1 fps :^).
- On this Project, given the hardware constraints of my computer, I opted for YOLOv5s and RCNN-mobilenet due to its balance between speed and accuracy, making it a suitable choice for real-time applications, they are also and small and both have support on the pytorch library (and decent documentation).

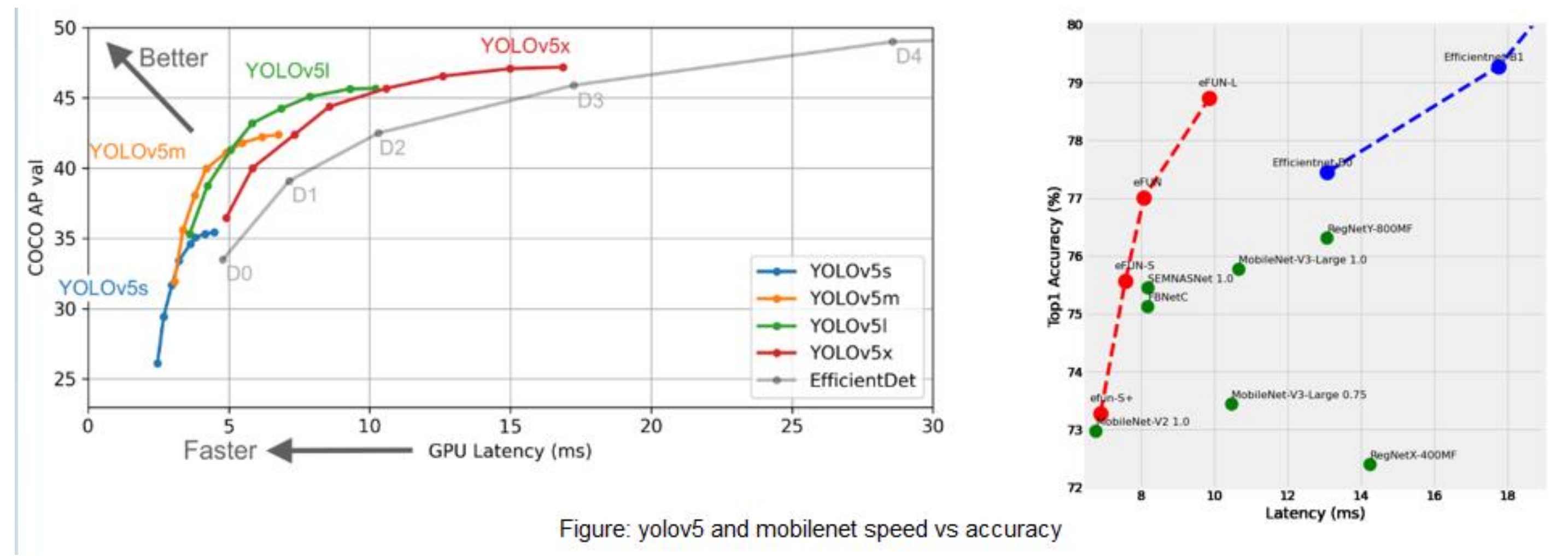
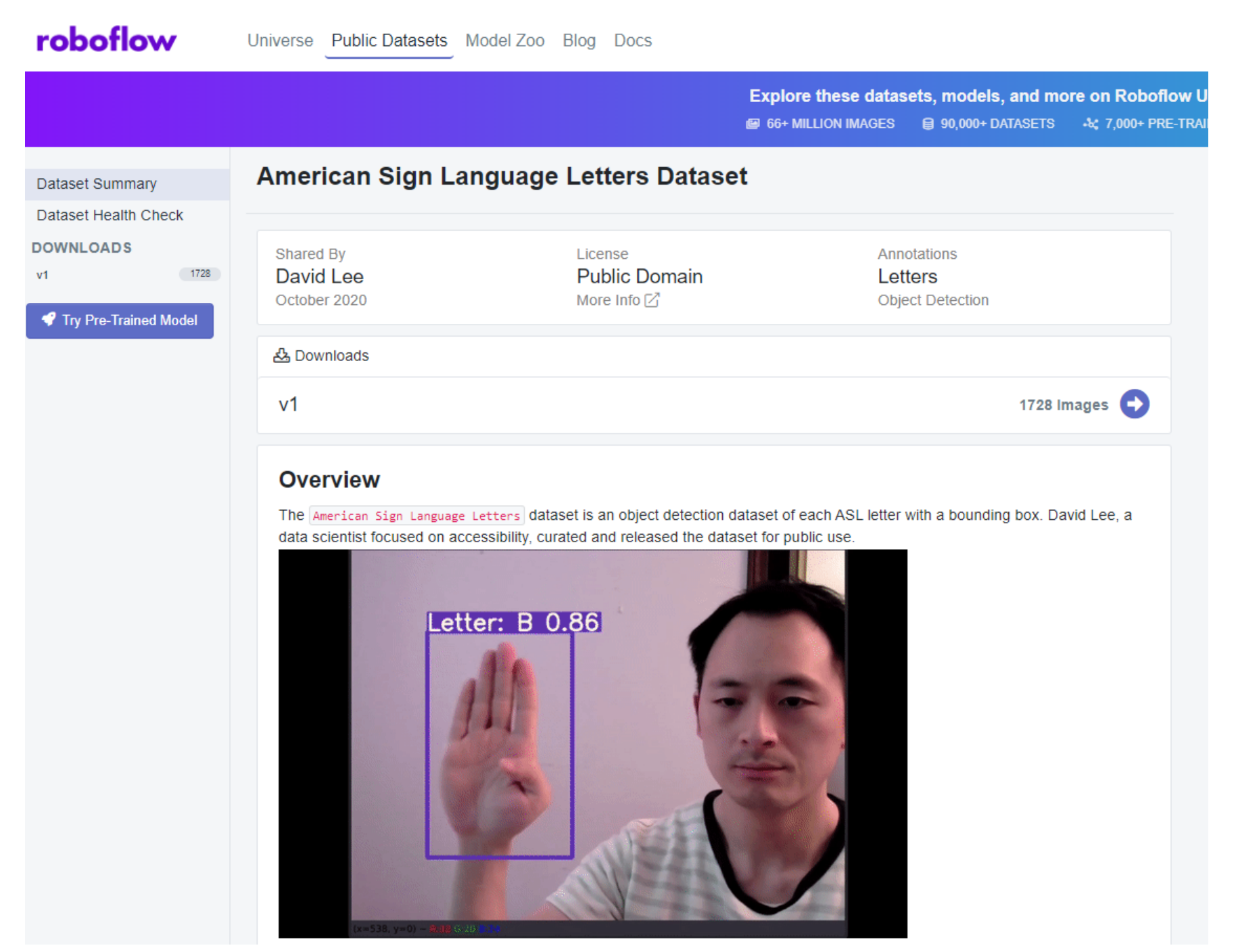


Figure: yolov5 and mobilenet speed vs accuracy

Dataset and preprocessing

- To train the model, it is needed to provided train images labeled with the position of the object and the class of the object. For this we most specified the coordinates of the bounding box and the name of the object in the right format, for Yolo it is needed to provide a .txt containing this information. To obtain a dataset containing images with its labels we can either manually label each image or use an available dataset online, such as roboflow's dataset



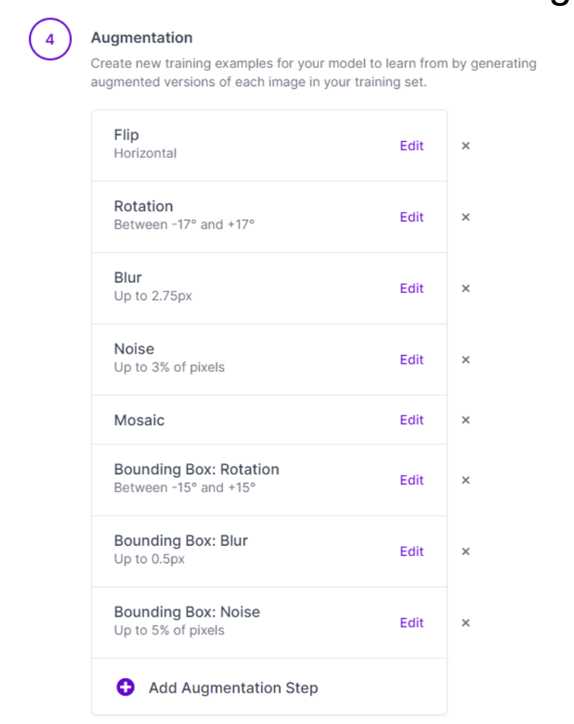
American Sign Language Letters Dataset

Shared By: David Lee, October 2020
License: Public Domain
Annotations: Letters, Object Detection

Downloads
v1 1728 images

Overview
The American Sign Language Letters dataset is an object detection dataset of each ASL letter with a bounding box. David Lee, a data scientist focused on accessibility, curated and released the dataset for public use

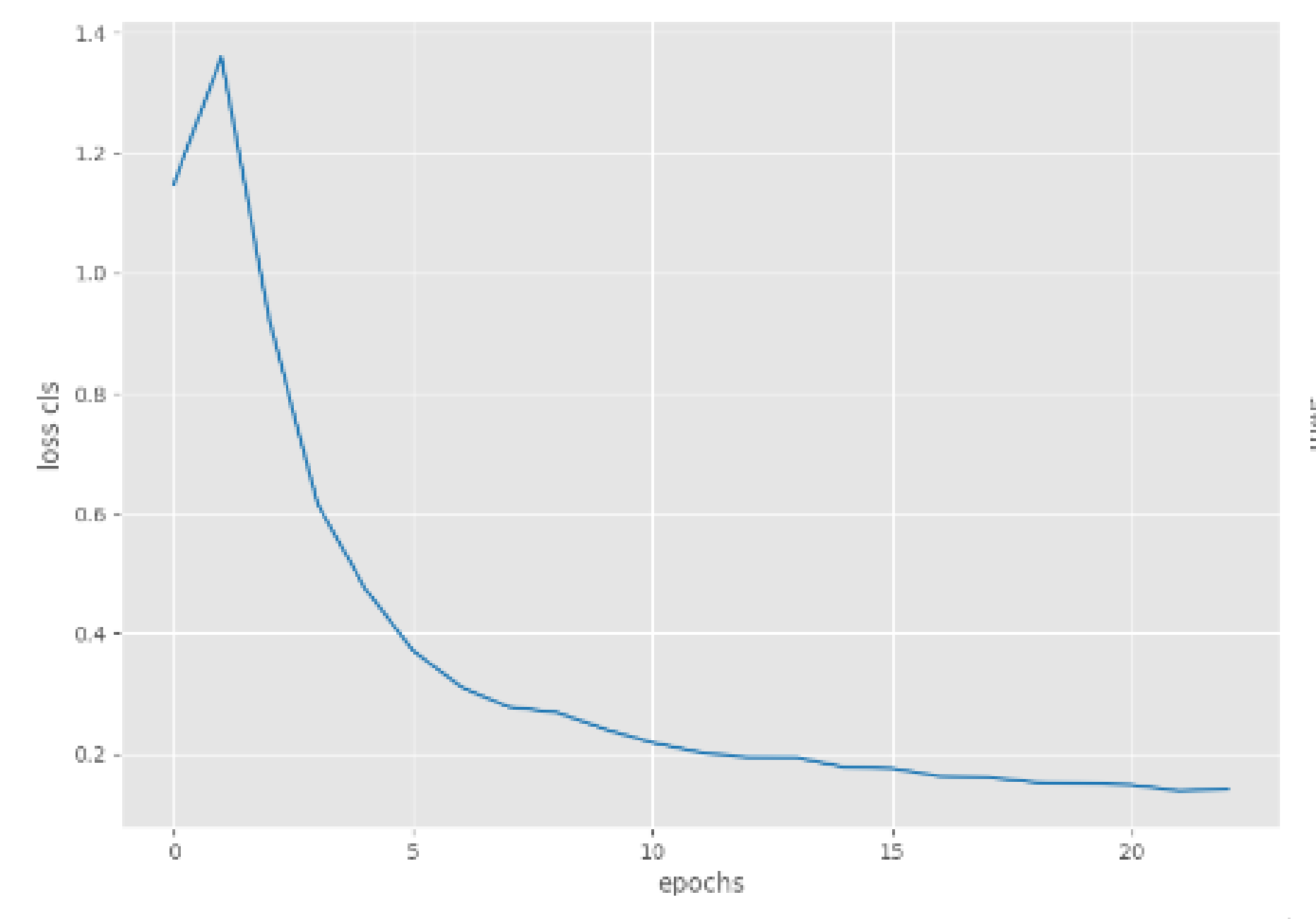
- To help the model generalize we use data augmentations, we are not experts on data augmentation, but thankfully on roboflow we can take an existing model and créate a new one by selecting our preprocess, even our bounding box coordinates are process to still match object's location, we add common augmentation techniques. The model is also already splited into train, validation and test. On my case, my camera is low quality, thus adding noise, we can help our model by adding noise and some blur to training images, thankfullly we can do this on roboflow

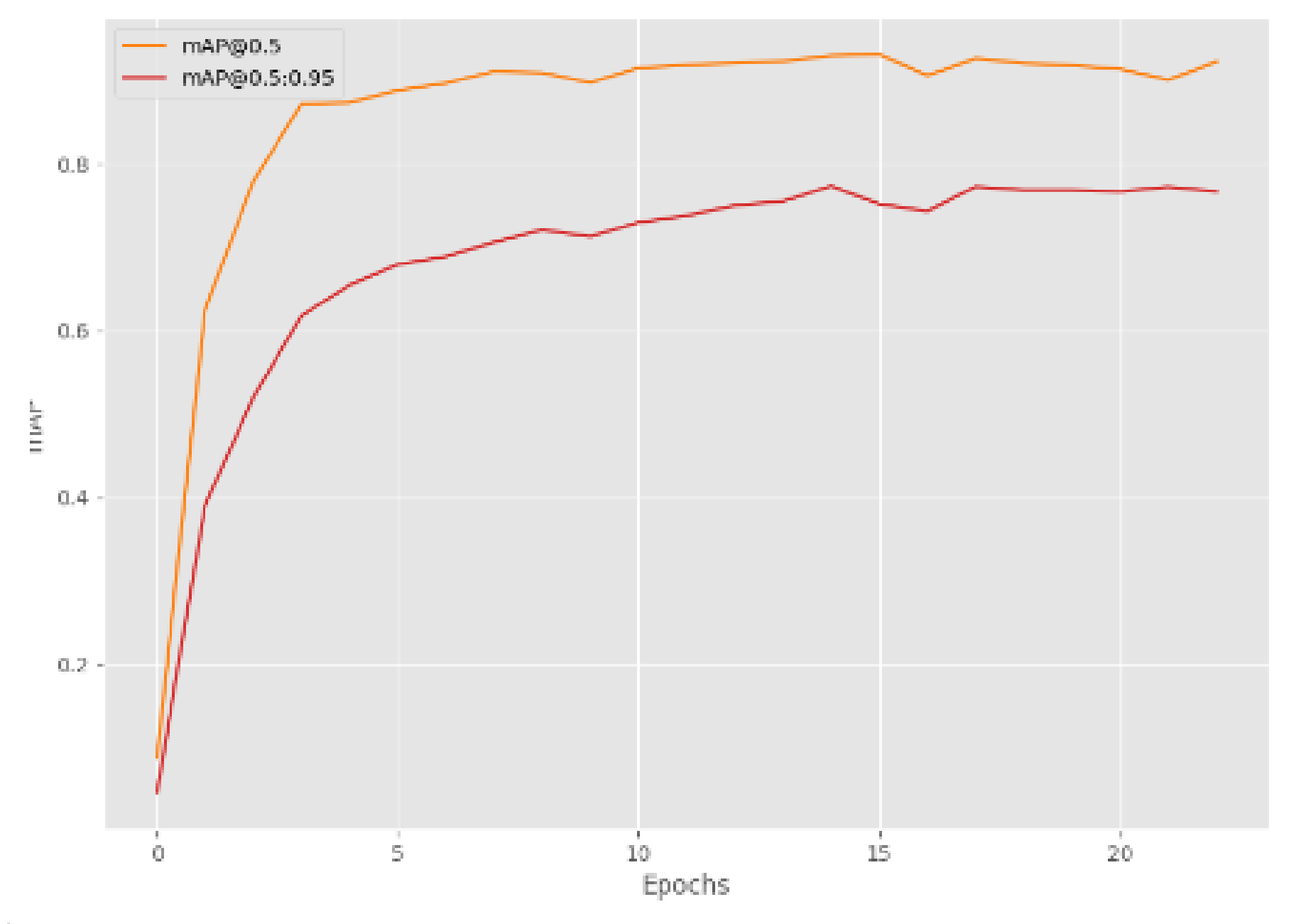

4 Augmentation
Create new training examples for your model to learn from by generating augmented versions of each image in your training set.

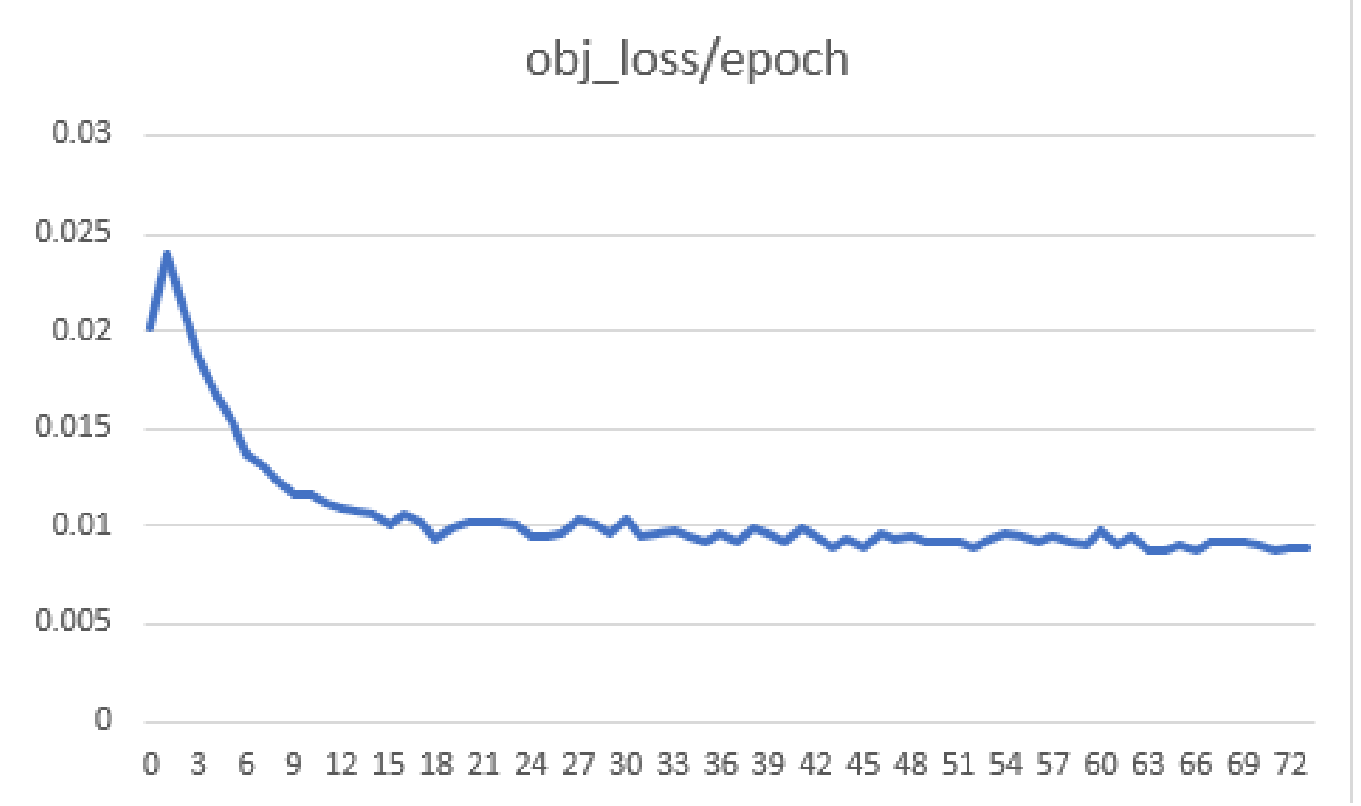
Flip: Horizontal
Rotation: Between -17° and +17°
Blur: Up to 2.75px
Noise: Up to 3% of pixels
Mosaic
Bounding Box: Rotation: Between -15° and +15°
Bounding Box: Blur: Up to 0.5px
Bounding Box: Noise: Up to 5% of pixels

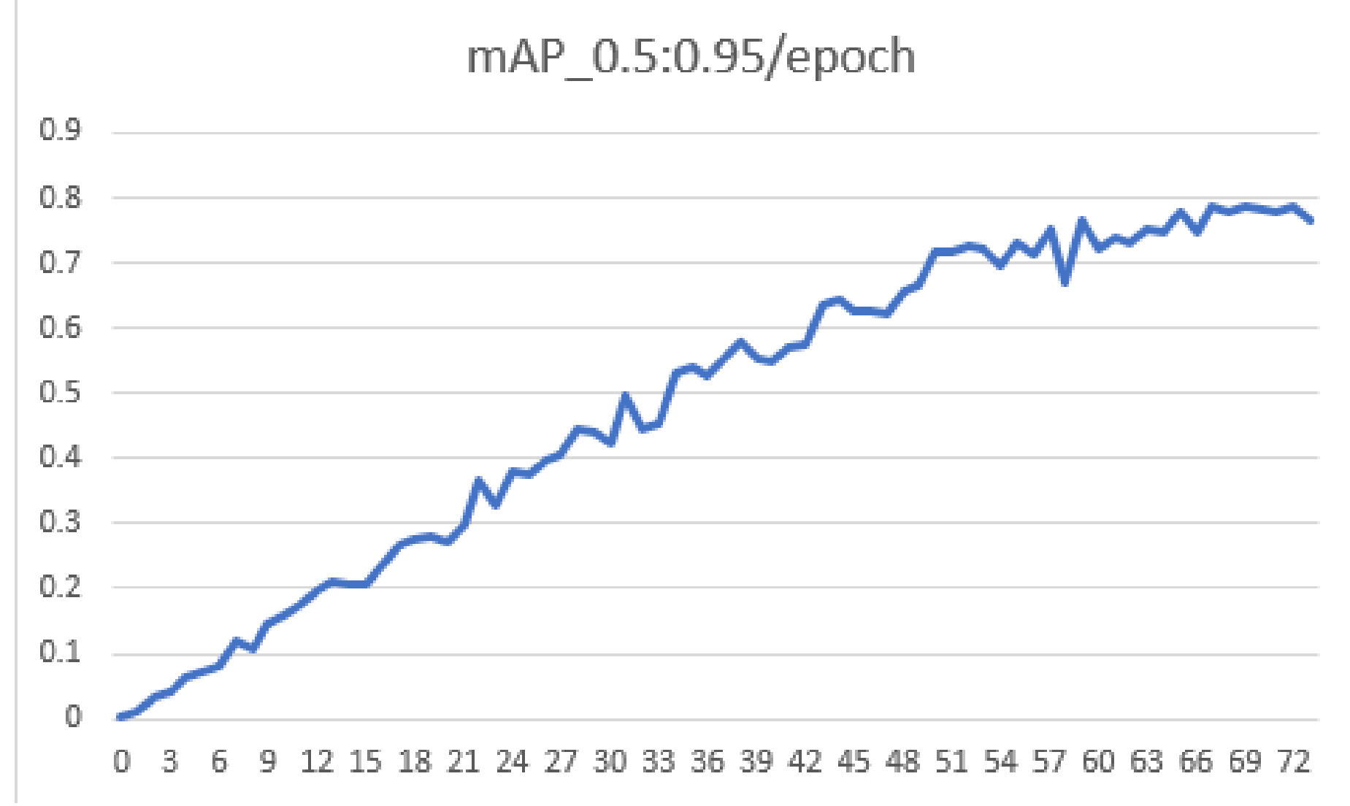
Training

- Once our data to train our model is ready, we can use the prebuilt libraries and scripts that the designer of the model made for us, each of the model creators provides in their project methods and classes to communicate with the important code inside their model. This way we can train our model by just calling the appropriate program and passing the right arguments, such as dataset path, epochs, data augmentations (if supported), workers, image size, file containing the head modifications and others. In turn, this trains the model on this parameters and returns the checkpoints/weights which we can load on the computer/device and start making predictions. In this project, training was performed on Google Collab, since GPU is not available, once the weights are obtained, we can run our model on a local device to start making predictions.
- To assess the performance of the model, I will be using mean Average Precision (mAP) across 0.5:0.9] as they provide a comprehensive understanding of the model's performance. When training the model, the hyperparameters are important, as well as interpret the resulting graphs for our model's performance, since this parameters can tell us if the model is learning correctly. For instance, we could overfit the model if we overtrain it and then the model will learn extremely well the dataset but will not make good prediction on real life environments, we can also run out of memory if our batch size is too big or get stuck at some mAP, we are not experts on how to tune our hyperparameters. Based on mAP[0.5 0.9] each model was tuned to get better performance based on recommendations available on the web.

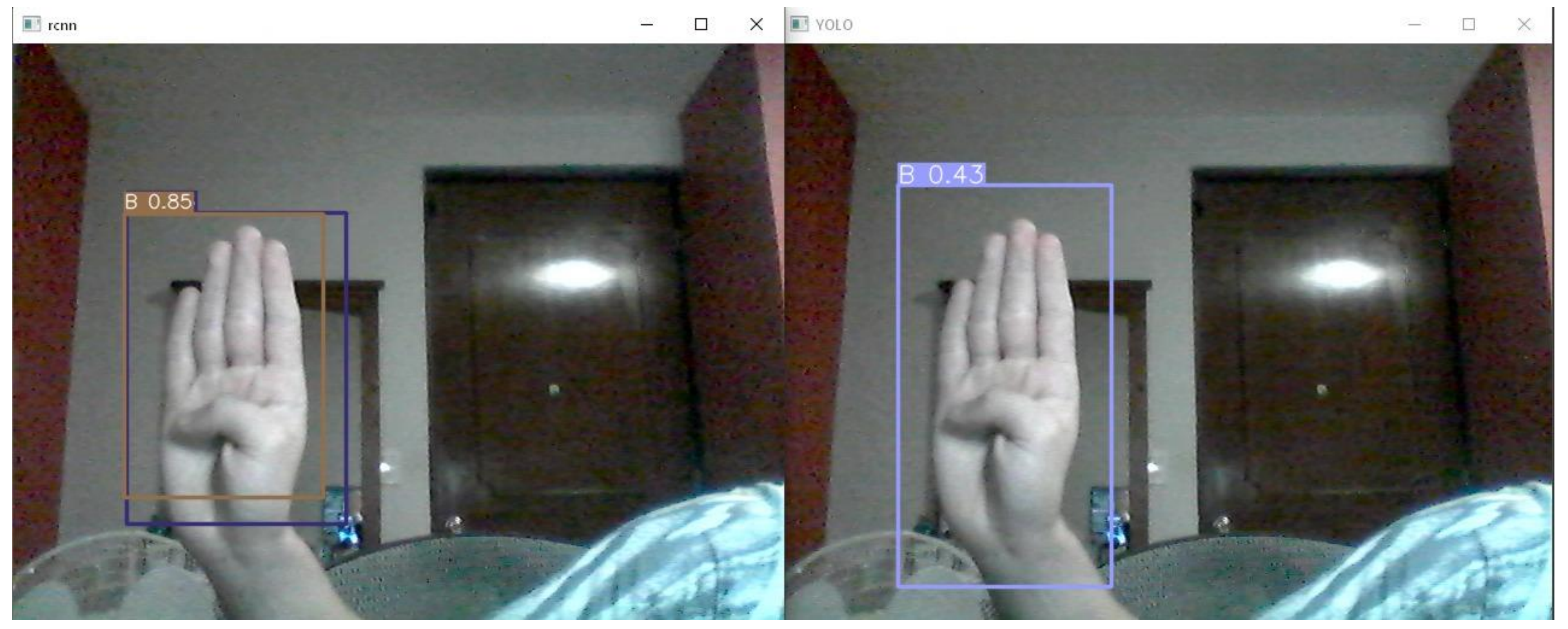








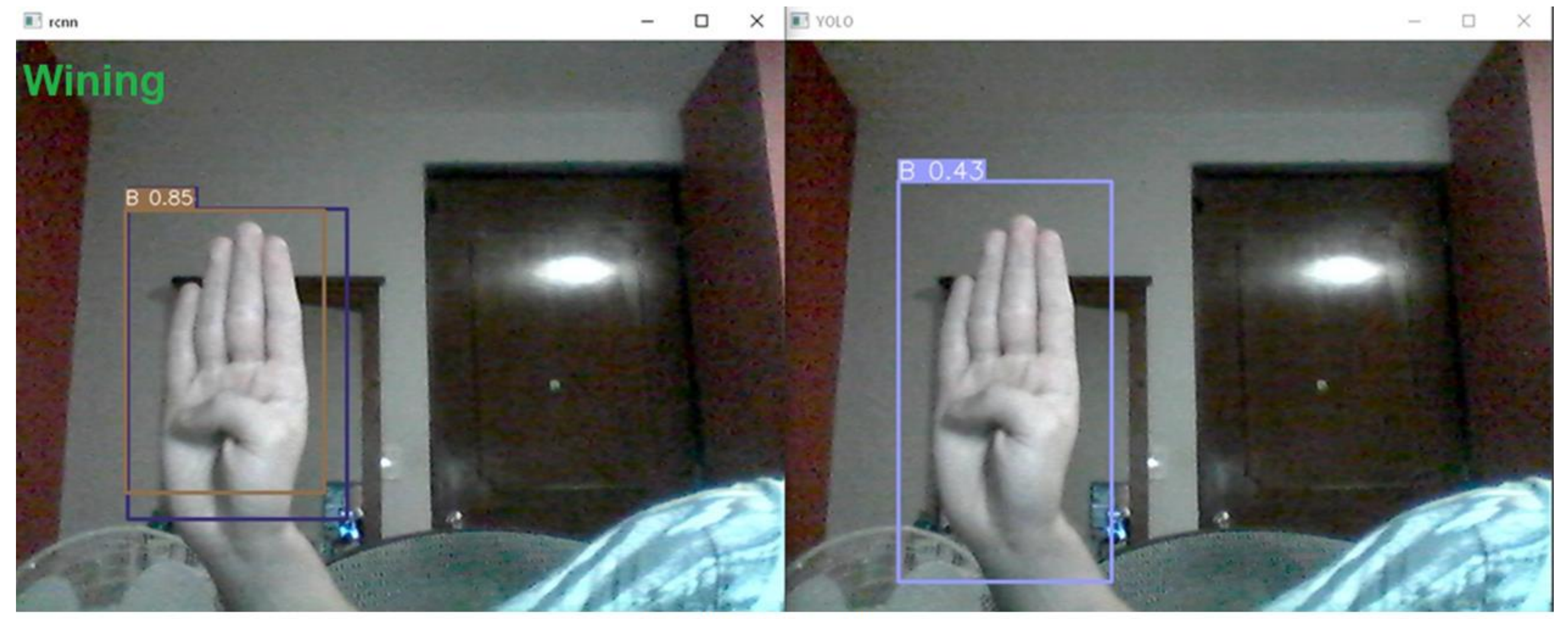
- Both models have drawbacks when testing, the FRCNN-mobilenet model detects with good confidence some classes, but for others (particularly the sign that are similar such as 'A' or 'X') is completely lost. On the other hand, yolo has low confidence score, but most of the time will get the right class even if we test it with similar hand signs.



- For both models it is not easy to detect the correct sign when the hand is too close, too far or there is low light, it is also not easy for the models to detect hand signs for different skin-colors, these problems could be reduced by using more augmentations so that our model can generalize more and more data samples (more diverse dataset).

The contest

- How can we decide which model is better if the program does not know what it is looking at? One way would be to tell the program what objects Will be shown in the next 60 seconds, for the fist 10 seconds 'A'. another method could be: if both models detect same sign (at least the object with highest score detected by each model) gets a point the one with more score or the model that detects the same object for the next 10 frames, get a point. If both models detect the same object, the one with highest score gets 2 points. Then for the model that is winning we print on image "winning". This is a simple method, but we need to be careful with the computer resources since the machine where this is running is not the best. Of course, this method has drawbacks, such as a model detecting the wrong object for 10 times getting a point, or both models detecting the wrong object and the one with highest score gets a point, so, considering that we will be presenting 1 sign at the time, if a model detects more than 1 object, then it gets -1 points. This simple method provides with visual information of which model is wining, with more computational resources, we could also use a heavier model such as Yolov5lL, adding a third model with better accuracy that will decide what object is being presented and then the model that detects it gets a point. Again, with the computational resources at our disposal we choose the simple method that thanks to redundancy, still gives a good insight of which method is best.



Figre: Models being tested

Conclusions:

- In conclusion, the comparison between YOLOv5 and Faster R-CNN with MobileNet for ASL (American Sign Language) sign language detection reveals that YOLOv5 outperforms Faster R-CNN MobileNet in terms of accuracy. Both models demonstrate the capability to detect signs effectively, but yolov5s gets the right sign more often, and they efficient enough to be run on CPU. The implementation process, however, posed challenges due to the differing APIs and data format requirements of YOLOv5 and Faster R-CNN MobileNet. The need for dataset preprocessing became crucial to facilitate optimal learning for both models, though more images examples could have been beneficial for both models, specially for RCNN-mobilnet since it did not generalize as expected. Despite the challenges, the superior performance of YOLOv5 suggests that its architecture and design contribute to better overall results for ASL sign language detection. The efficiency of both models in running on CPU makes them viable options for deployment in environments where computational resources are limited, even when the camera quality is low. In future work, it may be beneficial to explore additional optimizations, fine-tuning strategies, or consider alternative architectures to further enhance the performance and robustness of ASL sign language detection systems. It is not easy to test a model against other, how can we know which is right if we don't know the answer? For this redundancy results crucial. Additionally, considering the results observed, I personally would choose yolo model to work on Deep learning applications, its documentation and performance are amazing when compared to other models.