Research Report

Dean Alvarez - Summer 2018

https://github.com/DAlvarez-CACI/AI-Research

Introduction

Current state-of-the-art object detectors, such as Faster-RCNN, operate under the pretense that within a dataset all data points are properly labeled and well represented. However, in practice it can be difficult to obtain such a dataset. The hope of this research is to provide some insight on whether such a dataset is necessary. More concretely, the primary research objective of this investigation was to evaluate and compare the performance of various object detection models given two scenarios: the model was trained on a fully labeled (FL) dataset OR the model was trained on a single class (SC) dataset where all objects are labeled as “object”. This also led into some additional research objectives including: comparing SSDs and Faster-RCNNs, evaluating Retina-Net, and evaluating object detection performance on “messy” classes where objects are grouped in no meaningful way.

Methodology

For this investigation I used the TensorFlow Object detection API. This API provided me with pretrained models for transfer learning, config files, and evaluation utilities. Additionally, I used the COCO API for compatibility with the COCO dataset. My basic workflow was as follows:

- create TFrecord file labeled as needed (these were reusable, so I only had to do this a few times)
- retrieve model and config file from TensorFlow repository
- Alter config file to match what I am testing
- Train model
- Evaluate model on COCO dataset with COCO dataset metrics

The TFrecord files were made using a script that I modified to be able to write out continuously as well as enable labeling the data in various ways. For more specificity, many of the files I used (notably excluding the training data), are in the GitHub repository.

Findings

RCNN: SC vs FL

For Faster-RCNN models the highest performer was very clear: the FL model. However, this success is only because the Faster-RCNN had massive problems with training on SC data. It was bad to the point that all of the metrics (even after superfluous amounts of training) returned zero. The loss graph points to the problem being that for some reason or another the model was not learning. The extreme amounts of loss seen in the loss graph were coming from the Box-Classifier portion of the network; the region proposal network was working as intended, or at least seemed to be working as intended.

I tried a few different things to fix the problem, however, none of them proved successful. I tried adjusting the learning rate to prevent it from falling into a false minimum, however, this just postponed the networks failure (when I decreased the learning rate by a factor of ten it just took ten times as many steps to reach the point where the network started failing). I also tried testing different base models (res-Net instead of inception for example) to see if the poor performance was due to some sort of artifact from pretraining. This was not effective as this poor performance on SC data happened no matter what model the RCNN was based on. It is still possible that the model not learning is due to an artifact from pretraining. However, I think it is unlikely because intuitively one would think that two models that have different structure would not share the same artifacts from pretraining (even if they are trained on the same data). Unfortunately, I was not able to change the batch size to test if that is part of the problem (the network seemed to be locked to batch size one), however, I do not believe that it is since I tried training a SSD with batch size one and it trained normally (albeit I did not complete training; I just wanted to check to see if it trained normally).

Our current theory is that this behavior is some sort of edge case within RCNNs. This could mean multiple things however, ultimately it seems to be some sort of strange interaction between the SC data and the RCNN. Notably, this theory has not been tested.

SSD: SC vs PL45 vs FL

The results for the SSD models were more of what was expected. For starters, the SC model learned and was able to output valid (non-zero) COCO metrics. Additionally, the SC model outperformed the FL data as I had hypothesized. However, more interestingly, the 45 class (PL45) performed better in metrics that dealt with AP than both the FL and SC models. However, the PL45 model performed worse in terms AR than the SC model but better than the FL model. It should be noted that none of the models performed as well as you might have expected if you looked at the COCO dataset leaderboards, however, these models were just taken in base form and I did not tune their hyper parameters for maximum performance.

An interesting thing that I noticed is that the SSD models, especially the PL45 and SC models, seemed to learn very fast. This is probably due to the fact that these models were pretrained on the same dataset that I was training them on and that in theory they were completing simpler problems. Another slightly peculiar thing was that all of the SSD models experienced an increasing regularization loss.

Retina-Net: SC vs PL45 vs FL

The retina net, according to its accompanying paper, is a very high performing SSD with focal loss and a FPN. However, in my testing it performed worse than all of the other nets and much worse than it was reported to. Therefore, I think that the results I received should be taken with a heavy dose of salt.

Relatively speaking, the SC model of the Retina-Net performed best followed by the FL and then PL45. It should be noted that after a small amount of training the PL45 model had a massive spike in regularization loss. For some reason it had troubles with the PL45 model that it did not with others (and that the SSD did not have with PL45). I also think that if I had more time to let it train it might return to a regular level of loss as it had been making a slow decent from the massive loss spike early on in training. So once again, it is probably necessary to look at the results for the Retain-Net with a healthy dose of skepticism.

It was very curious to me that the Retina-Net was performing so abysmally. I have a theory as to why, but it is not confirmed. My biggest suspicion is that the poor performance is due to some sort of incompatibility or problem with the training method. It is possible that Retina-Net is built for training on the new TensorFlow trainer and when you try to force it to use the old trainer (as I did) it has sub optimal performance. Retina-Net, unlike the other models I worked on, used a sort of parallelization during training where it would split up the batch and then aggregate the results. However, this was seemingly done in sequence so perhaps it's more like pseudo-parallelization.

Conclusions

Overall, it is hard to make any sweeping conclusions about what the best labeling method is. The data I have points towards SC leading to better performance than FL and that the performance when trained the PL45 dataset depends heavily on the architecture of the model itself. However, it is hard to say if the coco evaluation metrics can even be directly compared. This is because the nature of the COCO metrics might penalize networks for misclassifying when what we care about is just how well the network actually detects objects regardless of classification. One conclusion that I can make fairly confidently is that with these types of neural networks is that despite their similar intended functions, seemingly small differences in their architecture can make a large difference in how the models performs in various scenarios.

Additionally, testing the 45-class dataset revealed that perhaps there is an optimal number of classes to have if all you care about is object detection scores. Especially since the classes were grouped arbitrarily, it seems that thoughtful grouping, perhaps using super-classes, could lead to a significant performance benefit.

Further Research Questions

What is the optimal number of classes?

As I mentioned before, the data I found points to there being some sort of optimal number of classes if you only care about the COCO metrics. This could be tested by looking at the performance models trained on more datasets, all with different numbers of classes and different methodologies of class grouping. If it was found that each model has its own optimal class number, it would mean that if someone wants to squeeze extra performance out of their model they could do it by finding the optimal number of classes.

Why do some models react differently to a change in data labeling than others?

In my testing many of the models seemed very sensitive to changes in the number of classes. RCNNs would not even work with one class and other models had large changes in regularization loss across different dataset types. It would be interesting to find out what causes this sensitivity because then you would (in theory) be able to leverage this and make models more suited to a given task.

What makes some models more robust / generally intelligent than others?

This sort of deals with the last question, but some models seemed more adaptable to changes in input data. It is my understanding that the more use cases a model has the more intelligent it is considered. It would be interesting to investigate what it is about certain models that makes them more robust to different types of input. If this proved successful you could (in theory) be able to apply it to many other models and create more intelligent and versatile AI. In theory the perfect object detector would, like us, be able to detect things no matter how you classify them. Finding out the cause of some models more general intelligence could be a step towards making a perfect object detector.