Final Competition of Deep Learning 2020 Spring

Final project (Master track)

---

1 Final project

As we're sadly approaching the end of this course, we thought it would be fun to fire up a competition. You'll need to form a group of three people. You can use the "Search for Teammates!" tool on Piazza (found here).

We have provided a Jupiter notebook that explains the data set and the evaluation procedures, and it's reported in this document as well.

You can find the data set, its companion notebook, and helper code in the folder `/scratch/jz3224/DLSP20Dataset` at HPC.

2 Leader board

We'll communicate on Piazza about how your models will be evaluated against our test data at a later time.

3 Deliverable

For the project you will need to:

(a) Perform a literature review: what work has been done in this area already? How does your work relate to previous work?

(b) Describe your initial idea: why it is new, interesting or useful. Include a description of your idea for a new architecture or task. This does not have to be final, you may find that as you collect results from your experiments, you will want to change your approach based on what works and what doesn't.

(c) Describe your results. This should include some kind of performance metrics, such as accuracy, AUC, reward, or whatever is appropriate to the task. Be sure to compare several different methods to put them all in context, do not simply report one number.

(d) Provide some kind of visualization or intuition: what is your model doing? Can you visualize the features that it is learning? What kind of examples does it succeed or fail on? What are its strengths and limitations?

3.1 Format

(a) Completed write-up. Not more than 4 pages, references included using ICML 2019 LaTeXlayout.

(b) 180 second video presentation (6 slides). You will receive a penalty of 1/60 for any additional second in the video length. You will receive a penalty of 1/2 for any additional slide. This means a video of 240 seconds or a presentation with 8 slides will automatically have a score of zero. Check here to learn how to deliver an effective presentation.

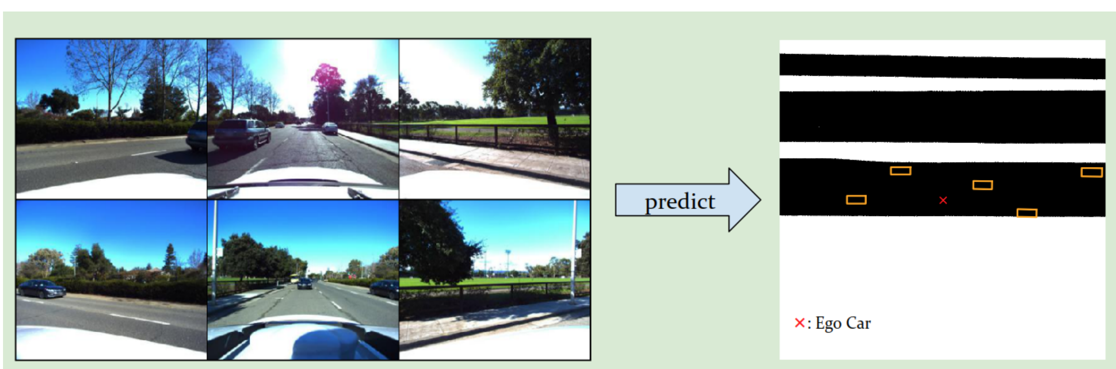
4 Final Submission

You will be able to submit your final project through NYU class, specifying the group formation in form of `NetID1`, `NetID2`, `NetID3`, by Monday 11 May 2020.

5 The Notebook

5.1 Goals

The objective is to train a model using images captured by six different cameras attached to the same car to generate a top down view of the surrounding area. The performance of the model will be evaluated by (1) the ability of detecting objects (like car, trucks, bicycles, etc.) and (2) the ability to draw the road map layout.



5.2 Data

You will be given two sets of data:

1. Unlabeled set: just images
2. Labeled set: images and the labels(bounding box and road map layout)

This notebook will help you understand the dataset.

[1]:
```
import os
import random

import numpy as np
import pandas as pd

import matplotlib
import matplotlib.pyplot as plt
matplotlib.rcParams['figure.figsize'] = [5, 5]
matplotlib.rcParams['figure.dpi'] = 200

import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision

from data_helper import UnlabeledDataset, LabeledDataset
from helper import collate_fn, draw_box
```

[2]:
```
random.seed(0)
np.random.seed(0)
torch.manual_seed(0);
```

[3]:
```
# All the images are saved in image_folder
# All the labels are saved in the annotation_csv file
image_folder = 'data'
annotation_csv = 'data/annotation.csv'
```

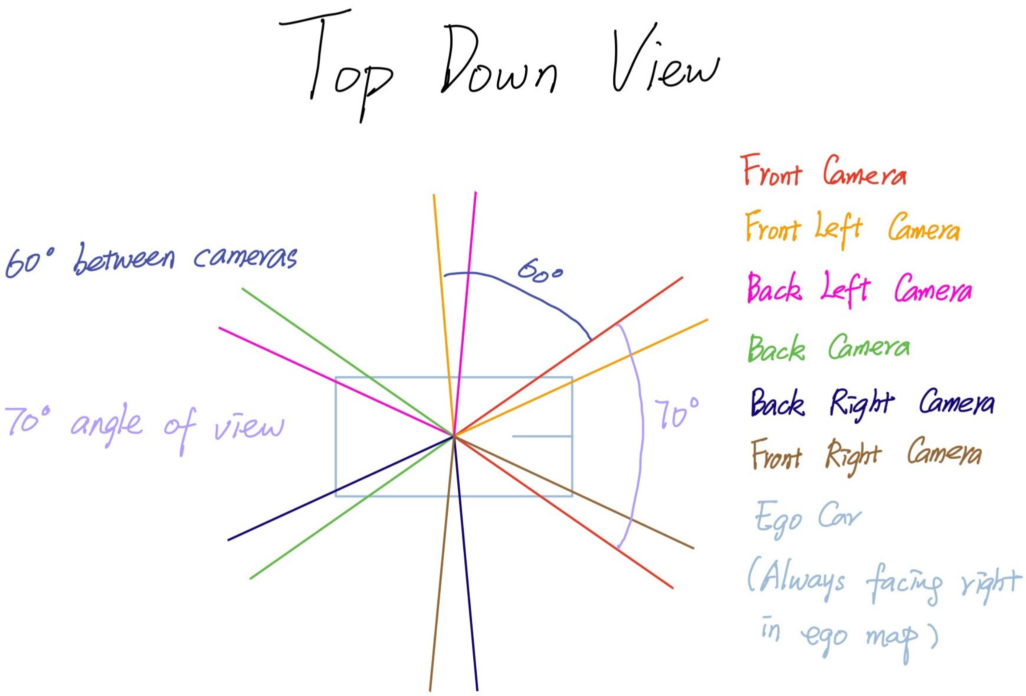
5.3 Dataset

You will get two different datasets:

1. an unlabeled dataset for pre-training
2. a labeled dataset for both training and validation

5.3.1 The dataset is organized into three levels: scene, sample and image

1. A scene is 25 seconds of a car's journey.
2. A sample is a snapshot of a scene at a given timeframe. Each scene will be divided into 126 samples, so about 0.2 seconds between consecutive samples.
3. Each sample contains 6 images captured by camera facing different orientation. Each camera will capture 70 degree view. To make it simple, you can safely assume that the angle between the cameras is 60 degrees



106 scenes in the unlabeled dataset and 28 scenes in the labeled dataset

[4]:
```
# You shouldn't change the unlabeled_scene_index
# The first 106 scenes are unlabeled
unlabeled_scene_index = np.arange(106)
# The scenes from 106 - 133 are labeled
# You should devide the labeled_scene_index into two subsets (training␣
↪and validation)
labeled_scene_index = np.arange(106, 134)
```

5.4 Unlabeled dataset

You get two ways to access the dataset, by sample or by image

5.4.1 Get Sample

[5]:
```
transform = torchvision.transforms.ToTensor()

unlabeled_trainset = UnlabeledDataset(image_folder=image_folder,␣
↪scene_index=labeled_scene_index, first_dim='sample',␣
↪transform=transform)
trainloader = torch.utils.data.DataLoader(unlabeled_trainset,␣
↪batch_size=3, shuffle=True, num_workers=2)
```

[6]:
```
# [batch_size, 6(images per sample), 3, H, W]
sample = iter(trainloader).next()
print(sample.shape)
```

```
torch.Size([3, 6, 3, 256, 306])
```

[7]:
```
# The 6 images orgenized in the following order:
# CAM_FRONT_LEFT, CAM_FRONT, CAM_FRONT_RIGHT, CAM_BACK_LEFT, CAM_BACK,␣
↪CAM_BACK_RIGHT
plt.imshow(torchvision.utils.make_grid(sample[2], nrow=3).numpy().
↪transpose(1, 2, 0))
plt.axis('off');
```

5.4.2 Get individual image

[8]:
```
unlabeled_trainset = UnlabeledDataset(image_folder=image_folder,␣
↪scene_index=unlabeled_scene_index, first_dim='image',␣
↪transform=transform)
trainloader = torch.utils.data.DataLoader(unlabeled_trainset,␣
↪batch_size=2, shuffle=True, num_workers=2)
```

[9]:
```
# [batch_size, 3, H, W]
image, camera_index = iter(trainloader).next()
print(image.shape)
```

```
torch.Size([2, 3, 256, 306])
```

[10]:
```
# Camera_index is to tell you which camera is used. The order is
```

```
# CAM_FRONT_LEFT, CAM_FRONT, CAM_FRONT_RIGHT, CAM_BACK_LEFT, CAM_BACK,␣
↪CAM_BACK_RIGHT
print(camera_index[0])
```

```
tensor(2)
```

[11]:
```
plt.imshow(image[0].numpy().transpose(1, 2, 0))
plt.axis('off');
```

5.5 Labeled dataset

[12]:
```
# The labeled dataset can only be retrieved by sample.
# And all the returned data are tuple of tensors, since bounding boxes␣
↪may have different size
# You can choose whether the loader returns the extra_info. It is␣
↪optional. You don't have to use it.
labeled_trainset = LabeledDataset(image_folder=image_folder,
                                  annotation_file=annotation_csv,
                                  scene_index=labeled_scene_index,
                                  transform=transform,
                                  extra_info=True
                                 )
```

```
trainloader = torch.utils.data.DataLoader(labeled_trainset, batch_size=2,␣
↪shuffle=True, num_workers=2, collate_fn=collate_fn)
```

[13]:
```
sample, target, road_image, extra = iter(trainloader).next()
print(torch.stack(sample).shape)
```

```
torch.Size([2, 6, 3, 256, 306])
```

There are two kind of labels

1. The bounding box of surrounding objects
2. The binary road_image

5.5.1 Bounding box

[14]:
```
# The shape of bounding box is [batch_size, N (the number of object), 2,␣
↪4]
print(target[0]['bounding_box'].shape)
```

```
torch.Size([16, 2, 4])
```

[15]:
```
# All bounding box are retangles
# Each bounding box is organized with four corners of the box
# All the values are in meter and bounded by 40 meters, and the origin␣
↪is the center of ego car
# the order of the four courners are front left, front right, back left␣
↪and back right
print(target[0]['bounding_box'][0])
```

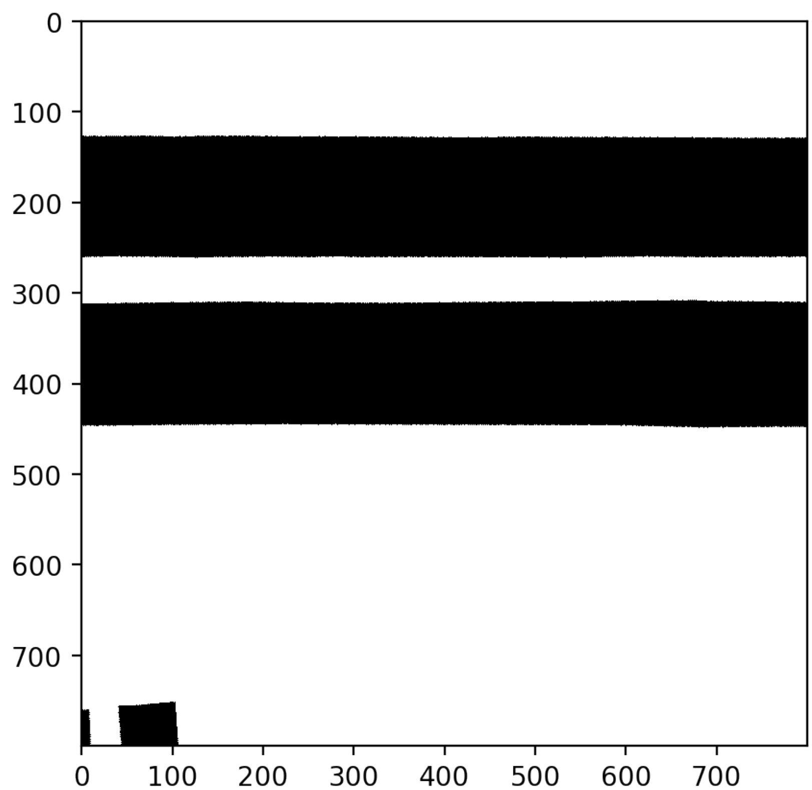
```
tensor([[29.1092, 29.1643, 20.9320, 20.9871],
        [-2.1322, -4.8175, -2.3005, -4.9858]], dtype=torch.float64)
```

[16]:
```
# Each bounding box has a category
# 'other_vehicle': 0,
# 'bicycle': 1,
# 'car': 2,
# 'pedestrian': 3,
# 'truck': 4,
# 'bus': 5,
# 'motorcycle': 6,
# 'emergency_vehicle': 7,
# 'animal': 8
print(target[0]['category'])
```

```
tensor([0, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0])
```

5.5.2 Road Map Layout

[17]:
```
# The road map layout is encoded into a binary array of size [800, 800]␣
↪per sample
# Each pixel is 0.1 meter in physiscal space, so 800 * 800 is 80m * 80m␣
↪centered at the ego car
# The ego car is located in the center of the map (400, 400) and it is␣
↪always facing the left

fig, ax = plt.subplots()

ax.imshow(road_image[0], cmap='binary');
```



[18]:
```
print(road_image[0])
```

```
tensor([[False, False, False,  ..., False, False, False],
        [False, False, False,  ..., False, False, False],
        [False, False, False,  ..., False, False, False],
        ...,
        [ True,  True,  True,  ..., False, False, False],
```

```
 [ True,  True,  True,  ..., False, False, False],
 [ True,  True,  True,  ..., False, False, False]])
```

5.5.3 Extra Info

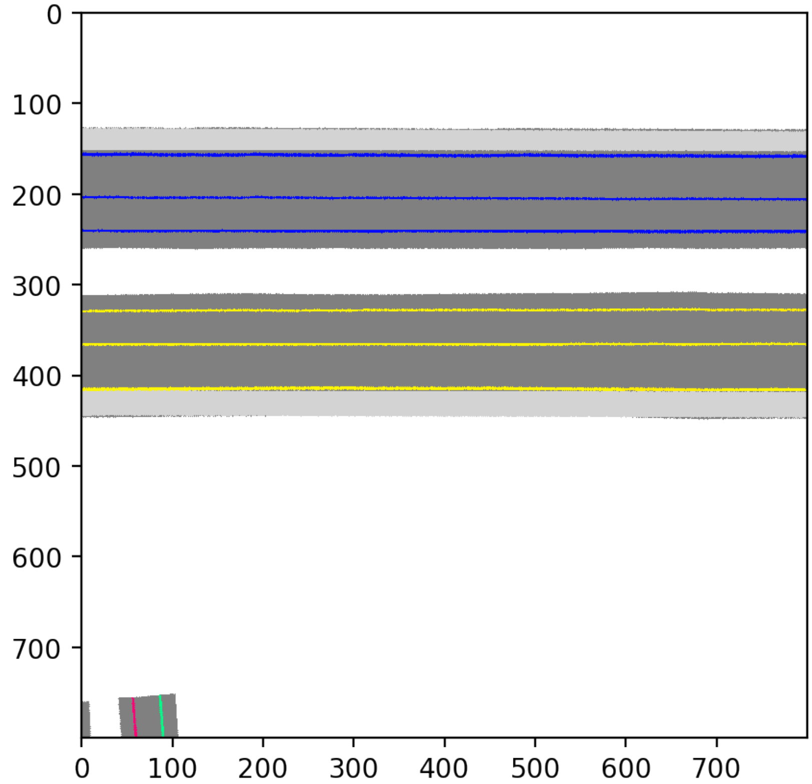
There is some extra information you can use in your model, but it is optional.

[19]:
```
# Action
# Action is the label that what the object is doing

# 'object_action_parked': 0,
# 'object_action_driving_straight_forward': 1,
# 'object_action_walking': 2,
# 'object_action_running': 3,
# 'object_action_lane_change_right': 4,
# 'object_action_stopped': 5,
# 'object_action_left_turn': 6,
# 'object_action_right_turn': 7,
# 'object_action_sitting': 8,
# 'object_action_standing': 9,
# 'object_action_gliding_on_wheels': 10,
# 'object_action_abnormal_or_traffic_violation': 11,
# 'object_action_lane_change_left': 12,
# 'object_action_other_motion': 13,
# 'object_action_reversing': 14,
# 'object_action_u_turn': 15,
# 'object_action_loss_of_control': 16
```

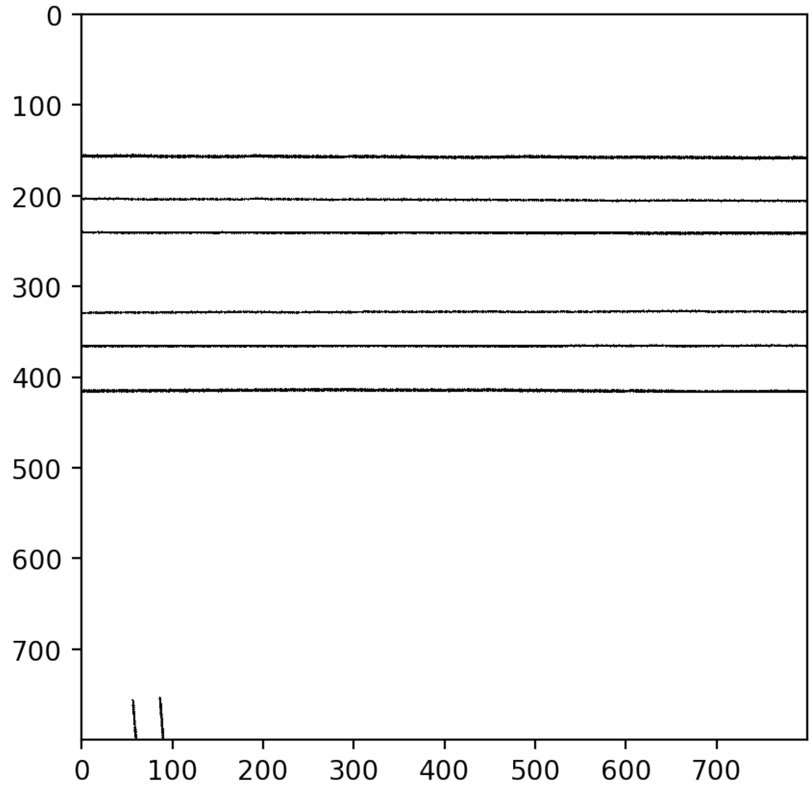
[20]:
```
print(extra[0]['action'])
```

```
tensor([0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0])
```

[21]:
```
# Ego Image
# A more detailed ego image
fig, ax = plt.subplots()

ax.imshow(extra[0]['ego_image'].numpy().transpose(1, 2, 0));
```

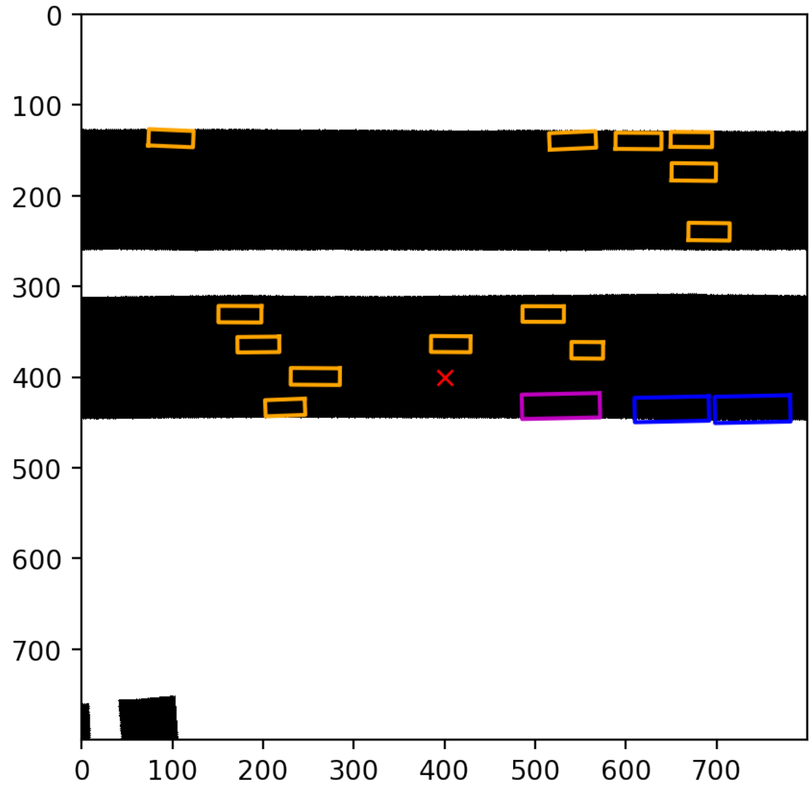


[22]:
```
# Lane Image
# Binary lane image
fig, ax = plt.subplots()

ax.imshow(extra[0]['lane_image'], cmap='binary');
```



5.6 Visualize the bounding box

[23]:
```
# The center of image is 400 * 400

fig, ax = plt.subplots()

color_list = ['b', 'g', 'orange', 'c', 'm', 'y', 'k', 'w', 'r']

ax.imshow(road_image[0], cmap ='binary');

# The ego car position
ax.plot(400, 400, 'x', color="red")

for i, bb in enumerate(target[0]['bounding_box']):
    # You can check the implementation of the draw box to understand how␣
 ↪it works
    draw_box(ax, bb, color=color_list[target[0]['category'][i]])
```



5.7 Evaluation

During the whole competition, you have three submission deadlines. The dates will be announced on Piazza. You will have to fill up the template 'data_loader.py' for evaluation. (see the comment inside data_loader.py' for more information)

There will be two leaderboards for the competition: The leaderboard for binary road map. We will evaluate your model's performance by using the average threat score (TS) across the test set:

$$\mathrm{TS} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

The leaderboard for object detection: We will evaluate your model's performance for object detection by using the average mean threat score at different intersection over union (IoU) thresholds. There will be five different thresholds (0.5, 0.6, 0.7, 0.8, 0.9). For each thresholds, we will calculate the threat score. The final score will be a weighted average of all the threat scores:

$$\text{Final Score} = \sum_t \frac{1}{t} \cdot \frac{\mathrm{TP}(t)}{\mathrm{TP}(t) + \mathrm{FP}(t) + \mathrm{FN}(t)}$$