Team Members - Daniel Griffin, Yudhister Satija

766 CV Deep Object Detection Project Proposal

Contents

Team Members - Daniel Griffin, Yudhister Satija

Midterm Report

Why we decided to change our proposal

Challenges we faced with the original proposal

Our original plan was to try to implement the networks from the paper ‘A Unified Multi-scale Deep CNN for Fast Object Detection’. However, we came to the conclusion that there were simply too many challenges to try to overcome in a reasonable amount of time. Some of these issues include the problem that the network might require too much time to implement or possibly even train (given we could implement it) within the given time restrictions. The paper also has many very particular nuances about how data is labeled, how data is sampled (special objectness bootstrapping), how the network is initialized, how the network is trained through multiple phases, separate stages of parameter modification etc. Some other aspects of overall complexity included the proposal network complexity, prediction network complexity, sampling methods, data set weighting and importance methods, hand tuned SGD, hand tuned weight transitions, unclear intermediate image scaling, deconvolution layers, etc.

We came to the conclusion that the paper's methods are so fine tuned that the resulting network is almost certainly overfitted to the data set used by the paper, and any extensions (if any could even be made) would have small value. Many papers in the deep net object detection field in fact seem to be this way, and indicate a somewhat “directionless” notion of general research that favors overfitting models specific to a specific task, or data set that likely don’t generalize well. This realization, in part, helped lead us to our updated proposal described in the next section.

Updated proposal

Problem?

How well do current cutting edge deep neural networks work in terms of accuracy, time to train, time to predict, kind of image, and sensitivity to noise? How statistically different are the most representative kinds of deep neural network object detection models in terms of performance? How do the deep network object detection models differ, how does this contribute to performance, and are there any rules, guidelines, or insights that can be used in general for developing deep neural network based object detection models? These are just a few important questions that require substantial analysis and benchmarking of existing deep neural network based object detection models.

We seek to compare and bench-mark several representative techniques, each of which may not be super-optimized. In doing so, we seek to gain some intuition about the relative limitations / advantages of different methods in terms of performance, robustness, and in dealing with different kinds of images.

Team Members - Daniel Griffin, Yudhister Satija

Importance?

While some simple benchmarking has been previously performed, there exists a need to thoroughly benchmark and explore today's existing cutting edge object detection models to understand how they differ, potential shortcomings, and how new models may be designed that improve upon issues that exist with today's models. While it is known that deep neural networks produce some of the highest detection performance, it is not yet fully understood if there are good rules or guidelines for designing modern detection networks, and the full capabilities or drawbacks of existing cutting edge detection networks are only understood at a cursory level. Deep analysis of today's cutting edge object detection neural networks is necessary to understand where improvements should be made for future research, which methods should be applied for different scenarios, when current methods perform within their suggested performance bounds, and the potentially unstated drawbacks of current methods.

State-of-the-art?

We aim to benchmark some of the most commonly used, modern deep neural networks including Mask R-CNN , RetinaNet, Faster R-CNN, RPN, Fast R-CNN, and R-FCN. Some of the pre-trained backbone networks we are looking to use include ResNeXt{50,101,152}, ResNet{50,101,152}, Feature Pyramid Networks, and VGG16. The datasets we aim to use include the COCO dataset, PASCAL VOC dataset, and the Kitti object detection dataset. These network architecture designs, and network backbones represent some of the most common cutting edge object detection deep neural networks that exist today. We hope to also be able to implement and test the YOLO (You only look once) network structure if we have time, as it is a network proposed to focus more on speedy predictions for near-real-time applications.

In terms of actual analysis, not much work has been performed into analysis of deep neural network models other than basic average performance metrics of individual models. Most work focuses more on proposing a new architecture for a deep network object detection model, rather than comparing its significant difference with existing methods, major fundamental drawbacks, and other ways in which the network can be fooled or broken. In terms of robustness and sensitivity to noise, even less work on deep network based object detection models has been performed.

Existing system or New approach?

We plan on benchmarking existing neural network designs and implementations that are representative of current deep neural network object detection based methods, with one particular goal being a proposal and outline of future research work that should be developed to address some of the issues we find. This might fall under the category of a "New approach". However, we also plan on performing deeper analysis of existing methods than has been previously performed to provide for a better overall understanding of how well modern methods compare, how similar and different the methods are, which approaches seem to work the best, and what shortcoming exist with existing methods.

Evaluation and Results plan?

Our project proposal centers around benchmarking and analyzing different deep object detection networks, so the evaluation phase is really the meat of our project (aside from getting everything up and running). The details of what we plan on benchmarking, what platforms we plan to use, what data sets we plan to use, and what metrics we plan to calculate are given in the “Plan for the rest of the semester” section below.

Current progress

Current results.

- Running on Condor:
    - Condor standard submit: We’ve set up all of the scripts and code necessary for running on the standard condor instances.
    - Condor Docker based submit: We’ve set up all of the scripts and code necessary for running on the docker based condor instances.
    - Condor Nvidia Docker GPU submit: We’ve set up all of the scripts and code necessary for running on the nvidia-docker GPU based condor instances.
- Running on AWS:
    - EC2 GPU instances: We’ve developed a process for provisioning AWS EC2 p2.xlarge GPU instances, along with everything that goes with that (VPC nets/subnets, security configs, S3 block storage instances, data center limits provisioning, etc…)
    - Docker everything: we’ve dockerized all of our code to run on the GPU based instances. (You can build on a non-GPU based instance, but running the deep nets requires GPUs since the tensor operators are implemented in terms of GPU operations)
- Running initial object detection network
    - Network Architecture, backbone: The object detection results provided below are for a ‘Mask R-CNN’ model with a ‘ResNet-101-FPN’ network initialization.
    - Images we’ve run on:
        - We’ve run the above specified network on some basic, annotated images and generated some resulting images, which we show below.

Team Members - Daniel Griffin, Yudhister Satija

Challenges.

- Refactoring our project and proposal.
    - So we had to rapidly change our direction when we realized the tedious complexity of our original project proposal, and search for a new potential project. We had to review lots of object detection papers and methods to come up with a new project to work on. Once we had an idea that we wanted to try to perform a set of benchmarking, we had to find candidate models, platforms, frameworks, datasets, and metrics that we thought we could achieveably use for our project. Some of the challenges of these are found in the next subsections.
- Running on Condor GPUs and getting access to resources.
    - Most of today's existing deep neural network, object detection based models require GPUs (A major drawback which I believe will eventually be remedied with FPGAs due to their high speed, low power nature which makes them a defacto choice for most industry scale, long term use). So, we talked to the University of Wisconsin's HPC group about running on the GPU based condor instances. We spent a good amount of time writing the submission and monitoring scripts for the GPU instances, and in particular the docker based GPU instances to simplify code submission. However, there are only 2 machines in HPC with 4 total GPUs that currently run docker, and the queue is so congested that using condor for our project became clearly impractical (I submitted a GPU based docker job about a week ago that still hasn't been run yet). I've talked with the condor HPC

staff, and they are working on getting docker running on the other 2 older instances (each with 6-8 GPUs each) that don't have Docker (But we can't use these without docker, as the nature of the code we are trying to run on the machines requires installation and system configuration permissions).

- Running on AWS, and the cost associated with it.
    - Since we couldn't run on the condor instances, we decided to turn to Amazon Web Services (AWS) to try to provision instances. After a few days of working with AWS EC2 instances, we finally developed a process for provisioning AWS GPU instance resources for running our nvidia-docker based code, and running some of our initial deep net object detection models (Many of which need GPUs for their operators, and up to 15GB of dedicated system memory). The downside with this is that **AWS GPU instances are $1/HR** to provision and use which is quite expensive for compute resources (And why we are trying to avoid training models that can each take anywhere from hours to days to train).
- Dataset non-uniformity.
    - While there exist some different datasets for benchmarking object detection, most data sets have different formats. We have decided to try to standardize to the COCO dataset API, and to try to convert some of the other data sets we would like to use to the COCO annotation specifications. This means that we will likely have to write scripts for generating COCO annotated images from each data set.

Plan for the rest of the semester.

Outline of goals and Project Specifics.

Below, we outline the specifics of what we will try to implement in terms of the deep net architectures, datasets, platforms, and metrics.

Insights, Design Guidelines, and Further Research Avenues:

1. Benchmark each network architecture and backbone structure individually for performance, resource, and network sensitivity metrics.
2. Compare benchmarks of each network and structure against each other for statistical significance.
3. Outline insights and attempt to develop general recipe-book object recognition network algorithm design for different scenarios, and propose new research directions.

Deep Net Object Detectors:

1. Deep Net Architectures:
    a. Mask R-CNN, RetinaNet, Faster R-CNN, RPN, Fast R-CNN, and R-FCN.
2. Deep Net Pre-Trained Networks:
    a. ResNeXt{50,101,152}, ResNet{50,101,152}, Feature Pyramid Networks, and VGG16.

3. Platforms for Running Code:
    a. AWS p2.xlarge GPU instances (Ubuntu 16.04 LTS images, Maybe DeepNet Images)
    b. Docker
        i. Vanilla Docker and Nividia-Docker
    c. Condor GPU instances (If HPC gets more docker instances up maybe)
4. Frameworks for Implementing Code:
    a. Caffe2, Tensorflow/Keras, Python+Libs

Datasets to Use:

1. Datasets:
    a. COCO http://cocodataset.org/#download
    b. PASCAL VOC http://host.robots.ox.ac.uk/pascal/VOC/
    c. Kitti http://www.cvlibs.net/datasets/kitti/eval_object.php
    d. Hand Annotated Coco Image Dataset https://github.com/visipedia/annotation_tools (To create our own images to break models)

Metrics to Track:

1. Accuracy Based Metrics
    a. Cross-Validated Precision, Recall, F1, Accuracy, Mean, Std Dev, PR Curves/Cost Sensitive Classification Analysis, Mean Average Precision, Intersection Over Union
    b. Pure performance metrics as shown above, but on an *untouched* held aside test set (which most analysis fails to do properly).
    c. T-Test between models to check for statistically significant differences in each of the metrics.
2. Resource Based Metrics
    a. Time to train (Maybe take this from other works since it takes a long time)
    b. Time to perform inference per image, per-object
    c. Memory usage of model
    d. Histogram of image performances
        i. Best Image Class Performance
        ii. Worst Image Class Performance
3. Noise Sensitivity
    a. Add gaussian and random noise to every image and compare change in performance.
    b. Add all noises until an image changes classification, or classification confidence gap drops below certain threshold or ratio.
    c. Sensitivity to orientation, scale, lighting conditions/ exposure (pixel intensity insensitivity)
    d. Links related to above sensitivity testing suggestions: (http://hera.inf-cv.uni-jena.de:6680/pdf/Rodner16_FRN.pdf, https://arxiv.org/pdf/1604.04004.pdf, https://en.wikipedia.org/wiki/Sensitivity_analysis)

Team Members - Daniel Griffin, Yudhister Satija

Further Notes and References

Possible Deep Net Detection Links:

1. https://github.com/kjw0612/awesome-deep-vision
2. https://github.com/Smorodov/Deep-learning-object-detection-links.
3. https://github.com/facebookresearch/Detectron
4. https://handong1587.github.io/deep_learning/2015/10/09/object-detection.html

Architecture Links:

1. https://github.com/caffe2/caffe2
2. http://chtc.cs.wisc.edu/approach.shtml, http://chtc.cs.wisc.edu/HPCuseguide.shtml

Data Sources:

1. Coco dataset
2. PASCAL VOC dataset
3. Kitti object detection data set (**No one has done this yet for coco based models**)
4. Hand Annotated Coco Image Dataset https://github.com/visipedia/annotation_tools (**Create our own images to break models**)

Metrics To Track:

1. Accuracy Based Metrics
    a. Cross-Validated Precision, Recall, F1, Accuracy, Mean, Std Dev, PR Curves/Cost Sensitive Classification Analysis, Mean Average Precision, Intersection Over Union
    b. Pure performance metrics as shown above, but on an *untouched* held aside test set (which most analysis fails to do properly).
    c. T-Test between models to check for statistically significant differences in each of the metrics.
2. Resource Based Metrics
    a. Time to train (Maybe take this from other works since it takes a long time)
    b. Time to perform inference per instance, per-object
    c. Memory usage of model
    d. Histogram of image performances
        i. Best Image Performance
        ii. Worst Image Performance
3. Noise Sensitivity
    a. Add gaussian and random noise to every image and compare change in performance.
    b. Add all noises until an image changes classification, or classification confidence gap drops below certain threshold or ratio.

c. Sensitivity to orientation, scale, lighting conditions/ exposure (pixel intensity insensitivity)
d. Links related to above sensitivity testing suggestions: (http://hera.inf-cv.uni-jena.de:6680/pdf/Rodner16_FRN.pdf, https://arxiv.org/pdf/1604.04004.pdf, https://en.wikipedia.org/wiki/Sensitivity_analysis)