

JETSON NANO 기반 차량 인식 포트폴리오

컴퓨터공학과 박종하

목차

1. 초기 환경 세팅
2. Linux 및 Container
3. Object detection
4. 추가적인 실습 (Classify, Semantic segmentation)
5. Transfer learning & Re-training Network
6. 차량 인식 시스템 개발
 - 6.1 차량 데이터 수집
 - 6.2 학습에 필요한 주요 알고리즘과 코드
 - 6.2.1 개요
 - 6.2.2 알고리즘과 코드
 - 6.3 최종 결과물
7. 기타 (certification)

학습 목표: Single Board Computer(SBC) (Jetson nano 를 이용) 를 통해 DNN 기반의 Object recognition 수행.

1. 초기 환경 세팅

1.1 필요한 사전 지식

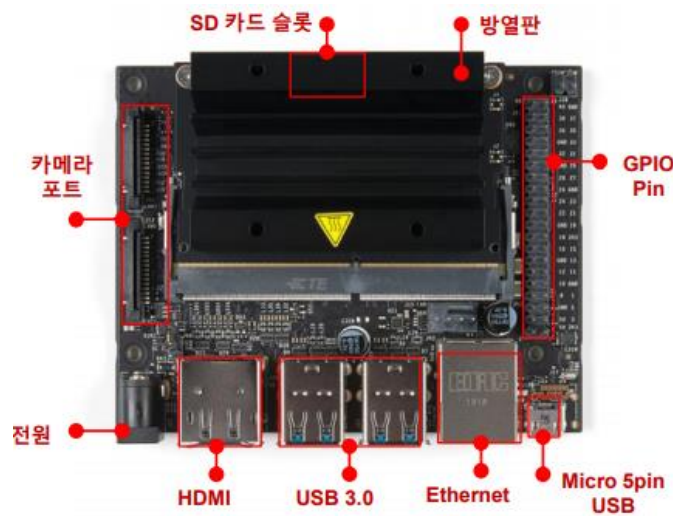
Linux, Docker, Pytorch, ONNX, TensorRT, CNN, Transfer learning, Data pre-processing, feature engineering 등을 학습.

1.2 Jetson 이란?

- 2019 년 6 월에 처음 출시, 임베디드 AI 개발을 위한 Single board computer (SBC)
- GPU 는 Nvidia Maxwell 기반 128 CUDA 코어 사용 (CPU 128 개와 유사한 연산 처리 가능)

Jetson nano development kit 에는 여러 종류가 있는데, 그 중 Jetson nano 를 이용한다.
Jetson 은 AI 개발 또는 로봇틱스 개발에 적합하다. 컴퓨터나 서버를 이용하기 보다는 Jetson 을 이용하는 이유는 이동식 Portable 시스템에서, Server + Cloud 에 비해 높은 반응 속도, 보안 확보, 비용 절감 등의 장점이 있기 때문이다.

1.3 Jetson 의 내부 구조



jetson nano 는 다음과 같은 구조를 이룬다.

1.4 Jetson 연결 방법

모니터에 직접 연결하는 방법과 원격으로 접속하는 Headless 한 방법이 있다. 모니터에 직접 연결하면, 쉽고 빠르다는 장점이 있지만, 이동식 시스템에 적용하기 어렵다. 반면 외부 PC 에 원격으로 접속하는 방법은 이동할 수 있지만, 느리다는 단점이 있다. 그중 원격으로 접속하는 Headless 한 방법을 이용하였다.

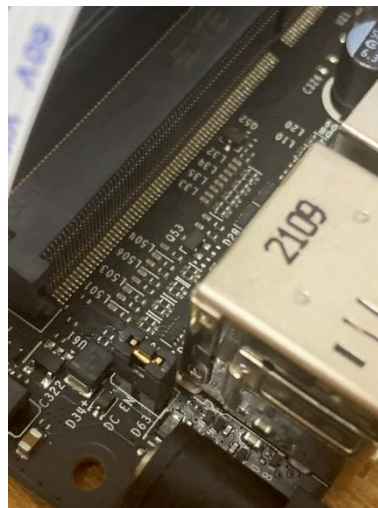
초기 부팅과 자동 로그인 세팅이 완료된 후, 외부 PC 에 연결을 하였다.

1.4.1. SD 카드 입력



검은색 방열판 아래에 SD카드를 포트에 삽입한다.

1.4.2 전원 근처 두 핀에 연결



1.4.3 전원 연결



1.4.4 Micro 5pin USB 연결



나는 노트북과 연결을 하였다.

1.4.5 Wifi 동글 USB port 연결



1.4.6 카메라 연결



라즈베리파이는 파란색이 앞이 보이도록 꽂아준다.

1.4.7 외부 PC 와 연결하기 위해 Putty 를 실행

USB 직렬 장치 번호를 통해, Jetson nano 원격 접속을 한다.

1.4.8 VNC 접속을 위한 설정

```
cd /usr/lib/systemd/user/graphical-session.target.wants  
  
sudo ln -s ../vino-server.service ./.  
  
gsettings set org.gnome.Vino prompt-enabled false  
  
gsettings set org.gnome.Vino require-encryption false  
  
gsettings set org.gnome.Vino authentication-methods "['vnc']"  
  
gsettings set org.gnome.Vino vnc-password $(echo -n 'sogang'|base64)  
  
sudo reboot
```

접속을 하기 위한 코드는 다음과 같다.

1.4.9 VNC 이용

VNC 를 이용하여 접속한다.

1.4.10 Nano editor 설치

인터넷 연결이 끝났으면, `sudo apt-get install nano` 를 이용하여 Nano editor 를 설치해준다.

1.4.11 CSI 카메라 테스트

```
gst-launch-1.0 nvarguscamerasrc sensor_mode=0 ! 'video/x-  
raw(memory:NVMM),width=3820, height=2
```

Gstream를 테스트하기 위한 코드인 `gst-launch`를 통해 확인할 수 있다. 이를 통해, 카메라 테스트를 할 수 있다

2. Linux 및 Container

2.1 Linux 설명

Linux 는 1991 년 9 월 17 일 리누스 토르발스가 처음 출시한 운영 체제 커널인 리눅스 커널에 기반을 둔 오픈 소스 유닉스 계열 운영 체제 계열이다.

2.2 Docker 설명

Docker: Linux Container 를 만들고 사용할 수 있도록 하는 오픈소스 기술 (Container 실행을 위한 운영체제)

Container: 라이브러리, 시스템 도구, 코드, 런타임 등 S/W 실행에 필요한 도구들을 포함하는 패키지

Jetson container 는 GPU 기반 AI 개발을 위한 CUDA/cuDNN/TensorRT/Pytorch/Tensorflow/JupyterLab/ROS/DeepStream 가 설치 되어있음

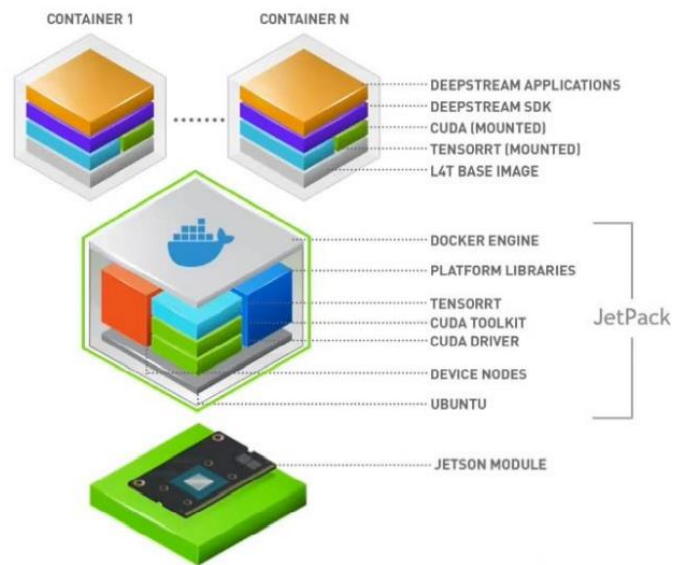


그림. Jetson container 구조

현재 하고 있는 앱 어플리케이션 CI/CD pipeline 자동화 과정에서 사용해서 꽤나 익숙하였다. Docker 의 장점은 어떤 환경에서도 빠르게 실행할 수 있다는 것!

2.3 Linux 환경에서 Docker container 설치

```
git clone --recursive https://github.com/dusty-nv/jetson-inference
cd jetson-inference
docker/run.sh
```

다음과 같은 코드를 입력하면 Docker container 설치가 가능하다!

2.4 Docker container 를 이용한 카메라 세팅

정상적으로 위의 과정을 실행하면, Docker container 내에서 (즉 셸 내에서) 카메라를 구동할 수 있다. 이는 다음과 같은 코드로 가능하다.

```
video-viewer --input-width=400 --input-height=300 csi://0
```

width, height 변수를 통해, video 화면 크기를 조절할 수 있다. 컨트롤 C 를 통해 gst Camera 파이프 라인을 종료할 수 있다.

3. Object detection

3.1 이론

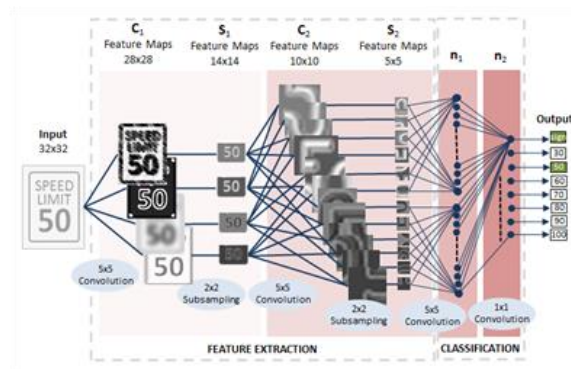
3.1.1 object detection

- input : 이미지 , 영상, 실시간
- DNN : 인공신경망으로, 라벨링된 입력데이터를 통해 높은 정확도의 학습 결과를 나타냄.
- output : 정확도, 무엇인지 출력

object detection 는 CNN 이 하는 Classification 외에도 localization 기능을 수행한다.

3.1.1 Convolutional neural network (CNN) – 합성곱 신경망
 시각적 이미지 처리 할 때 쓰인다. 입력 이미지로부터 특징을 추출하여 어떤 이미지인지 클래스 분류. 일반 신경망일 때, 입력을 하나의 데이터로 인식하여, 이미지의 특성을 찾지 못하여 올바른 성능 기대할 수 없음. 하지만 CNN 은 여러 개로 분할 하여 처리. **왜곡되더라도 부분적 특성을 추출!** 신경망 구조에서 합성곱 계층과 풀링 계층이 추가가 됨.

이미지 -필터 - 분류 학습기 -Classification 의 구조로 구성이 됨.



3.2 실습 구조 : Pytorch -> ONNX ->TensorRT

1. AI framework (Pytorch) 를 이용해 딥러닝 네트워크를 구성
2. Pre-trained network model
 Image classification, Object detection, Semantic segmentation
 나는 여기서 Object detection 의 SSD-Mobilenet 을 사용함
 SSD-Mobilenet-v2 (91 종류의 객체를 포함하는 MS COCO dataset 을 학습한 모델)
3. ONNX
 서로 다른 Framework 사이에서 네트워크 모델 이동할 수 있음
 ONNX 를 통해 Pytorch 에서 TensorRT 로 연결하기 위해 사용한다.
4. Tensor RT
 훈련된 네트워크 모델을 실시간으로 돌릴 수 있는 프로그램

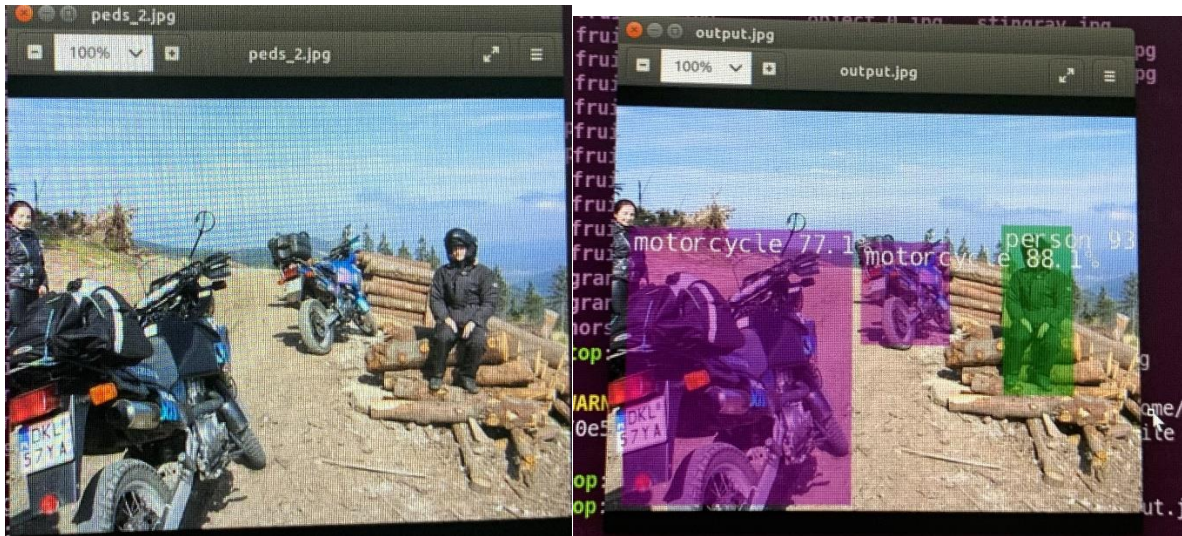
3.3 실습 결과

3.3.1 입력 : 이미지 -> 객체 인식

- Docker container 내로 접속한다.

```
detectnet.py --network=ssd-mobilenet-v2 data/images/peds_2.jpg
data/images/test/output.jpg
```

여기서는 peds_2.jpg 이미지를 처리한 결과가 output.jpg 로!



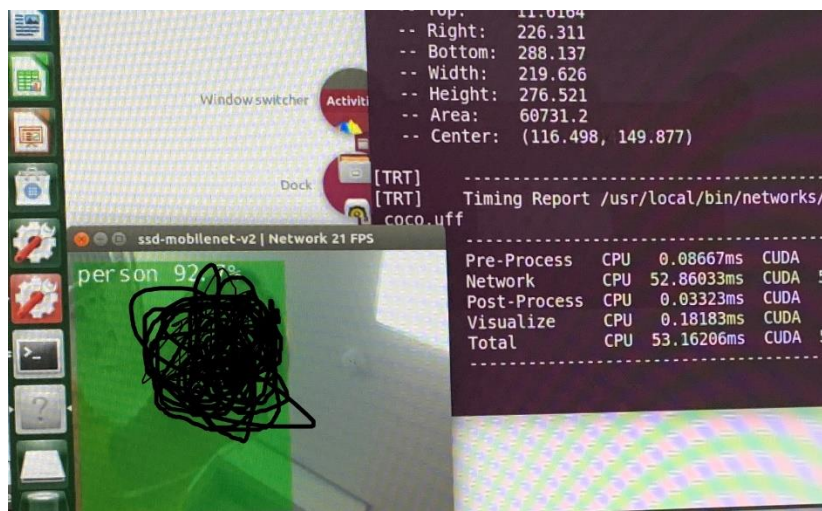
정확히 BOX 로 그려내어 localization 하고, Tag 로 Classification 되어 object detection 이 잘 일어남을 확인할 수 있다.

3.3.2 입력 : 실시간 -> 객체 인식

- Docker container 내로 접속한다.
그 후 다음과 같은 코드를 입력한다.

```
detectnet.py --network=ssd-mobilenet-v2 --input-width=400 --input-height=300
csi://0
```

그 결과, 카메라로 실시간으로 데이터 입력이 주어져도 객체를 잘 인식되는 것을 확인할 수 있다.

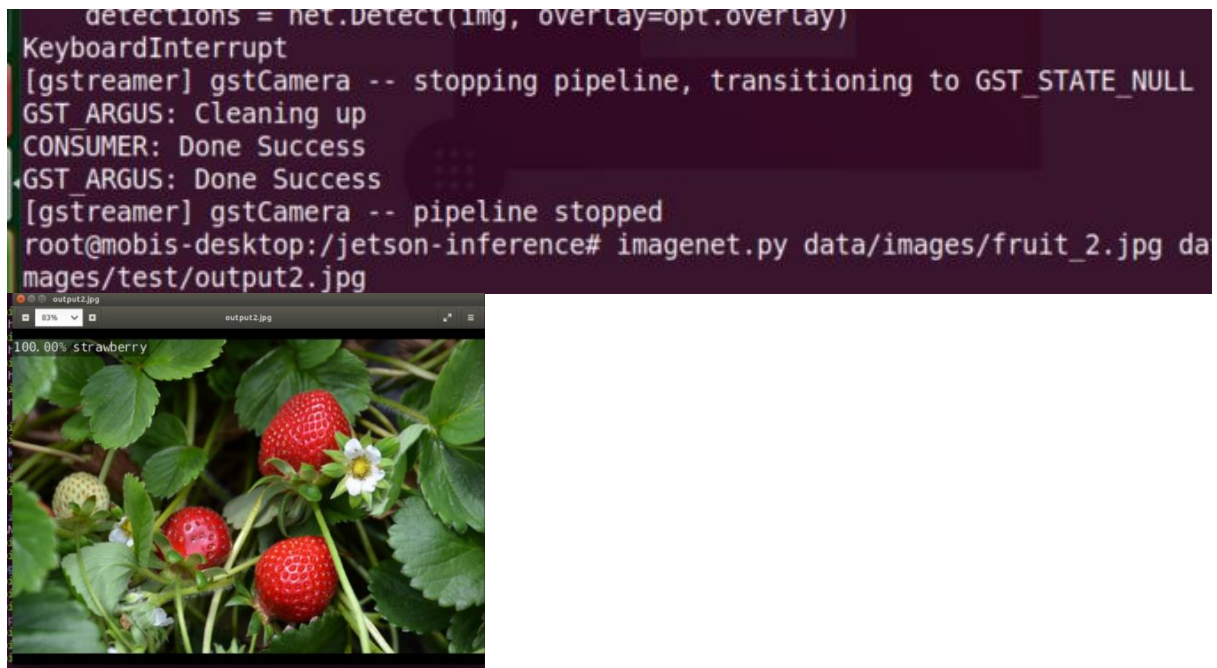


나를 사람으로 인식한 것을 확인할 수 있다.

4. 추가적인 실습(Classify, Semantic segmentation)

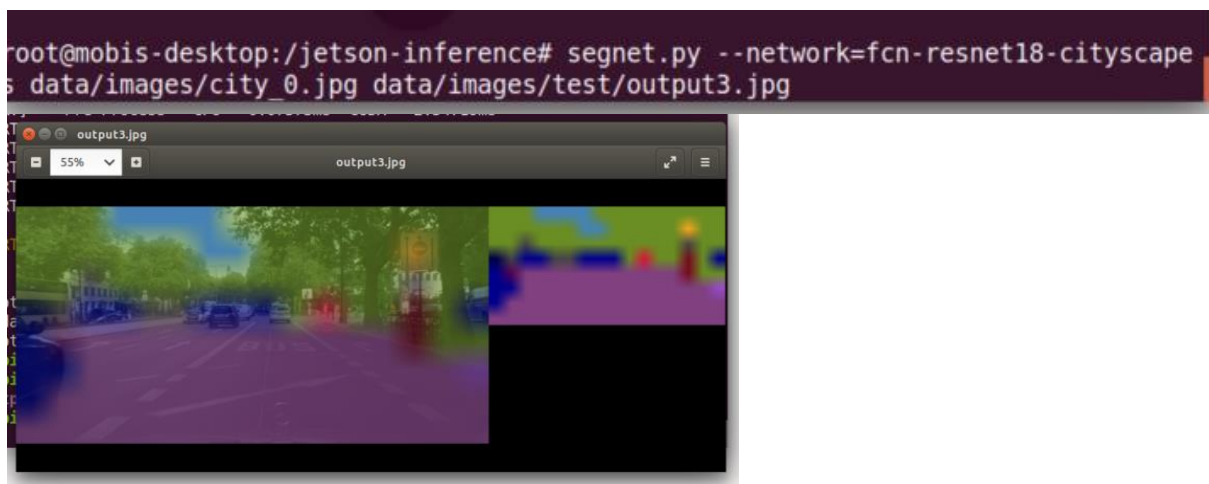
4.1. Classifying Images with ImageNet

```
imagenet.py images/strawberry_0.jpg images/test/output_1.jpg
```



결과가 딸기라고 잘 분류(Classification)됨을 확인할 수 있다.

4.2. Semantic Segmentation with SegNet



```
segnet.py --network=fcn-resnet18-cityscapes images/city_0.jpg images/test/output.jpg
```

세그멘테이션 코드 또한 정상적으로 작동함을 확인할 수 있다.

5. Transfer Learning

5.1 사전 이론

5.1.1 Transfer learning

research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem

목적에 맞게 적은 데이터를 이용하여, fine tuning 하여 높은 정확도 도출한다.

5.1.2 Fine tuning

딥러닝에선 Deep neural network 의 일부 파라미터를 조정하는 뜻으로 쓰임. 데이터 크기, 데이터 유사도에 따라 4 가지로 구분할 수 있음

5.1.3 실습 이론

- Training set, Test set : 전체 데이터 중, 네트워크 모델 훈련에 사용할 데이터셋과 훈련된 모델을 테스트할 때 사용하는 데이터 셋
- Batch size : 트레이닝 데이터 전체를 여러 개의 작은 그룹으로 나눌 때, 그룹 안에 속하는 데이터의 수 . 데이터를 통째로 이용하여 학습시키면, 비효율적인 리소스 사용으로 시간이 오래걸림
- Epochs : 전체 데이터가 학습에 사용된 횟수.

5.2. 실습 과정

5.2.1 Memory Swap

용량이 큰 프로그램을 사용하다 보면 freezing 현상이 생길 수도 있기 때문에 불편함을 없애고자 먼저 Memory Swap을 수행한다.

```
git clone https://github.com/JetsonHacksNano/installSwapfile
cd installSwapfile
./installSwapfile.sh
sudo reboot
```

5.2.1 Docker 밖에 데이터 저장

높은 정확도와 여러 Class 를 분류하기 위해서 또, CNN 과 같은 딥러닝 네트워크는 절대적으로 많은 Data set 에 중요성을 두고 있으므로 SD card 의 용량보다 더 큰 추가 공간을 위해 외장 메모리를 따로 구비하여 jetson nano 의 usb port 에 연결한다.

```
sudo apt-get install exfat-fuse exfat-utils
```

다음과 같은 코드로 외장 메모리 인식을 위한 파일을 설치할 수 있다.

```
cd jetson-inference
docker/run.sh --volume /media/mobis/mobis_usb:/media/mobis/mobis_usb
```

다음과 같이 Docker 내부에 접속 가능하다.

5.2.2 데이터 다운로드

Detectnet.py 를 통해 다운로드 되는 데이터는 다음과 같다.

1) Open Images

구글에서 제공하는 openimages 를 이용한다.

- 필요한 환경을 셋업.

```
sudo apt-get install python3-pip

sudo pip3 install boto3

cd jetson-inference/python/training/detection/ssd

wget https://nvidia.box.com/shared/static/djf5w54rjvpqocsiztzaandq1m3avr7c.pth -O
models/mobilenet-v1-ssd-mp-0_675.pth

pip3 install -v -r requirements.txt
```

이미지 데이터가 하드 디스크에 저장되도록 환경을 셋업한다.

- 다운로드 여부 확인 실험

```
python3 open_images_downloader.py --stats-only --max-annotations-per-class=10 --
class-names "Apple"
```

- 데이터 다운로드

```
python3 open_images_downloader.py --max-images=10 --class-names "Apple"
```

2) 외부 양식

<https://github.com/dusty-nv/jetson-inference/blob/master/docs/pytorch-collect-detection.md>
다음은 참고할 수 있다. Pascal VOC 형식을 지원한다.

이를 통해 외부의 데이터를 가져와서 object detection 을 확인할 수 있다.

5.2.3 Re-training

Docker 내부에서

- USB

```
python3 train_ssd.py --data=/media/mobis/mobis_usb --model-dir=models --batch-size=4
--epochs=5
```

- SD 카드

```
python3 train_ssd.py --data=data --model-dir=models --batch-size=4 --epochs=5
```

train_ssd.py 를 이용해 Re-training 을 실행하며 데이터 batch size, 수행 단계인 epochs 의 size 를 결정합니다.

5.2.4 Export ONNX

위치 : jetson-inference/python/training/detection/ssd

```
python3 onnx_export.py --model-dir=models
```

5.2.5 Test

위치 : jetson-inference/python/training/detection/ssd

- 이미지/ 비디오

```
detectnet.py --model=models/ssd-mobilenet.onnx --labels=models/labels.txt --input-blob=input_0 --output-cvg=scores --output-bbox=boxes "$IMAGES/testImage(*.jpg)"  
$IMAGES/output.jpg
```

- 카메라

```
detectnet.py --model=models/ssd-mobilenet.onnx --labels=models/labels.txt --input-blob=input_0 --output-cvg=scores --output-bbox=boxes --input-width=800 --input-height=400 csi://0
```

영상을 테스트하고 싶다면 영상을 입력으로 넣고, 출력에 output.mp4 를 하면 되며 여러 이미지를 한꺼번에 테스트 하고 싶은 경우, 입력에 testImage 를 비우고("\$IMAGES/"), 출력에 \$IMAGES/.jpg 로 쓰면 가능하다.

5.3 실습 결과

5.3.1 Docker 밖에 데이터 저장

```
192.168.55.1 (mobis@mobis-desktop) - VNC Viewer
mobis@mobis-desktop:~$ ls /media/mobis/Elements
신호등-도로표지판 인지 영상 (수도권)
$RECYCLE.BIN
System Volume Information
traffic
mobis@mobis-desktop:~$ ls
Desktop      examples.desktop  Pictures      Videos
Documents    jetson-inference  Public
Downloads    Music             Templates
mobis@mobis-desktop:~$ ls /media/mobis
Elements  mobis_usb  recup_dir.1
L4T-README  photorec.ses
mobis@mobis-desktop:~$
```

5.3.2 데이터 다운로드

```
Bounding box distribution:
-----
'validation' set statistics
-----
Image count: 0
Bounding box count: 0
Bounding box distribution:
-----
'test' set statistics
-----
Image count: 9
Bounding box count: 12
Bounding box distribution:
  Apple: 12/12 = 1.00
-----
Overall statistics
-----
Image count: 9
Bounding box count: 12
2021-07-23 13:59:37 - Saving 'train' data to data/sub-train-annotations-bbox.csv.
2021-07-23 13:59:37 - Saving 'validation' data to data/sub-validation-annotations-bbox.csv.
2021-07-23 13:59:37 - Saving 'test' data to data/sub-test-annotations-bbox.csv.
2021-07-23 13:59:37 - Starting to download 9 images.
2021-07-23 14:00:19 - Task Done.
mobis@mobis-desktop:~/jetson-inference/python/training/detection/ssd$
```

다음과 같이 다운로드가 잘 된 것을 확인할 수 있다.

5.3.3 Re-training

```
mb1-ssd-Epoch-0-Loss-10.532293221046185.pth
mb1-ssd-Epoch-0-Loss-10.61061894375345.pth
mb1-ssd-Epoch-0-Loss-11.700587811677352.pth
mb1-ssd-Epoch-1-Loss-9.245870901190717.pth
mb1-ssd-Epoch-1-Loss-9.664303483634159.pth
mb1-ssd-Epoch-2-Loss-10.586458765227219.pth
mb1-ssd-Epoch-2-Loss-9.359067046124002.pth
mb1-ssd-Epoch-3-Loss-8.224548091059146.pth
mb1-ssd-Epoch-3-Loss-9.18705254587634.pth
mb1-ssd-Epoch-4-Loss-8.795799898064654.pth
mb1-ssd-Epoch-5-Loss-8.280723654705545.pth
mb1-ssd-Epoch-6-Loss-7.829692861308223.pth
mb1-ssd-Epoch-7-Loss-7.905657861543738.pth
```

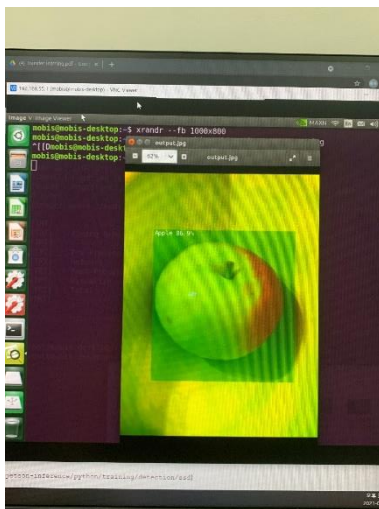

epoch 당 STEP 에 따라 학습을 수행하는 것을 보여준다.

5.3.4 Export ONNX

```
ssd-mobilenet.onnx  
ssd-mobilenet.onnx.1.1.7103.GPU.FP16.engine
```

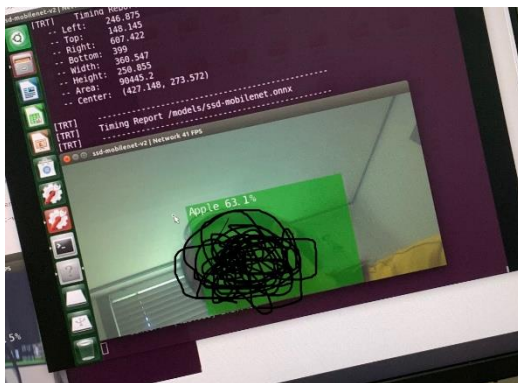
5.3.5 Test

- 이미지



알맞게 사과로 인식함을 확인할 수 있다.

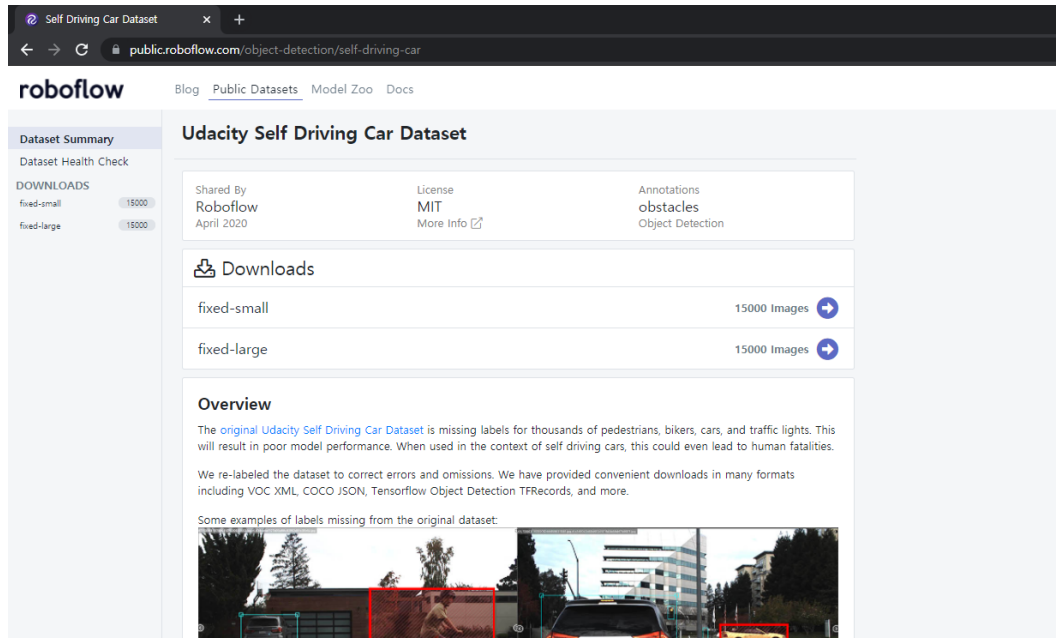
- 카메라



아직 사과 밖에 학습을 시키지 않아, 사람을 사과로 인식한다.

6. 차량 인식 시스템 개발

6.1 차량 데이터



Class Balance

car	64,399	<div></div>	over represented
pedestrian	10,806	<div></div>	
trafficLight-Red	6,870	<div></div>	
trafficLight-Green	5,465	<div></div>	under represented
truck	3,623	<div></div>	under represented
trafficLight	2,568	<div></div>	under represented
biker	1,864	<div></div>	under represented
trafficLight-RedLeft	1,751	<div></div>	under represented
trafficLight-GreenLeft	310	<div></div>	under represented
trafficLight-Yellow	272	<div></div>	under represented
trafficLight-YellowLeft	14	<div></div>	under represented

Images 는 총 30000 만장으로, 그중 15000 개의 데이터만 학습시켰다. Car 데이터에 대해 집중적으로 분포되어 있어 차량 인식에 필요한 데이터로 적합하였다. 한 그림당 여러 개의 객체가 존재할 수 있다.

8ed842a408 ▾ pytorch-ssd / train_ssd.py / <> Jump to ▾

dusty-nv updated command-line options

3 contributors

357 lines (311 sloc) | 15.7 KB

```
1 #
2 # train an SSD model on Pascal VOC or Open Images datasets
3 #
4 import os
5 import sys
6 import logging
7 import argparse
8 import itertools
9 import torch
10
11 from torch.utils.data import DataLoader, ConcatDataset
12 from torch.optim.lr_scheduler import CosineAnnealingLR, MultiStepLR
13
14 from vision.utils.misc import str2bool, Timer, freeze_net_layers, store_labels
15 from vision.ssd.ssd import MatchPrior
16 from vision.ssd.vgg_ssd import create_vgg_ssd
17 from vision.ssd.mobilenetv1_ssd import create_mobilenetv1_ssd
18 from vision.ssd.mobilenetv1_ssd_lite import create_mobilenetv1_ssd_lite
19 from vision.ssd.mobilenet_v2_ssd_lite import create_mobilenetv2_ssd_lite
20 from vision.ssd.squeezenet_ssd_lite import create_squeezenet_ssd_lite
21 from vision.datasets.voc_dataset import VOCDataset
22 from vision.datasets.open_images import OpenImagesDataset
23 from vision.nn.multibox_loss import MultiBoxLoss
24 from vision.ssd.config import vgg_ssd_config
25 from vision.ssd.config import mobilenetv1_ssd_config
26 from vision.ssd.config import squeezenet_ssd_config
27 from vision.ssd.data_preprocessing import TrainAugmentation, TestTransform
28
29 parser = argparse.ArgumentParser(
30     description='Single Shot MultiBox Detector Training With PyTorch')
31
32 # Params for datasets
33 parser.add_argument("--dataset-type", default="open_images", type=str,
34     help='Specify dataset type. Currently supports voc and open_images.')
35 parser.add_argument('--datasets', '--data', nargs='+', default=["data"], help='Dataset directory path')
36 parser.add_argument('--balance-data', action='store_true',
37     help='Balance training data by down-sampling more frequent labels.')
38
```

모델을 훈련시키는 코드인 train_ssd.py 를 보면 openimage 에서 이미지를 다운 받아 학습시키는 방법과, pascal voc 형태의 외부데이터를 끌어와 학습시키는 방법이 있다.

```
1 <annotation>
2   <folder></folder>
3   <filename>1478019952686311006_jpg.rf.54e2d12dbabc46be3c78995b6eaf3fee.jpg</filename>
4   <path>1478019952686311006_jpg.rf.54e2d12dbabc46be3c78995b6eaf3fee.jpg</path>
5   <source>
6     <database>roboflow.ai</database>
7   </source>
8   <size>
9     <width>512</width>
10    <height>512</height>
11    <depth>3</depth>
12  </size>
13  <segmented>0</segmented>
14  <object>
15    <name>truck</name>
16    <pose>Unspecified</pose>
17    <truncated>0</truncated>
18    <difficult>0</difficult>
19    <occluded>0</occluded>
20    <bndbox>
21      <xmin>219</xmin>
22      <xmax>227</xmax>
23      <ymin>244</ymin>
24      <ymax>258</ymax>
25    </bndbox>
26  </object>
27  <object>
28    <name>car</name>
29    <pose>Unspecified</pose>
30    <truncated>0</truncated>
31    <difficult>0</difficult>
32    <occluded>0</occluded>
33    <bndbox>
34      <xmin>230</xmin>
35      <xmax>242</xmax>
36      <ymin>247</ymin>
37      <ymax>261</ymax>
38    </bndbox>
39  </object>
40  <object>
41    <name>car</name>
42    <pose>Unspecified</pose>
43    <truncated>0</truncated>
44    <difficult>0</difficult>
45    <occluded>0</occluded>
46    <bndbox>
47      <xmin>254</xmin>
48      <xmax>269</xmax>
49      <ymin>246</ymin>
```

따라서, 다음과 같이 pascal voc 형태의 데이터를 다운 받았다.

	이름	수정한 날짜	유형	크기
	Annotations	2021-07-27 오후 7:44	파일 폴더	
	ImageSets	2021-07-27 오후 6:51	파일 폴더	
	JPEGImages	2021-07-27 오후 7:44	파일 폴더	
	generate.py	2021-07-24 오후 8:19	Python 원본 파일	2KB
	labels.txt	2021-07-27 오후 7:01	텍스트 문서	1KB

다음과 같이 JPEG, xml 파일별로,pascal VOC 형식으로 배치하여 jetson nano data 경로에 저장시켜준다.

또한 학습에 필요한 train set 과 validataion set 을 나누기 위해 이를 위한 코드를 작성하였다.

```

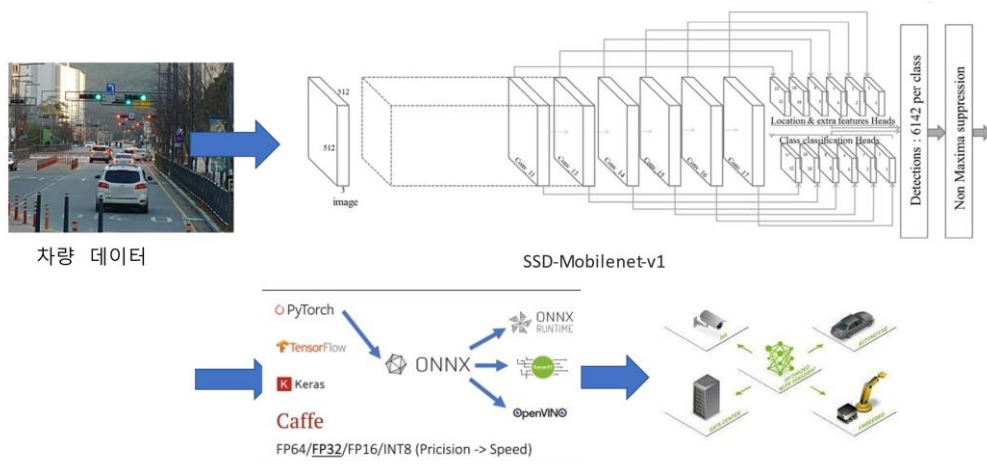
generate.py X
C: > Users > bbc27 > OneDrive > 바탕 화면 > traffic3 > generate.py
4
5
6 root_path = os.getcwd()
7
8 xmlfilepath = root_path + '/Annotations'
9
10 txtsavepath = root_path + '/ImageSets/Main'
11
12 if not os.path.exists(root_path):
13     print("cannot find such directory: " + root_path)
14     exit()
15
16 if not os.path.exists(txtsavepath):
17     os.makedirs(txtsavepath)
18
19 trainval_percent = 0.9
20 train_percent = 0.8
21 total_xml = os.listdir(xmlfilepath)
22 num = len(total_xml)
23 list = range(num)
24 tv = int(num * trainval_percent)
25 tr = int(tv * train_percent)
26 trainval = random.sample(list, tv)
27 train = random.sample(trainval, tr)
28
29 print("train and val size:", tv)
30 print("train size:", tr)
31
32 ftrainval = open(txtsavepath + '/trainval.txt', 'w')
33 ftest = open(txtsavepath + '/test.txt', 'w')
34 ftrain = open(txtsavepath + '/train.txt', 'w')
35 fval = open(txtsavepath + '/val.txt', 'w')
36
37 for i in list:
38     name = total_xml[i][:-4] + '\n'
39     if i in trainval:
40         ftrainval.write(name)
41         if i in train:

```

6.2 학습에 필요한 주요 알고리즘과 코드

6.2.1 개요

개요



먼저 차량에 대한 데이터를 수집하고, SSD-Mobilenet-v1 네트워크 모델을 바탕으로, 새로운 데이터를 학습시키는 transfer learning 을 진행한다. 데이터를 학습시켜 모델을 만들고, 실시간 처리를 위해 ONNX 로 네트워크 모델을 이전하여, TensorRT 에서 불러온다.

6.2.2 알고리즘과 코드

6.2.2.1 Detectnet.py

1. Detectnet.py

```
import argparse
import sys

# parse the command line
parser = argparse.ArgumentParser(description="locate objects in a live camera stream using an object detection DNN.",
                                formatter_class=argparse.RawTextHelpFormatter, epilog=jetson.inference.detectnet.Usage() +
                                jetson.utils.videoSource.Usage() + jetson.utils.videoOutput.Usage() + jetson.utils.logUsage())

parser.add_argument("input_url", type=str, default="", nargs='?', help="URI of the input stream")
parser.add_argument("output_url", type=str, default="", nargs='?', help="URI of the output stream")
parser.add_argument("--network", type=str, default="ssd-mobilenet-v2", help="pre-trained model to load (see below for options)")
parser.add_argument("--overlay", type=str, default="box,labels,conf", help="detection overlay flags (e.g. --overlay=box,labels,conf)\nval")
parser.add_argument("--threshold", type=float, default=0.5, help="minimum detection threshold to use")

is_headless = ["--headless"] if sys.argv[0].find('console.py') != -1 else []

try:
    opt = parser.parse_known_args()[0]
except:
    pass

# load the object detection network
net = jetson.inference.detectnet(opt.network, sys.argv, opt.threshold)

# create video sources & outputs
input = jetson.utils.videoSource(opt.input_url, argv=sys.argv)
output = jetson.utils.videoOutput(opt.output_url, argv=sys.argv+is_headless)

# process frames until the user exits
while True:
    # capture the next image
    img = input.Capture()

    # detect objects in the image (with overlay)
    detections = net.Detect(img, overlay=opt.overlay)

    # print the detections
    print("detected (%d) objects in image".format(len(detections)))

    for detection in detections:
        print(detection)

    # render the image
    output.Render(img)

    # update the title bar
    output.SetStatus("%s | Network FPS: %f" % (opt.network, net.GetNetworkFPS()))

    # print out performance info
    net.PrintProfilerLines()

    # exit on input/output EOS
    if not input.IsStreaming() or not output.IsStreaming():
        break
```

Detectnet.py 는 고정된 이미지에 객체를 인식할 수 있도록 하는 코드로, 좌측을 보시면 인자를 설정하여 네트워크, input, output url 등을 설정할 수 있다. 코드를 간단하게 설명하자면, 인자로 받은 네트워크를 jetson 의 객체 인식 네트워크로 올리고 Jetson.utils.videoSource 함수를 통해 input 과 output 을 생성한다. 밑에 while 문은 카메라를 사용할 때 멈추지 않고 지속적으로 frame 을 받기 위해 사용한다. 각 이미지를 캡처하고, 이미지 안에 있는 오브젝트 들을 detect 합니다. 그리고 분석한 결과를 print 해준다. 그 후 output image 를 생성하는 방식으로 구성된다.

6.2.2.2 train_ssd.py

1) 알고리즘

2. train_ssd.py

```
def train(loader, net, criterion, optimizer, device, debug_steps=100, epoch=-1):
    net.train(True)
    running_loss = 0.0
    running_regression_loss = 0.0
    running_classification_loss = 0.0
    for i, data in enumerate(loader):
        images, boxes, labels = data
        images = images.to(device)
        boxes = boxes.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()
        confidence, locations = net(images)
        regression_loss, classification_loss = criterion(confidence, locations, labels, boxes) # TODO CHANGE BOXES
        loss = regression_loss + classification_loss
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        running_regression_loss += regression_loss.item()
        running_classification_loss += classification_loss.item()
        if i and i % debug_steps == 0:
            avg_loss = running_loss / debug_steps
            avg_reg_loss = running_regression_loss / debug_steps
            avg_cls_loss = running_classification_loss / debug_steps
            logging.info(
                f"Epoch: {epoch}, Step: {i}/{len(loader)}, " +
                f"Avg Loss: {avg_loss:.4f}, " +
                f"Avg Regression Loss {avg_reg_loss:.4f}, " +
                f"Avg Classification Loss: {avg_cls_loss:.4f}"
            )
            running_loss = 0.0
            running_regression_loss = 0.0
            running_classification_loss = 0.0

# train for the desired number of epochs
logging.info(f"Start training from epoch {last_epoch + 1}.")

for epoch in range(last_epoch + 1, args.num_epochs):
    scheduler.step()
    train(train_loader, net, criterion, optimizer,
          device=DEVICE, debug_steps=args.debug_steps, epoch=epoch)

    if epoch % args.validation_epochs == 0 or epoch == args.num_epochs - 1:
        val_loss, val_regression_loss, val_classification_loss = test(val_loader, net, criterion, DEVICE)
        logging.info(
            f"Epoch: {epoch}, " +
            f"Validation Loss: {val_loss:.4f}, " +
            f"Validation Regression Loss {val_regression_loss:.4f}, " +
            f"Validation Classification Loss: {val_classification_loss:.4f}"
        )
        model_path = os.path.join(args.checkpoint_folder, f"{args.net}-{epoch}-{epoch}-Loss-{val_loss}.pth")
        net.save(model_path)
        logging.info(f"Saved model {model_path}")

logging.info("Task done, exiting program.")
```

Train_ssd.py 는 새로운 데이터를 훈련시켜 모델을 만들기 위해 사용한다. Data 를 이용해 train dataset 과 validation dataset 을 구성하고 네트워크를 생성하고, 원하는 epoch 만큼 반복문을 돌며 train 과 test 를 진행하여 모델을 생성한다.

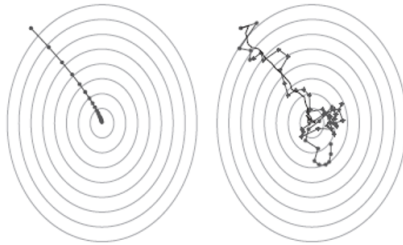
2) SGD

```
# Params for SGD
parser.add_argument('--lr', '--learning-rate', default=0.01, type=float,
                    help='initial learning rate')
parser.add_argument('--momentum', default=0.9, type=float,
                    help='Momentum value for optim')
parser.add_argument('--weight-decay', default=5e-4, type=float,
                    help='Weight decay for SGD')
parser.add_argument('--gamma', default=0.1, type=float,
                    help='Gamma update for SGD')
parser.add_argument('--base-net-lr', default=0.001, type=float,
                    help='initial learning rate for base net, or None to use --lr')
parser.add_argument('--extra-layers-lr', default=None, type=float,
                    help='initial learning rate for the layers not in base net and prediction heads.')
```

Neural Network 의 Weight 를 조정하는 과정에서, 네트워크에서 내놓는 결과값과 실제 값 사이의 차이를 정의하는 Loss Function 의 값을 최소화하기 위해 기울기를 이용하는 Gradient Descent 라는 방법을 사용한다. 다음과 같이 train_ssd.py 에서는 Loss Function 을 계산할 때, 전체 데이터(Batch) 대신 일부 데이터의 모음(Mini-Batch)를 사용하여 Loss Function 을 계산하는 SGD 를 이용한다. Batch Gradient Descent 보다 다소 부정확할 수는 있지만, 계산 속도가 훨씬 빠르기 때문에 같은 시간에 더 많은 step 을 갈 수 있으며, 여러 번 반복할 경우 Batch 처리한 결과로 수렴한다. 또한 Batch Gradient Descent 에서 빠질 Local Minima 에 빠지지 않고 더 좋은 방향으로 수렴할 가능성도 높다.

Base Knowledge

[Stochastic Gradient Descent]

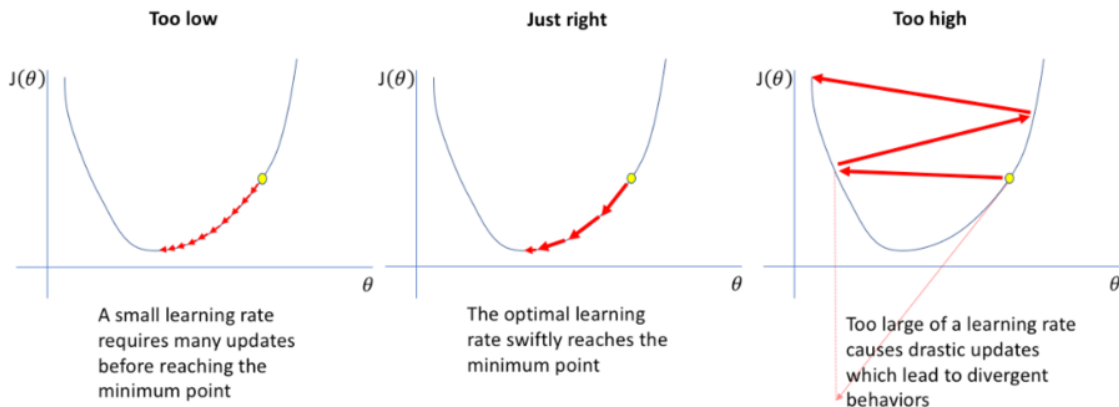


<BGD, SGD>

- ✓ Gradient Descent를 전체 데이터(Batch)가 아닌 **일부 데이터의 모음(Mini-Batch)**를 사용하는 방법
- ✓ BGD(Batch Gradient Descent)는 하나의 step을 위해 전체 데이터를 계산 하므로 **계산량이 많음**
- ✓ SGD(Stochastic Gradient Descent)는 Mini-Batch를 사용하여 다소 부정확할 수는 있지만 **계산 속도가 훨씬 빠르기** 때문에, 같은 시간에 더 많은 Step을 나아갈 수 있음
- ✓ Local Minima에 빠지지 않고 Global Minima에 수렴할 가능성이 더 높음

3) Learning rate scheduler

learning rate 는 gradient 의 보폭을 말한다. learning rate 는 성능에 꽤나 영향을 주는 요소(learning rate 를 잘못 설정하면 아예 학습이 안되기도 한다.)이기 때문에 learning rate 를 어떻게 설정할 지가 중요하다.



처음부터 끝까지 같은 learning rate 를 사용할 수도 있지만, 학습과정에서 learning rate 를 조정하는 learning rate scheduler 를 사용할 수도 있다. 처음엔 큰 learning rate(보폭)으로 빠르게 optimize 를 하고 최적값에 가까워질수록 learning rate(보폭)를 줄여 미세조정을 하는 것이 학습이 잘된다고 알려져있다. learning rate 를 decay 하는 방법이외에도 learning rate 를 줄였다 늘렸다 하는 것이 더 성능향상에 도움이 된다는 연구결과도 있다.

```
# Scheduler
parser.add_argument('--scheduler', default="cosine", type=str,
                    help="Scheduler for SGD. It can one of multi-step and cosine")

# Params for Multi-step Scheduler
parser.add_argument('--milestones', default="80,100", type=str,
                    help="milestones for MultiStepLR")

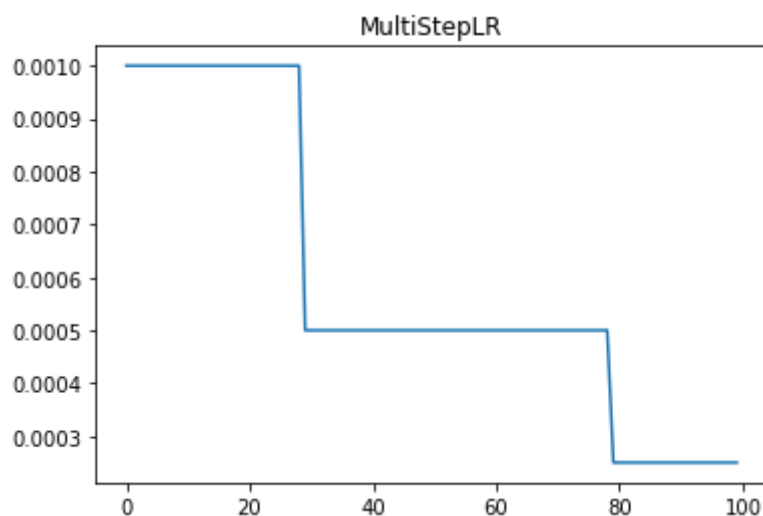
# Params for Cosine Annealing
parser.add_argument('--t-max', default=100, type=float,
                    help='T_max value for Cosine Annealing Scheduler.')
```

Train_ssd.py 코드에서 살펴보면 다음과 같이 scheduler 가 Multi-step Scheduler 방식과 Cosine Annealing 방식이 있음을 확인할 수 있다.

- Multi-step Scheduler

- MultiStepLR : step size 가 아니라 learning rate 를 감소시킬 epoch 을 지정해준다.
- milestones: learning rate 줄일 epoch index 의 list
- gamma: gamma 비율로 lr 을 감소시킨다.

$$lr_{\text{epoch}} = \begin{cases} \text{Gamma} * lr_{\text{epoch} - 1}, & \text{if epoch in [milestones]} \\ lr_{\text{epoch} - 1}, & \text{otherwise} \end{cases}$$



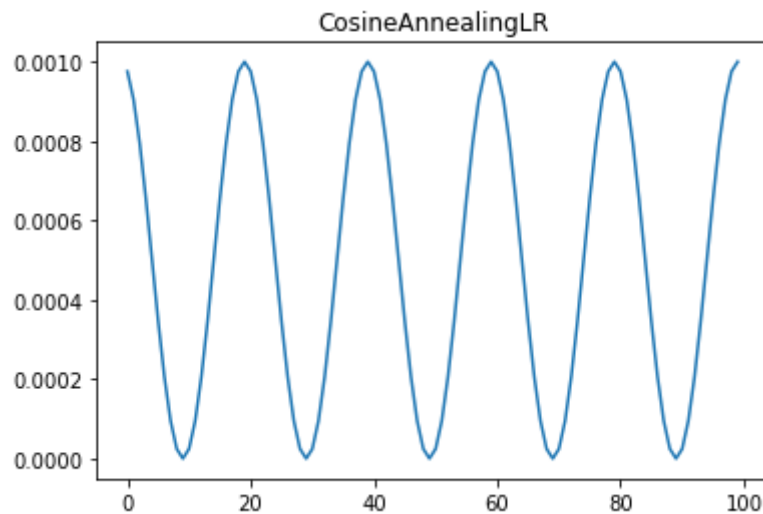
Train_ssd 의 milestone 는 default 가 80,100 구간이므로, epoch 를 100 번은 실행 시켜야 gamma 값을 곱하며 lr 을 감소시킬 수 있다. 따라서 이 scheduler

를 실행 시킬 때, 100 번 이상의 epoch 를 학습시키기에 불가능 하다면 milestone 구간을 변경시키는 방법으로 train 을 할 필요성이 있다.

- Cosine Annealing Scheduler

- learing rate 가 cos 함수를 따라서 eta_min 까지 떨어졌다 다시 초기 learning rate 까지 올라온다.
- optimizer: 이전에 정의한 optimizer 변수명을 넣어준다.
- T_max: 최대 iteration 횟수
- eta_min: 최소로 떨어질 수있는 learning rate default=0

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos\left(\frac{T_{cur}}{T_{\max}}\pi\right) \right)$$



Train_ssd 파일에서는 t-max 의 default 값이 100 으로 설정되어있다. 이는 코사인 함수가 주기의 1/4 를 돌기전에 학습이 끝나버려 감소하는 형태의 learning rate 를 보일 것이다. 현실적으로 epoch 를 100 번까지 돌리기에는 무리가 있어서 t-max 를 달리하여, 실험시켜보았다.

3. onnx_export.py

```
# format output model paths
if args.model_dir:
    args.model_dir = os.path.expanduser(args.model_dir)

# find the checkpoint with the lowest loss
if not args.input:
    best_loss = 10000
    for file in os.listdir(args.model_dir):
        if not file.endswith(".pth"):
            continue
        loss = float(file.split(".pth")[0].split("_loss=")[-1])
        if loss < best_loss:
            best_loss = loss
            args.input = os.path.join(args.model_dir, file)
    except ValueError:
        continue
    print("found best checkpoint with loss {:.4f} {}".format(best_loss, args.input))

# append the model dir (if needed)
if not os.path.isabs(args.input):
    args.input = os.path.join(args.model_dir, args.input)

if not os.path.isabs(args.output):
    args.output = os.path.join(args.model_dir, args.output)

# determine the number of classes
class_names = [name.strip() for name in open(args.labels).readlines()]
num_classes = len(class_names)

# construct the network architecture
print("creating network")
net = args.net
print("num classes")
print(num_classes)

if args.net == "vgg16-aid":
    net = create_vgg16_aid(num_classes, is_test=True)
elif args.net == "res101-aid" or args.net == "res101-aid-aid":
    net = create_res101_aid(num_classes, is_test=True)
elif args.net == "res101-aid-aid":
    net = create_res101_aid_aid(num_classes, is_test=True)
elif args.net == "res101-aid-aid":
    net = create_res101_aid_aid(num_classes, is_test=True)
elif args.net == "res101-aid-aid":
    net = create_res101_aid_aid(num_classes, is_test=True)
elif args.net == "res101-aid-aid":
    net = create_res101_aid_aid(num_classes, is_test=True)
else:
    print("The net type is wrong. It should be one of vgg16-aid, res101-aid and res101-aid-aid.")
    sys.exit(1)

# load the model checkpoint
print("loading checkpoint")
print(args.input)

# load the model checkpoint
print("loading checkpoint: " + args.input)

net.load(args.input)
net.to(device)
net.eval()

# create example image data
dummy_input = torch.randn(args.batch_size, 3, args.height, args.width).cuda()

# format output model path
if not args.output:
    args.output = args.net + ".onnx"

if args.model_dir and args.output.find("/") == -1 and args.output.find("\\") == -1:
    args.output = os.path.join(args.model_dir, args.output)

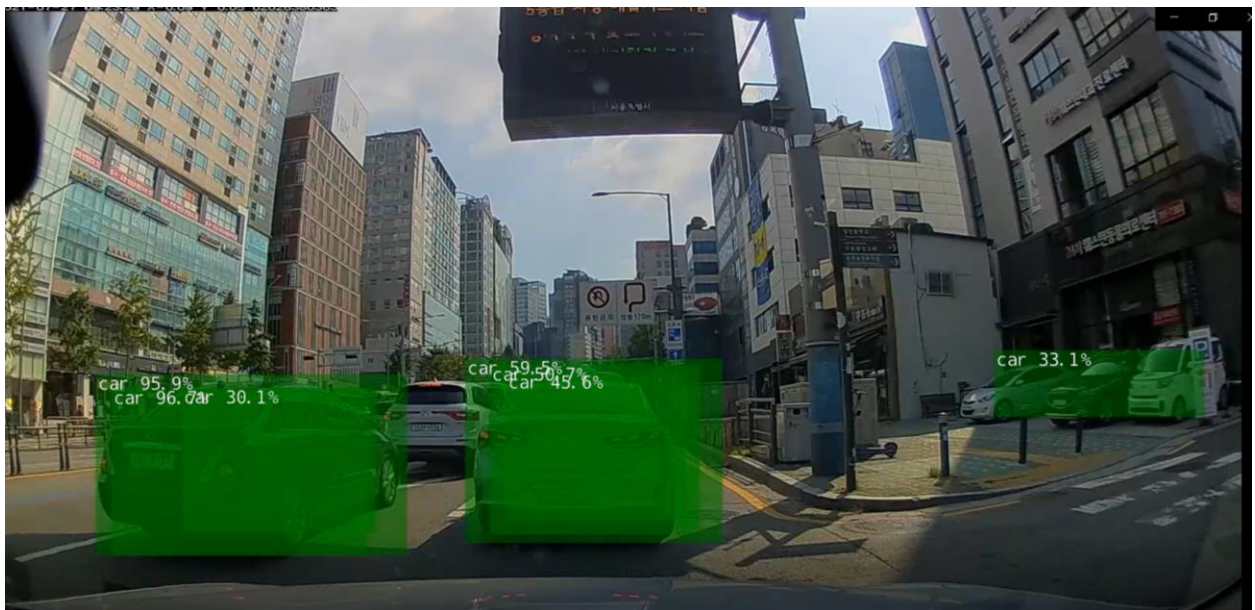
# export to ONNX
input_names = ["input_0"]
output_names = ["scores", "boxes"]

print("exporting model to ONNX...")
torch.onnx.export(net, dummy_input, args.output, verbose=True, input_names=input_names, output_names=output_names)
print("model exported to: {}".format(args.output))
print("task done. exitline nnnnnnn")
```

마지막으로 Onnx_export.py 는 훈련된 모델을 Tensorflow 환경에서 로드할 수 있도록, PyTorch 에서 ONNX 로 이전시켜주는 코드이다. 입력 받은 model path 를 통해, 가장 작은 lowest loss 를 찾고, 클래스의 개수를 결정 및 네트워크 구조를 생성한다. 찾았던 네트워크 체크 포인트를 device 에 올리고, 입력 받은 model 위치 경로로, ONNX 파일을 생성하고 이동시킨다. 이를 통해 이미지는 물론 실시간으로 영상을 인식할 수 있게 한다.

6.3 최종 결과물

목차 5 번에 나타나 있는 transfer learning 방법으로 모델을 re-training 시켰다. 목차 6.2 에 나와있는 이론들을 참고하여, schedule 를 달리해가며 loss 를 줄이는 방식을 달리 적용시켜보며 학습을 진행하였다. 이에 따라 완성된 최종결과물은 다음과 같다. 자세한 결과는 동영상에도 기재되어 있다.



7. 기타 (Certification)

2021. 7. 26.

DLI C-IV-02 Certificate | Deep Learning Institute

NVIDIA DEEP LEARNING INSTITUTE CERTIFICATE OF COMPETENCY

This certificate is awarded to
JONGHA PARK

for demonstrating competence in the completion of
**GETTING STARTED WITH DEEPSTREAM FOR
VIDEO ANALYTICS ON JETSON NANO**



Will Ramey
Senior Director, Developer Programs, NVIDIA

2021
Year issued



<https://courses.nvidia.com/certificates/b239d6cf44c45fd819f2b30507d8414>

1/1

2021. 7. 26.

DLI S-RX-02 Certificate | Deep Learning Institute

NVIDIA DEEP LEARNING INSTITUTE CERTIFICATE OF COMPETENCY

This certificate is awarded to
JONGHA PARK

for demonstrating competence in the completion of
**GETTING STARTED WITH AI ON JETSON
NANO**



Will Ramey
Senior Director, Developer Programs, NVIDIA

2021
Year issued



<https://courses.nvidia.com/certificates/c4c10fcb1d1542b39e934f89cb463029>

1/1