Final Project Report for CS 175, Spring 2018

Project Title: Hand Detection
Project Number: Group 8

Student Name(s)
Jiliang Ni, 81224742, jiliangn@uci.edu
Chuyi Wang, 71535393, chuyiw@uci.edu

1. Introduction and Problem Statement (1 or 2 paragraphs):

In this paper, we will show our efforts on detecting hand positions. We mainly used TensorFlow Object Detection API to pursue our goal. To find the "hands" in the picture, we first resize the data, transfer the data to TFRecords format, then use TensorFlow to train a model, finally test the model on new images and draw the bounding boxes around the hands in the images.

2. Related works (1 or 2 paragraphs):

At the beginning, we tried to use CV2 and YOLO to achieve this project. However, we found the environment settings of Windows are not suitable for YOLO. Then, we began to do other research and we found that many other researchers have great contributions in hand detection area. Most topics related to our topic are hand detection and pose/gesture detection in pictures or videos. Finally, we found the TensorFlow Object Detection API. We watched the related tutorials of TensorFlow Object Detection API on YouTube.

What's more, there are several famous datasets for hand that are used by other researches, like Egohand dataset, Oxford hand dataset. Victor Dibia, an HCI researcher, used the Egohand dataset to train and build a real-time hand-detection with neural network TensorFlow. His process of building such a detection model has three steps: transferring the datasets into TensorFlow format, training a hand-detection model called 'ssd_mobilenet_vi_coco' and using this detector to track hands.

We also evaluated several object-detector libraries: darkflow, TensorFlow object detection API. The darkflow is a real-time object detection and classification library. It uses YOLO (in darknet) with different weight files to detect objects.

However, we finally decided to use TensorFlow object detection API, because there are a lot of good models in model zoo.

3. Datasets:

We used the dataset collected from the CS 175. This dataset collected the videos of all the students in this class. The videos are transferred into single pictures. There are 21 joint position data for each single hand. Each finger has 4 joint points and each hand has one wrist point same as figure 1 below. The

dataset downloaded from piazza have two main parts. The “color” directory contains image frames and the “annotation.json” file contains the image name as the key and the 21 coordinates of joint points as the value. The joint points of hand image is shown below (Figure 1).

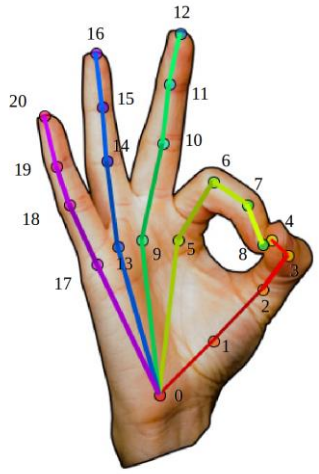


Figure 1. The joint points of hand image

4. Description of Technical Approach：

Our original plan was very different from what we did later. The first plan is to use techniques used in previous assignments, like create a five layer of convolution network model by ourselves. However, the process of implementation didn’t go well. So, we decided to use sources from existing libraries.

Then we tried to use the YOLO to deal with the problem. But we finally found the environment settings for YOLO cannot be built well in Windows system.

Finally, we found that the TensorFlow Object Detection API is a good tool for dealing with this project. Then we watched three tutorials on YouTube (https://www.youtube.com/watch?v=COlbP62-B-U&t=76s). After learning how to use this API, we first resized our raw data from 1920*1080 to 320*180 (we did not include these images in the zip file, because they are too large). Then we wrote a python script to transfer the resized data to TFRecords format, which is needed for the API.

After preprocessing the data, we downloaded a TensorFlow RCNN configuration from model zoo. Then we kept changing the parameters and layers in the configuration to find which combination of parameters is the best one.

The RCNN is an object detection algorithm proposed by Ross Girshick in 2015. It builds on previous work to efficiently classify object proposals using deep convolutional networks. We compared the convolutional networks which is used in our assignment 4 and the RCNN. We finally found that the RCNN was more efficiently and the most difference between them is the CNN in our assignment 4 can only deal with classification problems. But the RCNN can deal with problems like object detection, which can predict many values at same time.

The first configuration we used is faster_rcnn_nas_lowproposals_coco.config. The coco means "Common Object in Context". It has a lot of models for machining learning. But after use this configuration to build model, we found it was not really good based on the performance.

Therefore, we tried another configuration file, which is faster_rcnn_resnet50_coco.config. We changed the batch_size to 1, which made the training process faster. Although it cannot guarantee the loss can be reduced for each iteration, it indeed made the model faster than before and improved its performance. And this time, we changed the num_examples to 8000, which was 2000 before. Because it can run faster than before, so we can input more examples. Indeed, it improved the performance well. I will introduce the performance later.

For the final evaluation, we split the data to 4 folds and used the first 3 folds as the training data and the rest one fold as validation data. We also tried the cross-validation, but that ran so slowly. So, we did not include that part in our project.

4. Software:[at least ½ a page]

We used the Anaconda3 to create an virtual environment and then we downloaded the TensorFlow library. We also used the Jupyter which was included in Anaconda. We also used the cv2 to read the images. The most import tool we used is the TensorFlow Object Detection API, which is from github. The link is https://github.com/tensorflow/models/tree/master/research/object_detection (which is also shown in the reference part at the end of paper).

We also used Google Drive to communicate with each other and wrote the report in the Google Drive. For labeling the images, we used the labeling tool to deal with the images.

We also used the Python3 to write scripts to preprocess the data, like resizing the images and transferrin the raw data to TFRecords format.

For the model configuration file, we used the model zoo to choose the configuration which is most suitable for our project.

5. Experiments and Evaluation:

First, we wrote python script to compute the coordinates of the bounding boxes of each image using the 21 joint points given in the annotation.json file.

Then, we used cv2 to read the images and resized them. Then, we transferred the data to TFRecords. The code is in the "Get_TFRecord.py" file, which is in the models-master\research directory. We set the new size as width 320 and height 180. Because the original size is 1920 * 1080, we multiplied 0.16666667 to the original size to get 320 * 180.

Then we loaded the annotation.json file. For each key in annotation.json which represents the name of an image, we extracted the value of it, which is the combination of coordinates of bounding box. Then we made the image as X and the value corresponding to it as Y value.

Then we used tf.train.Example to save the transferred data as TrainingData.record and ValidationData.record in the directory of "object_detection/TrainingData".

After preprocessing the data, we downloaded the configuration file faster_rcnn_nas_lowproposals_coco.config and tried to use this configuration file to make a new model. However, after many times changing the parameters of the model, we still did not get a good result. Then, we downloaded another configuration file from model zoo, which is faster_rcnn_resnet50_coco.config.

This time, when changed the batch size to 1, we found it improved the speed of training process. Therefore, we changed the num_examples from 2000 to 8000, because it is possible for our computers to train the model with more data.

Then, we found the model improved a lot and it's average accuracy is about 80%, which is much better than the model produced by the first configuration file (that one only has 50% accuracy).

My photo shows below and there is 92% and 98% around the bounding boxes, which means that the object inside of the box is really likely a hand.

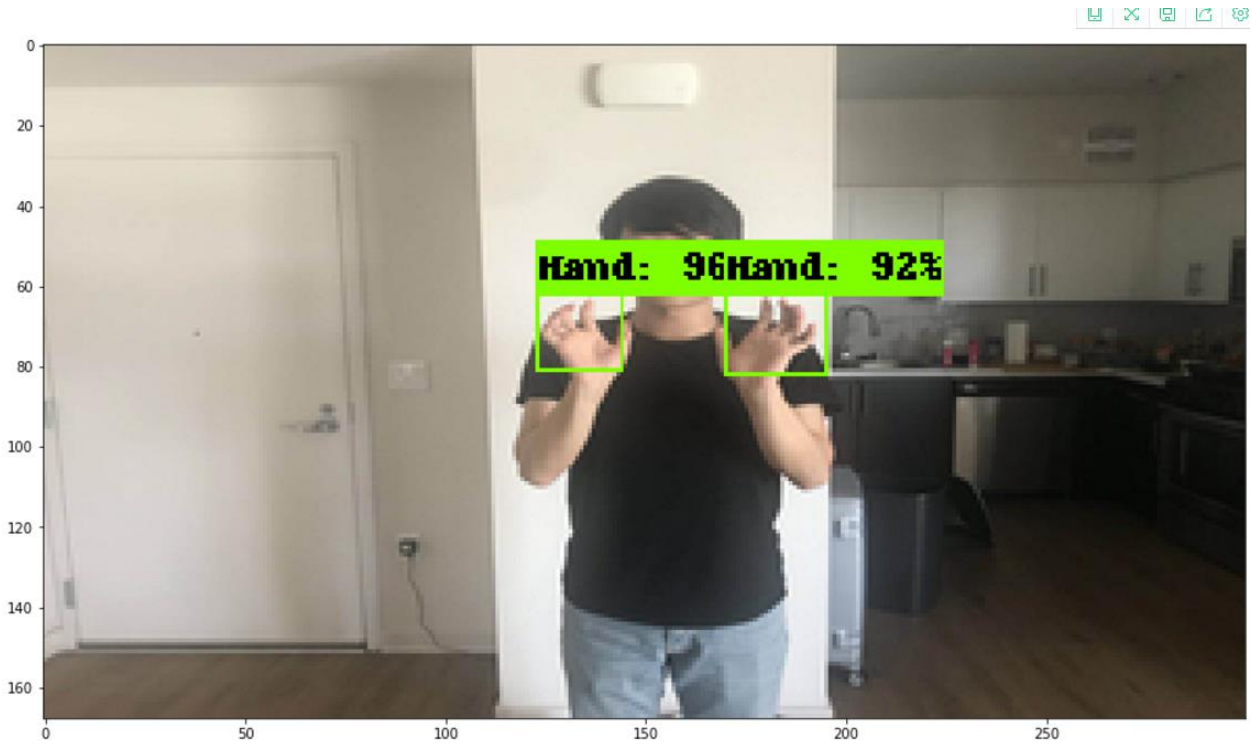
During the process of changing parameters, we found that the batch size is not necessary a big number. Although sometimes a big batch size will make the loss always reduce, a small batch size can increase the speed of training process, which makes it is possible for our computers to train the model with more data and finally increase the accuracy.

We also changed the num_steps to study what effect it made on the performance. And we found it would affect the speed of training process a lot but had a little effect on the final performance of the model.

For the methods about the evaluation, we split the data to 4 folds. And we used the first 3 folds as training data and the rest one fold as validation data. We also tried to use the cross-validation method, but it ran so slowly. So, we just used that for learning purpose. We did not include that in our report.

6. Discussion and Conclusion: [at least ½ a page]

Finally, we got 80% accuracy for our RCNN model. And as shown in the project.ipnb file, the model can predict hands in an image with a really high confidence. Generally, it can successfully predicts hands with about more than 90% confidence.



Through this project, we found that RCNN is really better than CNN we used in our assignment 4. It is a higher version of CNN. It is not limited to deal with the classification problems but it can also solve some problems like object detection, which means it can predict many values for one input data.

Also, we thought it must be better when the batch size for gradient descent process is higher. However, after finishing this project, we found that in practice, there is a trade-off between accuracy and speed. If we choose a smaller batch size, then it will train faster. Therefore, we can give more data to train the model, which results the increase of the accuracy. So, the real case in practice may be really different from the knowledge we learned from classes.

References:

https://towardsdatascience.com/how-to-build-a-real-time-hand-detector-using-neural-networks-ssd-on-tensorflow-d6bac0e4b2ce

https://www.youtube.com/watch?v=COIbP62-B-U&t=0s&list=PLQVvvaa0QuDcNK5GeCQnxYnSSaar2tpku&index=2

https://github.com/tensorflow/models/tree/master/research/object_detection

https://github.com/victordibia/handtracking

https://arxiv.org/pdf/1506.02640.pdf