CS 230 Project Final Report: Object detection using regression-type model vs. faster region-convolutional neural networks

Alfred Cheung (kccheung), Yuetao Lv (lvy), and Yang Zhao (yangz16)

Project Category: Computer Vision

1 Introduction

Autonomous driving is one of the hottest applications of artificial intelligence and deep learning. While different companies each stress a different approach to achieving fully unattended autonomous driving equivalent to a human driver in all respects – some may stress more camera-based approaches while others rely more on sensory technology such as lidar and radar – *an essential problem to overcome is still to be able to detect various objects given an image or video. Obviously, this is necessary for detecting pedestrians, other cars, etc. when the autonomous vehicle is on the streets!*

While modern convolutional neural network architectures have been very successful at this task of single object detection, in autonomous driving, it not sufficient to just be able to detect an object accurately. In addition to accuracy, *speed* is also a major factor. One can imagine an autonomous vehicle collecting live footage of its surroundings constantly as it moves. It is necessary to process this live stream of frames and detect objects in them efficiently.

One algorithm aimed at overcoming this challenge is called faster region-convolutional neural network [1] (faster R-CNN), which falls under the category of region-based methods. These methods rely on sampling only a relatively small subset of proposed regions out of all possible regions in an input image to detect potential objects. Then, on each region, an image classification using a CNN architecture is performed (with a softmax output layer) to determine what object is in that region (cat, dog, person, etc.). This approach is what is meant by "region" CNN and it has the benefits of: (1) being able to detect many objects in a given image, and (2) being able to perform the detection task accurately.

The problem with region-based methods though is that it can be very slow to perform image classification on all the possible sub-regions within an input image. Further improvements are then made in how the proposed regions are selected. This is the basis of the faster R-CNN algorithm. It was briefly discussed in the course. We will further discuss how it works in Section 4.

Exactly how much faster is faster R-CNN? **The goal of this project is to compare the speed of this algorithm to simple regression-type object detection baseline algorithms.** Regression-based baselines frame the object detection task as a regression problem – the goal is to predict the x and y coordinates of the center of an object as *regression variables*. These baselines involve no region-proposal steps and can be very fast.

2 Data and Computational Resources

2.1 Data

Originally, we planned to film our own videos of moving objects. However, after discussing with our TA, we were encouraged to look for existing datasets online. Indeed, we found a very good dataset. The best part is that all images in this source are already labeled with the correct bounding box for the object being tracked. This dataset can be found at: *https://www.kaggle.com/kmader/videoobjecttracking/version/1?*.

The "true labels" for each image is given as four numbers, the x and y coordinates of the top left corner of the bounding box (x_{TL}, y_{TL}), and the x and y coordinates of the bottom right corner of the bounding box (x_{BR}, y_{BR}). By taking the average of these two coordinates, we can find the center (x_{center}, y_{center}) of where the object is located:

$$x_{center} = \frac{x_{TL} + x_{BR}}{2}$$
$$y_{center} = \frac{y_{TL} + y_{BR}}{2}. \tag{1}$$

We will focus on one particular dataset which shows a panda moving around in its enclosure. We choose this to start off with because: (1) there is only one object (i.e. the panda) being tracked; (2) the panda moves relatively slowly; and (3) the background does not change. In the original dataset, there are 3000 images. We performed data augmentation by reflecting each image about a vertical axis at the center of each image. Thus, we have a total of 6000 panda images.

In addition to this data, for the faster R-CNN, we will use an existing model that is pre-trained on data from the PASCAL Visual Object Classes Challenge 2007. Thus, we are implicitly using data from this dataset (even though we did not train it ourselves!).

2.2 Computational Resources

Originally, we used our personal computers and CPUs to train and test on models. However, we quickly found that once the architecture becomes complex (the VGG-16 model for example contains of on the order of 10 million trainable parameters), using CPUs become unfeasible given the limited time we have. Even after using transfer learning to fix parameters, it would have taken days to train on the remaining 10000 parameters.

To solve this problem, we set up virtual machines on Google Cloud with each machine using a NVIDIA Tesla K80 GPU. This was found to be critical in completing our project. The same task which took several days on CPUs took only around half an hour running on the GPU.

3 Regression-Based Baseline Models

3.1 Baseline Model 1: Fully-Connected Neural Network

As a first baseline model, we have implemented a fully connected neural network to track the positions of the panda as a regression problem. The architecture is very simple as we did not want to spend too much time optimizing a lot of weights of a baseline model. The key steps in our baseline model are:

1. Flatten input image into a one dimensional array. I.e. if the original input image shape is $(n_x, n_y, 3)$, the resulting flattened input shape is $(n_x \times n_y \times 3, 1)$.
2. First fully connected hidden layer with 128 neurons using relu activation. The result is batch normalized.
3. Second fully connected hidden layer with 64 neurons using relu activation. The result is batch normalized.
4. Final output layer with two neurons. These represent the predicted center coordinates for the object. They are also passed into a relu activation.

The evaluation metric we use is a squared loss defined as:

$$Loss = \frac{1}{m} \sum_{i=1}^{m} (y_{true} - y_{predicted})^2, \tag{2}$$

where y_{true} and $y_{predicted}$ are the true and predicted center coordinates for a given object. Hence, they are both of length 2. m is the number of examples being averaged over for the loss.

3.2 Baseline Model 2: CNN with Transfer Learning

Considering the fact that the data set has limited number of frames to train a large neural network, as well as the nature of our task that is to localize objects in a given frame, we tried to use VGG-16, a well-established CNN for classification tasks to conduct transfer learning [2]. The weights are pre-trained from the data set "ImageNet".

The way we conduct transfer learning is to use the output from the second-last fully connected layer of VGG-16 as encoded features of the input image. We added a 2-neuron fully connected layer there to predict the center of pandas in the input image.

As for the evaluation metric, we adopted the same loss function as baseline model I, as shown in Eq. 2.

3.3 Performance of the baseline models

We trained the baseline models on 5500 training examples and evaluated on 500 testing images. The training examples are further divided into training set(99%) and dev set(1%). After iterating through enough epochs, the training loss could decrease to around 500, while the testing loss still hung around 5000. Results from baseline model II is shown in Fig. 1. As we could see, the predictions from baseline model II biased a lot from

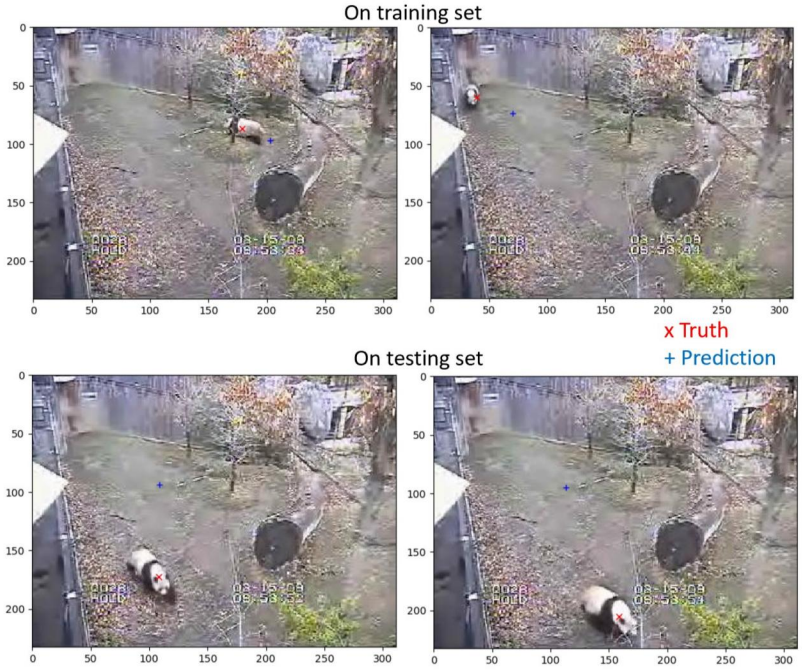

Figure 1: The top two images are from the training data while the bottom two images are from the testing data using baseline model 2. Clearly, the predictions are much more accurate for the training images but are much worse for the testing images. This is reflected in the very large loss we obtained for the testing data set, suggesting overfitting of our simple baseline model.

the actual center of the panda. We tried to feed the images to the pre-trained VGG-16 model, and the model could identify the images as pandas. As far as we're concerned, CNN is good at performing classification tasks. However, in terms of object detection, encoded features of the whole image from CNN might not be adequate to localize positions of the objects tracked in the image. From this conclusion, we realized the *importance of region proposal* in object detection models such as YOLO and Fast/Faster RCNN.

4 Faster R-CNN

4.1 How it works

We start by discussing briefly the main ideas behind faster R-CNN. A main time bottleneck of region-proposal based approaches to object detection is in proposing a good set of regions that might have objects in them. Each of these proposed regions must then be fed into a classifier to determine if there is an object in it. The classifier itself tends to be a large, complex CNN that can extract relevant features for object classification. Thus, there are *two* computationally heavy parts: (1) a region proposal computation, and (2) a feature extractor/classification computation.

The central premise behind the faster R-CNN algorithm is then the following: have a convolutional neural network which is able to perform both tasks (1) and (2) simulataneously! This "dual purpose" network is called a Region Proposal Network (RPN). Essentially, it is a CNN which produces as output key features from an input image. The same features are then used to make region proposals *and* to use in the classification task for each proposed region. By having a single RPN which accomplishes both of these tasks, time can be saved. A schematic diagram of the method is shown in Fig. 2 taken from Ref. [1].

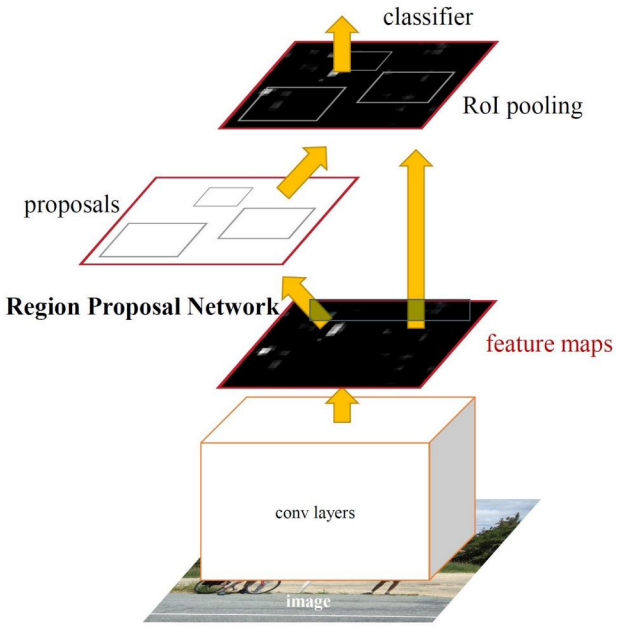

Figure 2: Taken from Ref. [1]. The key main idea behind the faster R-CNN algorithm is to use a single CNN, called the region proposal network, which generates features that can be used for proposing region and in classifying objects within the regions.

4.2 Results

The faster R-CNN implementation we used was obtained from an open source code available on Github that was developed by the original inventors of faster R-CNN. The image classification architecture they use is a VGG-16 model trained on the PASCAL Visual Object Classes Challenge 2007. The link to the repository is: *https://github.com/rbgirshick/py-faster-rcnn.*

Originally, we planned to test on the same set of panda images as for our baseline models. However, we encountered a major issue: the pre-trained faster R-CNN model was not trained on pandas and thus did not have panda as a classification category. We could not train a new model in time. For this reason, we regrettably could not make a direct comparison with the baselines using the panda images. We will instead predict on some dog images and comment on the speed of the faster R-CNN implementation as compared to the baselines.

In Fig. 3, we show a typical result of the faster R-CNN algorithm when asked to predict the presence and bounding boxes of dogs. 203 regions were proposed and tested for the presence of a dog. A proposed region is said to have a dog if the probability for dog is greater than 0.8. The pre-trained model is able to successfully predict both dogs. We performed the same task 100 times and averaged the prediction time over those trials. On average, the prediction task took 0.31 seconds. For regression-based the baseline model II, the prediction task took 0.011 seconds.

At first glance, it appears that the baseline is much faster than the faster R-CNN. However, we must remember that for a single input image, faster R-CNN proposes many regions (in Fig. 3, 203 regions) and does classification on each of them. Thus, 0.31 seconds represents the total time taken to make the proposals and classify all potential objects in each one, at the same time achieving very high accuracy. Through this comparison, we see how good the performance of faster R-CNN is in both speed and accuracy.

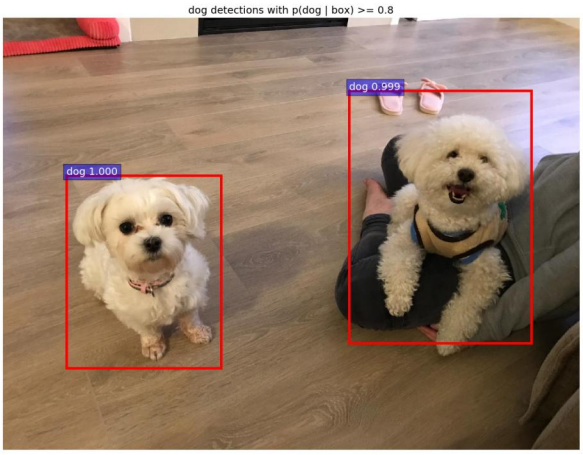

Figure 3: Prediction result using the pre-trained VGG-16 faster R-CNN model obtained from Github. The prediction task took an average of 0.31 seconds when averaged over 100 trials. (The dog on the right belongs to one of the authors)

The link to our GitHub repository is: *https://github.com/acheung623/CS_230_Project.git.*

5 Contributions

All authors contributed equally to this work, from selecting the project, to carrying out the calculations, to writing the reports and making the poster.

References

[1] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[2] How to use the pre-trained vgg model to classify objects in photographs. https://machinelearningmastery.com/use-pre-trained-vgg-model-classify-objects-photographs/. Accessed: 2019-03-01.