KF5012 Software Engineering Practice

Iterative Development

Poe Dameron

Pipeline evaluation

The first stage was to find and prepare a publicly available database for our project. After doing some research, we chose one set of images that best fit the specifications of our idea. The database represented various locations from a drone's perspective with lots of potential objects that our code would identify.

The next step was to prepare the dataset by dividing the data into two subsets: training and test set. Using this procedure allowed us to control our training accuracy on an independent set.

Then, to fully prepare our data for neural network, the images from our databases were reshaped into tensors and resized.

The final step was to divide the photos into batches and use the Data Loader to import the photos into our programme.

Due to the characteristics of our data (images) and the fear of losing relevant information, we have decided that the above pre-processing techniques are safe and sufficient for our purposes.

Model Evaluations

The assumptions of our project-imposed requirements that we had to check before finally choosing one of the models. Among many, our attention was focused on two specific ones: RetinaNet and Faster R-CNN.

Model	mAP	mAR
RetinaNet	0.0570	0.0468
Faster R-CNN	0.1345	0.1756

Figure 1 – Comparison of pyTorch implementations of Faster R-CNN and RetinaNet by mean Average Precision (mAP) and mean Average Recall (mAR)

As seen from Figure 1, RetinaNet achieves nearly half of the mAP and barely a third of the mAR of Faster R-CNN from preliminary tests. It should be noted that RetinaNet required a train percentage of 0.02 (2%) to be able to achieve these scores while Faster R-CNN only required 0.01, making it require twice the normal training size for testing.

Some models that were considered favourable were found in the TensorFlow Hub model library that may have been successful when fine-tuned to our dataset (such as the TF implementation of SSD using MobileNet v2 trained on COCO 2017). However, the main implementation of our project was made in pyTorch, and creating a parallel implementation using TensorFlow was proving difficult, as it required many functions (like one to transform the dataset into a format like a TFRecord that is usable by the TF models) that were drastically different to pyTorch functions already created. Early testing of some

pre-trained models from TensorFlow Hub also showed that they were hard to fine-tune, so the implementation was cancelled to focus on development of the pyTorch implementation.

Parameter Justifications

Considering the amount of time it would take to train a model from scratch, the team decided to fine-tune a pretrained model. The choice of the model finalized, we needed to find the best values for the hyperparameters which is why a shared excel sheet with every one's chosen hyperparameter to optimize was created. This did facilitate and organize the search as everyone knew what parameters they were expected to train on. A csv file was linked to an output function to record the model's output and its parameters. To allow access to the same training files and code, a GitHub and a Google Drive were created and associated to the Colab notebook. Training the model needed coordination because of Google Colab's GPU usage limit. Consequently, was established a cycle to take advantage of everyone's GPU availability on Google Colab. This rotation was available in the Poe Dameron Microsoft Teams tasks.

	Best performance			Worst performance		
	Value	Average Precision	Average Recall	Value	Average Precision	Average Recall
Learning rate	0.005	0.135	0.176	0.05	0.046	0.055
Weight decay	0.0005	0.135	0.176	0.0001	0.109	0.135
Momentum	0.9	0.135	0.176	0.95	0.125	0.179
Step size	3	0.135	0.176	1	0.102	0.135
Gamma	0.0001	0.142	0.199	1	0.128	0.175
Batch size	2	0.1345	0.176	6	0.078	0.110

Figure 2 – Summary of the results of the search for the best hyperparameters

Optimizer	Best performance		Worst performance	
	Average Precision	Average Recall	Average Precision	Average Recall
Adam	0.124	0.162	0	0
SGD	0.142	0.199	0.046	0.055

Figure 3 – Summary of the results of the search for the best optimizer

Reproducible code

Our main tool for work and sharing code was GitHub. To view the contents of our repository, please follow the link:

https://github.com/POE-DAMERON/Glie-44

Many files (like the compressed database, video outputs and model checkpoints) could not be stored on GitHub due to size constraints, so the files were added to a Google Drive that could be mounted and accessed in Google Colab. This drive can be seen here:

https://drive.google.com/drive/folders/1IMQUReiJeOlQxcvlQy8GqVn505PKYDhR?usp=sharing