Computer Vision Project3

2nd semester of 2022 Yonsei University Computer Vision

**Self Introduction**

Affiliation – 연세대학교 컴퓨터과학과

Name – 김신현

ID – 2016147588

Email – ksh415622@naver.com

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

—————————— ◆ ——————————

**1 Introduction**

본 보고서는 음식 이미지를 분류하는 뉴런 네트워크를 구현하고 데이터 preprocessing 및 네트워크 아키텍처에 대한 설명으로 구성되어 있다.

**2 environment**

**Language**: Python

**OS**: Window10

**GPU:** RTX 3060ti (8GB)

**CPU:**  Intel i5 12400

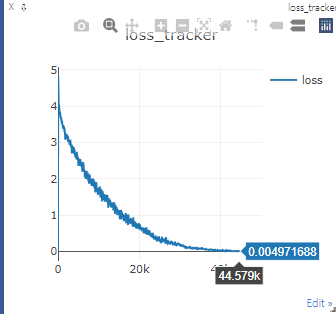
**Editor:** jupyter notebook & VsCode (Local)

**RAM :** 32 GB

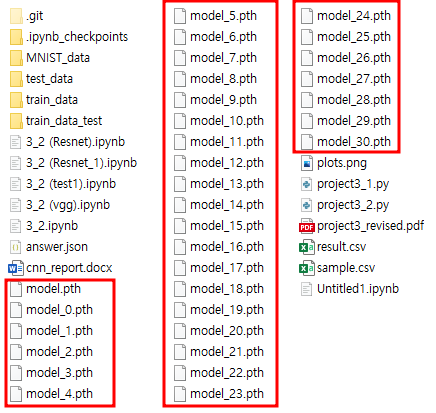
**3 Result (RMS)**

**3.1 Leaderboard**

- 주어진 testcase에는 0.71428의 결과가 나왔다.

**3.2 Train Loss**

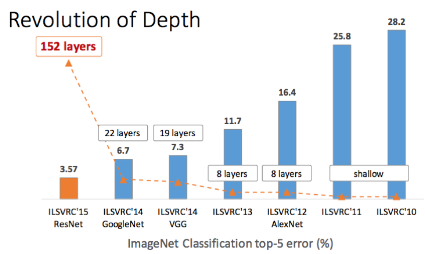
- 총 30 epoch을 학습하였고 학습이 거듭 될수록 CrossEntropyLoss가 계속해서 줄어드는 것을 확인할 수 있었다

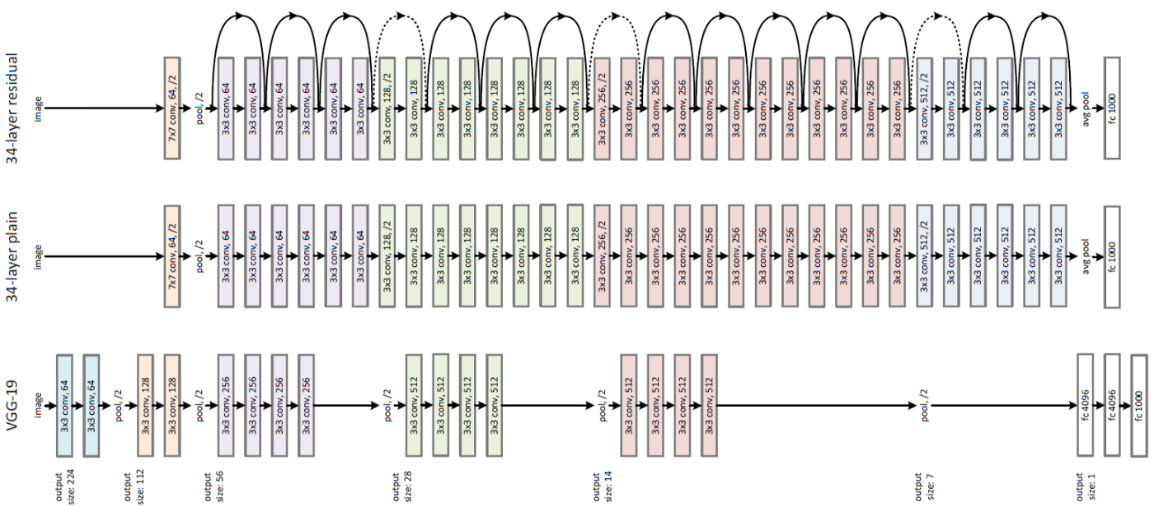


- epoch이 거듭될수록 Overfitting이 될 수 있다 판단하여 위 사진과 같이 각 epoch이 끝날 때마다 파라미터를 저장하였다

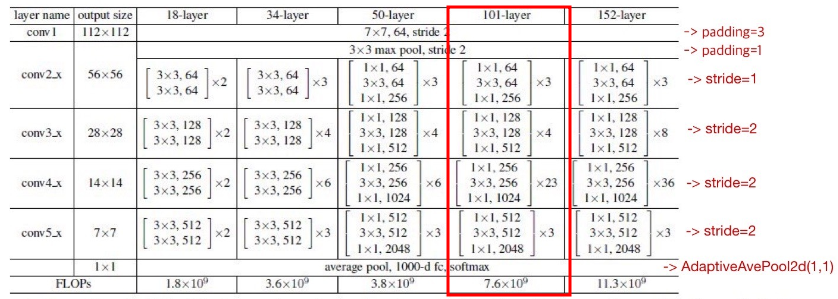
- 20~30 epoch의 총 10개의 파라미터들을 load하여 test 데이터를 입력해본 결과 25번 째 epoch을 거친 후 학습 된 모델이 약 0.71428의 accuracy로 가장 높은 정답율을 보였다.

**4 IMPLEMENTATION**

**4.1 ResNet101:**

- 본 프로젝트에선 총 101개의 layer로 구성되어 있는 ResNet101의 아키텍처를 참고하여 구현하였다.

**Fig. 1. ResNet Architecture 1**

- ResNet 은 Residual Connection(identical connection)기법을 활용하여 short cut을 만들어 layer가 매우 깊어 졌을 때도 빠르게 학습하여 파라미터를 수렴시킬 수 있다.

**Fig. 2. ResNet Architecture 2**

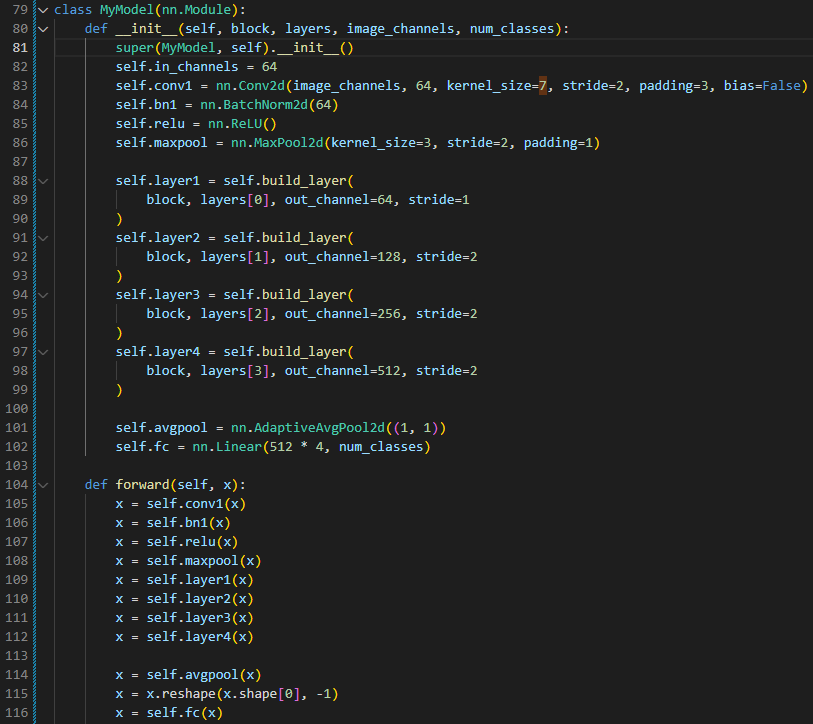
- 위 사진과 같이 본인이 사용한 ResNet101의 구조는 빨간 박스 영역과 같이 구현 되어있다.

- ResNet101은 위 그림과 같이 각각 다른 모양의 블록들이 3,4,23,3개씩 있다, 자세한 구현 내용은 아래에서 다룬다.

**4.2 ResNet Implementation**

- block & MyModel 총 두개의 클래스로 모델을 정의하였다

**4.2.1 MyModel Class**

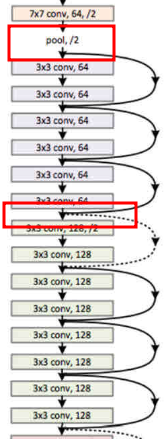
**** **-** block의 구현은 Fig.2의 설명에 따라 구현하였다.

**- self.conv1:** Fig.2의 7\*7, 64, stride 2, padding 3의 convolution커널을 의미.

**- self.maxpool:** Fig.2의 3\*3, stride 2, padding 1의 maxpooling 커널을 의미

- **self.avgpool & fc:** Fig.2의 80-dimension avgpool & fully connected 커널을 의미.

 - **build\_layer**: redisual block을 만드는 함수

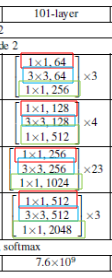
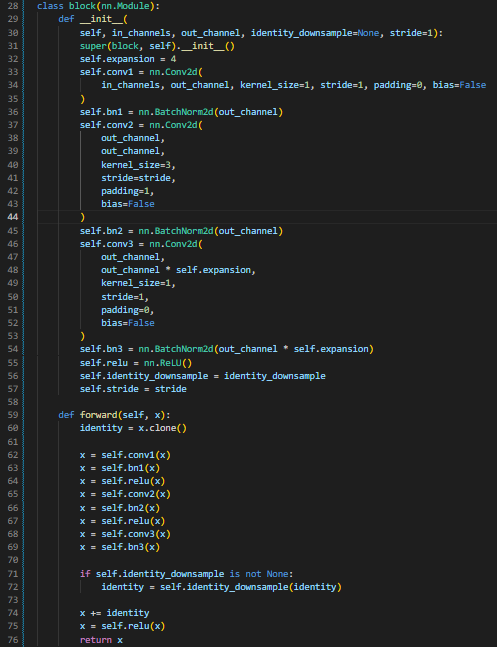
- **line 124:** stride가 1이 아니고 ouput channel이 input의 4배가*(아래 block class 에서 self.expansion = 4)* 아닐 때, downsampling을 한다.

downsmapling이란 pooling, convolution layer를 통해 featuremap이 축소되어 잔차연결할 때 크기가 맞지 않아 연산을 할 수 없다.

따라서 1\*1 convolution(bottle neck) layer를 하나 더 연결해준다.

- **line 136:** 첫 블록은 위에서 말한 downsampling 여부를 판단하고 한 블록을 추가해준다,

다음으로 남아있는 블록 수 만큼 모두 stride를 1로, downsampling없이 쌓는다.

 **4.2.2 BlockClass**

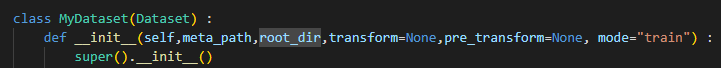
- **self.conv1:** residual block 내의 모든 1\*1\*out\_channel 사이즈에서 첫 번째 convolution laye를 의미 (오른쪽 이미지 빨간색 블록)

**- self.conv2**: residual block 내의 모든 3\*3\*out\_channel conv layer의미(오른쪽 이미지 파란색 블록)

**- self.conv3**: residual block 내의 모든 1\*1\*out\_channel 사이즈에서 세 번째 convolution laye를 의미 (오른쪽 이미지 녹색 블록)

- **line 74:** residual connection 부분.

**4.3 Data Set (MyDataset):**

- MyDataset 클래스로 총 5개의 파라미터를 받는다

**a**. **meta\_path**: answer.json

**b.** **root\_dir**: train or test set directory

**c.** **pre\_transform**: dataset에 image 데이터를 저장하기 전 전처리용 transform

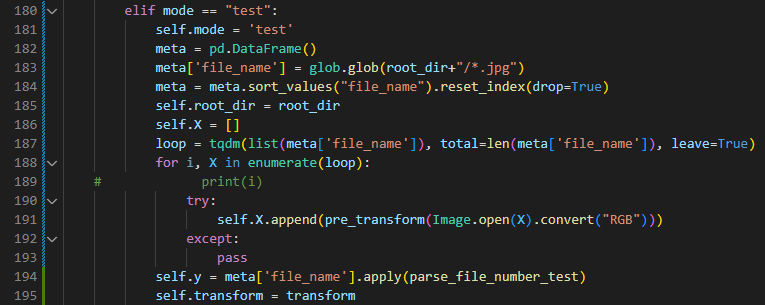
**d.** **transform**: 이미지 학습 시 사용될 전처리용 transform

**e: mode:** train용으로 사용될 경우 answer.json 파일을 읽어야 하기 때문에 구분용으로 사용되는 파라미터, 총 “train”,”test” 두가지 값을 받는다.

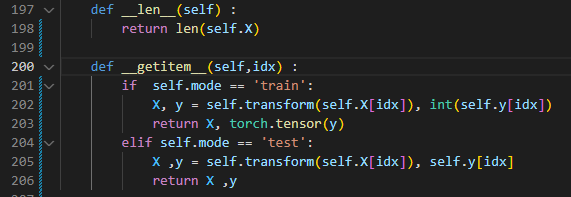
**-** 위 코드는 train의 경우 dataset을 빌드하는 과정이다.

**-** line 162: answer.json을 읽어 dataframe으로 변환시킨다

- line 172: 학습 속도 향상을 위해 데이터를 학습할 때 이미지를 읽어오는 것이 아닌 dataset생성 시 모든 학습용 이미지 데이터들을 RAM에 올린다.

**-** 위 코드는 test의 경우 dataset을 빌드하는 과정이다.

- line 188: train의 경우와 동일하게 dataset을 빌드하는 과정에서 미리 이미지 데이터를 모두 RAM에 올린다.

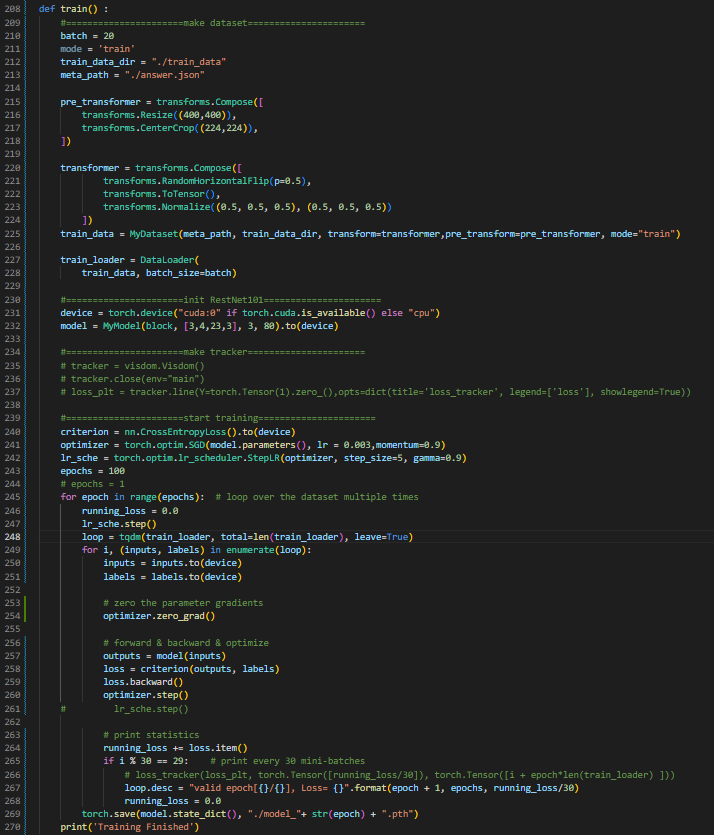


- line 197: 데이터셋 길이는 X의 길이, 즉 이미지 픽셀 데이터를 기준으로 구현하였다

예) train의 경우 len(train\_dataset) = 40000

- line 203&206: train와 test에서 return 하는 X는 이미지 픽셀 데이터로 동일하지만 y의 경우

Train은 카테고리를 반환하고 test의 경우는 이미지 파일 명을 반환한다.

**4.4 train** 

- **pre\_transformer: 4.3**에서설명한 내용으로 dataset을 구성할 때 먼저 이미지마다 사이즈가 다르기 때문에 400, 400의 크기로 통일한 후 중앙에서 224\*224만큼의 이미지만 사용한다

- **transformer:** 해당 transformer는 이미지를 dataset에서 불러와 학습시킬 때 사용되는 것으로 정확도를 올리기 위해 좌우 반전 및 정규화 두가지를 적용하였다.

- **line 242:** 효율적인 학습을 위해 경사가 최적화될수록 learning rate를 줄여나가 미세하게 조정해주는 lr\_scheduler를 사용하였다.

- **line 235~237 & line 266:**

먼저 다른 콘솔창에서 visdom을 입력하여 로컬 서버를 실행시킨 뒤 해당 부분들의 주석을 풀고 실행시키면 loss를 tracking할 수 있다.

**5 Approach**

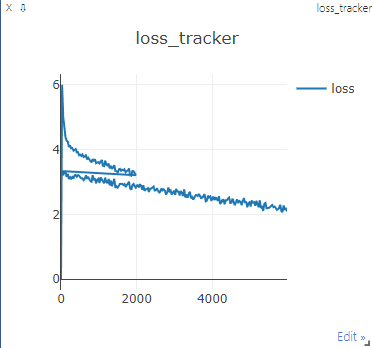
**5.1 Why Choose ResNet101**

- ResNet은 18, 34, 50 ,101, 152 총 5가지의 hidden layer수로 구성된 버전이 있다

- 본 프로젝트에서는 40000개의 training data가 있으므로 152개 layer로는 학습할 데이터가 조금 부족하다 생각되었다.

- Memory 및 Computing Power가 부족하고 또한 총 80개의 class 밖에 없기 때문에 152보다는 작은 101개의 layer로 구성된 ResNet101을 사용하였다.

**5.2 How to Choose Batch Size**



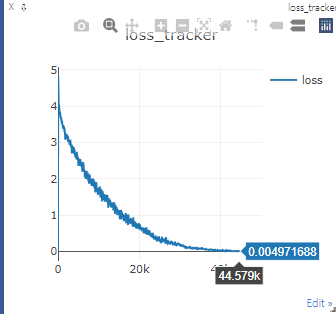
* 위 이미지는 batch size 5로 하였을 때의 loss tracking 이미지이다
* 각 30 batch마다 loss를 기록하도록 설정하였다.
* 그래프를 보면 알 수 있듯이 노이즈가 조금 많다고 판단되어 batch size를 20으로 늘렸다
* 또한 batch size가 20 이상일 경우 **2 Enviroment**에서 기재한 본인이 사용하는 환경에서는 GPU 메모리가 부족하기 때문에 사용가능한 최대 batch size인 20으로 학습하였다.

**5.3 How to Choose Learning rate**

- 초기 learing rate를 0.001보다 조금 더 큰 0.003으로 설정했다

- 이후 lr\_scheduler를 사용하여 초기에는 학습을 조금 더 빠르게 하고 이후 minimum에 가까월 질수록 learning rate가 줄어들어 더욱 세밀한 minimum을 찾도록 하였다.

**5.4 How to Avoid Overfitting**



- 위 그림은 총 30epoch을 학습하였을 때 loss tracking한 결과이다.

- 학습이 진행될수록 Loss가 매우 작아지는 것을 확인할 수 있다

- 때문에 더 많은 epoch을 학습했을 때 Overfitting이 더욱 심해질 것 같아 30에서 학습을 멈췄다

- 이후 train()에서 확인할 수 있듯이 각 epoch마다 모델을 저장하였기 때문에 23~29 epoch의 학습을 거친 모델을 load하여 test set의 class들을 각각 예측한 후 submit하여 결과를 비교해봤다.

- 위 그래프는 test loss를 비교해본 결과로 25 epoch을 학습했을 때의 모델이 가장 좋은 성능을 보였기 때문에 이를 체택하였다.

- 또한 transformer를 정의할 때 50%의 확률로 좌우 반전시켜 학습시켜 모델이 더욱 견고하도록 설계하였다.

**6 How to Run**

**6.1 How to Test**



- 코드 수정 없이 python project3\_2.py로 실행하면 된다

**6.2 How to Train**



- main 함수에서 train() 부분의 주석을 해제한 후 python project3\_2.py로 실행하면 된다

**References**

1. “ResNet 클래스 정의하기” https://ingu627.github.io/code/ResNet\_scratch\_pytorch/
2. “ResNet Architecture” <https://bskyvision.com/644>
3. “Pytorch Documentation” <https://pytorch.org/docs/stable/index.html>
4. “Pytorch ResNet” https://www.youtube.com/watch?v=DkNIBBBvcPs