Real-time Object Detection and Classification for ASL alphabet

Rain Juhl
Stanford University
rjuhl@stanford.edu

Eric Feng
Stanford University
ericfeng@stanford.edu

A. Abstract

In this paper we will experimenting with real time American Sign Language (ASL) Object Detection and Classification. We will be utilizing Roboflows American Sign Language Dataset, which contains 1728 images of hands spelling letters A-Z. Utilizing three different models: a Faster RCNN model with a ResNet backbone, a Faster RCNN model with a mobileNetv3 Backbone, and a YOLOv5 model, we analyse the results of the models, comparing the trade offs between performance and computational resources of each of our models. We ultimately are able to present a model that is consistent with classifications of the ASL alphabet, that is robust to unique hands in varying unique backgrounds, and that is able to draw bounding boxes and make classifications in real-time on GPU or even CPU. There are only 250,000 - 500,000 ASL speakers in the world. We hope that this compact fast running model can be utilized as a starting point to build simple easy tools to teach/transcribe sign language that can be run on consumer grade computers.

B. Introduction

American Sign Language (ASL) is a main component utilized within the deaf community to communicate with others in America. According to the SIPP, nearly 10,000,000 persons are hard of hearing, with nearly 1,000,000 of those people functionally deaf. However, in contrast, only 250000 - 500000 Americans speak American Sign language. We hope that the models that we present can serve as a starting point for products that can serve to make communicating with other easier for deaf people, or serve as a tool to teach others American Sign Language. With the results that we present in this paper, we show that the possibility of ASL object detection and classification on widely accessible modern day consumer grade CPUs may soon be a possibility. We chose to pursue utilizing object detection in this domain as we reasoned it better allows for real time transcription. By allowing the user to observe what parts of a live feed the model is making the prediction from, we allow the user to identify what part of an image the model makes the prediction from, and quickly allow users to see when the model is truly understanding them.

B.1. Problem statement

In this project we experimented with 3 different models. The 3 models are a ResNet Faster RCNN, a mobileNet Faster RCNN, and finally YOLOv5. For each one of the models we implemented, the input to the model is a color image (3 x H x W) of a hand, along with the image of the hand, labels for bounding boxes around the hand were also included, the bounded boxes were passed in as an x coordinate a y coordinate, and the length and width of the box. Each bounded box also had a integer label passed that corresponds to the letter of the alphabet that the ASL sign corresponds to: IE the label 1 corresponds to the letter A, label 2 corresponds to B, ..., the label 26 corresponds to Z. The model then outputs a range of guesses for where a bounding boxes will be drawn as lists of 4 values. These 4 values correspond to the 2 x and y coordinates necessary to construct the corners of a bounding box, the model also returns a predication of the classification of the ASL image. Corresponding to each of the bounding box guesses are scores on how certain the model is of the prediction.

C. Related Works

Object detection and classification has been an area of focus for computer vision scientists for a long time, just 20 years ago it was thought of as the paragon of computer vision. [20] Since then the field has come a long way.

Current deep object detection/classification models fall to into one of two categories. The first of which are are two stages detection algorithms. The most notable and state of the art models that fall into this category are fast RCNN and Faster RCNN [6] [16], Pyramid Networks [18], and most recently the G-RCNN [9].

Both the fast RCNN and faster RCNN models are successors to the RCNN model [15]. It was comprised of a region proposal system, a CNN to extract features from the proposed regions, and a collection of class unique SVMs. This paper was the first to show the effectiveness of CNNs for object detection tasks [15]. Its predecessor the fast

RCNN builds upon the architecture and tackles many of the drawbacks found in the RCNN. Proposals being fed through feature extraction system independently and multi-stage architecture made inference time slow, and training difficult. The Fast RCNN used ROI Pooling to extract equal length vectors from proposals which allowed the CNN calculations to be shared across proposals. Additionally, the Fast RCNN was combined into a single network making training easier [6]. Finally, the Faster R-CNN changed the proposal system, using a new system called an RPM which is also a CNN based method. This allowed CNN computation to be shared along the proposal method and the previously described Fast R-CNN method making the model even faster and hence the name [16].

The next development was the Pyramid Networks/FPNs which utilized the Faster RCNN but reintroduced feature pyramids but integrated them with a new proposed FPM structure. This structure leveraged the pyramid feature extraction but at a fraction of the previous computational cost. This allowed the model to generalize better especially when it needed to identify multiple objects of different sizes for just a marginal computational cost [18].

Finally, the most recent development for two stage detection methods is the G-RCNN. The G-RCNN improves on the base RCNN by adding a gate to the residual connection that modulates the amount contextual information that is passed through. This change improved the RCNN and made it competitive with the other state-of-the-art models [9].

The second category of models are known as one-stage detection algorithms. Models such as YOLO (all five versions) and most notable YOLOv5 [11], SSD [19], and RetinaNet [4].

All YOLO models are designed to maximize mean average precison (mAP) and share the same core components which consists of the backbone: a CNN to extract features, a neck: which consist of a group of layers that works on the features before passing them onto the prediction layer (uses a pyramid network/FPN which we discussed earlier), and a Head: uses the features extracted from neck in conjunction with the bounding box predictions to give the final output of the bounding box. The first YOLO model consisted of all these parts and used a grid for acquiring bounding box proposals [11]. YOLOv2 instead used anchor boxes and model is randomly resized during training to improve generalization to images of a variety of aspect ratios [10]. YOLOv3 combined both RCNN and FPN while switching to new architecture for its backbone [5]. YOLOv4 added what is colloquially known as a Bag of Freebies (BOF) in the machine learning community. BOF consists of many techniques that researches have found that generally increase a models performance. For YOLOv4 this was a series of data augmentation techniques, a series of regularization techniques, and a series of normalization techniques [3]. Finally, YOLOv5 has no paper has been shown to just as fast and powerful as YOLOv4 if not better.

SSD or single shot detection just has a backbone comprised of a pre trained feature extractor (often a ResNET) and a head which is comprised of CNN layers that are used to find bounding boxes and classify them [19]. Overall, comparison between YOLO and SSD show that YOLO is often faster and SSD is slightly more precise.

Finally, RetinaNet is also comprised of a backbone which computes a feature maps using CNNS and is made up of a subnet that classifies the bounding box. Additoinally, RetinaNet uses focus loss which concentrates on hard data examples [4]. RetinaNet has been found to more precise than YOLO but is slower and cannot detect multiple bounding box and classify them simultaneously.

Overall both categories have shown large improvement over the years in object classification and detection. When comparing the two categories the two stage methods tend to be more precise in the predictions but the one stage methods are not far off and tend to be faster.

Additionally, researches have set out to tackle ASL translation specifically. Often they rely on of the techniques we have just gone through. Using YOLOv5 with some augmentations researchers were able to achieve 0.987 map@0.5 when classifying the ASL alphabet and the first ten numbers [17]. Furthermore, in the paper ASL Recognition with Metric-Learning based Lightweight Network researchers develop a model with few parameters that achieves a mAP of 87.79 on 100 different sizes. It does so by using backbone that is a edge-oriented MobileNet-V3 and using a residual spatio-temporal attention module in conjunction with auxiliary loss [8].

D. Methods

D.1. Faster R-CNN ResNet-50

The Faster R-CNN model is a two stage object detection algorithom, the model is composed if two main parts:

The model first utilizes a trained deep learning convolutional neural network in order to propose regions to the models known as the Region Proposal Network (RPN) and the second model is the fast RCNN detector that uses the proposed regions to make the final region detection decisions.

For the baseline model, we utilize the ResNet-50 model [7] as the backbone convolutional network model that initially extracts the features of the image for the region proposal network.

For our implementation, we we utilized the pretrained ResNet backbone model. The pretrained model was trained on mini-batches of 3 channel RGB images from ImageNet. Images were preprocessed to be normalized using the mean

of [.485., .456, .406] and standard deviation of [.229, .224, .225]. We change the number of output nodes for the classification to be 27 classes and finally we freeze the parameters of the first 5 layers of the ResNet backbone network.

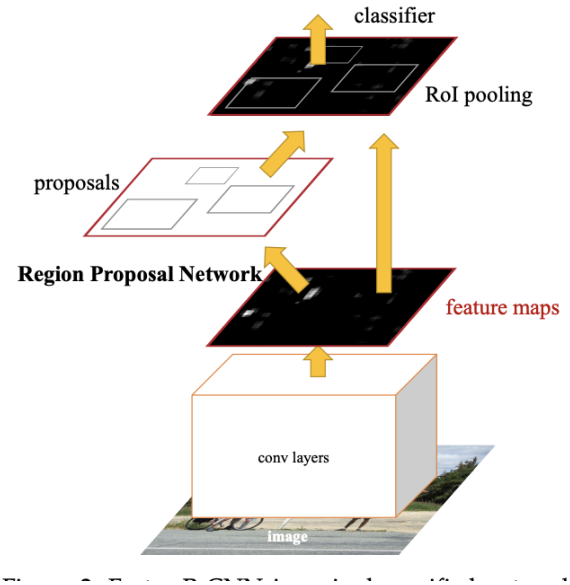


Figure 1. The architecture of Faster R-CNN models. It is a two stage architecture. The RPN first purposes regions utilizing a convolutional neural network, then a fast RCNN detector, makes the final region detection decisions.

D.2. Faster R-CNN MobileNet

The main unique aspect of the model is the MobileNet backbone of the R-CNN. The MobileNet model is a model designed to be used with CPUs and mobile devices. It is specifically tuned through a combination of hardware-aware network architecture tuning. An important unique aspect of the mobileNet architecture is that it utilizes depthwise separable convolutions, replacing normal convolutions.

For a standard convolutional layer, it takes $D_F \times D_F \times M$ feature map F, and produces $D_F \times D_F \times N$ feature map G. where D_F is the spatial width and height of a square input feature map, M is the number of input channels, and D_G is the spatial width and height of a square. The convolution is thus parameterized by kernel K of size: $D_K \times D_K$

$$\mathbf{G}_{k,l,n} = \sum_{i,j,m} \mathbf{K}_{i,k,m,n} \cdot \mathbf{F}_{k+i-1,l+j-1,m} \qquad (1)$$

Therefore the computational cost of a standard convolution is the following:

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \qquad (2)$$

depth-wise separable convolutions replace the normal convolutions with a two step operation. First is the depth wise convolution, where each $D_F x D_F$ filter is applied per each input channel. These filter convolutions are much more efficient to calculate, however the the depthwise convolution only filters per channel, thus to combine them, point wise convolutions are then applied. Point wise convolution, which is a single 1x1 convolution that is used to create a linear combination of the depthwise layer

We see that the computation cost of the depth-wise separable convolution is now greatly reduced in comparison with normal convolutions:

$$D_K \dot{D}_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \qquad (3)$$

In our implementation of the Faster R-CNN mobileNet, we utilized the pretrained mobilenet model from the torchvision library, similar to the Faster R-CNN ResNet-50 model, the pretrained model was trained on mini-batches of 3 channel RGB images from ImageNet. Images were preprocessed to be normalized using the mean of [.485., .456, .406] and standard deviation of [.229, .224, .225]. We change the number of output nodes for the classification to be 27 classes and finally we freeze the parameters of the first 3 layers of the mobileNet backbone network.

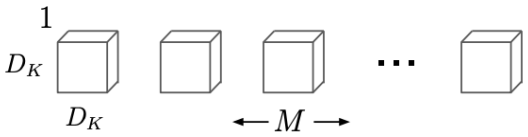


Figure 2. depthwise convolutional Filters

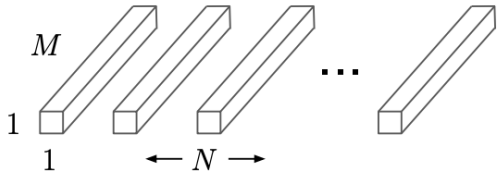
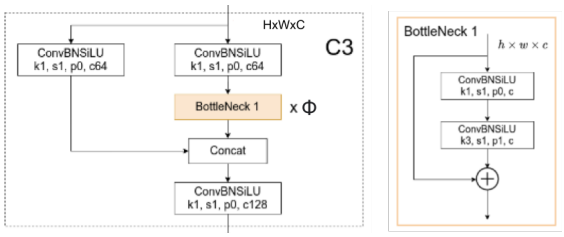


Figure 3. 1×1 Pointwise convolutional layers in context of the Depthwise Seperable Convolution.

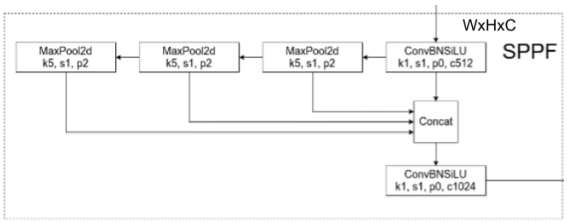
D.3. YOLOv5

YOLOv5 like all models can be split into a backbone, neck, and head. The backbone is a New CSP-Darknet 53 which in composed of mutiple CNN and C3, layers as well as a SPPF layer. First lets break down the C3, layer which looks as follows. (adapted from the Yolov5 github) [2].



$\phi =$ number of bottleneck layers (changes with each layer)

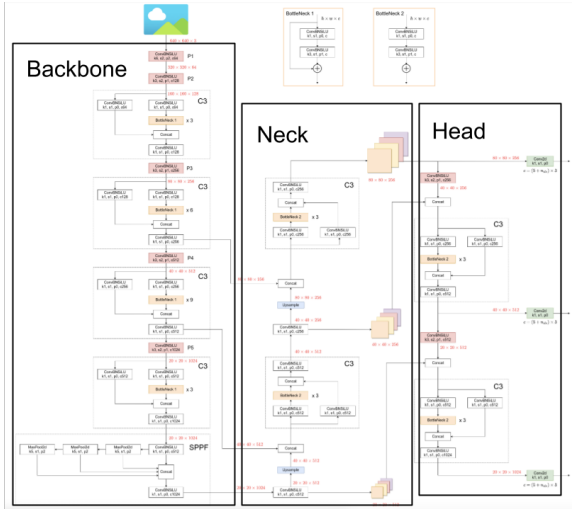
Note that ConvBNSiLU is just a convolutional layer followed by a batch normalization layer, and then finally a sigmoid linear unit layer as an activation, or $x\sigma(x)$. As shown to the right of the figure the bottleneck is simply two ConvBNSiLU and residual connection that gets directly summed together. Below is an illustration, again adapted from them the Yolov5 github [2], that depicts the SPPF layer which takes as input the final the output of the final C3 layer.



All of the building blocks of this layer we have seen before (MaxPool and ConvBNSiLU); however, they are arranged differently. The maxpool build off each other, progressively capturing the max over a larger area, and get concatenated together with the ConvBNSiLU. The concatenated output gets passed through another ConvBNSiLU layer which then returned the final output for the SPPF layer.

The neck is also made of many C3 blocks. The C3 blocks use a slightly modified bottleneck block that is identical to the bottleneck found in the neck except it is missing the residual connection. Here there is a new layer block called upsample, which increases the resolution of the image.

The head is very similar to the neck but it includes a final Conv2d layer at each depth to give the output. The full flow of the model can be seen in the diagram adapted from the Yolov5 github below [2].



To compute loss Yolov5 uses weighted sum of three different losses: Class and Objectness Loss which uses Binary Cross Entropy Loss (BCE), and Location Loss which is CIoU Loss. The goal of CIoU loss is quantify the quality of the predicted bounding boxes. It does this by factoring in the overlap between the prediction and ground truth, comparing central point in the prediction and ground truth, and finally contrasting the aspect ratios between the prediction and ground truth. The Loss formula can be broken down into these three parts and written as the following:

$$Loss_{CIoU} = S + D + \alpha V. \tag{4}$$

The first term (S) measures the box overlap and given the prediction as P and ground truth as G is defined as follows:

$$S = 1 - \frac{P \cap G}{P \cup G} \tag{5}$$

The next term (D) is effected by the central point in the bounding box. Let c be distance for the far corners of the bounding box and let $\rho(P, G)$ be the distance between the center points of prediction and ground truth than D is as follow:

$$D = \frac{\rho^2(P, G)}{c^2} \tag{6}$$

Finally the aspect ratio similarities are quantified by αV. Let P^w, P^h be the height and width of prediction respectively and G^w, G^h be the height and width of the ground truth respectively then α and V are as follows:

$$V = \frac{4}{\pi^2}(\arctan(\frac{G^w}{G^h}) - \arctan(\frac{P^w}{P^h})) \tag{7}$$

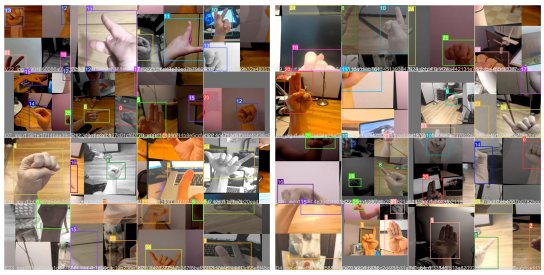
$$\alpha = \begin{cases} 0 & S > 0.5 \\ \frac{V}{S+V} & S \leq 0.5 \end{cases} \tag{8}$$

Finally, the full equation for the loss can be described below where each lambda weights the importance of each loss

function output. For our purposes we used $\lambda_1 = 4, \lambda_2 = 1$, and $\lambda_3 = 0.4$.

$$Loss = \lambda_1 Loss_{Class} + \lambda_2 Objectness + \lambda_3 CIoU. \quad (9)$$

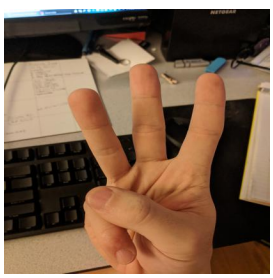
For our implementation of Yolov5 we used a smaller version to increase speed. IN this version the the depth was decreased by a factor of $\frac{1}{3}$ and the width (number of channels) was multiplied by a factor of $\frac{1}{2}$. During training the dataset also went through a variety of extra augmentations including a random affine, mossiacs of different images, and a mixup of different images. Below in an example of mosaic created and trained on.



The model was then finetuned with the backbone frozen for 60 epochs and the output layer was changed from 80 classes to 26. After training the model was then configured to predict a single label for each bounding box with a confidence threshold of 0.2. To evaluate the model we looked at loss, precision, and recall during training, and calculated the precision, recall, f1 score, for each class on the test set. In addition, we generated a confusion matrix. We also measure the average inference speed on CPU (Collab) for batches of size 1. Furthermore, we generated a Saliency map for an image. Finally, this was one of the models we integrated into an online platform to perform live detection and classification on a CPU.

E. Dataset and Features

To train and evaluate all of our models we used the American Sign Language Letters Dataset [1]. The dataset consists of images of ASL letters from a variety of people in a variety of back grounds and has 27 different class labels. One for each letter in the alphabet and a class denoting that no ASL letter is present in the frame. A typical image is looks like the ones below.



Note that both are Ws and the difference in backgrounds. The full dataset contains 1512 images in the train set, 144 images in the validation set, and 216 images in the test set. Each images is labeled with the correct letter classification and bounding box. Below is an example of what an image looks like with its bounding box.

The images were resized to be 416x416 and then normalized. We normalized using mean and variance found in ImageNet dataset which is often the standard. During training images were also augmented, utilizing a random horizontal flip (which does not effect meaning of ASL letters), a crop with 0-20% zoom, a rotation of ± 5, a vertical/horizontal shear of ± 5, a greyscale to 10% of the images, a adjustment to brightness levels of $\pm 25\%$, and blur of up to 1.25 pixels.

F. Experiments/Results/Discussion

F.1. Hyperparameters and optimization choices

We experimented with various different techniques for training the models. All the models except for Yolov5 were pretrained in batch sizes of 3, we started with training our models with batch sizes of 3, however we did decide to attempt to train the model with other batch sizes. In our experiments we saw that larger batch sizes empirically lead the models to have worse overall generalized, as the models tended to have increasing validation loss within the first few epochs. Through thorough experimentation with hyperparameters, we found that batch sizes of two produced the best results in the FRCNN models. For Yolov5 the larger batches sizes gave the numerous batch normalization layers a better sample so we used a batch size of 64 to get the most instances without over-stressing the GPU.

Given the limited time and resources, we first conducted our experiments with ADAM optimizer, as the Adam optimizer is able to adjust its own parameters while training, thus hypothetically making it more robust to choices in chosen hyper parameters. However in our small dataset, Adam proved to not work as well as we hoped. Performing grid search for hyperparamters, we saw common patterns of numeric instability and early stagnation in training. We hy-

pothesize that given more time and resources more carefully produced hyperparameters for ADAM would likely have produced better performance in our tests, however, SGD with momentum and scheduled learning decay generated the best results for our domain of application. For the two FRCNNs we used similar hyper parameters with a lr of .005, momentum coefficient of .9 and weight decay of .0005, YOLO slightly deviated from the FCRNN with a lr of .01.

F.2. Metrics

Our object detection models have two tasks, to identify the bounded regions of the box for an object, and additionally make a prediction of the label of the box. The Mean average precision is the most common way of evaluating the performance of both tasks of an object detection model.

For object detection models, we are not only evaluating the classification of abilities of the model itself, localization of the object must also evaluated. Therefore we need a sightly different evaluation metric. This evaluation metric is known as the mean Average Precision (mAP).

The general idea is that we need to combine a way to judge detection along with the classification, the way that the mean Average precision is able to calculate the correctness of a bounding box is known as the Intersection over Union (IoU), also known as the Jaccard Index. The (IoU) for predicted bounding box A and ground truth label is calculated using the following:

$$\text{IoU}(A, B) = \frac{\text{area}(A \cap B)}{\text{area}(A \cup B)} \tag{10}$$

Utilizing IoU values, we explore different ways of calculating True Positives, and False Positives. Given a certain value of IoU, we can consider it to be a True Positive. For example, if we set the threshold to be .5, if an example's IoU $> .5$, it is considered to be a True positive, otherwise, it is considered to be a false positive.

Precision is calculated as the ratio of true positives to the number of total guesses of positives. This can be evaluated as the following:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{11}$$

In this paper, we will be utilizing the same metrics as Microsoft's COCO challenge, therefore we will be calculating metrics from a range of 0.5 to .95 in step sizes of .05. Average precision AP of a class is then calculated by averaging the precision measurements within a class across all IoU thresholds. Then finally, mean average precision of n classes for classes k is calculated as the following:

$$\text{mAP} = \frac{1}{n} \sum_{k=1}^{k=n} AP_k \tag{12}$$

We also delve deeper into the the per-class classification accuracy, recall, precision, and f-1 scores. Accuracy is defined as the fraction of total predictions that the model gets correct. Recall is defined as the measurement of the proportion of actual positives that was identified correctly, precision is the same as defined above, and finally, f-1 scores is the the harmonic mean of both recall and precision. Mathematically, the metrics are defined as the following

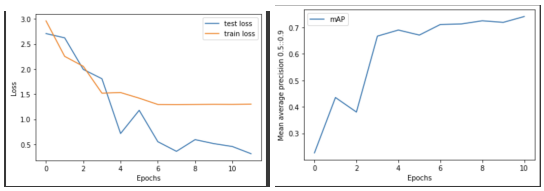
$$\text{accurracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{13}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{14}$$

$$\text{F1-mean} = \frac{2(\text{Precision} * \text{Recall})}{\text{Precision} + \text{Recall}} \tag{15}$$

F.3. Training Faster R-CNN models

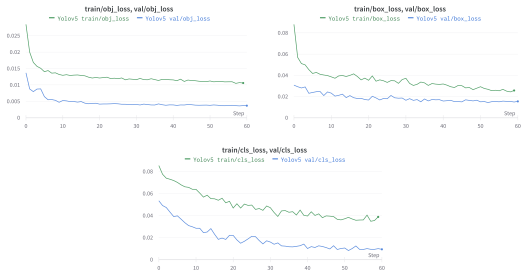
While training the models, we tracked the sum of all losses across the classifier loss, the objective loss, the box loss and the rpn box loss on the training set and test set. In order to avoid training models that over fit the training data, if there was an increase in test loss during evaluation, we would stop the training of the model early. In order to to ensure that this choice was a reasonable choice, we also tracked the mean average precision at each epoch. We observed that when early stopping happened, the values of test mAP had either started to stagnate or decrease, thus this provided us a sign that stopping the training early was a reasonable choice in order to avoid model over fitting to the training data.



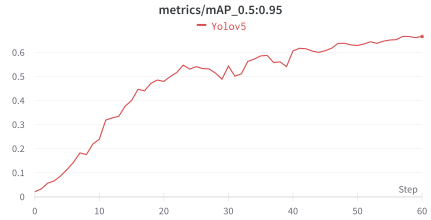
Above is the loss and mAP for the mobileNet FRCNN.

F.4. YOLOv5

During training we kept track of performance of the main metrics on the validation set as well as loss to make sure we weren't over fitting. First consider the loss graphs, which are split up into Objective Loss, Box Loss, and CLS Loss.



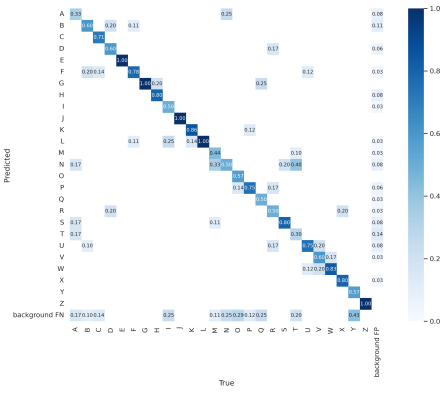
Here we see objective loss stabilized quickly while the other took a few more epochs. Additionally, we see that 60 epochs seems to be a good number of epochs to train for since all the losses have started to level out. Minor improvements could likely be made with another 60 epochs but it was not worth the additional computational time. Furthermore, we see that no overfitting has occurred and that the loss in validation set is often slightly lower. This is likely due to the many aggressive augmentation applied to the training images that help with generalization. Furthermore, during training we kept track of mAP@0.5:0.95.

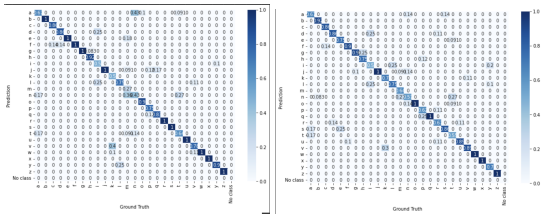


Through examining the mAP@0.5:0.95 over epochs we see that there is indeed a inverse correlation with the loss as expected. We also see that like the loss it starts to level out.

F.5. Evaluation

For test time evaluations we looked at the normalized confusion matrix, final object classification precision/recall, and final f1 scores all for each class.





Yolov5 (Top), ResNet FRCNN (Left), MobileNet FRCNN (Right)

For all three confusion matrices, shows that overall the models did well on the test. Most of the mass is on the diagonal indicating it is outputting the right class the majority of the time. When observing the confusion matrices, common patterns of misclassification tend to show. From the confusion matrices, we are able to identify and common classes that are often confused for one each other. For example We see that M and N are often confused with each other no matter which model we are testing it on. Notice that for all three models, we allow for the model to make the decision to classify detected images as non images, we allow this choice for when no labels are clear, with no scores past the threshold of .2. For both RCNNs this does not occur in the test set, however for YOLO, this does occasionally happen.

Model	Avrg Precision	Avrg Recall	F1 Score	mAP@0.5:0.95
Yolov5	0.65	0.737	0.717	0.501
ResNet FRCNN	0.84	0.83	0.82	0.735
MobileNet FRCNN	0.81	0.80	0.80	0.741

This table gives a the average scores the per class scores can be seen in Table 1,2, and 3 in the Appendix.

We can observe that ResNet Faster RCNN performs the best at the classification task. Across the board in average class precision, average class Recall and the average class F1 scores the baseline ResNet FRCNN outperforms the Yolov5 network and the MobileNet FRCNN. This does not extend to all cases when observing these metrics on a class to class basis (view the Appendix to see the metrics on a class to class level). We can also note that the mobileNet is able to achieve slightly better object detection mAP, beating out both Yolov5 and slightly edging out the ResNet FRCNN model.

Space/runtime eval	Parameters (in MB)	Mean CPU Inference Time(s)	Mean GPU Inference Time (s)
Faster RCNN ResNet-50	159.69	4.15	.1796
YOLOv5	27.9	.307	.0067
Faster RCNN mobileNet	72.89	.167	.0176

We also decided to focus on the computational costs of these models, we ca see that the FRCNN ResNet model was clearly more expensive to maintain and evaluate on. its parameters are over double the FRCNN, and its inference time was magnitudes higher than the other model's

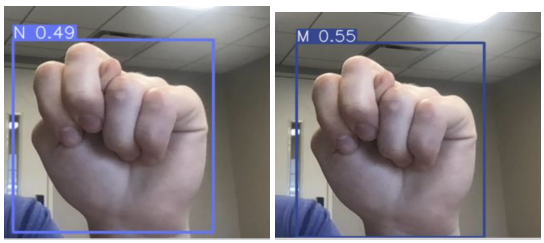
inference times on the CPU and on the GPU. The mobile FRCNN was able to clearly out perform YOLOv5 and the ResNet FRCNN on CPU infrence. Finally the YOLOv5 model shines in its low parameter count and its quick inferences on the GPU.

F.6. Qualitative Evaluations

In our examples, we generate saliency maps (Figures 5, 6, and 7 in the Appendix) to get an intuition on what underlying features in the images the model looks for when making the bounded box and classification. Through examination of the saliency maps it is clear that they are all focusing on the hands. However, looking even closer at the ResNet model we see that the digits on the hand can be made out. This extra detail seems to have translated to better performance as the ResNet had the highest classification scores.

Further delving into our data, we saw some examples that the data may have had some invalid ASL letters. For examples, in figure x, we observe a hand displaying the ASL letter y, however, the hand is facing away from the camera. This is not valid ASL, and furthermore, this mistake is unique to only this example in the dataset. Further issues in the data may have caused issues with learning, especially given the limited size of the datasets especially within each class.

When observing the ASL alphabet we can see many issues that seem reasonable, across all of the models, we see that the the highest inter class confusion in classification comes from classes that are very similar. For example, the letters A,E,M,N, S,and T are all very similar to each other, the letters are all closed fists, however the difference between the classes is the nuanced placement of where the thumb is in the fist. Especially given that the thumb may be completely obscured by the other fingers in some of the signs, the differences between some classes are truly nuanced and could easily confuse a human observer.



Both images are very similar but with a slight change the prediction changes with relatively high confidence. On the left we see that it correctly predicts N and on the left we see it incorrectly predict M indicates the model is still struggling to figure out what makes N and M fundamentally different.

One surprising result that we found was the ability for the models to correctly classify the letters J and Z in our demo. As these letters require motion to be represented one would believe that the model would not be able to make reasonable predictions, however our models, are all able to make reasonably correct guesses on these letters.

Another qualitative note is through observation working with the live translation the distance from the camera and towards the face/body appears to effect the prediction. If the hand is too far or too near the camera it has a harder classifying the sing correctly. Additionally, having your hand out away from your body and face marginally increases performance. This is likely due to the limitations of our data, as most images in the dataset are pictures of hands from the same relative distance.

G. Conclusion/Future Work

We can see that each of the models that we test have their own merits: the ResNet FRCNN performs the best in classification tasks obvious choice if processing time in not an issue. however on the negative side, it is infeasible to run live on a CPU and has very low frame rate on a GPU and was by far the largest model in terms of parameters. The mobileNet FRCNN architecture was able to produce the best object detection metrics while maintaining extremely fast performance on the CPU, however it has more parameters than Yolov5 and performs worse in classification in comparison to the ResNet FRCNN. Finally, Yolov5 has much less parameters than the others and is lightning fast on a GPU.

In the future, expanding the dataset would help the model generalize better. A dataset with a larger range of distances from the camera would help mitigate the problem of hands far from the camera being classified poorly. Additionally we believe a more expansive hyperparameter search should yield marginally better results.

The alphabet is just the beginning, to be able to fully achieve the goal of assisting hearing impaired individuals with communication and teaching ASL, words need to be identified and classified as well. However for now these results show preliminary results of the capabilities for these modern models to applied in order to create tools for the hearing impaired community.

In order to show our models in action, we created a web application to play with the mobileNet FRCNN and Yolov5 ALS models. You can view our live demo at http://cs231ndemo.rainjuhl.su.domains/

References

[1] American sign language letters dataset. https://public.roboflow.com/object-detection/american-sign-language-letters/1/augmentation. Accessed: 2022-02-06. 5

[2] Yolov5. https://github.com/ultralytics/yolov5, 2022. 3, 4, 9

[3] Hong-Yuan Mark Liao Alexey Bochkovskiy, Chien-Yao Wang. Yolov4: Optimal speed and accuracy of object detection, 2016. 2

[4] Tsung-Yi Lin Priya Goyal Ross Girshick Kaiming He Piotr Dollar. Focal loss for dense object detection, 2018. 2

[5] Joseph Redmon Ali Farhadi. Yolov3: An incremental improvement, 2018. 2

[6] Ross Girshick. Fast r-cnn, 2015. 1, 2

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. http://arxiv.org/abs/1512.03385, 2015. 2

[8] Evgeny Izutov. Asl recognition with metric-learning based lightweight network, 2020. 2

[9] Senior Member IEEE Jianfeng Wang, Xiaolin Hu. Convolutional neural networks with gated recurrent connections, 2021. 1, 2

[10] Ali Farhadi Joseph Redmon. Yolo9000: Better, faster, stronger, 2016. 2

[11] Ross Girshick Ali Farhadi Joseph Redmon, Santosh Divvala. You only look once: Unified, real-time object detection, 2016. 2

[12] matpotlib. 9

[13] Pillow. 9

[14] Pytorch. 9

[15] Trevor Darrell Jitendra Malik Ross Girshick, Jeff Donahue. Rich feature hierarchies for accurate object detection and semantic segmentation tech report (v5), 2014. 1

[16] Ross Girshick Shaoqing Ren, Kaiming He and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016. 1, 2

[17] MD. Eleas Ahme Tasnim Ferdous Dima. Using yolov5 algorithm to detect and recognize american sign language, 2021. 2

[18] Ross Girshick1 Kaiming He1 Bharath Hariharan-Serge Belongie Tsung-Yi Lin1, Piotr Dollar. Feature pyramid networks for object detection, 2017. 1, 2

[19] Dumitru Erhan Christian Szegedy Scott Reed Cheng-Yang Fu Alexander C. Berg Wei Liu, Dragomir Anguelov. Ssd: Single shot multibox detector, 2016. 2

[20] Member IEEE Yuhong Guo Zhengxia Zou, Zhenwei Shi and IEEE Jieping Ye, Senior Member. Object detection in 20 years: A survey, 2019. 1

H. Libraries Used

For this project we used Pytorch/Torchvision [14], yolov5 repository [2], matplotlib [12], and PIL [13].

I. Appendix

I.1. Table 1 (YOLO)

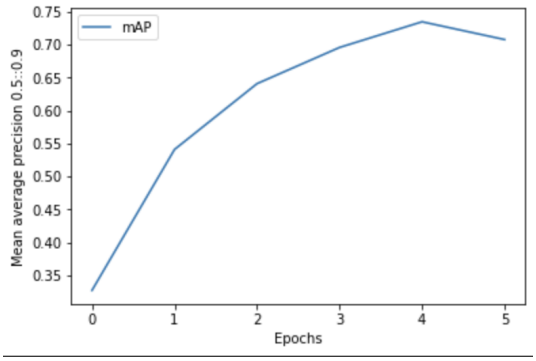
Letter	Precision	Recall	F1 Score
A	0.891	0.667	0.762
B	0.817	0.8	0.808
C	1	0.629	0.772
D	0.539	0.6	0.568
E	0.367	1	0.537
F	0.737	0.889	0.806
G	0.867	0.9	0.883
H	0.813	1	0.897
I	0.459	0.5	0.479
J	0.814	1	0.897
K	0.653	0.811	0.723
L	0.349	1	0.517
M	0.546	0.444	0.490
N	0.166	0.5	0.249
O	0.736	0.571	0.643
P	1	0.658	0.794
Q	0.442	0.5	0.469
R	0.371	0.5	0.426
S	0.542	1	0.703
T	0.752	0.607	0.672
U	0.556	0.75	0.639
V	0.643	0.8	0.712
W	0.379	0.5	0.431
X	0.911	1	0.953
Y	0.79	0.542	0.643
Z	0.738	1	0.849

I.2. Table 2 (FRCNN ResNet-50)

Letter	Precision	Recall	F1 Score
A	0.444	0.666	0.533
B	1	1	0.1
C	1	0.857	0.923
D	0.857	0.857	0.856
E	0.666	1	0.8
F	0.833	1	0.909
G	0.909	1	0.952
H	1	0.916	0.956
I	0.666	0.5	0.571
J	0.8	1	0.888
K	1	0.5	0.666
L	0.6	0.75	0.666
M	1	0.272	0.428
N	0.272	0.428	0.333
O	1	0.9	0.947
P	1	0.75	0.857
Q	0.833	0.833	0.833
R	1	1	1
S	1	1	1
T	0.7	0.636	0.666
U	1	0.777	1
V	0.636	1	0.7
W	0.8	1	0.888
X	1	1	1
Y	0.9	0.9	0.9
Z	1	1	1

I.3. mobileNetv3 FRCNN Table 3

Letter	Precision	Recall	F1 Score
A	0.67	67	0.67
B	1	0.92	0.96
C	1	0.86	0.92
D	0.75	0.86	0.8
E	0.75	0.75	0.75
F	0.9	0.9	0.9
G	0.75	0.9	0.82
H	0.9	0.75	0.82
I	0.4	0.5	0.44
J	0.80	1	0.89
K	0.88	0.7	0.78
L	0.6	0.75	0.67
M	1	0.64	0.78
N	0.36	0.57	0.44
O	0.83	1	0.91
P	0.83	0.63	0.71
Q	0.83	1	0.85
R	0.75	0.67	0.71
S	0.75	0.86	0.80
T	0.75	0.55	0.67
U	0.86	0.89	0.84
V	0.72	0.89	0.80
W	1	1	0.1
X	1	1	0.1
Y	1	0.7	0.82
Z	1	1	0.1



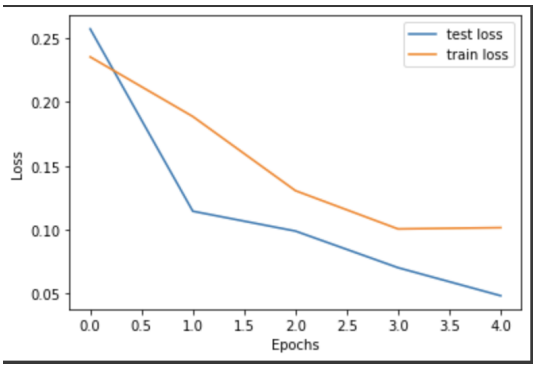


Figure 4. Training Metrics. Image 1.) ResNet-50 RCNN valid mAP per epoch, Image 2.) ResNet-50 summed Training/valid loss per Epoch

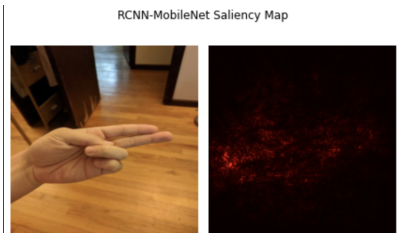


Figure 5. Saliency map for RCNN mobileNet saliency map.

Figure 6. Saliency map for for faster RCNN with Resnet51

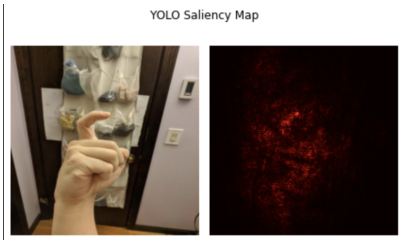


Figure 7. Saliency map for YOLOv5