American Sign Language Classifier

Alexander Roman, Sarah Padilla

University of California Merced

Professor: Xiaoyi Lu

Table of Contents

Abstract

Artificial intelligence (AI) is the field of computer science and engineering focused on the creation of intelligent machines that work and react like humans. Machine learning (ML) is a subfield of AI that involves the use of statistical techniques to enable computers to learn from data, without being explicitly programmed. In other words, ML algorithms enable computers to improve their performance on a specific task through experience, without human intervention. AI and ML have the potential to revolutionize many industries and in this case, AI/ML is leveraged in an Object Classification Task to translate common American Sign Language gestures in images. To achieve the aforemention task, a model was train using Faster-RCNN with a ResNet 50 backbone[1]. Faster RCNN (Regional Convolutional Neural Network) is a type of deep learning neural network architecture that is used for object detection tasks. It consists of two main components: a region proposal network (RPN) and a classifier. The former is responsible for finding a set of regions in image that are likely to contain the object while the latter helps separate the objects from the background to create bounding box coordinates for the objects. A ResNet 50 backbone is a pretrained deep convolutional neural network that serves as the underlying feature extractor for the Faster RCNN architecture. The ResNet 50 backbone is used to extract features from the input image that are fed into the RPN and classifier components of the Faster RCNN architecture. The model was trained on a custom labeled dataset on a GPU for a variety of epochs and datasets to improve the model - all while using a variety of performance metrics to assess the quality of the model.

[1] Due to dependency issues using Tensorflow, the Pytorch Framework was used to implement the architecture, training, and deployment of the model as it was deemed comparable to Tensorflow.

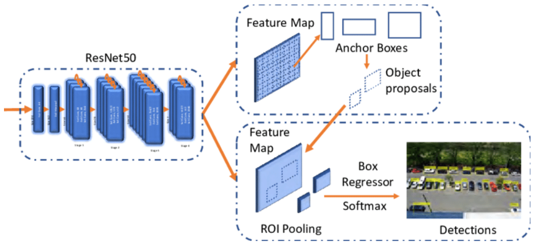
Knowledge Attained

There were three main areas in which key knowledge was attained to complete the task.

Data Collection

To collect suitable data for training, the team had to adhere to best data collection and labeling practices to ensure that ASL classification task would be plausible, and that the data was correctly labeled so the model could extract the best features during training. The data also had to be diverse enough to inference on never before seen images as well.

Training & Inferencing



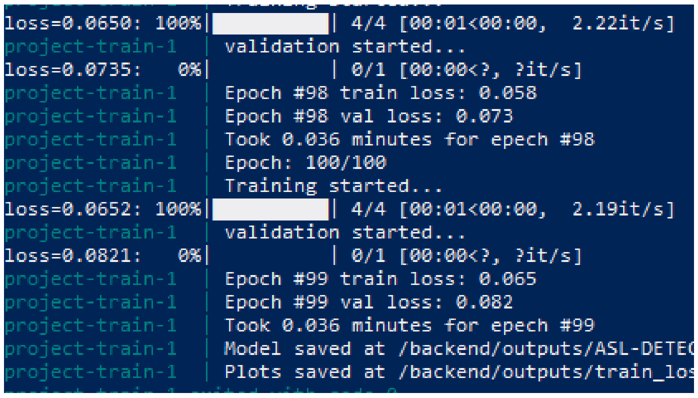
To properly train the model, knowledge about various DPP/CNN architectures was needed to choose the most suitable architectures for training. For our usecase the application had no inference performance limitations, and the best quality was desirable - which is why FRCNN was chosen as it tends to produce better quality results but at a higher latency when compared to faster architectures like SSD MobileNet(v) or Yolo. Knowledge of how to setup the training pipeline was needed to the model could be iterated upon and improved. An inference pipeline was also setup to ensure the model was deployable in a real world application.

Validation & Performance

To find out how the model was performing during/after each training session, knowledge of various performance metrics was needed. Training loss is an important metric to track during the training process because it can help to identify whether the model is learning effectively and whether it is overfitting or underfitting to the training data. If the training loss is too high, it may indicate that the model is not learning effectively and may need to be redesigned or trained for longer. On the other hand, if the training loss is too low, it may indicate that the model is overfitting to the training data and may not generalize well to unseen data. In addition to training loss, the validation loss (which is calculated using a validation set) and various performance metrics (such as precision, and recall). A high precision value indicates that the classifier is good at correctly identifying positive examples and is not prone to false positive predictions. A high recall value indicates that the classifier is good at identifying all of the positive examples in the dataset and is not prone to false negative predictions. These metrics can provide additional insight into the model's behavior and help to guide the training process. Intersection over Union (IOU) was also employed to determine how accurate predicted bounding boxes were.

Training

To train the model, data (images) had to be collected and classified first. To start three common ASL gestures were chosen: *“I love you”, “Thank you”, “Yes”.* Theses gestures were chosen because they only require 2 or less motions describe the phrases (gestures with more complicated gestures would require parsing video frames and labeling). Each team member took 5-10 selfies of gesture - ensuring the gesture hand, position, and angle were varied. To label the images, an open source software called LabelImg was used to created bounding boxes and labels for all the training images. The data was then exported into the original jpg files with a corresponding XML file. To feed the data into the training piple, the XML file was used to extract information about the ground truth and various transformation functions(blurs,transforms,etc) were applied to the original image to try to generalize the model. The first training set was with 15 images and 5 epochs to assess the effectiveness to the training pipeline. Then the epochs were increased to 100 better generalize the model. Finally, the training set was increased along with epochs after each validation set to attempt to correct issues in the model. During each training session the, training loss was measured to assess whether the model was correctly learning the features, or if it would overfit the model (this helped guide how many epochs to use). The model was set to a learning rate of 0.001, and weight decay of 0.0005.



```
loss=0.0650: 100%|██████████| 4/4 [00:01<00:00,  2.22it/s]
project-train-1  | validation started...
loss=0.0735:   0%|          | 0/1 [00:00<?, ?it/s]
project-train-1  | Epoch #98 train loss: 0.058
project-train-1  | Epoch #98 val loss: 0.073
project-train-1  | Took 0.036 minutes for epech #98
project-train-1  | Epoch: 100/100
project-train-1  | Training started...
loss=0.0652: 100%|██████████| 4/4 [00:01<00:00,  2.19it/s]
project-train-1  | validation started...
loss=0.0821:   0%|          | 0/1 [00:00<?, ?it/s]
project-train-1  | Epoch #99 train loss: 0.065
project-train-1  | Epoch #99 val loss: 0.082
project-train-1  | Took 0.036 minutes for epech #99
project-train-1  | Model saved at /backend/outputs/ASL-DETEC
project-train-1  | Plots saved at /backend/outputs/train_los
```

Inference

To perform inferencing, the a version of the model was saved every 2 epochs. The model with the best loss value for that training session was then used in the inferencing model. The dataset used during inferencing contained images that were excluded from the training dataset. Inferencing was also used to measure the performance of the model with various metrics, such as precision and recall. To deploy the model in common application like the web, mobile, or a backend service - the model would just need to be exported into the client's ML framework where FRCNN is employed.

Evaluation

To evaluate the efficacy and performance of the model various common metrics were used such as loss, precision, recall, and IOU.

Workload Section

The model was initially trained with 15 images, 5 epochs, and took about 100ms to train[2]. The second workload was with 15 images, 100 epochs. The third work load was with 30 images, 100 epochs. The last workload was with 30 images and 2,000 epochs.

Platform

The underlying system was a windows device with an Intel i7-4790k cpu, 16gb of ram, and a nvidia RTX 3060 12gb vram. The training, and inferencing pipelines were all contained in a docker container with an Ubuntu OS. The docker container was configured to use nvidia cuda to speed up training the the gpu cuda cores. The docker container was also test on a windows device with no gpu (dedicated), and a M1 Macbook with no gpu.

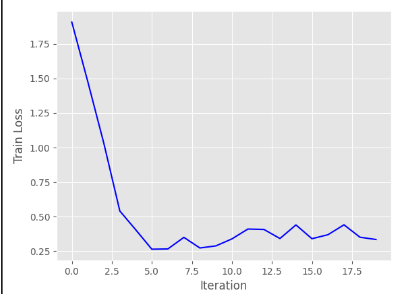
Metrics

As mentioned before, training loss was measured to see how the model was learning during training. Precision was measured on new data to see how the model could perform on positive examples while recall was measured to see if the model was prone to false negatives. Finally, intersection over union was used to see how accurate the model was in location of a classified object.

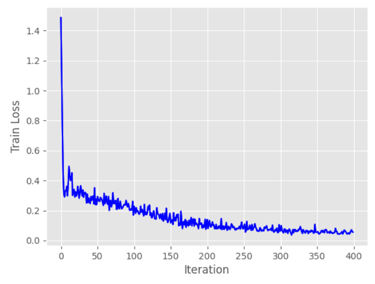
Results

Loss

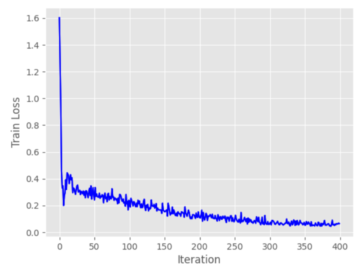
[2] All workloads were done with the 3 classifications: “i love you”, “thank you”, “yes”



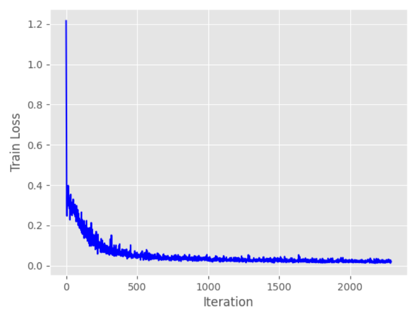
- Workload 1



- Workload 2



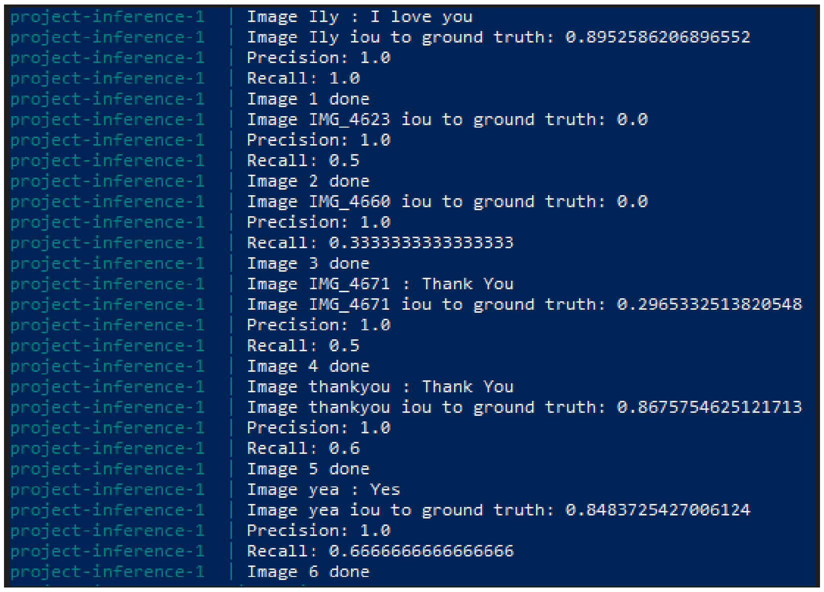
- Workload 3



- Workload 4

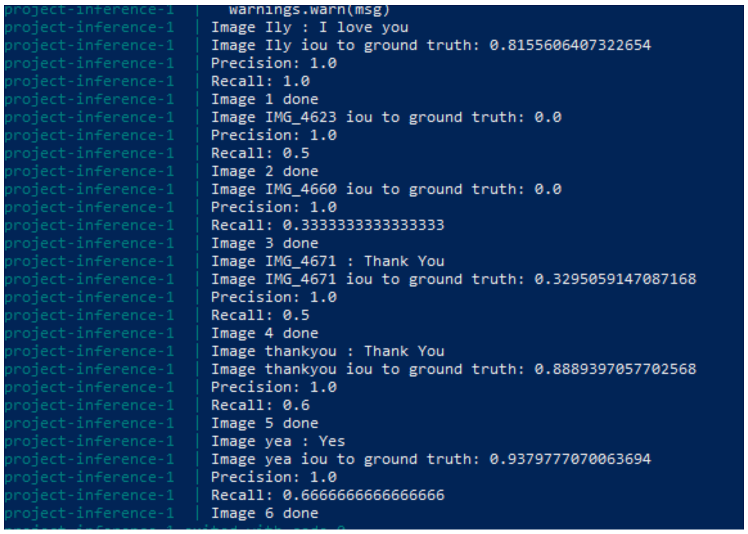
The workloads above each show the training loss. Workload 4 should have continued for 500 epochs, but the loss suggested it was no longer learning at around 500 epochs, and it started to overfit.

Precision, Recall, and IOU



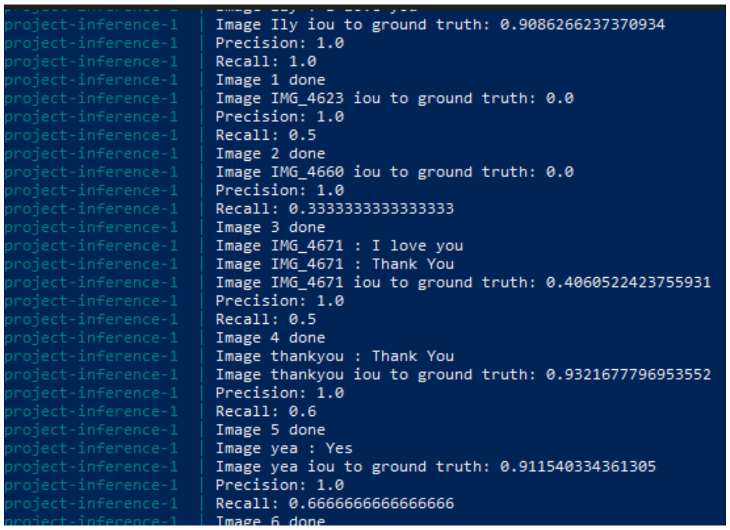
```
project-inference-1  | Image Ily : I love you
project-inference-1  | Image Ily iou to ground truth: 0.8952586206896552
project-inference-1  | Precision: 1.0
project-inference-1  | Recall: 1.0
project-inference-1  | Image 1 done
project-inference-1  | Image IMG_4623 iou to ground truth: 0.0
project-inference-1  | Precision: 1.0
project-inference-1  | Recall: 0.5
project-inference-1  | Image 2 done
project-inference-1  | Image IMG_4660 iou to ground truth: 0.0
project-inference-1  | Precision: 1.0
project-inference-1  | Recall: 0.3333333333333333
project-inference-1  | Image 3 done
project-inference-1  | Image IMG_4671 : Thank You
project-inference-1  | Image IMG_4671 iou to ground truth: 0.2965332513820548
project-inference-1  | Precision: 1.0
project-inference-1  | Recall: 0.5
project-inference-1  | Image 4 done
project-inference-1  | Image thankyou : Thank You
project-inference-1  | Image thankyou iou to ground truth: 0.8675754625121713
project-inference-1  | Precision: 1.0
project-inference-1  | Recall: 0.6
project-inference-1  | Image 5 done
project-inference-1  | Image yea : Yes
project-inference-1  | Image yea iou to ground truth: 0.8483725427006124
project-inference-1  | Precision: 1.0
project-inference-1  | Recall: 0.6666666666666666
project-inference-1  | Image 6 done
```

- Workload 2 -



```
project-inference-1  |   warnings.warn(msg)
project-inference-1  | Image Ily : I love you
project-inference-1  | Image Ily iou to ground truth: 0.8155606407322654
project-inference-1  | Precision: 1.0
project-inference-1  | Recall: 1.0
project-inference-1  | Image 1 done
project-inference-1  | Image IMG_4623 iou to ground truth: 0.0
project-inference-1  | Precision: 1.0
project-inference-1  | Recall: 0.5
project-inference-1  | Image 2 done
project-inference-1  | Image IMG_4660 iou to ground truth: 0.0
project-inference-1  | Precision: 1.0
project-inference-1  | Recall: 0.3333333333333333
project-inference-1  | Image 3 done
project-inference-1  | Image IMG_4671 : Thank You
project-inference-1  | Image IMG_4671 iou to ground truth: 0.3295059147087168
project-inference-1  | Precision: 1.0
project-inference-1  | Recall: 0.5
project-inference-1  | Image 4 done
project-inference-1  | Image thankyou : Thank You
project-inference-1  | Image thankyou iou to ground truth: 0.8889397057702568
project-inference-1  | Precision: 1.0
project-inference-1  | Recall: 0.6
project-inference-1  | Image 5 done
project-inference-1  | Image yea : Yes
project-inference-1  | Image yea iou to ground truth: 0.9379777070063694
project-inference-1  | Precision: 1.0
project-inference-1  | Recall: 0.6666666666666666
project-inference-1  | Image 6 done
```

- workload 3



```
project-inference-1  | Image Ily iou to ground truth: 0.9086266237370934
project-inference-1  | Precision: 1.0
project-inference-1  | Recall: 1.0
project-inference-1  | Image 1 done
project-inference-1  | Image IMG_4623 iou to ground truth: 0.0
project-inference-1  | Precision: 1.0
project-inference-1  | Recall: 0.5
project-inference-1  | Image 2 done
project-inference-1  | Image IMG_4660 iou to ground truth: 0.0
project-inference-1  | Precision: 1.0
project-inference-1  | Recall: 0.3333333333333333
project-inference-1  | Image 3 done
project-inference-1  | Image IMG_4671 : I love you
project-inference-1  | Image IMG_4671 : Thank You
project-inference-1  | Image IMG_4671 iou to ground truth: 0.4060522423755931
project-inference-1  | Precision: 1.0
project-inference-1  | Recall: 0.5
project-inference-1  | Image 4 done
project-inference-1  | Image thankyou : Thank You
project-inference-1  | Image thankyou iou to ground truth: 0.9321677796953552
project-inference-1  | Precision: 1.0
project-inference-1  | Recall: 0.6
project-inference-1  | Image 5 done
project-inference-1  | Image yea : Yes
project-inference-1  | Image yea iou to ground truth: 0.911540334361305
project-inference-1  | Precision: 1.0
project-inference-1  | Recall: 0.6666666666666666
project-inference-1  | Image 6 done
```

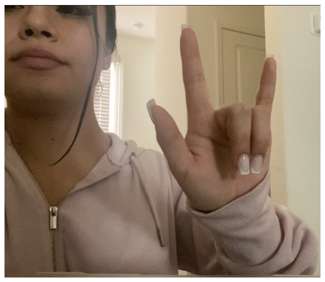
- workload 4

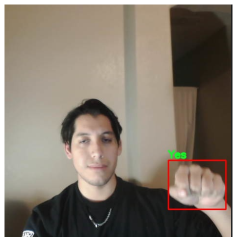
The metrics above do not include workload one because it failed to classify any test image.

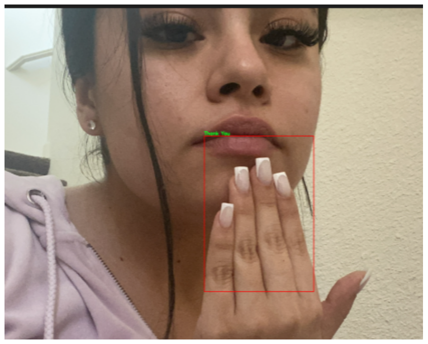
Observations

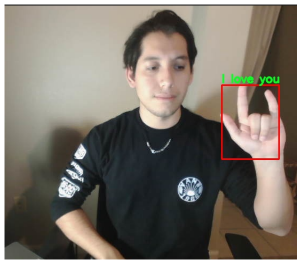
With the loss at each workload - one can see that the model continues to learn in workloads 1 - 3. The precision (value of 1.0) indicates that the model is good at detecting the correct sign language gestures when they are in the image. Still, the recall suggests that the model is failing to detect any sign language gesture even when some relevant language gesture is present. In workload 4 the epochs start to level out after epoch 500 which suggests that the model was overfitting and no longer learning. The final workload precision and recall suggest that the model needs more and more diverse data to improve the recall. One upside of the final workload is that it improved the IOU scores across all test images meaning that it was better to predict where the gesture was located in the picture.

The final images show a sample of the validation images. One can see that the model was able to classify gestures but completely missed a gesture in one image.









Conclusion

This entire process details how to collect the data, train, inference, and evaluate a model for ASL gesture classification using an FRCNN. After running 4 workloads with various dataset counts, and epochs - it was concluded by the recall of 0.66 that the model failed to detect gestures at all when it was supposed. This means that the model is not generalizable to a wide variety of angles, lighting, distances, etc. To attempt to fix this issue - the model would need much more data with more diverse data points as well. However, the model does prove that the ASL classification task is doable.

References

Acharya, D. (n.d.). *The architecture of Faster-Rcnn containing a ResNet50 backbone.* Parking Occupancy Detection and Slot Delineation Using Deep Learning: A Tutorial. Retrieved from https://www.researchgate.net/figure/The-architecture-of-Faster-RCNN-containing-a-ResNet50-backbone_fig8_355576303.

Rath, S. R. R. S. R., O., … *, N. (2022, September 25). *Custom object detection using Pytorch Faster RCNN*. DebuggerCafe. Retrieved December 16, 2022, from https://debuggercafe.com/custom-object-detection-using-pytorch-faster-rcnn/

Brownlee, J. (2020, August 1). *How to calculate precision, recall, and F-measure for imbalanced classification*. MachineLearningMastery.com. Retrieved December 16, 2022, from https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/#:~:text=Consider%20a%20model%20that%20predicts,95%20%2F%20(95%20%2B%2055)