JSR프로젝트 - Nvidia의 End-to-End Deep Learning을 바탕으로 구현

1. Model - CNN(Convolutional Neural Network)

CNN model :

➔ Filter를 이용하여 합성곱 연산 수행

➔ 필터에 따라 이미지의 다른 특징을 추출

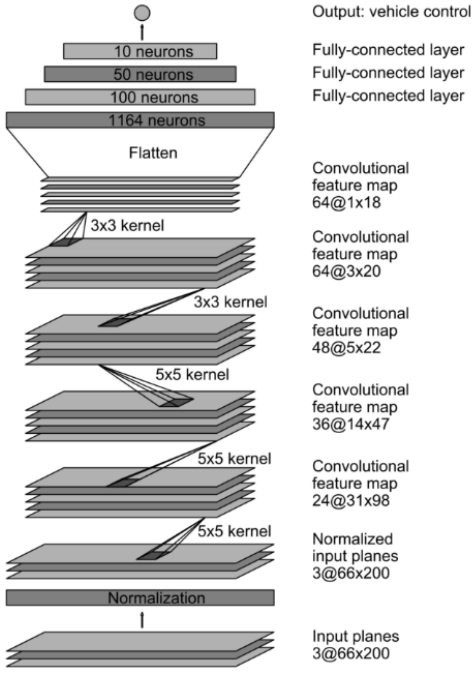


그림 1. CNN architecture

(출처 : https://developer.nvidia.com/blog/deep-learning-self-driving-cars/)

➔ Pytorch 1.4.0 버전 환경 (https://pytorch.org/docs/1.4.0/nn.html)

- **convolution**

```
class end2end(nn.Module):

    def __init__(self):
        super(end2end, self).__init__()
        conv1 = nn.Conv2d(in_channels=3, out_channels=24, kernel_size=(5, 5), stride=(2, 2), padding=(0, 0), dilation=(1, 1))
        conv2 = nn.Conv2d(in_channels=24, out_channels=36, kernel_size=(5, 5), stride=(2, 2), padding=(0, 0), dilation=(1, 1))
        conv3 = nn.Conv2d(in_channels=36, out_channels=48, kernel_size=(5, 5), stride=(2, 2), padding=(0, 0), dilation=(1, 1))
        conv4 = nn.Conv2d(in_channels=48, out_channels=64, kernel_size=(3, 3), stride=(1, 1), padding=(0, 0), dilation=(1, 1))
        conv5 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=(3, 3), stride=(1, 1), padding=(0, 0), dilation=(1, 1))
```

각 채널의 크기에 맞게 in channel과 out channel 설정

```
H = 3
W = 20

kernel_size = (3, 3)
stride = (1, 1)
padding = (0, 0)
dilation = (1, 1)

Hout = ((H + (2 * padding[0]) - dilation[0] * (kernel_size[0] - 1) - 1) / stride[0]) + 1
Wout = ((W + (2 * padding[1]) - dilation[1] * (kernel_size[1] - 1) - 1) / stride[1]) + 1

print(Hout, Wout)
```

위의 계산 과정을 거쳐 66x200에서 31x98 순서로 나오도록 stride를 설정

- **Flatten**

```
fc1 = nn.Linear(1152, 100)
fc2 = nn.Linear(100, 50)
fc3 = nn.Linear(50, 10)
fc4 = nn.Linear(10, 1)
```

최종적으로 output이 1이 될 때까지 layer를 펴주는 작업

- **Forward**

```
def forward(self, x):
    x = self.conv_module(x)
    x = torch.flatten(x, start_dim=1)
    y = self.fc_module(x)

    return y
```

X가 들어오면 convolution을 먼저 통과하고, torch.flatten 함수를 통해 tensor([[1,2,3], [4,5,6]])형식의 배열로 바꾼 뒤 flatten을 통해 최종 output을 도출

2. Image normalization

```
vedio_files=glob.glob("*.mkv")
caps=[cv2.VideoCapture(name) for name in vedio_files]

#cap_length=max([int(cap.get(cv2.CAP_PROP_FRAME_COUNT)) for cap in caps])


if not os.path.isdir("./image/"):
    os.mkdir("./image/")
```

확장자가 .mkv인 파일을 읽어 프레임을 읽는다.

```
for cap in caps:
    i=0
    while True:
            cap_name=str(cap).split(" ")[1][:-1]
            ret, img = cap.read()

            if not ret:
               break

        img=cv2.resize(img, dsize=(200,112))

        img=img[46:,:]

        cv2.imwrite("image/"+str(i)+"-"+cap_name+".jpg",img)

        i+=1
```

동영상으로부터 프레임을 얻으면 image 크기 720, 1280이고, 이것을 cnn모델의 input 형식(66x200)에 맞추어 (200, x) 크기로 줄여준다.

1280 : 720 = 200 : x

X=112.5(112)

Resize를 통해 현재 이미지 크기를 200, 112크기로 줄인다.

112-66= 46이므로 66부분을 버린 나머지 부분을 ROI로 설정

ROI영역으로 정리된 이미지를 image 폴더에 저장한다.

```
for cap, vedio_file in zip(caps, vedio_files):
    current_name=str(vedio_file).split(".")[0]
    cap_name=str(cap).split(" ")[1][:-1]
    os.rename(current_name+".mkv", cap_name+".mkv")
    os.rename(current_name+".csv", cap_name+".csv")
```

영상과 csv파일을 랜덤 이름으로 바꾼다.

3. Train

➔ Google colab으로 학습

```
def study_model_save(epoch, batch_cnt, model):
    if not os.path.isdir("./save/"):
        os.mkdir("./save/")
    SavePath_main = os.getcwd()+"/save/main_model_"+str(epoch).zfill(6)+"_"+str(batch_cnt).zfill(6)+".pth"
    SaveBuffer = io.BytesIO()
    torch.save(model.state_dict(), SaveBuffer, _use_new_zipfile_serialization=False)
    with open(SavePath_main, "wb") as f:
        f.write(SaveBuffer.getvalue())


def study_model_load(episode, model, cnt, device):
    LoadPath_main = os.getcwd()+"/save/main_model_"+str(episode).zfill(6)+"_"+str(cnt).zfill(6)+".pth"
    with open(LoadPath_main, 'rb') as f:
        LoadBuffer = io.BytesIO(f.read())
    model.load_state_dict(torch.load(LoadBuffer, map_location=device))
    return model
```

image폴더에서 이미지를 읽고 학습된 파일을 save 폴더에 저장.

```
csv_files = glob.glob("*.csv")
csv_data = []
for csv_file in csv_files:
    f = open(csv_file, 'r')
    reader = csv.reader(f)
    #reader.next()
    next(reader)
    for row in reader:
            if len(row)!=3:
                break
        csv_data.append((csv_file[:-4], row[1], row[2]))

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

csv파일에서 현재 이미지 프레임 번호와 해당 노출 값을 읽어 csv_data에 저장

```
batch_size = 100

x_batch = []
y_batch = []

epochs = 2000
epoch = 203
cnt = 1

net = end2end().to(device)
#net = study_model_load(202, net, 82, device).to(device)

loss_function = nn.MSELoss()
optimizer = torch.optim.Adam(net.parameters(), lr=1e-4)
```

Batch 크기는 100마다 지정하고 epoch가 2000이 될 때까지 학습시킴

Loss – mean squared error 구하기 위해 사용(예측 값과 타겟 값 사이의 에러)

nn.MSELoss()를 통해 측정

torch.optim.Adam(net.parameters(), lr=le-4)

parameters : 매개 변수 그룹을 최적화 하거나 정의하기 위한 매개 변수 반복

lr : 학습률 (기본 값 le-3)

```
while (epoch < epochs):

    bc = 0
    random.shuffle(csv_data)
```

2000번이 될 때까지 학습시키며, 매번 csv파일을 랜덤으로 섞는다.

```
for ccss in csv_data:
    if (cnt % batch_size) == 0:
        cnt = 1
        x = torch.FloatTensor(x_batch).to(device)
        y = torch.FloatTensor(y_batch).to(device)

        outputs = net(x)

        optimizer.zero_grad()
        loss = loss_function(outputs, y)
        loss.backward()
        optimizer.step()

        print("epoch : {} / {} | step : {} / 245 | loss : {}".format(epoch, epochs, bc, loss / 100))
        x_batch = []
        y_batch = []
        bc += 1
```

Tensor type으로 x_batch, y_batch를 device에 올려주고 outputs으로 출력 값 도출

학습 수행 전 미분 값을 0으로 초기화 하고 네트워크 연산 결과와 실제 결과를 cross entropy 연산하여 손실함수 값 도출

가중치 w와 편향 b에 대한 기울기 계산 후 업데이트

```
name = "image/"+ccss[1]+"-"+ccss[0]+".jpg"
img = Image.open(name)
img = img.convert('YCbCr')
img = np.array(img)
img = img.transpose((2, 0, 1)) / 255.0
img = x_batch.append(img.tolist())
label = y_batch.append([float(ccss[2])])

cnt += 1
time.sleep(0.02)
```

이미지 폴더에서 이미지를 불러와 YUV형태로 바꾼 뒤 x_batch에 저장하고 y_batch에는 csv 값을 저장한다.

```
if (epoch % 100) == 0:
    study_model_save(epoch, cnt, net)
```

Epoch가 100만큼 증가할 때마다 학습 파일을 저장한다.

4. 데이터 수집 방법

```
device_dir = "/dev/videoCAM"
```

현재 usb카메라의 device 이름 지정한다.

```
video_name = "/home/nvidia/Desktop/test.mkv"
csv_name = "/home/nvidia/Desktop/test.csv"
```

자동차가 주행하면 해당 영상과 csv파일을 동시에 저장한다. 데이터 수집을 위해 여기에서는 "자이카 조이스틱"으로 수동으로 트랙을 주행 시키면서 녹화하였다.

```
#make csv file
out = cv2.VideoWriter(video_name,cv2.VideoWriter_fourcc('M','J','P','G'), 10, (640, 480))
f = open(csv_name, 'w')
wr = csv.writer(f)
wr.writerow(["ts_micro", "frame_index", "threshold"])
```

CSV파일을 만들고 현재 시간, 프레임 번호, 해당 프레임의 노출 값을 저장한다.

```
def camera_callback(data):
    global cv_image
    global bridge
    cv_image = bridge.imgmsg_to_cv2(data, "bgr8")
```

Subscribe를 통해 data를 받으면 거기로부터 image를 추출하고 이 이미지를 바탕으로 다음 과정을 진행한다.

```
while True:

    if cv_image.size !=(640*480*3):
            continue

    #set exposure from 5 to 70
    if cnt>=70:
            cnt=5
    exposure_set(cnt)

    va=exposure_get()

    #make csv file
    wr.writerow([time.time(), fre , cnt])
    out.write(cv_image)
```

이미지가 들어오면 해당 프레임 번호를 fre로 세고, cnt 를 노출 값으로 지정하여 5부터 70까지 범위에서 노출 값을 변경해준다. 이후 cnt값을 exposure_set 함수를 통해 카메라에 세팅한다.

```
def exposure_set(value):
    global device_dir

    if str(type(value)) != "<type 'int'>":
        return

    command = "v4l2-ctl -d "+ device_dir +" -c exposure_absolute="
    command += str(value)
    os.system(command)
```

cnt값을 받아 v4l2명령어로 디바이스에 직접 명령을 내려 노출도를 바꿔준다.

5. 결과

```
net = end2end().to(device)
net = self.study_model_load(1000, 77, net, device)
```

학습된 .pth파일을 model을 통해 load 한다.

```
def study_model_load(episode, batch_cnt, model, device):


    LoadPath_main = os.getcwd()+"/save/main_model_"+str(episode).zfill(6)+"_"+str(batch_cnt).zfill(6)+ ".pth"

    with open(LoadPath_main, 'rb') as f:
        LoadBuffer = io.BytesIO(f.read())

    model.load_state_dict(torch.load(LoadBuffer, map_location=device))


    return model
```

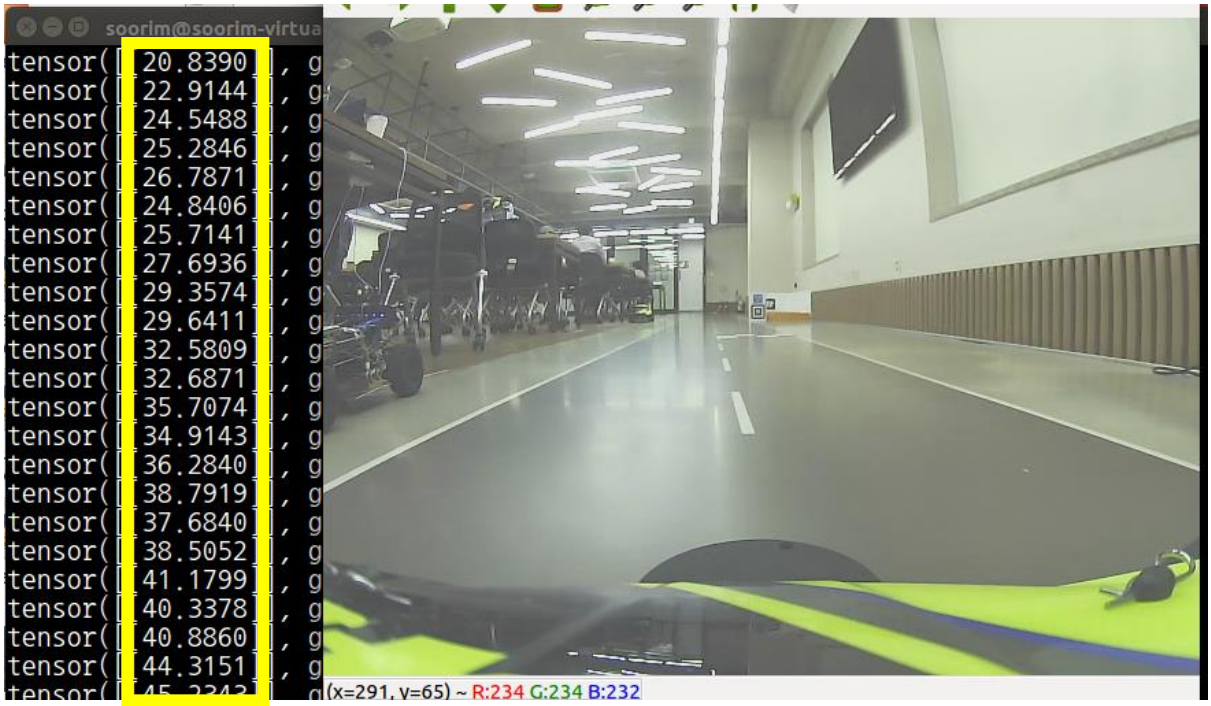
모델을 로드하여 최종 값을 도출해낸다.

```
frame = cv2.cvtColor(cv_image, cv2.COLOR_BGR2YUV)
frame = cv2.resize(frame, dsize=(200, 112))

frame = frame[46:, :]

frame = frame.transpose((2, 0, 1)) / 255.0
t_frame = torch.FloatTensor([frame]).to(device)

ex=net(t_frame)
```

이후 들어온 이미지를 학습시킬 때의 이미지 크기와 똑 같이 만들어주고 end2end모델을 통해 출력 값을 도출한다. 여기에서 출력 값은 해당 이미지의 노출 값이 된다.

```
input_file = "0x7f0961a722d0.mkv"
```

녹화한 영상 중 하나를 input file로 지정해주고 확인한다.



화면의 밝기가 달라지면 노출 값이 해당 영상에 따라 바뀐다.