Analyzing xView Dataset:

Individual Project Report - Diana Holcomb

Machine Learning II DATS 6203 - 11

Introduction

The goal of this project was to build a convolutional neural network (CNN) in PyTorch to detect objects in high-resolution satellite imagery, and to compare that with existing Tensorflow output. I chose the xView dataset, not knowing what a leap it would be from our friendly MNIST and fashionMNIST class examples.

We split the work pretty evenly. Jiarong Che set up the image chipper and picked out our desired CNN architecture. She also got all of the example Tensorflow code working with the Vanilla model. Jiajun Wu figured out how to evaluate our PyTorch findings and how to compare the models - basically, figuring out how mean average precision, aka mAP, works. He also got all of the example Tensorflow code working with the MultiRes_Aug model. I worked on loading the xView data into Jiarong Che's CNN, and got all of the example Tensorflow code working with the MultiRes model.

Individual Work

Loading the xView data into PyTorch's DataLoader was my job and it was very difficult and time consuming. I worry that my failure to get it done as quickly as I had anticipated held my group back from some of their activities. Every PyTorch example we did in class had its own torchvision dataset for convenient use with a DataLoader, and I failed to realize that. XView does not yet have a dataset. Figuring out how to create a dataset object from scratch was challenging in two ways: the main challenge was in how PyTorch prefers its imagery formatting. NumPy and image-specific libraries such as PIL parse an image as (height, width, channels). PyTorch needs images to be formatted as (channels, height, width). I not-so-quickly realized this, but finally implemented a transform and got us back on track. Additionally, most online PyTorch examples used datasets that were already included in Torchvision, so examples of how to create my own were sparse and very problem-specific. I wasted many cycles trying

different solutions I found on Stack Overflow and the PyTorch forums, only to find that they didn't work with our specific xView data issues.

I also had to create my own custom collate function to solve another issue I encountered: converting our target array to a PyTorch tensor. In the xView dataset, not only can many annotations occur in the same image (car, boat, building, etc.), there is also a variable amount of annotations per image - one image may contain 12 cars while another contains 3000 buildings. To solve this issue and get PyTorch to ingest our data, I decided to pad every batch of targets to the length of the longest target array in the batch. I found several examples on Stack Overflow and the PyTorch forums, but much like the DataSet they did not do the trick, and I had to write my own from scratch.

Finally, I helped a lot with calculating the exact size of inputs, outputs, layers, etc to make sure that the CNN could understand the data.

Results

I learned a lot about big data and preprocessing on this project. Initially our dataset seemed small to me, with only ~1400 images included. I work with imagery at work, although more from an ETL perspective than an ML one, so only 1400 TIFFs didn't phase me at first - that would only take me 2-3 hours to put into GeoServer and S3, after all! I was really wrong. The size of those images and the amount of chips we generated from them instead resulted in much more data than I anticipated - it wouldn't fit on Jiarong or Jiajun's laptop, so I had to SCP the data to their GPC and AWS instances, respectively. It was very tricky to avoid out-of-memory errors, and also my input and layer size calculations were much bigger than I had initially thought. Even just processing the ground truth geoJSON file took a while on every single run. I made the mistake initially of letting all 379 training images be processed, but after a couple days of waiting 45 minutes at a time to just see an error further down in the code, I started limiting the size to 20 until I sorted out the bugs.

I hit my downfall in the loss calculation part of the network. I finally had every layer the correct size, and every tensor the proper type, but I kept getting a "multi-target not supported" error at the backwards step. In class, Professor Jafari pointed out it was probably my class names. While xView has 60 numerically-designated classes, it is actually a subset of a much bigger dataset of around ~100 classes that Digital Globe (the company who created it) stripped down for the competition. Unfortunately that meant that the class names were not 0-59, as expected by CrossEntropyLoss, but instead an arbitrary list of 60 numbers that could be anywhere between 1 and 100.

Future Work and Conclusion

Given more time, I'd really like to update the ground truth target ids and get past my issue with the loss function. Once that happens, all of the things I really wanted to do could happen! I could finally compare output against Tensorflow. I could set up GridSearchCV and try various optimizers in PyTorch, and then let GPC churn for a few (possibly expensive) days to see how accuracy and convergence time change. I'd love more time to find a better way to deal with the variable target array lengths than simple padding - that was a panicked 3am decision a few nights ago just to get some forward momentum. Finally, trying oversampling and undersampling like I learned in Machine Learning I sounds cool. The distribution of the classes is not even, and accounting for that would boost accuracy.

Processing xView data in PyTorch proved to be an overly ambitious problem decision. I regret stressing my teammates out with my dataset choice, since I think this is the first time any of us have presented a project that didn't fully work, and it's my part that doesn't work. The lack of multi-class per image and xView-specific examples online really held me back, and made me wish I had chosen a one image/one class problem instead. My current challenge of properly calculating loss from a dataset with a variable length of target arrays can hopefully be solved by the in-class suggestion of fixing the target class names, luckily.

Other People's Code

Total lines of code (including comments and other people's code): 444

Lines of code from xView Pre-Processing Code: lines 39-174 = 135 lines

Lines of code from Torchvision DataSet class: lines 225-243, 262-298 = 54 lines

(135 + 54 / 444) * 100 = **42.56%**

References

xView Competition website: https://challenge.xviewdataset.org

xView dataset and ground truth geojson:
https://challenge.xviewdataset.org/download-links

Pretrained Tensorflow models: https://github.com/DIUx-xView/baseline/releases

xView Pre-Processing code: https://github.com/DIUx-xView/data_utilities

xView Baseline TensorFlow code: https://github.com/DIUx-xView/baseline

xView Baseline Docker container: https://hub.docker.com/r/xview2018/baseline/tags

Code

https://github.com/catsbergers/Final-Project-Group-2