IIPL 인턴 1주차 보고

정윤희

INDEX

01 과제 수행 사항

02 개인 수행 사항

03 향후 활동

과제 수행 사항

수행사항 **기본적인 PyTorch 구조 이해 및 cs231n 내용 숙지**

- PyTorch?

  파이썬 기반의 과학 계산을 위한 패키지

  - Numpy의 연산을 GPU로 가속
  - Deep learning 연구에 있어서 개발을 좀 더 유연하고 빠르게 수행하도록 함

- 기본 개념
  - Tensor 클래스 ($n_1 \times n_2 \times \cdots \times n_d$ 개의 숫자를 d차원으로 정렬한 객체)
  - Autograd -> 역전파 통한 미분값 자동 계산
  - Module -> PyTorch에서 딥러닝 모델 구성하는 base class
    Input, Output, Forward, Backward 정의

https://tutorials.pytorch.kr/beginner/introyt/introyt1_tutorial.html

과제 수행 사항

- Dataset과 Dataloader

  데이터의 샘플과 label 저장 / 데이터 세트를 미니배치 단위로 iterable 객체로 감쌈

  Map-style dataset, Iterable-style dataset으로 구분

- Custom Dataset (원하는 형태로 데이터 가공하기 위함)

  1. torch.utils.Dataset 클래스 상속받는 자식 클래스 생성

  2. 세 가지 메소드 필요

  __init__(self, 인수) 데이터셋 전처리

  (label, data, transform 등의 값들 초기화함)

  __len__(self) 데이터셋 길이. 즉, 샘플의 수 반환

  __getitem__(self, idx) 데이터셋에서 특정 1개의 샘플 가져옴. (idx에 해당하는 샘플을 정의한 형태로 변환해 돌려줌)

```
class CustomM(Dataset):
    def __init__(self, mnist_dataset):
        self.mnist_dataset = mnist_dataset

    def __len__(self):
        return len(self.mnist_dataset)

    def __getitem__(self, idx):
        image, label = self.mnist_dataset[idx]
        return image, label
```

과제 수행 사항

- MNIST -> 28*28 크기의 grayscale 이미지가 50,000개의 train, 10,000개의 test로 구분된 set로, 숫자 0~9까지 10개의 정답 class로 구성.

CLASS torchvision.datasets.MNIST(*root: Union[str, Path], train: bool = True, transform: Optional[Callable] = None, target_transform: Optional[Callable] = None, download: bool = False*) [SOURCE]

MNIST Dataset.

Parameters:

- **root** (str or `pathlib.Path`) – Root directory of dataset where `MNIST/raw/train-images-idx3-ubyte` and `MNIST/raw/t10k-images-idx3-ubyte` exist.
- **train** (*bool, optional*) – If True, creates dataset from `train-images-idx3-ubyte`, otherwise from `t10k-images-idx3-ubyte`.
- **download** (*bool, optional*) – If True, downloads the dataset from the internet and puts it in root directory. If dataset is already downloaded, it is not downloaded again.
- **transform** (*callable, optional*) – A function/transform that takes in a PIL image and returns a transformed version. E.g, `transforms.RandomCrop`
- **target_transform** (*callable, optional*) – A function/transform that takes in the target and transforms it.

과제 수행 사항

```
# MNIST 데이터셋 로드
mnist_train = datasets.MNIST(root='./', train=True, download=True, transform=transforms.ToTensor())
mnist_test = datasets.MNIST(root='./', train=False, download=True, transform=transforms.ToTensor())

# Custom Dataset 객체 생성
train_dataset = CustomM(mnist_train)
test_dataset = CustomM(mnist_test)
```

```
DataLoader(dataset, batch_size=1, shuffle=False, sampler=None,
           batch_sampler=None, num_workers=0, collate_fn=None,
           pin_memory=False, drop_last=False, timeout=0,
           worker_init_fn=None, *, prefetch_factor=2,
           persistent_workers=False)
```

```
train_loader = DataLoader(train_dataset, batch_size=256, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=256, shuffle=False)
```

https://tutorials.pytorch.kr/beginner/introyt/introyt1_tutorial.html

과제 수행 사항

- Library import

```
import torch
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms as transforms
from torchvision import datasets, models

import numpy as np
import matplotlib.pyplot as plt
```

- optim: 가중치에 대한 error값을 최소로 하기 위해 최적화 기법 사용 (SGD, Adam…)

- transforms: 데이터 전처리 위한 함수 (형변환, norm…)

```
model = models.resnet18(pretrained=True)

model.conv1 = torch.nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
model.fc = torch.nn.Linear(model.fc.in_features, 10)
```

- ImageNet에 학습된 weight를 불러옴

- Conv2D(채널 사이즈, 쓰고자 하는 필터 개수, 필터 크기…)

https://pytorch.org/vision/stable/models.html

과제 수행 사항

Training

```
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)
```

```
train_losses = []
test_accuracies = []

for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()

        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    train_loss = running_loss / len(train_loader)
    train_losses.append(train_loss)

    print(f"Epoch {epoch+1}, Loss: {running_loss/len(train_loader)}")
```

```
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    test_accuracies.append(accuracy)
```

과제 수행 사항

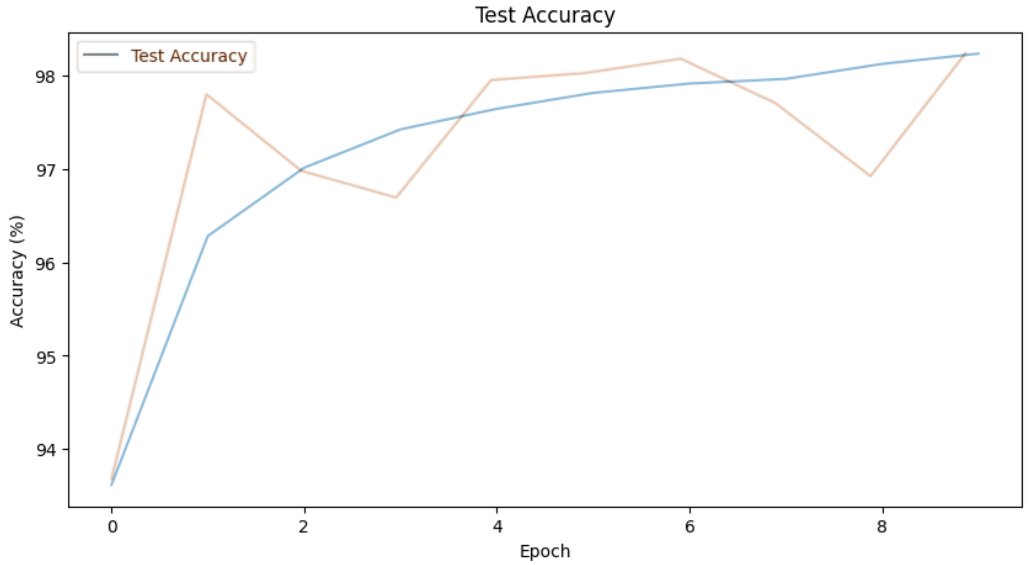
- Model(ResNet-18)

  ResNet은 18~152 layer로 설계되어 있으며, depth가 증가하였음에도 VGG-16 / 19 모델보다 더 낮은 복잡성을 가지는 것이 특징.

Epoch 10

Batch size	Adam LR = 0.001	SGD LR = 0.001
16	0.019	0.03
64	0.02	0.04
256	0.017	0.11

https://inhovation97.tistory.com/32



과제 수행 사항

- cs231n 4강

  - 3가지 예시를 통한 가중치 계산하는 방법 다룸(Back propagation)
  - Neural Network

- GD: 가장 작은 loss를 만드는 가중치를 찾아가는 과정인 optimization에 사용되는 방식
- 모델 복잡해질 수록 모든 값 미분 힘듦 -> computational graphs
- Backpropagation: 오차를 거슬러 올라가며 각각의 파라미터에 대한 gradient 계산하는 과정
  - local gradient 구하고 chain rule 이용해 재귀적으로 upstream gradient 구함
- Computational graphs의 gate 일반화(쉽게 편미분값 반환 가능)
  - Add gate: 현재의 가중치 다음 단계로 뿌림 (gradient distributor)

과제 수행 사항

- Max gate: 큰 값에 부여
- mul gate: 서로의 값을 바꿔 줌

Neural Network

- input과 output 사이 hidden layer를 추가함으로써 한 개의 클래스에도 여러 개의 분류기가 생겨 다양한 특징을 잡아 분류할 수 있게 됨

- FC layer(Dense layer) -> 1차원 배열의 형태로 평탄화된 행렬 통해 이미지 분류에 사용
  - 이미지 분류 시 3차원에서 1차원으로의 데이터 변환을 했어야 함 -> 데이터 형상 무시

- 이미지 분류에 왜 FC layer 사용?
  - 고차원의 feature map을 학습하여 최종적인 분류 수행 위함
  - Image가 고해상도가 될 경우 매개변수 수가 많아져 과적합 발생 가능성 존재

과제 수행 사항

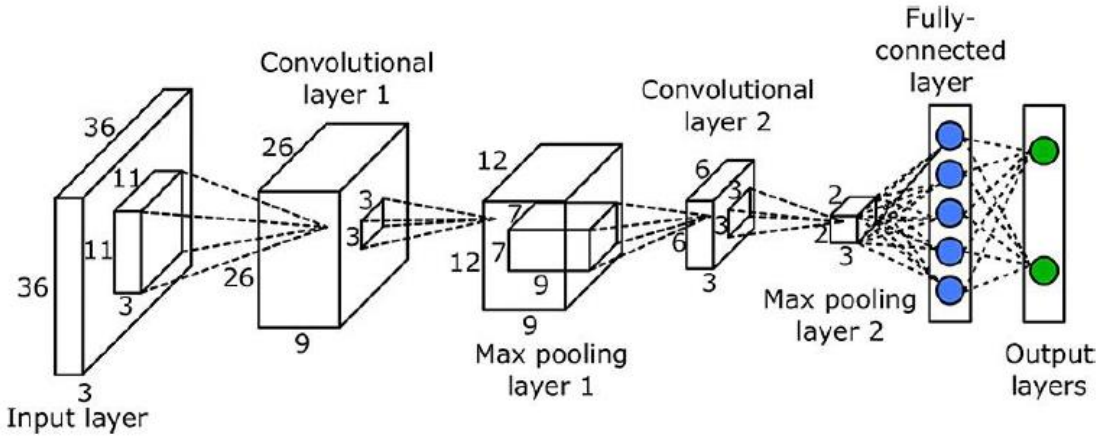
- cs231n 5강

  - CNN의 역사
  - CNN architecture

- Convolution Layer는 FC Layer에서 포함하지 못했던 공간 정보를 포함
  - Dot product를 통해 이미지 전체가 아닌 공간별 특징을 추출하기 때문
- 계층별 필터마다 무엇을 시각화한 건지?
  - 이미지가 어떻게 생겨야 해당 뉴런의 활성화를 최대화시킬 수 있는지 나타낸 것.
- Stride
  - Activation map의 사이즈를 줄이고, FC의 파라미터 수도 줄이는 목적

과제 수행 사항

- ConvNet 구조



Input layer → Conv ~ Pooling layer → FC layer ~ Output layer

- Input layer: 이미지 입력
- Conv ~ Pool: 이미지 특징 뽑아내고 차원 낮추는 과정
- FC: 결과 점수화시켜 어떤 분류에 해당하는 이미지인지 결정

- Pooling Layer
  - Spatial 차원에 대한 downsampling을 위해 사용

개인 수행 사항

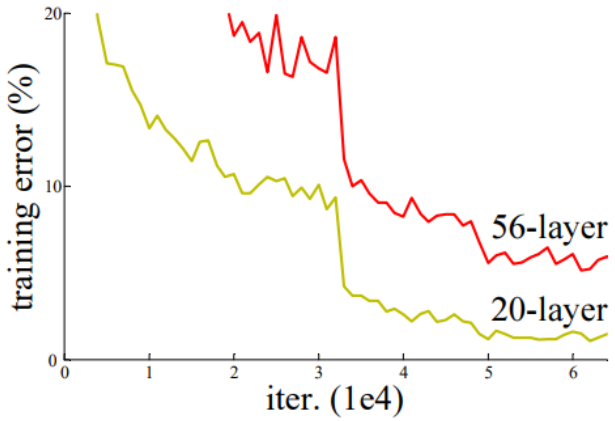
- ResNet(Deep Residual Learning for Image Recognition) 논문 리뷰

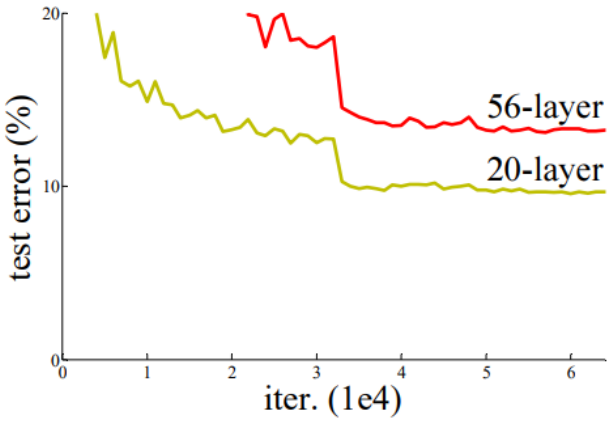
  https://arxiv.org/pdf/1512.03385.pdf

  Main idea

  Model을 깊게 쌓을수록 성능이 좋아질 것을 기대함.

  but, overfitting이나 vanishing gradient 등의 문제로 성능이 항상 좋지는 않다는 것 인지.

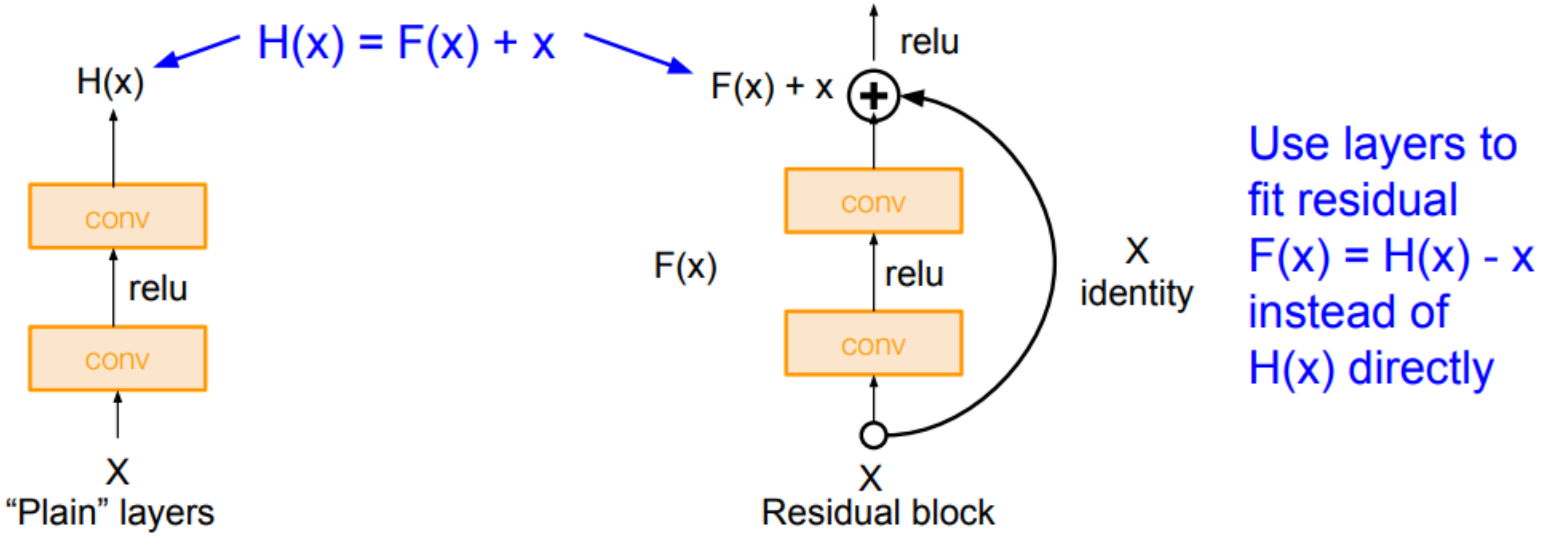




Deep한 모델일수록 optimize가 어렵다는 가설을 세움

-> 불필요하게 deep하다면 skip할 수 있도록 edge가 추가된 residual block을 만들자

개인 수행 사항



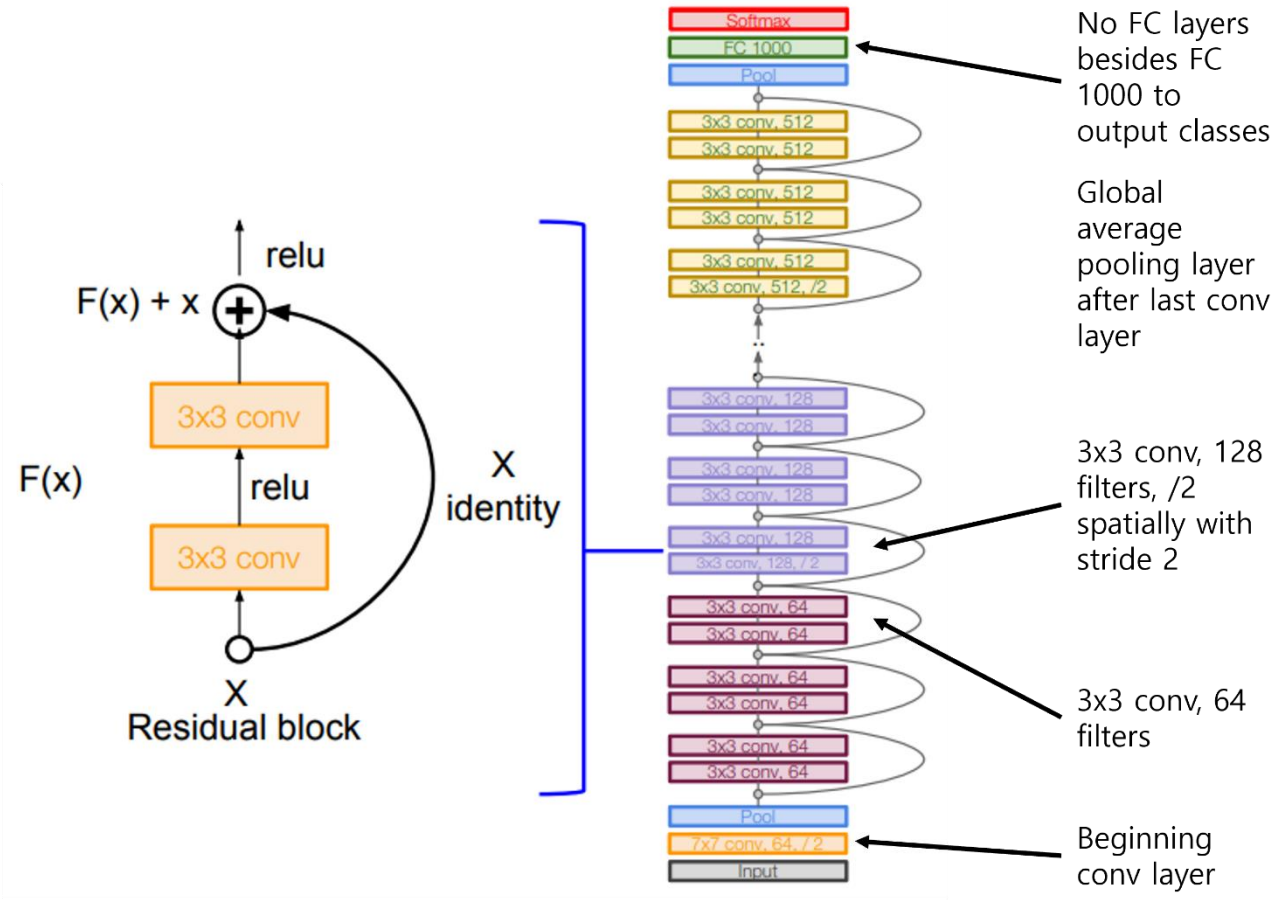
x가 input으로 들어올 때, 어떤 함수를 통해 매핑되어 target인 y와 같아지도록 학습.

$$H(x) - y = 0$$

Skip connection을 두어 H(x) = F(x) + x 형태로 매핑하고, F(x)가 0이 되도록 학습

*왜 0이 되도록 학습? 기존 모델처럼 특정 값인 y보다 0을 목표로 하는 것이 학습이 잘 됨.

개인 수행 사항



Details

- Batch norm을 본격적으로 사용하고, dropout을 제외함.
- SGD + Momentum(0.9)
- Xavier/2 initialization
- Learning rate: 0.1
- Mini-batch size: 256

향후 활동

향후활동

- CIFAR-10 이미지 데이터셋을 pre-trained model 사용해 training 진행.
- cs231n 6-7장 공부
- VGG Net(Very Deep Convolutional Networks for Large-scale Image Recognition) 논문 리뷰
- 94% on CIFAR-10 in 3.29 Seconds on a Single GPU 논문 리뷰

정윤희

감사합니다