Final Project Report for CS 175, Spring 2018

Project Title: Hand Detection

Project Number: 38

Name	Student ID	Email
Ling Jin	22588158	lingj6@uci.edu
Jiahao Yao	91945818	jiahy@uci.edu
Lizhen Lin	82226790	lizhenl@uci.edu

1. Introduction and Problem Statement

Object detection is an important topic in computer vision and image processing. With recent advancements in deep learning object detection is much easier than ever before. Object detection can be applied in many areas such as people counting for data collection and visual surveillance systems. In order to better understand object detection we decided to work on hand detection for our project.

Hand detection is just one type of object detection which has the goal of detecting the location of all hands in an image. In other words, we take as input a still image of a human with their hands visible in some fashion and as an output draw an accurate and precise rectangle around the hands that appear in the image. Our motivation for this project is that having an efficient and accurate hand detector algorithm would be incredibly useful for a variety of further projects. Humans, via gestures, can communicate a lot of information and for a computer to be able to understand these gestures the first necessary step is an algorithm to locate and isolate the human hand.

2. Related Work:

The Haar Cascade training method which was proposed by Paul Viola and Michael Jones was one of first feasible methods for detecting hands effectively in real time images. This method was then used by groups like Ramadijanti and his team as part of a full process. Ramadijanti used web cameras to capture images of hands, the Haar Cascade Training method to detect the hands present in these images and then used the Kalman Filter to predict the finger positions in these images. The success of this hand detection is based on many factors such as the training of positive and negative data as well as the sheer amount of data used for training. However, because this method relies on having many small negative files, the program is forced to keep reading all small files directly from disk. As a result, the Haar Cascade training is very slow during its learning phase.

Therefore, instead of using the above training method in our project, we decided to use the faster R-CNN and SSD to train our model. The recent success of region proposal methods and region-based convolutional neural networks (R-CNNs) have greatly improved object detection. Combining the deep fully convolutional network and fast R-CNN detector, the network becomes single and unified. We also used SSD (Single Shot Detector) to train our models. By running convolutional network on input images only once to get a feature map, SSD maintains a balance between accuracy and speed.

Reference:

1.Mourad, B., Tarik, A., Karim, A., & Pascal, E. (2014). Real-Time System of Hand Detection And Gesture Recognition In Cyber Presence Interactive System For E-Learning. *Bousaaid Mourad et al Int. Journal of Engineering Research and Applications, 2248-9622, Vol 4, Issue 9 (Version 1), September 2014, pp.*
2.Sutoyo, R., Prayoga, B., Fifilia, Suryani, D & Shodiq, M. (2015). The Implementation of Hand Detection and Recognition to Help Presentation Processes. *International Conference on Computer Science and Computational Intelligence (ICCSCI 2015)*.

3.Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine intelligence,* 39(6), 1137-1149.

3. Data Sets

We used the EgoHands dataset from Indiana University. (Bambach, Sven, et al. "Lending a hand: Detecting hands and recognizing activities in complex egocentric interactions." Proceedings of the IEEE International Conference on Computer Vision. 2015.) This dataset contains 48 Google Glass videos of complex, first-person interactions between two people. This dataset has two size groups. The larger one consists of 48 videos and 2700 labeled frames per video. The smaller one consists of 48 videos and 100 labeled frames per video. We chose the smaller one for our project since it was easier for us to use for our model and reduced the time needed to train our model.

All frames were in JPEG format (720*1280px). Other information such as ground-truth labels consisted of pixel-level masks for each hand-type were stored in Matlab files. We chose this dataset for three main reasons. Firstly, all labelled images were high quality and pixel-level segmentations of hands. This feature made the images more clear and thus making it easier to train our model. Secondly, since the people in the pictures were playing games such as chess and cards, we had information that allowed our model to distinguish between several peoples' hands, as well as identifying left and right hands. Furthermore, the different angles of the hands and variety of hand poses in the pictures also increased the variety of our dataset. Third and finally, the dataset is large and contains lots of ground-truth labeled hands. This feature makes the dataset more reliable, comprehensive and practical.

Figure 1. Sample images in EgoHands Dataset

4. Description of Technical Approach

Faster R-CNN: Faster r-cnn has two networks: one is the region proposal network (RPN) for generating proposals and the other is a cnn using the proposal generated from RPN to detect objects. The RPN ranks the boxes/anchors and proposes the ones most likely containing a hand. The anchor box configuration we have for our model is 9 anchor boxes (3 boxes with scale 128x128 256x256 512x512, 3 aspect ratios 1:1 2:1 1:2) at each of the locations. We choose one position with a stride of 16. Then we use the RPN to predict which of these anchor boxes have the highest probability of having a hand in it. RPN outputs many anchor boxes that are then examined by our classifier and regressor to check for the occurrence of a hand.

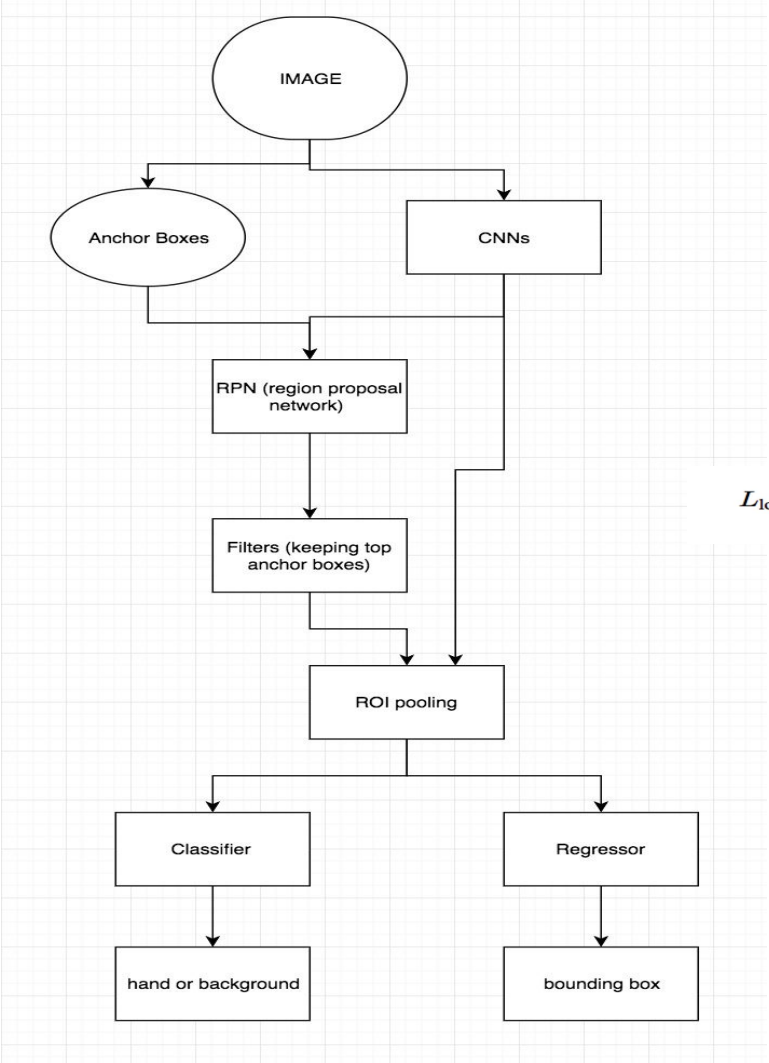

Figure 2: Faster R-CNN pipeline

For our classifier, since our data is already labeled and preprocessed we compared the ground-truth box using softmax classification. For our regressor, since our preprocessed data did not have a ground-truth box for them we run it through a smooth-L1 loss function with the top-left (x,y) position of the box and the log of the heights and widths.

$$L_{\text{loc}}(t^u, v) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i^u - v_i),$$

After RPN we get a lot of proposed regions (boxes) with different sizes, which would be hard for us to work with later. We then used a method called region of interest (ROI) pooling to reduce the boxes into the same size. ROI pooling splits the input into 8 roughly equal regions and then we applied max-pooling to every region.

SSD: First we used an RPN to generate anchor boxes. Then we used a fully connected layer to classify those anchors. We did these two operations simultaneously predicting the bounding box and classifying them as a hand. In detail, we first pass the input into a series of convolutional layers to get the proposed

region, then for each location of these regions we used a 3x3 filter to evaluate a small set of bounding boxes. For each box we predict the bounding box offset and the probabilities of it being a hand. We match the ground truth box with predicted boxes base on the Jaccard index (intersection over union) to find the boxes with probabilities over 0.5. Then we only keep the boxes with the highest probability and best fitting for the hand, and we use the negative examples with the highest loss to train the next iteration.

5. Software

Number	Software Name	Functionality	Source
1	egohands_dataset_clean.py	The input is all the images used to train from different directories. This script adds the directory name before the original name of the images. Since images from different directories may have the same name, this action will avoid duplication and make its easier to train later. Furthermore, it also converts the matlab file which contains label information such as x/y coordinates into a csv file which can be used later in the training process.	Victordibia's github handtracking
2	generate_tfrecord.py	The input is a csv file which contains information of every training image and the output is a tfrecord file which can be used directly in the training process.	Dat Tran's github
3	Tools from Tensorflow Object Detection API	● Label_map_util.py: The input is some label maps of images. The output are images with "Hand" labels. This script loads a label map, checks its validity, and then converts label map into "Hand" category. ● Visualization_util.py: The input is a testing image without any information. This script does not return a value. It only modifies the image itself and performs some visualization on the image. The image after modifying will have boxes which predict the position of hands. ● Eval.py: The input is test_tfrecord which are data about the true position of hands in test dataset. This	Tensorflow Object Detection API from github

		script will compare the true boxes and predicted boxes to check the accuracy of our models. ● Export_inference_graph.py: This script is a tool to export an object detection model for inference. This tool prepares an object detection tensorflow graph for inference using model configuration and an optional trained checkpoint. It outputs an inference graph, associated checkpoint files, a frozen inference graph and a model.	
4	Train.py	Take pipeline.config as input files. Modified to fit our own training model	● Tensorflow Object Detection API from github and ● own writing
5	project.ipynb	Loads the trained model and images to detect hands within the image.	● Own
6	real_time_hand.py	Loads the trained model and use internal webcams to capture frames. Then detects if hands are within the frame.	● Own

6. Experiments and Evaluation

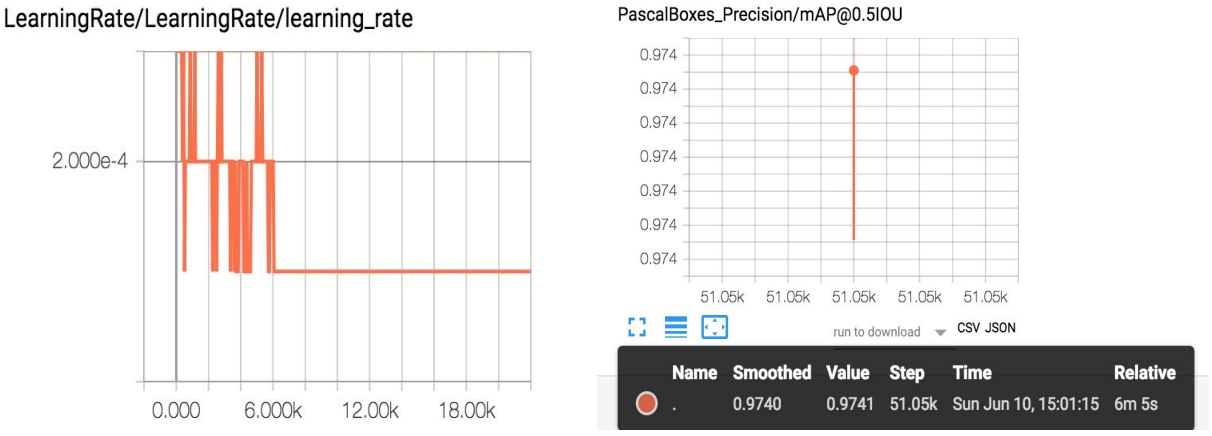

Figure 3. Faster R-CNN Learning Rate/Precision

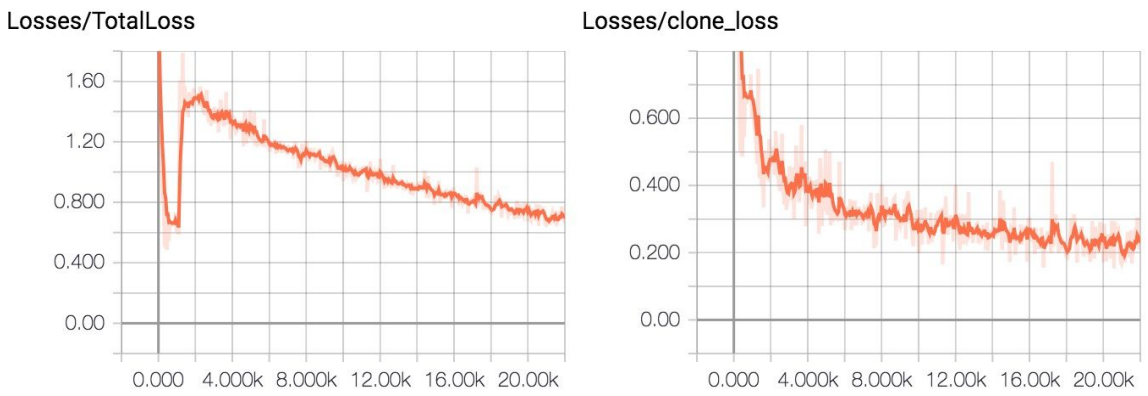

Figure 4. Faster R-CNN Loss

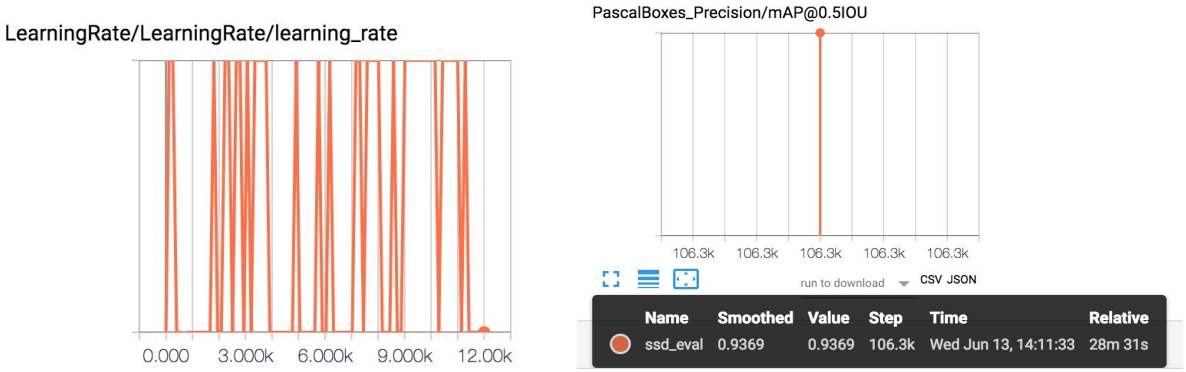

Figure 5. SSD Learning Rate

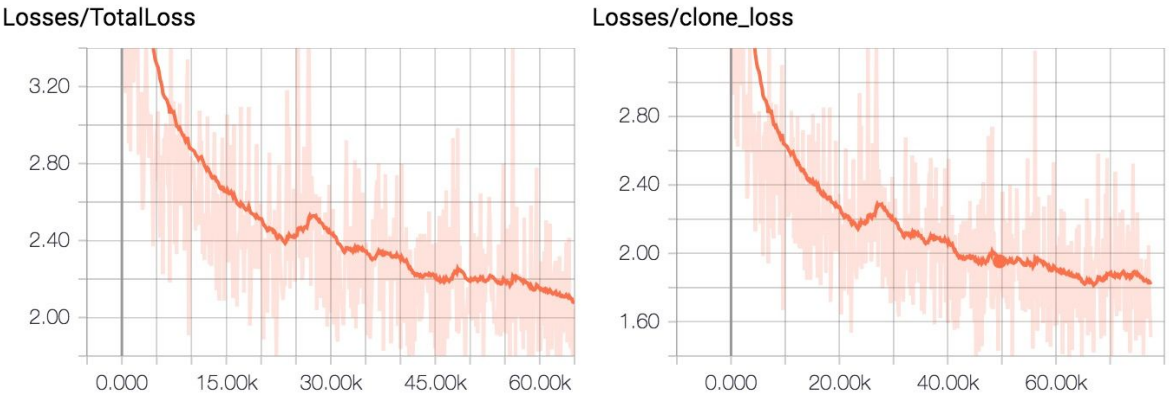

Figure 6. SSD Loss

First we went online trying to see what type of data was available. Based on our initial research we decided to use the Ego Hands dataset. Again this was mainly because the images have higher quality, pixel-level segmentation of hands and already have all the related data saved in a matlab file format. Overall it was very easy and convenient to use. Initially, we tried to write the script to load all the matlab file ourselves but due to an unfortunate lack of the matlab knowledge we decided to use the script

egohands_dataset_clean.py from Victordibia's github. This handled all the downloading, extracting, renaming and loading of the matlab files. After converting the data into csv files, we learned from from another script called generate_tfrecord.py how to add labels to each aspect of the data and also how to add all the boxes' coordinates as a array. However, we did not use all the labels from the original scripts, we only kept the filename, image(encoded) xmins[], xmaxs[], ymins[], ymax[], and classes. Since we already know the image format and class_text. After this we were able to start loading the tfrecord into our jupyter notebook.

We evaluate our method with the 20% validation data. On the graphs above it shows that faster rcnn with 20,000 steps with total loss about 0.7 and the validation accuracy 0.97. On the ssd model with 100,000 steps with total loss about 2.1 and the validation accuracy 0.93. We learned that training faster rcnn is much slower than ssd but results in a higher accuracy.

7. Discussion and Conclusion

From this project, we now know how to use Faster R-CNN and SSD to train our models and do hand detection. We also got a firsthand look at the tradeoff between Faster R-CNN and SSD in terms of speed and accuracy. We first expected the speeds of the two models to be the same. However, we later found the model with the SSD method does the training process quicker than Faster R-CNN does. We spent 12 hours training 100,000 steps with SSD but needed 24 hours to train only 20,000 steps with Faster R-CNN. In other words, our model with SSD spends 0.75s in each step and our model with Faster R-CNN spends 2s in each step. This is because SSD does not have back propagation but Faster R-CNN has both back and forward propagation. However, our model with Faster R-CNN is more accurate than our model with SSD.

Our current model may still have some problems when the hands are overlapping or in dark background environments. This is because the hands in our training dataset were all separated and the images were all good quality and bright. If we were in charge of a research lab, we would collect more images which contained overlapping hands and hands in darker environments for our next training dataset. Also, we would increase the size of the input training dataset to improve the accuracy of our model. Furthermore, since hand detection is only a small part in object detection, we would also like to extend the topic to object differentiation and be able to detect different types of objects. For instance, given a set of images containing planes, ships and hands, we would like our model to differentiate the type of object and detect the position of each type of object in the image.

Appendix

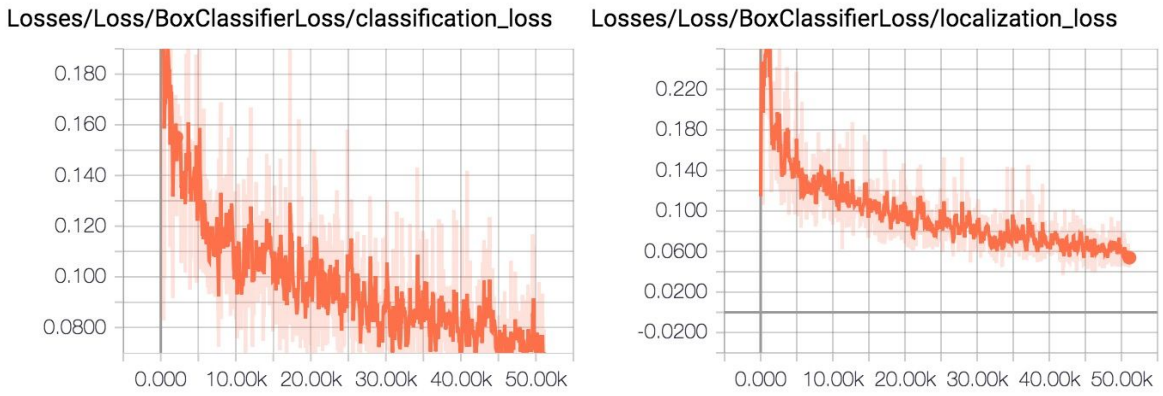

Figure 7. Anchor box classifier loss

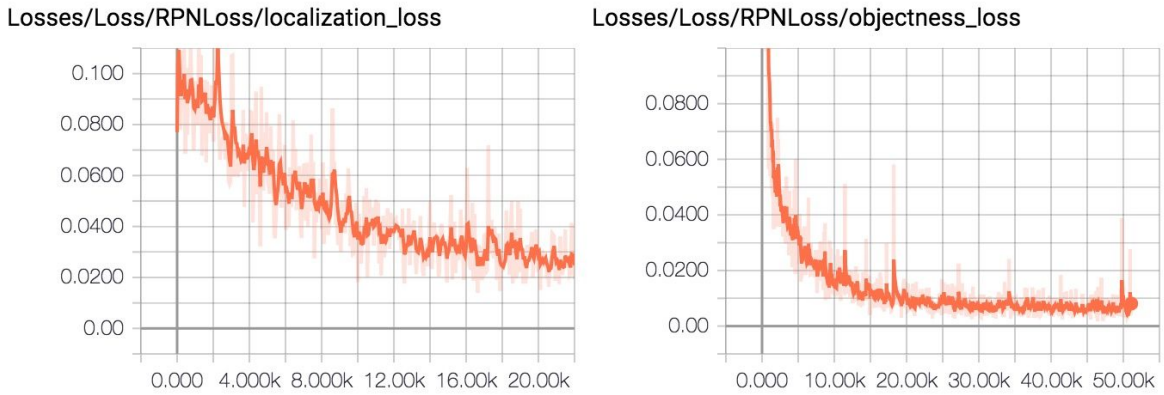

Figure 8. RPN loss

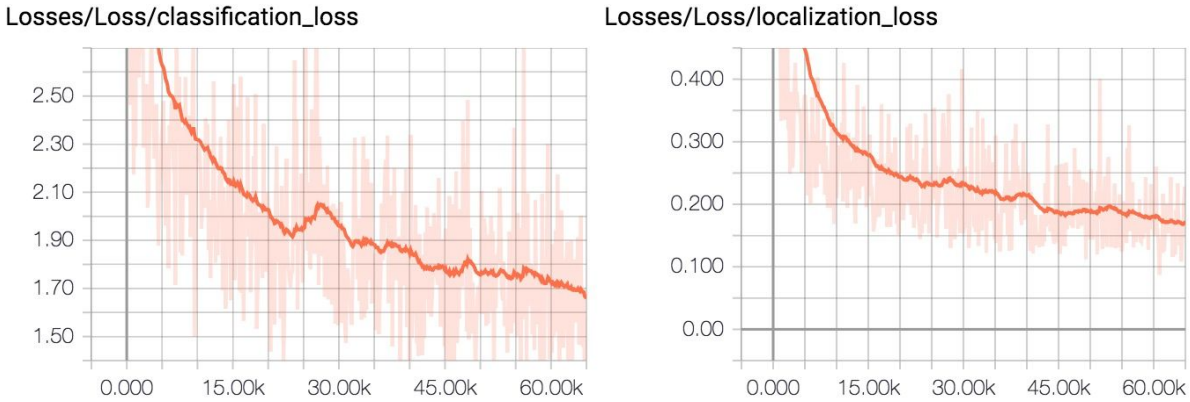

Figure 9. SSD classification