YOLO(V3, V3-tiny, V3-edge) 모델 테스팅 및 분석

Requirments : YOLOv3, docker, Linux

- **docker 개발 환경 구축**
  - docker 에 nvidia driver 설치 여부 확인(nvidia-smi)
  - cuda를 docker image로 받아서 컨테이너 설치

```
docker run -itd –gpus all –ipc host -v /home/testworks/works/deep.learning/edge_folder:/workspace/src
 –name edge_AI nvidia/cuda:11.4.2-devel-ubuntu20.04
```

  - 컨테이너 실행

```
docker exec -it edge_AI /bin/bash
```

- **YOLOv3 사용환경 구축**
  - **darknet 내려 받기**

```
git clone https://github.com/pjreddie/darknet
```

  - **다운 받은 폴더 안에서 makefile 수정**
    - **Makefile**

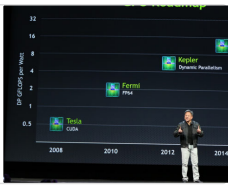
```
GPU=1

CUDNN=0

OPENCV = 0

OPENMP=0

DEBUG=0

ARCH= -gencode arch=compute_35,code=sm_35 \

-gencode arch=compute_50,code=[sm_50,compute_50] \

-gencode arch=compute_52,code=[sm_52,compute_52] \

-gencode arch=compute_70,code=[sm_70,compute_70] \

-gencode arch=compute_75,code=[sm_75,compute_75] \

-gencode arch=compute_80,code=[sm_80,compute_80] \

-gencode arch=compute_80,code=[sm_80,compute_80]
```

    - GPU를 사용하기 위해 0 → 1로 바꿔준다
    - ARCH 변수는 nvcc 컴파일러에 사용되는 값으로 GPU모델과 cuda 버전에 따라 다르게 설정해야 오류 없이 최대의 성능을 낼 수 있다.
      - 아래 주소에 가면 GPU 모델과 cuda 버전에 따른 값들을 확인할 수 있으며 그대로 써주면 된다.

Matching CUDA arch and CUDA gencode for various NVIDIA architectures - Arnon Shimoni

I've seen some confusion regarding NVIDIA's nvcc sm flags and what they're used for:When compiling with NVCC, the arch flag (") specifies the name of the NVIDIA GPU architecture that the CUDA files will be compiled for.Gencodes (") allows for more PTX generations and can be repeated many times for

https://arnon.dk/matching-sm-architectures-arch-and-gencode-for-various-nvidia-cards/



- GPU에 관련된 정보는 nvidia-smi 혹은 nvidia-smi -q 명령어를 통해 확인할 수 있다.
- CUDA 버전 확인은 nvcc --version 명령으로 확인 가능

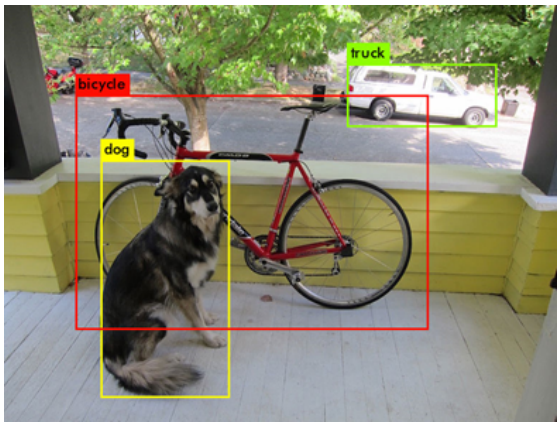
- **make**
- **download pre-trained weight file**

```
wget https://pjreddie.com/media/files/yolov3.weights
```

- test image를 이용한 testing

```
./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg
./darknet detector test cfg/yolov3.cfg yolov3.weights data/dog.jpg
```

  - 위의 두 명령은 같은 작업을 수행



  - default로 제공해주는 가중치(weights)는 yolov3이다.
  - 그 외에 yolov3-tiny 를 테스트하고 싶으면 가중치를 다운 받아야 한다.

```
wget https://pjreddie.com/media/files/yolov3-tiny.weights
```

  - 테스팅 방법은 yolov3와 동일

```
./darknet detect cfg/yolov3-tiny.cfg yolov3-tiny.weights data/dog.jpg
```

  - darknet을 받으면 cfg폴더에 각종 파일(yolov3.cfg, yolov3-tiny.cfg 등)들이 존재

- **학습을 위한 dataset 구성**
  - 데이터셋은 다음과 같은 구조로 구성

```
├── images
│   ├── train2014
│   │   ├── COCO_train2014_000000000009.jpg
│   │   ├── COCO_train2014_000000000025.jpg
│   │   └── ...
│   └── val2014
│       ├── COCO_val2014_000000000042.jpg
│       ├── COCO_val2014_000000000073.jpg
│       └── ...
├── labels
│   ├── train2014
│   │   ├── COCO_train2014_000000000009.txt
│   │   ├── COCO_train2014_000000000025.txt
│   │   └── ...
│   └── val2014
│       ├── COCO_val2014_000000000042.txt
│       ├── COCO_val2014_000000000073.txt
│       └── ...
├── train.txt
└── valid.txt
```

- dataset 다운받고 위와 같은 구조로 설계
  - clone coco API

```
- git clone https://github.com/pdollar/coco
- cd coco
- mkdir images
- cd images
```

  - download images

```
- wget -c https://pjreddie.com/media/files/train2014.zip
- wget -c https://pjreddie.com/media/files/val2014.zip
- unzip -q train2014.zip
- unzip -q val2014.zip
- cd ..
```

  - downlaod coco meta data(train.txt, valid.txt, labels)

```
- wget -c https://pjreddie.com/media/files/instances_train-val2014.zip
- wget -c https://pjreddie.com/media/files/coco/5k.part
- wget -c https://pjreddie.com/media/files/coco/trainvalno5k.part
- wget -c https://pjreddie.com/media/files/coco/labels.tgz
- tar xzf labels.tgz
- unzip -q instances_train-val2014.zip
```

- train.txt, valid.txt 파일 생성 : .part —> .txt

```
- train/valid.txt 파일은 다음과 같은 형식으로 이미지 위치를 저장

    ./images/train2014/COCO_train2014_000000000009.jpg

- paste <(awk "{print \"$PWD\"}" <5k.part) 5k.part | tr -d '\t' > valid.txt
- paste <(awk "{print \"$PWD\"}" <trainvalno5k.part) trainvalno5k.part | tr -d '\t' > train.txt
```

- class는 data/coco.names에서 확인
- cfg/coco.data파일을 수정
  - class 개수 설정:
    - classes = 80(coco의 경우)
  - train, valid 경로 설정
    - train = /workspace/src/darknet/coco/train.txt
    - valid = /workspace/src/darknet/coco/valid.txt
  - 객체 이름에 대한 데이터 위치 설정
    - names = data/coco.names
  - backup 위치 지정
    - backup = /darknet/backtup/
  - 수정 예시

```
classes = 80
train = /workspace/src/darknet/coco/train.txt
valid = /workspace/src/darknet/coco/valid.txt
names = data/coco.names
backup = /workspace/src/darknet/backup/
```

- edge AI를 위한 config 설정

```
[net]
######## Testing
#batch=1
#subdivisions=1
######## Training
batch=64
subdivisions=1
width=256
height=256
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
policy=steps
burn_in=1000
max_batches = 160200
steps=128000,144000
scales=.1,.1

[convolutional]
batch_normalize=1
filters=8
size=3
stride=1
```

```
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
filters=16
size=3
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
filters=64
size=3
stride=1
pad=1
activation=leaky
[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=1

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

###########

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
```

```
activation=leaky

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=leaky

[upsample]
stride=2

[route]
layers = -1, 8

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=255
activation=linear

[yolo]
mask = 0,1,2
anchors = 21,20, 50,47, 115,107
classes=80
num=3
jitter=.3
ignore_thresh = .5
truth_thresh = 1
random=1
```

- 위의 config 파일은 coco 를 기준으로 클래스(80)를 정의한다.
- 만일 클래스 이름과 개수가 바뀌면
  - data/coco.names 변경
  - config 파일 마직막 부분의 [yolo]에서
    - class 개수를 수정
  - max_batches와 steps 부분을 수정
    - max_batches = class x 2000 가 default

- **YOLO 학습 및 log data 수집**
  - 학습을 위한 가중치 받기

```
wget https://pjreddie.com/media/files/darknet53.conv.74
```

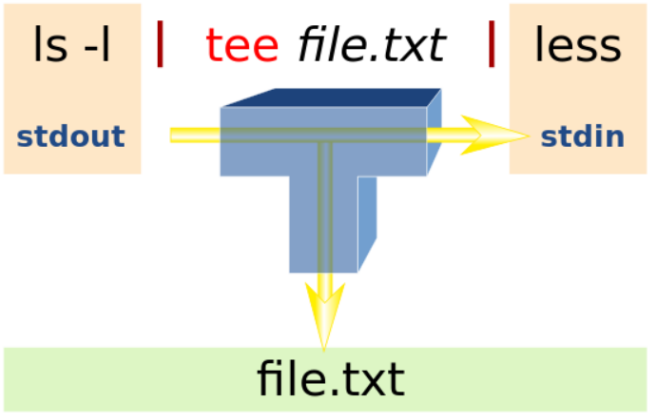
  - examples/detector.c 파일에서 가중치 생성 주기 설정(선택사항)
    - 아래 화면 139 라인에서 10000을 다른 값으로 바꾸면(예:100000) 그 주기마다 가중치가 생성됨

```
132 #ifdef GPU
133             if(ngpus != 1) sync_nets(nets, ngpus, 0);
134 #endif
135             char buff[256];
136             sprintf(buff, "%s/%s.backup", backup_directory, base);
137             save_weights(net, buff);
138         }
139         if(i%10000==0 || (i < 1000 && i%100 == 0)){
140 #ifdef GPU
141             if(ngpus != 1) sync_nets(nets, ngpus, 0);
142 #endif
143             char buff[256];
144             sprintf(buff, "%s/%s_%d.weights", backup_directory, base, i);
145             save_weights(net, buff);
146         }
```

- 학습 시작

```
./darknet detector train cfg/coco.data cfg/YOLOv3-edge.cfg darknet53.conv74 -gpus 1,2 | tee backup/train.log
```

- tee명령은 아래 사진과 같이 출력결과를 파일과 화면에 동시에 출력해준다.

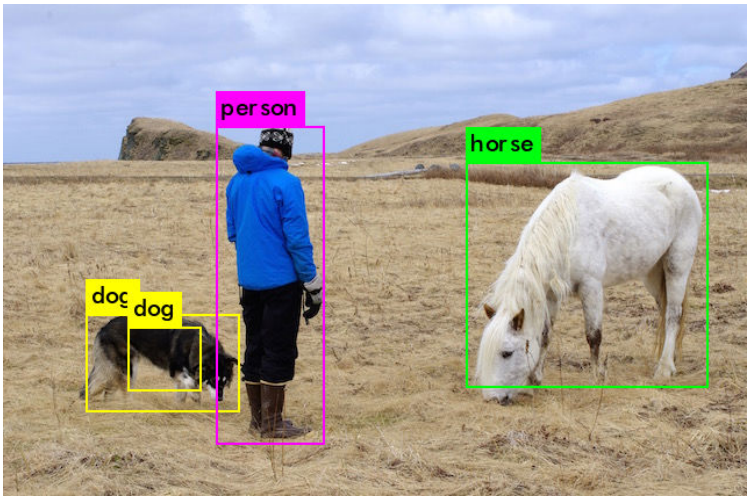


- log data 확인

```
Region 19 Avg IOU: 0.380078, Class: 0.501038, Obj: 0.501237, No Obj: 0.501272, .5R: 0.261628, .75R: 0.013566,  count: 516
Region 19 Avg IOU: 0.391488, Class: 0.501204, Obj: 0.500497, No Obj: 0.501272, .5R: 0.250000, .75R: 0.026923,  count: 520
19: 1050.015625, 645.881531 avg, 0.000000 rate, 1.180288 seconds, 2432 images
Loaded: 0.000065 seconds
Region 19 Avg IOU: 0.371186, Class: 0.501351, Obj: 0.501361, No Obj: 0.501272, .5R: 0.242563, .75R: 0.025172,  count: 437
Region 19 Avg IOU: 0.378779, Class: 0.501099, Obj: 0.500996, No Obj: 0.501272, .5R: 0.235632, .75R: 0.024904,  count: 522
Syncing... Done!
24: 1035.173462, 684.810730 avg, 0.000000 rate, 0.582088 seconds, 3072 images
Loaded: 0.000075 seconds
Region 19 Avg IOU: 0.384205, Class: 0.500854, Obj: 0.500701, No Obj: 0.501272, .5R: 0.291353, .75R: 0.024436,  count: 532
Region 19 Avg IOU: 0.408792, Class: 0.500437, Obj: 0.499588, No Obj: 0.501272, .5R: 0.315650, .75R: 0.047745,  count: 377
25: 1029.475342, 719.277222 avg, 0.000000 rate, 0.511656 seconds, 3200 images
Loaded: 0.000067 seconds
Region 19 Avg IOU: 0.405072, Class: 0.501078, Obj: 0.500220, No Obj: 0.501272, .5R: 0.324380, .75R: 0.039256,  count: 484
Region 19 Avg IOU: 0.385631, Class: 0.501173, Obj: 0.500780, No Obj: 0.501272, .5R: 0.266533, .75R: 0.020040,  count: 499
26: 1042.409058, 751.590393 avg, 0.000000 rate, 0.517428 seconds, 3328 images
Loaded: 0.000092 seconds
Region 19 Avg IOU: 0.406810, Class: 0.501086, Obj: 0.500401, No Obj: 0.501272, .5R: 0.300505, .75R: 0.037879,  count: 396
Region 19 Avg IOU: 0.386672, Class: 0.501188, Obj: 0.500921, No Obj: 0.501272, .5R: 0.290553, .75R: 0.017825,  count: 561
27: 1035.857056, 780.017090 avg, 0.000000 rate, 0.515751 seconds, 3456 images
Loaded: 0.000093 seconds
Region 19 Avg IOU: 0.393334, Class: 0.501412, Obj: 0.500737, No Obj: 0.501272, .5R: 0.251037, .75R: 0.035270,  count: 482
Region 19 Avg IOU: 0.381309, Class: 0.501135, Obj: 0.500748, No Obj: 0.501272, .5R: 0.257606, .75R: 0.018256,  count: 493
Syncing... Done!
32: 1039.739014, 805.989258 avg, 0.000000 rate, 0.603808 seconds, 4096 images
Loaded: 0.000081 seconds
```

- **학습 결과 확인**
  - 테스트한 결과는 /darknet/predictions.jpg로 출력

```
./darknet detect cfg/yolov3-edge.cfg backup/yolov3-edge_final.weights data/person.jpg
```



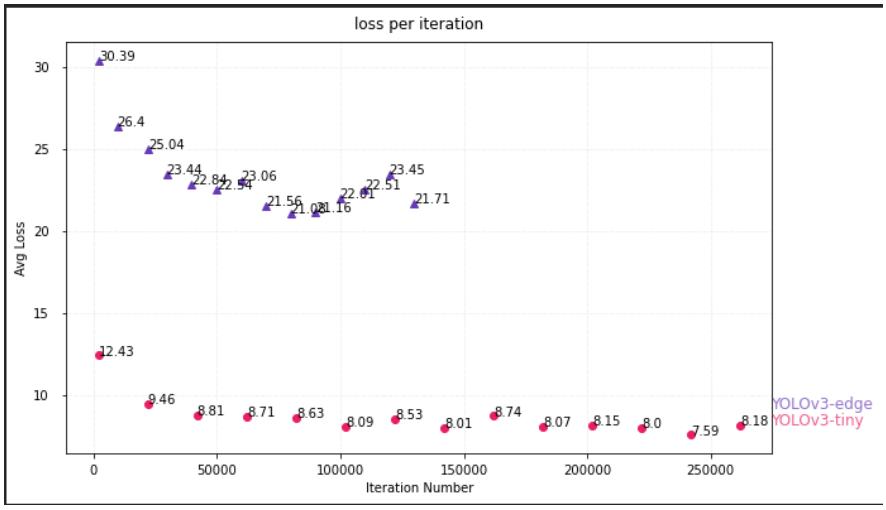
  - log data를 분석하여 원하는 간격으로 Avg-loss값을 측정하고 학습시간 확인
  - 아래 코드는 yolo-tiny.log 와 yolo-edge.log를 비교하여 분석하는 코드

```
import sys
import matplotlib.pyplot as plt
from tqdm import tqdm
import math

tiny_lines = []
edge_lines = []
tiny_time = []
edge_time = []

# log 데이터 불러오기
for line in tqdm(open('tiny_train.log')):
    if "avg" in line:
        tiny_lines.append(line)
    elif "Loaded" in line:
        tiny_time.append(line)

for line in tqdm(open('edge_train.log')):
    if "avg" in line:
        edge_lines.append(line)
    elif "Loaded" in line:
        edge_time.append(line)

# log data에서 avg losss만 추출
def extract_log_avg(tiny_lines, edge_lines):
    tiny_iterations = []
    tiny_avg_loss = []
    edge_iterations = []
    edge_avg_loss = []

    tiny_line_cnt=len(tiny_lines)
    edge_line_cnt = len(edge_lines)

    if edge_line_cnt < tiny_line_cnt:
        tiny_line_cnt = edge_line_cnt
        tiny_lines = tiny_lines[:edge_line_cnt]

    else:
        edge_line_cnt = tiny_line_cnt
        edge_lines = edge_lines[:tiny_line_cnt]
```

```
    for i in tqdm(range(tiny_line_cnt)):
        tiny_lineParts = tiny_lines[i].split(',')
        edge_lineParts = edge_lines[i].split(',')
        tiny_iterations.append(int(tiny_lineParts[0].split(':')[0]))
        tiny_avg_loss.append(float(tiny_lineParts[1].split()[0]))
        edge_iterations.append(int(edge_lineParts[0].split(':')[0]))
        edge_avg_loss.append(float(edge_lineParts[1].split()[0]))

    return tiny_iterations, tiny_avg_loss, edge_iterations, edge_avg_loss

# 시간만 추출
def extract_log_time(tiny_time, edge_time):
    size = None
    if len(tiny_time) > len(edge_time):
        size = len(edge_time)
        tiny_time = tiny_time[:size]
    else:
        size = len(tiny_time)
        edge_time = edge_time[:size]

    tiny_x = []
    edge_x = []
    tiny_y = []
    edge_y = []

    for i in tqdm(range(size)):
        tiny_lineParts = tiny_time[i].split(' ')
        edge_lineParts = edge_time[i].split(' ')
        if float(tiny_lineParts[1]) >= 0.0005 or float(edge_lineParts[1]) >= 0.0005:
            continue
        tiny_y.append(float(tiny_lineParts[1]))
        edge_y.append(float(edge_lineParts[1]))
        tiny_x.append(i)
        edge_x.append(i)
    return tiny_x, tiny_y, edge_x, edge_y


def draw_avg_graph():

    tiny_x, tiny_y, edge_x, edge_y = extract_log_avg(tiny_lines, edge_lines)
    tiny_min = min(tiny_y)
    t_batch = tiny_x[tiny_y.index(tiny_min)]
    edge_min = min(edge_y)
    e_batch = edge_x[edge_y.index(edge_min)]
    print(tiny_min, t_batch, edge_min, e_batch)
    fig, ax = plt.subplots(1,1 , figsize = (10,6)) #start = 0
    ax.xaxis.set_tick_params(pad = 5)
    ax.yaxis.set_tick_params(pad = 10)
    ax.grid(b = True, color ='grey',
            linestyle ='-.', linewidth = 0.5,
            alpha = 0.2)
    plt.figure(figsize = (10,6))
    start = 1000
    term = 10000

    for i in range(start, len(tiny_x), term):
        ax.plot(tiny_x[i:i+1], tiny_y[i:i+1], color="#e91e63", marker = 'o', linestyle = '-')
        ax.plot(edge_x[i:i+1], edge_y[i:i+1], color="#673ab7", marker = '^', linestyle = '-')
        #print(iterations[i], avg_loss[i])m
        ax.text(tiny_x[i], tiny_y[i], str(round(tiny_y[i], 2)))
        ax.text(edge_x[i], edge_y[i], str(round(edge_y[i], 2)))

    ax.set_xlabel('Iteration Number')
    ax.set_ylabel('Avg Loss')
    ax.set_title('loss per iteration', pad = 10)

    fig.text(0.9, 0.2, "YOLOv3-tiny", fontsize = 12, color = '#e91e63', ha = 'left', va='top', alpha = 0.7)
    fig.text(0.9, 0.2, "YOLOv3-edge", fontsize = 12, color = '#673ab7', ha = 'left', va='bottom', alpha = 0.7)

def draw_time_graph():
    tiny_x, tiny_y, edge_x, edge_y = extract_log_time(tiny_time, edge_time)
    fig1, ax1 = plt.subplots(1,1 , figsize = (10,6)) #start = 0
    ax1.xaxis.set_tick_params(pad = 5)
    ax1.yaxis.set_tick_params(pad = 10)
    ax1.grid(b = True, color ='grey',
            linestyle ='-.', linewidth = 0.5,
            alpha = 0.2)
    tiny_x = tiny_x[::4000]
    tiny_y = tiny_y[::4000]
    edge_x = edge_x[::4000]
```

```
    edge_y = edge_y[::4000]
    ax1.set_xlabel('batch Number')
    ax1.set_ylabel('execution time')
    ax1.set_title('time per batch(second)', pad=10)
    ax1.plot(tiny_x, tiny_y, color="#e91e63", marker = 'o', linestyle = '-')
    ax1.plot(edge_x, edge_y, color="#673ab7", marker = '^', linestyle = '-')

    fig1.text(0.9, 0.2, "YOLOv3-tiny", fontsize = 12, color = '#e91e63', ha = 'left', va='top', alpha = 0.7)
    fig1.text(0.9, 0.2, "YOLOv3-edge", fontsize = 12, color = '#673ab7', ha = 'left', va='bottom', alpha = 0.7)

if __name__ == '__main__':
  draw_avg_graph()
  draw_time_graph()
```

- Avg loss 분석 결과



- 학습시간 분석 결과