CNN IMAGE CLASSIFICATION USING KITTI DATASET MIDTERM REPORT

1032743 An Shuyang

June 4, 2018

1 Introduction

This project is mainly about the deep learning implement of the *Object Detection Evaluation 2012*[1] dataset, which consisting of 7481 training images and 7518 test images, comprising a total of 80.256 labeled objects[5].

The project includes data preprocessing, exploration, baseline training and hyper-parameter tunning sections. The training is running in Google cloud virtual machine with 8 vCPUs and 24 GB memory. However, Tesla k80 GPU is not available when this project was implemented due to a limitation to new account user.

2 Data exploration

2.1 Raw data processing

From the provided development kit, each row in label file corresponds to one object, and the 15 columns represents type, truncated, occluded, alpha, bounding box (4 columns), dimensions (3 columns), location (3 columns), rotation_y and score respectively.

A python script is written to read each row in label text, get the coordinates of bounding box, open the corresponding image and draw all the bounding boxes on it. For simplicity, the *patch* is imported from *matplotlib* and it provides the default function *Rectangle* to draw a box. Some examples could be seen in Fig 1, 2 and 3.

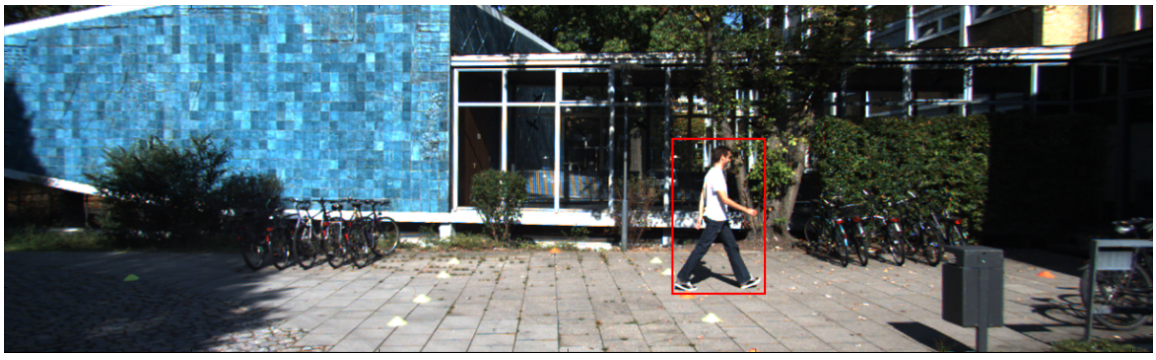


Figure 1: draw bounding box

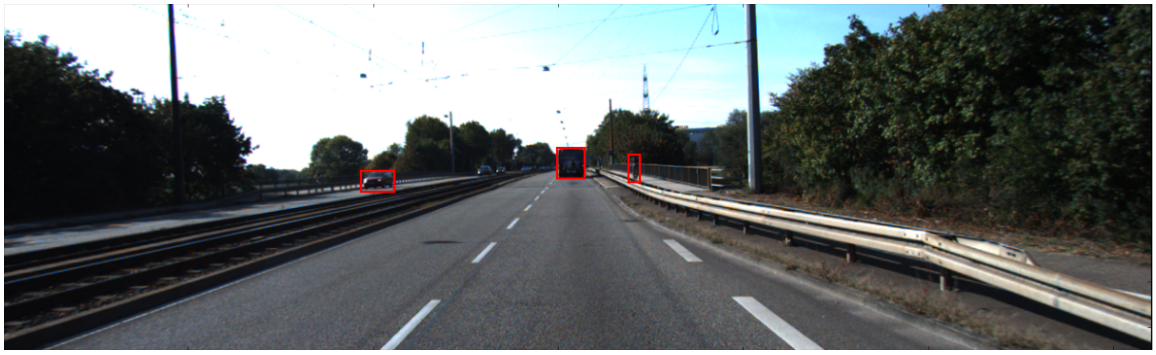


Figure 2: draw bounding box

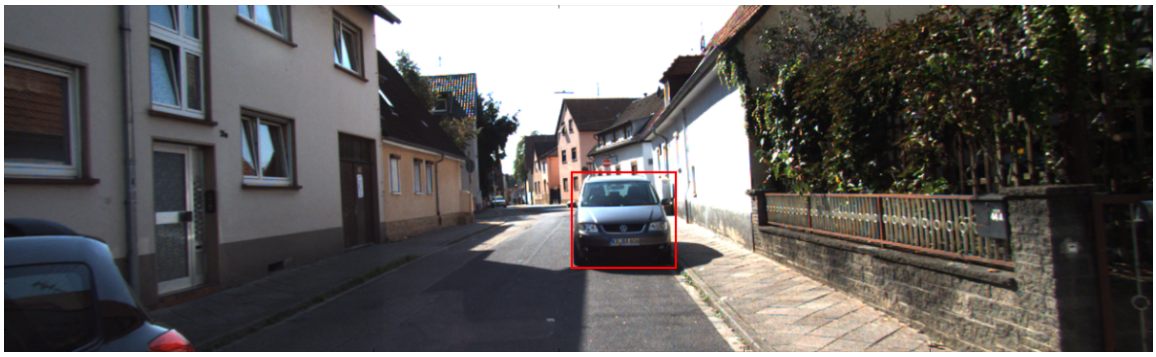


Figure 3: draw bounding box

Then all the objects in each image is cropped based on the bounding box coordinates and saved in another directory. To distinguish objects from file name, a prefix of object class is given, and also a suffix in that there might be more than one same object in a single image. The cropped file name looks like <object class><original filename><suffix number><.png>.

A h5py format dataset is also created. Array of images and their numeric labels are stored in two lists, and shuffled before writing into a h5py file. Each h5py dataset contains two "keys", namely array of images and their labels.

2.2 Data exploration

As seen in table 1, there are 8 classes in total and their size was obtained using *size* function of PIL.Image.

Table 1: Image Class and Size

	Car	Van	Truck	Pedestrian	People_sitting	Cyclist	Tram	Misc
Number	28712	2913	1093	4487	222	1625	510	972
Average Size	122,66	110,84	100,86	44,104	84,102	56,77	154,104	91,75

The average size of all the classes is (102, 73) and the all of the cropped images are resized to this dimension.

However, I doubt whether this operation is reasonable.The width-height-ration of images are quite different and by simply resizing to average size, they may not be able to present original features. After doing some searching I know the basic strategy to handle such problem, one is by resizing or cropping images and another is by *Spatial Pyramid Pooling* in model, which is not implemented in this project.

2.3 Splitting dataset for training and validation

It's required to split images into two dataset, for validation and training. Rather than simply select 25% from the whole dataset, which may cause some missing data with a low percentage, a better strategy is used to randomly select 25% for validation from each class, and later saved as the h5py file. The number of each class for validation and their percentage could be seen in table 2 below.

Table 2: Split for validation & train

	Car	Van	Truck	Pedestrian	People_sitting	Cyclist	Tram	Misc
Total Number	28712	2913	1093	4487	222	1625	510	972
Validation	7178	729	274	1122	56	407	128	243
Percentage %	25.00	25.03	25.07	25.01	25.23	25.05	25.10	25.00

The average RGB values of images are also explored:
Average RGB per split: Training: (73.60, 76.74, 81.60), Validation: (73.32, 76.47, 81.37)
Average RGB overall: (73.54, 76.67, 81.55)

3 Baseline CNN training and hyper-parameter tuning

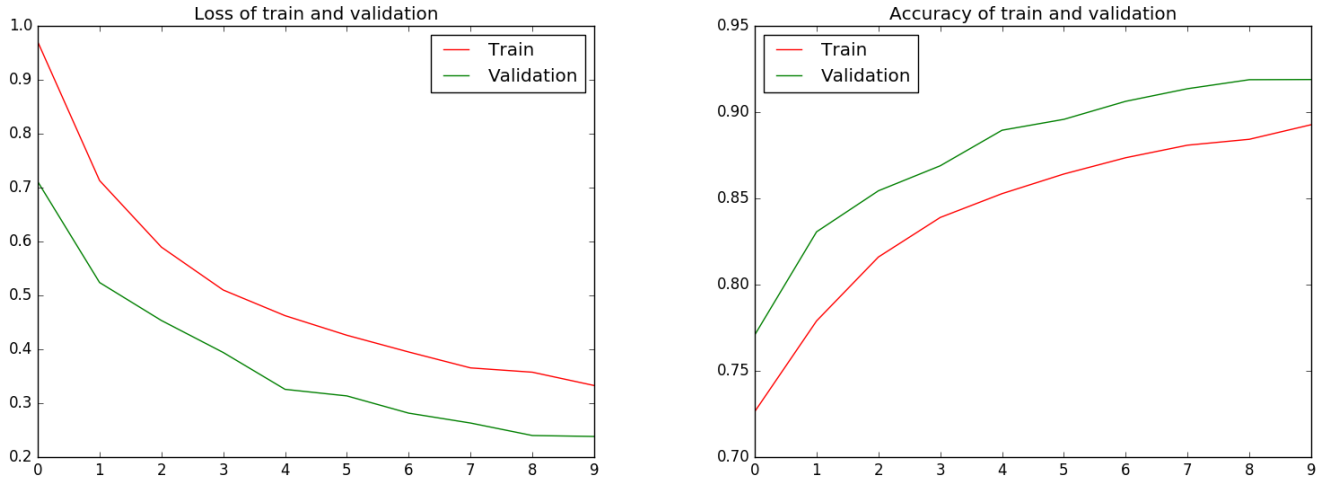
3.1 Input pipeline and Baseline CNN training

Pipeline - ImageDataGenerator

The tensorflow provides a *ImageDataGenerator* function[4] to feed data from directory to model. This is somehow slower compared with loading all data into ram at the same time, but overcomes the shortcoming that large dataset exceeding ram limitation.

History object

The tensorflow defines a class named *Callback* that records events into a *History* object[3], which is automatically applied to every keras model. To check the key of history's dictionary, it contains four variable, namely *loss*, *acc*, *val_loss* and *val_acc*. Use *pipeline* method to train KITTI and record its event in history, which could be seen in figure 4.

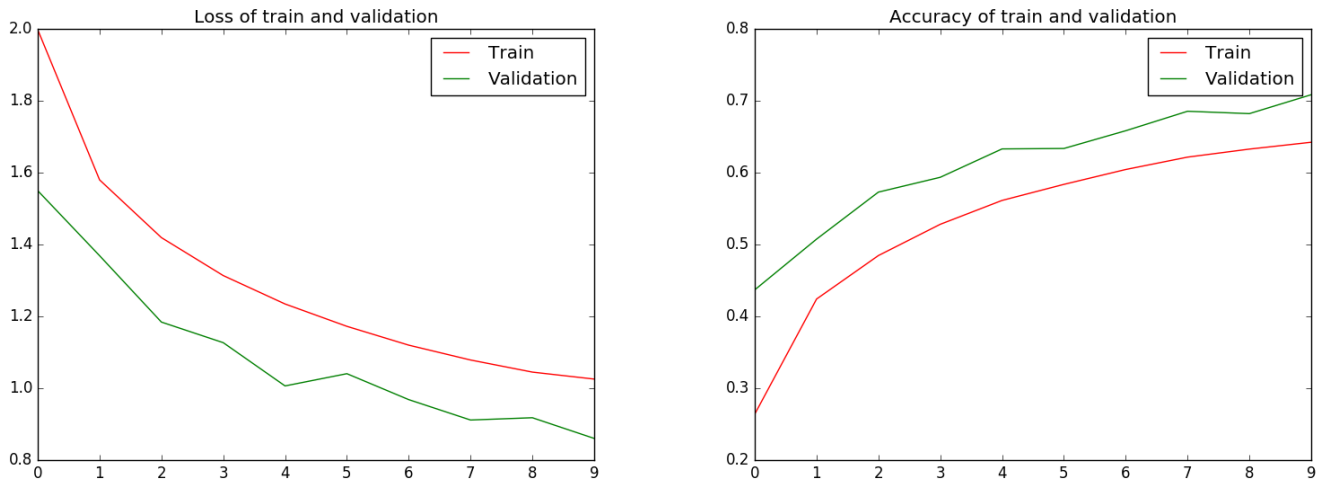


(a) Loss

(b) Accuracy

Figure 4: KITTI Baseline Loss and Accuracy

CIFAR10 is also tested with the pipeline strategy and its results as below in figure 5:



(a) Loss

(b) Accuracy

Figure 5: CIFAR10 Baseline Loss and Accuracy

Accuracy per class

The accuracy per class is obtained by using *sklearn*[2], a data analysis tool in python. A function called *classification_report* is imported from sklearn, and it has two input arguments, the test and the prediction. As seen in table 3, the accuracy of each class are not the same with each other. This is partially due to different size in each class.
The overall validation loss is 0.238, the accuracy is 0.920.

Table 3: Accuracy per class

Class	0	1	2	3	4	5	6	7	Average
Precision	0.93	0.85	0.88	0.90	0.91	0.87	0.90	0.81	0.92
Recall	0.99	0.56	0.76	0.92	0.36	0.79	0.84	0.63	0.92
F1-score	0.96	0.67	0.81	0.91	0.51	0.83	0.84	0.71	0.91

3.2 Hyper-parameter tuning and input augmentations

Proposal improvement

- Add momentum to SGD optimizer
- Use Adam optimizer instead of SGD
- Use adaptive learning rate or scheduler
- Add batch normalization layer after Conv2D and before Activation layer

Results

I build three models with different improvement approach and run them on Google cloud virtual machine, but always get wrong result. It seems the training is not complete and stuck at first class as in figure 6, however, the dataset is shuffled and the baseline works fine. The GPU will be available within this week, which would be helpful with the final project.

	precision	recall	f1-score	support
0	0.71	1.00	0.83	7178
1	0.00	0.00	0.00	729
2	0.00	0.00	0.00	274
3	0.00	0.00	0.00	1122
4	0.00	0.00	0.00	56
5	0.00	0.00	0.00	407
6	0.00	0.00	0.00	128
7	0.00	0.00	0.00	243
avg / total	0.50	0.71	0.59	10137

Figure 6: Wrong results

References

[1] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[2] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[3] *tf.keras.callbacks.History*. `https://s3.eu-central-1.amazonaws.com/avg-kitti/devkit_object.zip`.

[4] *tf.keras.preprocessing.image.ImageDataGenerator*. `https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator`.

[5] *The KITTI Vision Benchmark Suite - Object Detection Evaluation 2012*. `http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=2d`.