KENTECH
Korea Institute of Energy Technology

Visionary Course – Energy AI
Week 12

Nov 14, 2022
Sohee Kim, Giwon Sur

KENTECH
Korea Institute of Energy Technology

Week 12a – Inference for road following on Jetson Nano

JetRacer: Let's start autonomous driving!

➡ Today we will test the pretrained model on the road with 2D-lined track.

Workspace:

“localhost:8888/lab/tree/jetracer/notebooks/road_following.ipynb”

Overview of road_following.ipynb

1

First, create the model. This must match the model used in the interactive training notebook.

```
import torch
import torchvision

CATEGORIES = ['apex']

device = torch.device('cuda')
model = torchvision.models.resnet18(pretrained=False)
model.fc = torch.nn.Linear(512, 2 * len(CATEGORIES))
model = model.cuda().eval().half()
```

Define model

2

Next, load the saved model. Enter the model path you used to save.

```
model.load_state_dict(torch.load('road_following_model.pth'))
```

Convert and optimize the model using `torch2trt` for faster inference with TensorRT. Please see the torch2trt readme for more details

> This optimization process can take a couple minutes to complete.

```
from torch2trt import torch2trt

data = torch.zeros((1, 3, 224, 224)).cuda().half()

model_trt = torch2trt(model, [data], fp16_mode=True)
```

Save the optimized model using the cell below

```
torch.save(model_trt.state_dict(), 'road_following_model_trt.pth')
```

Load the optimized model by executing the cell below

```
import torch
from torch2trt import TRTModule

model_trt = TRTModule()
model_trt.load_state_dict(torch.load('road_following_model_trt.pth'))
```

Load pretrained model

3

Create the racecar class

```
from jetracer.nvidia_racecar import NvidiaRacecar

car = NvidiaRacecar()
```

Create the camera class.

```
from jetcam.csi_camera import CSICamera

camera = CSICamera(width=224, height=224, capture_fps=65)
```

Configure racecar & camera

Finally, execute the cell below to make the racecar move forward, steering the racecar based on the x value of the apex.

Here are some tips,

- If the car wobbles left and right, lower the steering gain
- If the car misses turns, raise the steering gain
- If the car tends right, make the steering bias more negative (in small increments like -0.05)
- If the car tends left, make the steering bias more postive (in small increments +0.05)

4

```
from utils import preprocess
import numpy as np

STEERING_GAIN = 0.75
STEERING_BIAS = 0.00

car.throttle = 0.15

while True:
    image = camera.read()
    image = preprocess(image).half()
    output = model_trt(image).detach().cpu().numpy().flatten()
    x = float(output[0])
    car.steering = x * STEERING_GAIN + STEERING_BIAS
```

Iterative inference

1. Define model

```python
import torch
import torchvision

CATEGORIES = ['apex']

device = torch.device('cuda')
model = torchvision.models.resnet18(pretrained=False)
model.fc = torch.nn.Linear(512, 2 * len(CATEGORIES))
model = model.cuda().eval().half()
```

Import modules to define model

'apex' directs a point with x, y coordination in camera image

Declare resnet18 to define model
Set fc layer's output size as 2 (x, y)

Set data type to float16

Set model's mode as evaluation

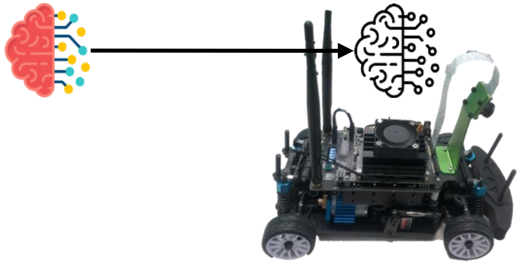
Load model to GPU

2. Load pretrained model

```
model.load_state_dict(torch.load('road_following_model.pth'))
```

Load pretrained model from '~.pth' file.

model(resnet18)



```
from torch2trt import torch2trt

data = torch.zeros((1, 3, 224, 224)).cuda().half()

model_trt = torch2trt(model, [data], fp16_mode=True)
```

Import Torch2trt for better performance

Declare sample data with the same size as target data

Convert model from pytorch to TensorRT

```
torch.save(model_trt.state_dict(), 'road_following_model_trt.pth')
```

Save converted model parameters

```
import torch
from torch2trt import TRTModule

model_trt = TRTModule()
model_trt.load_state_dict(torch.load('road_following_model_trt.pth'))
```

Declare trt model which stores the original model

Load saved model parameters

3. Configure race car and camera

```
from jetracer.nvidia_racecar import NvidiaRacecar

car = NvidiaRacecar()
```

Create race car class

```
from jetcam.csi_camera import CSICamera

camera = CSICamera(width=224, height=224, capture_fps=65)
```

Create camera class

4. Iterative inference

```python
from utils import preprocess
import numpy as np

STEERING_GAIN = 0.75
STEERING_BIAS = 0.00

car.throttle = 0.15

while True:
    image = camera.read()
    image = preprocess(image).half()
    output = model_trt(image).detach().cpu().numpy().flatten()
    x = float(output[0])
    car.steering = x * STEERING_GAIN + STEERING_BIAS
```

Set fixed multiplier and adder for steering

Set throttle to fixed value

Read real-time image from camera

Feed image values and get the output from the model

From output, car inferences steering value

```python
import torch
import torchvision.transforms as transforms
import torch.nn.functional as F
import cv2
import PIL.Image
import numpy as np

mean = torch.Tensor([0.485, 0.456, 0.406]).cuda()
std = torch.Tensor([0.229, 0.224, 0.225]).cuda()

def preprocess(image):
    device = torch.device('cuda')
    image = PIL.Image.fromarray(image)
    image = transforms.functional.to_tensor(image).to(device)
    image.sub_(mean[:, None, None]).div_(std[:, None, None])
    return image[None, ...]
```

Return normalized image values

Experiments :

1. Upload pretrained model.

Q.1.1. Download trained model "road_following_model_gwsur.pth" from LMS.

Q.1.2. Visit to localhost:8888/lab/tree/jetracer/notebooks/road_following.ipynb

Q.1.3. Upload the model pth file and write the models name on code
model.load_state_dict(torch.load('road_following_model_gwsur.pth'))

2. Observe and execute the model.

Q.2.1. Unfold the track in suitable place. And put the car on the track.

Q.2.2. Run codes along with the cell.

Q.2.3(optional). Reference and use "teleoperation.ipynb" code to prevent missing car.

Q.2.4. Observe the car's trace. Are the steering and throttle are suitable?

Q.2.5. Control the value of throttle (slower or faster).

Experiments :

3. Make variable environments and execute the model.

Q.3.1. Place the car in opposite direction of Q2 and execute model.
Is the model following the road similar?

Q.3.2. Place a small obstacle on the road. How's the car doing?