<인공지능 최종 텀프로젝트 보고서>

MNIST 인식 성능 향상 프로젝트

인공지능 19 년도 2 학기

유하진 교수님

컴퓨터과학부 2015920057 하경민

목차

MNIST 인식 성능 향상 프로젝트

1. 프로젝트 개요

교재를 통하여 MNIST 데이터 세트라는 필기 문자(숫자) 이미지 데이터를 분류하였다. 이 코드를 여러가지 방법으로 개선하여 test data에 대하여 오류 1% 이하로 개선하는 것이 이 프로젝트의 목표이다.

<수행 목표>
- Tensorflow code를 여러가지 방법으로 확장 (자유 선택)
- Convolution Hidden layer 수 증가 (필수)
- Filter size = 3x3 (필수)

options:
- Node 수 증가
- Filter 수
- Activation function
- Data augmentation
- …

<제출 자료>
- 보고서, code, 결과물

2. 프로젝트 수행 과정 요약

1) 첫번째는 교재의 예시 코드를 사용하여 먼저 성능을 확인

INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> DROPOUT > SOFTMAX
CONV 필터 크기는 5x5 이고, Stride 는 1x1 이고, 2층이며 필터 수는 각각 32개, 64개이다.
POOL 필터 크기는 2x2 이고, Stride는 2x2 이고, MAX POOL을 사용한다

2) 교재의 코드를 수행 목표의 필수 조건 2 가지로 수정

INPUT -> [CONV -> RELU -> POOL]*3 -> FC -> RELU -> DROPOUT > SOFTMAX
CONV 필터 크기는 3x3 이고, Stride 는 1x1 이고, 3층이며 필터 수는 각각 32개, 64개, 128개이다.
POOL 필터 크기는 2x2 이고, Stride는 2x2 이고, MAX POOL을 사용한다.

3) Pooling 필터의 크기를 3x3 으로 수정

INPUT -> [CONV -> RELU -> POOL]*3 -> FC -> RELU -> DROPOUT > SOFTMAX
CONV 필터 크기는 3x3 이고, Stride 는 1x1 이고, 3층이며 필터 수는 각각 32개, 64개, 128개이다.
POOL 필터 크기는 3x3 이고, Stride는 2x2 이고, MAX POOL을 사용한다.

4) AVG Pooling 과 L2 Norm Pooling 을 사용

INPUT -> [CONV -> RELU -> POOL]*3 -> FC -> RELU -> DROPOUT > SOFTMAX
CONV 필터 크기는 3x3 이고, Stride 는 1x1 이고, 3층이며 필터 수는 각각 32개, 64개이다, 128개이다.
POOL 필터 크기는 2x2 이고, Stride는 2x2 이고, AVG POOL을 사용한다.

INPUT -> [CONV -> RELU -> POOL]*3 -> FC -> RELU -> DROPOUT > SOFTMAX
CONV 필터 크기는 3x3 이고, Stride 는 1x1 이고, 3층이며 필터 수는 각각 32개, 64개, 128개이다.
POOL 필터 크기는 2x2 이고, Stride는 2x2 이고, L2 NORM POOL을 사용한다.

5) VGGNet 과 GoogleNet 을 참고

INPUT -> [CONVi_1 -> RELU -> CONVi_2 -> RELU -> POOL]*3 -> FC -> RELU -> DROPOUT > SOFTMAX
CONV1_1 필터 크기는 3x3x1 이고, Stride 는 1x1 이고, 개수는 1개이다.
CONV1_2 필터 크기는 3x3x32 이고, Stride 는 1x1 이고, 개수는 1개이다.
CONV2_1 필터 크기는 3x3x32 이고, Stride 는 1x1 이고, 개수는 1개이다.
CONV2_2 필터 크기는 3x3x64 이고, Stride 는 1x1 이고, 개수는 1개이다.
CONV3_1 필터 크기는 3x3x64 이고, Stride 는 1x1 이고, 개수는 1개이다.
CONV3_2 필터 크기는 3x3x128 이고, Stride 는 1x1 이고, 개수는 1개이다.
POOL 필터 크기는 2x2 이고, Stride는 2x2 이고, MAX POOL을 사용한다.

6) 'All Convolution Net'이라는 논문을 참고하여 Conv 로만 층을 구성

INPUT -> [CONV -> RELU]*3 -> FC -> RELU -> DROPOUT > SOFTMAX
CONV 필터 크기는 3x3 이고, Stride 는 2x2 이고, 3층이며 필터 수는 각각 32개, 64개이다.
POOLING은 사용하지 않는다

3. 과정별 세부 설명

1) 교재 내의 예제 코드 : Ch05 MNIST double layer CNN classification.ipynb

INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> DROPOUT > SOFTMAX
CONV 필터 크기는 5x5 이고, Stride 는 1x1 이고, 2층이며 필터 수는 각각 32개, 64개이다.
POOL 필터 크기는 2x2 이고, Stride는 2x2 이고, MAX POOL을 사용한다

교재의 Github 저장소에서 'Ch05 MNIST double layer CNN classification.ipynb' 이 파일을 그대로 가져와서 실행해보았다.

2) Conv 층 3 개로 증가, Conv 필터크기 3x3 으로 수정

INPUT -> [CONV -> RELU -> POOL]*3 -> FC -> RELU -> DROPOUT > SOFTMAX
CONV 필터 크기는 3x3 이고, Stride 는 1x1 이고, 3층이며 필터 수는 각각 32개, 64개, 128개이다.
POOL 필터 크기는 2x2 이고, Stride는 2x2 이고, MAX POOL을 사용한다.

주어진 필수 조건에 따라, 단순하게 Conv 층을 3 개로 증가하고, 필터 크기를 5x5 에서 3x3 으로 수정하였다. 이는 성능의 향상이 있을 것이라고 예상하였고, 그 이유는 다음과 같다.

CNN 에서는, 큰 크기를 가지는 CONV 레이어 하나 대신 여러 개의 작은 필터를 가진 CONV 레이어를 쌓는 것이 좋다. 3x3 크기의 CONV 레이어 3 개를 쌓는다고 생각해보자 (물론 각 레이어 사이에는 비선형 함수를 넣어준다). 이 경우 첫 번째 CONV 레이어의 각 뉴런은 입력 볼륨의 3x3 영역을 보게 된다. 두 번째 CONV 레이어의 각 뉴런은 첫 번째 CONV 레이어의 3x3 영역을 보게 되어 결론적으로 입력 볼륨의 5x5 영역을 보게 되는 효과가 있다. 비슷하게, 세 번째 CONV 레이어의 각 뉴런은 두 번째 CONV 레이어의 3x3 영역을 보게 되어 입력 볼륨의 7x7 영역을 보는 것과 같아진다.

이런 방식으로 3 개의 3x3 CONV 레이어를 사용하는 대신 7x7 의 크기를 가지는 CONV 레이어 하나를 사용한다고 생각해 보자. 이 경우에도 각 뉴런은 입력 볼륨의 7x7 영역을 크기 필드로 갖게 되지만 몇 가지 단점이 존재한다. 먼저, CONV 레이어 3 개를 쌓은 경우에는 중간 중간 비선형 함수의 영향으로 표현력 높은 feature 를 만드는 반면, 하나의 (7x7) CONV 레이어만 갖는 경우 각 뉴런은 입력에 대해 선형 함수를 적용하게 된다. 두 번째로, 모든 볼륨이 CC 개의 채널(또는 깊이)을 갖는다고 가정한다면, 7x7 CONV 레이어의 경우 C×(7×7×C)=49C2C×(7×7×C)=49C2 개의 파라미터를 갖게 된다. 반면 3 개의 3x3 CONV 레이어의 경우는 3×(C×(3×3×))=27C23×(C×(3×3×))=27C2 개의 파라미터만 갖게 된다.

직관적으로, 하나의 큰 필터를 갖는 CONV 레이어보다, 작은 필터를 갖는 여러 개의 CONV 레이어를 쌓는 것이 더 적은 파라미터만 사용하면서도 입력으로부터 더 좋은 feature 를 추출하게 해준다. 단점이 있다면, backpropagation 을 할 때 CONV 레이어의 중간 결과들을 저장하기 위해 더 많은 메모리 공간을 잡고 있어야 한다는 것이다.

3) Pooling Layer 의 크기 3x3, Stride 2x2

INPUT -> [CONV -> RELU -> POOL]*3 -> FC -> RELU -> DROPOUT > SOFTMAX
CONV 필터 크기는 3x3 이고, Stride 는 1x1 이고, 3층이며 필터 수는 각각 32개, 64개, 128개이다.
POOL 필터 크기는 3x3 이고, Stride는 2x2 이고, MAX POOL을 사용한다.

현재는 대부분의 Pooling 필터가 2x2 크기에 stride 를 2 로 놓고 사용한다. 하지만 덜 사용되는 옵션으로 3x3 크기의 필터로, stride 에 2 를 넣고 풀링을 수행하는 방법이 있다고 해서 사용해보았다.

보통 3 보다 큰 크기로 풀링을 수행할 경우 너무 많은 정보를 버리게 되므로 거의 사용하지 않는다고 한다. 많은 정보 손실은 곧 성능 하락으로 이어지기 때문이다.

4) AVG Pooling, L2 Norm Pooling 사용

INPUT -> [CONV -> RELU -> POOL]*3 -> FC -> RELU -> DROPOUT > SOFTMAX
CONV 필터 크기는 3x3 이고, Stride 는 1x1 이고, 3층이며 필터 수는 각각 32개, 64개이다, 128개이다.
POOL 필터 크기는 2x2 이고, Stride는 2x2 이고, AVG POOL을 사용한다.

INPUT -> [CONV -> RELU -> POOL]*3 -> FC -> RELU -> DROPOUT > SOFTMAX
CONV 필터 크기는 3x3 이고, Stride 는 1x1 이고, 3층이며 필터 수는 각각 32개, 64개, 128개이다.
POOL 필터 크기는 2x2 이고, Stride는 2x2 이고, L2 NORM POOL을 사용한다.

Tensorflow에 AVG 풀링 내장함수도 존재하는 것을 보고 사용해 보았다. 예전에는 AVG 풀링이 많이 쓰였으나 최근에는 Max 풀링이 더 좋은 성능을 보여 더 많이 쓰인다고 한다. 그리고 L2-norm 풀링도 가능한 것 같아서 사용해 보았다. L1-norm 은 사용하는 사례가 없길래 제외했다.

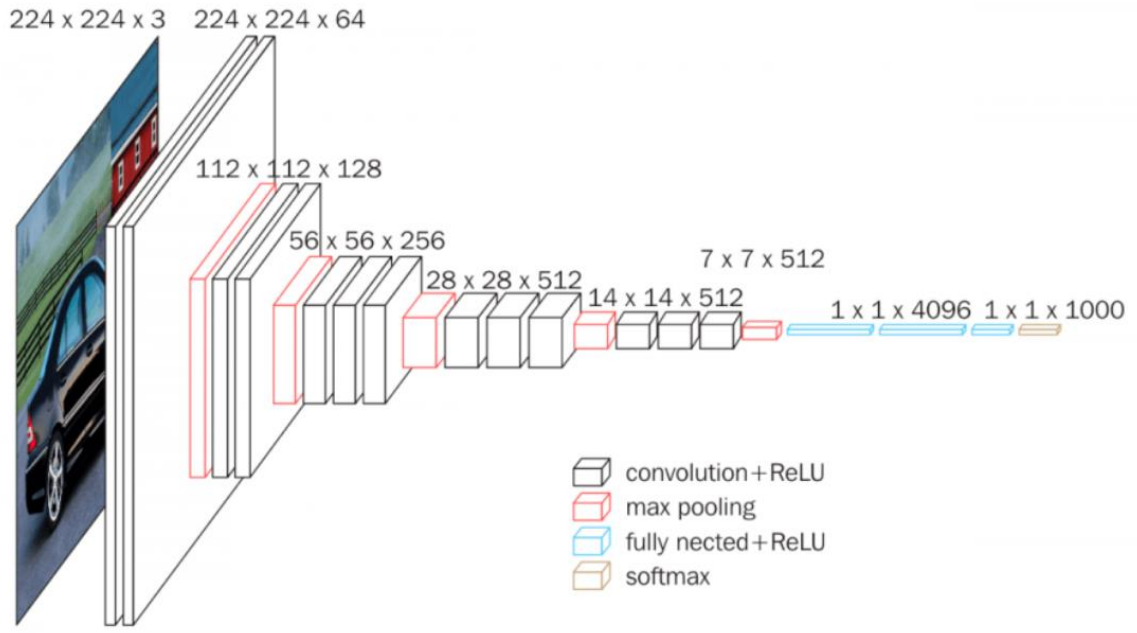
5) VggNet, GoogLeNet 참고 코드

INPUT -> [CONVi_1 -> RELU -> CONVi_2 -> RELU -> POOL]*3 -> FC -> RELU -> DROPOUT > SOFTMAX
CONV1_1 필터 크기는 3x3x1 이고, Stride 는 1x1 이고, 개수는 1개이다.
CONV1_2 필터 크기는 3x3x32 이고, Stride 는 1x1 이고, 개수는 1개이다.
CONV2_1 필터 크기는 3x3x32 이고, Stride 는 1x1 이고, 개수는 1개이다.
CONV2_2 필터 크기는 3x3x64 이고, Stride 는 1x1 이고, 개수는 1개이다.
CONV3_1 필터 크기는 3x3x64 이고, Stride 는 1x1 이고, 개수는 1개이다.
CONV3_2 필터 크기는 3x3x128 이고, Stride 는 1x1 이고, 개수는 1개이다.
POOL 필터 크기는 2x2 이고, Stride는 2x2 이고, MAX POOL을 사용한다.

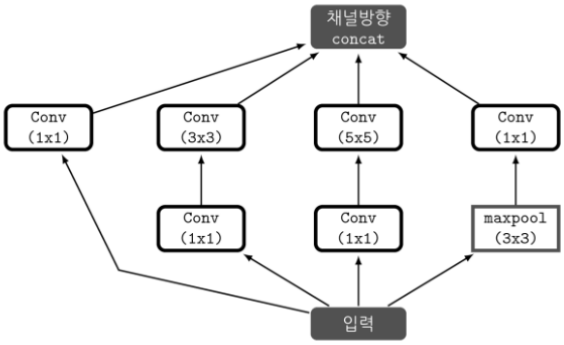
CNN의 대표적인 모델중에 GoogLeNet과, VGGNet이란 모델이 있어서, 해당 모델들을 공부하고, 비슷하게 따라 만들어 보았다. Conv -> ReLU -> Pool 을 사용하지 않고, Conv -> Conv -> ReLU -> Pool 과 같이 만들어 보았다.

VGGNet 이란, Karen Simonyan 과 Andrew Zisserman 이 만든 ILSVRC 2014 에서 2 등을 한 네트워크 모델이다. 이 모델의 가장 큰 기여는 네트워크의 깊이가 좋은 성능에 있어 매우 중요한 요소라는 것을 보여준 것이다. 이들이 제안한 여러 개 모델 중 가장 좋은 것은 16 개의 CONV/FC 레이어로 이뤄지며, 모든 컨볼루션은 3x3, 모든 풀링은 2x2 만으로 이뤄져 있다. 비록 GoogLeNet 보다 이미지 분류 성능은 약간 낮지만, 여러 Transfer Learning 과제에서 더 좋은 성능을 보인다는 것이 나중에 밝혀졌다. 그래서 VGGNet 은 최근에 이미지 feature 추출을 위해 가장 많이 사용되고 있다. Caffe 를 사용하면 Pretrained model 을 받아 바로 사용하는 것도 가능하다. VGGNet 의 단점은, 매우 많은 메모리를 사용하며 (140M) 많은 연산을 한다는 것이다.

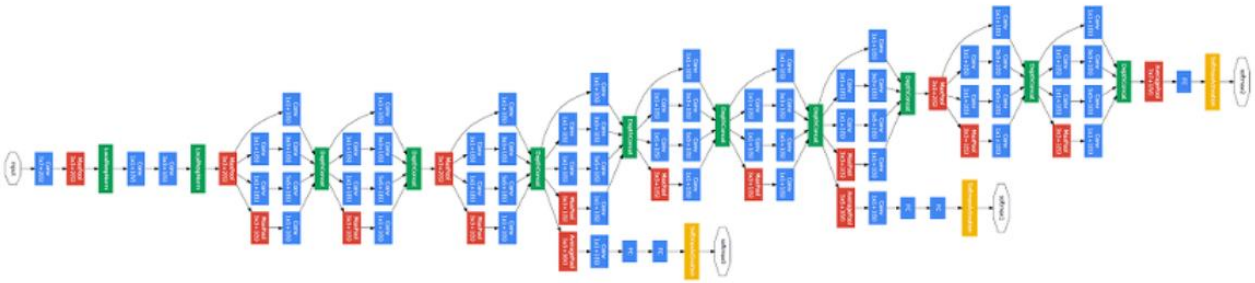
<VGGNet 모델의 구조를 나타낸 그림>



GoogLeNet 은 구글의 Szegedy et al.가 만든 ILSVRC 2014 의 승자 모델이다. 이 모델의 가장 큰 기여는 파라미터의 개수를 엄청나게 줄여주는 Inception module 을 제안한 것이다 (4M, AlexNet 의 경우 60M). 뿐만 아니라, ConvNet 마지막에 FC 레이어 대신 Average 풀링을 사용해 별로 중요하지 않아 보이는 파라미터들을 많이 줄이게 된다. 구글이 고민한 것은 어떻게 노드간의 연결을 줄이면서, 행렬 연산은 Dense 연산을 하도록 처리하는가 였다. 이 모듈을 통해 GoogleNet 이 이렇게 깊은 망을 만들고도 학습이 가능했던 것이다.



<GoogLeNet 의 구조를 나타낸 그림>



6) 'All Convolution Net' 논문 참고 코드

INPUT -> [CONV -> RELU]*3 -> FC -> RELU -> DROPOUT > SOFTMAX
CONV 필터 크기는 3x3 이고, Stride 는 2x2 이고, 3층이며 필터 수는 각각 32개, 64개이다.
POOLING은 사용하지 않는다

'Striving for Simplicity: The All Convolutional Net'라는 논문에서 컨볼루션 레이어만 반복하며 풀링 레이어를 사용하지 않는 방식을 제안한다. Representation 의 크기를 줄이기 위해 가끔씩 큰 stride 를 가진 컨볼루션 레이어를 사용한다. 풀링 레이어가 보통 이미지의 크기를 심하게 줄이기 때문에, 최근 추세는 풀링 레이어를 사용하지 않는 쪽으로 발전하고 있다고 한다.

이를 참고하여, Conv 층으로만 구성된 CNN 을 디자인해보았다.

4. 결론

예제코드

```
Step: 500, Loss: 1459.561768, Accuracy: 0.954900
Step: 1000, Loss: 955.638245, Accuracy: 0.970900
Step: 1500, Loss: 767.265503, Accuracy: 0.973600
Step: 2000, Loss: 656.851135, Accuracy: 0.979100
Step: 2500, Loss: 580.280396, Accuracy: 0.980200
Step: 3000, Loss: 510.467285, Accuracy: 0.983900
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-pa
Instructions for updating:
Use standard file APIs to delete files with this prefix.
Step: 3500, Loss: 462.890320, Accuracy: 0.983500
Step: 4000, Loss: 437.105347, Accuracy: 0.985900
Step: 4500, Loss: 399.214722, Accuracy: 0.986800
Step: 5000, Loss: 381.797821, Accuracy: 0.986800
Step: 5500, Loss: 389.958832, Accuracy: 0.985600
Step: 6000, Loss: 377.912903, Accuracy: 0.987000
Step: 6500, Loss: 356.498169, Accuracy: 0.987100
Step: 7000, Loss: 341.254578, Accuracy: 0.988800
Step: 7500, Loss: 355.357300, Accuracy: 0.987800
Step: 8000, Loss: 413.995575, Accuracy: 0.986600
Step: 8500, Loss: 363.434570, Accuracy: 0.987100
Step: 9000, Loss: 297.517395, Accuracy: 0.990000
Step: 9500, Loss: 329.583832, Accuracy: 0.989400
Step: 10000, Loss: 325.033051, Accuracy: 0.989000
Step: 10500, Loss: 309.635651, Accuracy: 0.989800
Step: 11000, Loss: 297.066467, Accuracy: 0.990400
Step: 11500, Loss: 297.743225, Accuracy: 0.990300
Step: 12000, Loss: 298.554993, Accuracy: 0.990200
Step: 12500, Loss: 284.765533, Accuracy: 0.989400
Step: 13000, Loss: 319.756012, Accuracy: 0.989900
Step: 13500, Loss: 307.381561, Accuracy: 0.990400
Step: 14000, Loss: 334.749725, Accuracy: 0.989800
Step: 14500, Loss: 325.808594, Accuracy: 0.989400
Step: 15000, Loss: 265.995331, Accuracy: 0.991300
Step: 15500, Loss: 264.316040, Accuracy: 0.991700
Step: 16000, Loss: 310.278748, Accuracy: 0.989800
Step: 16500, Loss: 264.514130, Accuracy: 0.991800
Step: 17000, Loss: 273.701233, Accuracy: 0.991700
Step: 17500, Loss: 299.669739, Accuracy: 0.991000
Step: 18000, Loss: 265.314911, Accuracy: 0.991300
Step: 18500, Loss: 276.278015, Accuracy: 0.991400
Step: 19000, Loss: 283.320068, Accuracy: 0.990900
Step: 19500, Loss: 273.913452, Accuracy: 0.991600
Step: 20000, Loss: 270.747498, Accuracy: 0.991800
```

단순 변형 코드

```
Step: 500, Loss: 1895.239990, Accuracy: 0.945700
Step: 1000, Loss: 1052.232056, Accuracy: 0.968400
Step: 1500, Loss: 855.754761, Accuracy: 0.971100
Step: 2000, Loss: 681.735840, Accuracy: 0.978600
Step: 2500, Loss: 642.358643, Accuracy: 0.978900
Step: 3000, Loss: 599.184082, Accuracy: 0.980600
Step: 3500, Loss: 493.112549, Accuracy: 0.983700
Step: 4000, Loss: 495.016754, Accuracy: 0.984700
Step: 4500, Loss: 456.670349, Accuracy: 0.983600
Step: 5000, Loss: 413.723877, Accuracy: 0.987300
Step: 5500, Loss: 443.164001, Accuracy: 0.986000
Step: 6000, Loss: 399.314331, Accuracy: 0.987100
Step: 6500, Loss: 379.892181, Accuracy: 0.987600
Step: 7000, Loss: 368.163055, Accuracy: 0.987600
Step: 7500, Loss: 367.828125, Accuracy: 0.987400
Step: 8000, Loss: 403.276062, Accuracy: 0.987600
Step: 8500, Loss: 329.194824, Accuracy: 0.990200
Step: 9000, Loss: 324.198547, Accuracy: 0.989100
Step: 9500, Loss: 318.039948, Accuracy: 0.990300
Step: 10000, Loss: 367.245728, Accuracy: 0.988000
Step: 10500, Loss: 330.343628, Accuracy: 0.989400
Step: 11000, Loss: 302.776978, Accuracy: 0.990800
Step: 11500, Loss: 305.716003, Accuracy: 0.990900
Step: 12000, Loss: 330.956360, Accuracy: 0.990400
Step: 12500, Loss: 326.489960, Accuracy: 0.989600
Step: 13000, Loss: 295.805450, Accuracy: 0.991000
Step: 13500, Loss: 312.282501, Accuracy: 0.990600
Step: 14000, Loss: 279.742828, Accuracy: 0.991700
Step: 14500, Loss: 321.050568, Accuracy: 0.990100
Step: 15000, Loss: 264.983368, Accuracy: 0.991500
Step: 15500, Loss: 279.445587, Accuracy: 0.991400
Step: 16000, Loss: 281.666718, Accuracy: 0.992000
Step: 16500, Loss: 339.679321, Accuracy: 0.989300
Step: 17000, Loss: 312.785645, Accuracy: 0.990800
Step: 17500, Loss: 260.366882, Accuracy: 0.991900
Step: 18000, Loss: 249.814789, Accuracy: 0.992700
Step: 18500, Loss: 265.322906, Accuracy: 0.991700
Step: 19000, Loss: 298.007324, Accuracy: 0.991100
Step: 19500, Loss: 299.596649, Accuracy: 0.991000
Step: 20000, Loss: 285.303284, Accuracy: 0.991200
```

풀링 필터 크기 변경

```
Step: 500, Loss: 1827.890137, Accuracy: 0.945800
Step: 1000, Loss: 917.305359, Accuracy: 0.970900
Step: 1500, Loss: 737.673950, Accuracy: 0.975100
Step: 2000, Loss: 602.606384, Accuracy: 0.978700
Step: 2500, Loss: 548.882446, Accuracy: 0.982100
Step: 3000, Loss: 507.734314, Accuracy: 0.982700
Step: 3500, Loss: 435.700195, Accuracy: 0.984500
Step: 4000, Loss: 434.273041, Accuracy: 0.985300
Step: 4500, Loss: 382.583435, Accuracy: 0.986400
Step: 5000, Loss: 375.800079, Accuracy: 0.986200
Step: 5500, Loss: 356.439880, Accuracy: 0.987700
Step: 6000, Loss: 299.028717, Accuracy: 0.989300
Step: 6500, Loss: 276.727356, Accuracy: 0.990000
Step: 7000, Loss: 366.474670, Accuracy: 0.988000
Step: 7500, Loss: 309.766602, Accuracy: 0.988500
Step: 8000, Loss: 330.614105, Accuracy: 0.988000
Step: 8500, Loss: 257.149261, Accuracy: 0.990600
Step: 9000, Loss: 252.673569, Accuracy: 0.990900
Step: 9500, Loss: 263.249634, Accuracy: 0.990300
Step: 10000, Loss: 278.231049, Accuracy: 0.990100
Step: 10500, Loss: 241.033356, Accuracy: 0.991600
Step: 11000, Loss: 252.993011, Accuracy: 0.991100
Step: 11500, Loss: 262.320038, Accuracy: 0.990700
Step: 12000, Loss: 253.016205, Accuracy: 0.990700
Step: 12500, Loss: 261.096802, Accuracy: 0.990900
Step: 13000, Loss: 241.358994, Accuracy: 0.991700
Step: 13500, Loss: 275.054901, Accuracy: 0.991200
Step: 14000, Loss: 271.393738, Accuracy: 0.990800
Step: 14500, Loss: 201.894043, Accuracy: 0.992600
Step: 15000, Loss: 242.323471, Accuracy: 0.991300
Step: 15500, Loss: 290.191956, Accuracy: 0.989400
Step: 16000, Loss: 295.437256, Accuracy: 0.988900
Step: 16500, Loss: 233.381653, Accuracy: 0.991300
Step: 17000, Loss: 247.970978, Accuracy: 0.991700
Step: 17500, Loss: 217.136627, Accuracy: 0.993200
Step: 18000, Loss: 206.154633, Accuracy: 0.992700
Step: 18500, Loss: 236.579712, Accuracy: 0.991400
Step: 19000, Loss: 293.702118, Accuracy: 0.989900
Step: 19500, Loss: 208.520081, Accuracy: 0.992500
Step: 20000, Loss: 233.135422, Accuracy: 0.992100
```

AVG POOL

```
Step: 500, Loss: 3271.672607, Accuracy: 0.904600
Step: 1000, Loss: 2060.479492, Accuracy: 0.940700
Step: 1500, Loss: 1493.189697, Accuracy: 0.954400
Step: 2000, Loss: 1236.320801, Accuracy: 0.961500
Step: 2500, Loss: 1052.539307, Accuracy: 0.966000
Step: 3000, Loss: 906.708130, Accuracy: 0.970500
Step: 3500, Loss: 817.527283, Accuracy: 0.974100
Step: 4000, Loss: 770.357117, Accuracy: 0.974300
Step: 4500, Loss: 666.304321, Accuracy: 0.977400
Step: 5000, Loss: 602.917542, Accuracy: 0.979400
Step: 5500, Loss: 663.407288, Accuracy: 0.977900
Step: 6000, Loss: 565.956238, Accuracy: 0.980200
Step: 6500, Loss: 522.153442, Accuracy: 0.983600
Step: 7000, Loss: 530.214050, Accuracy: 0.981600
Step: 7500, Loss: 506.202026, Accuracy: 0.983100
Step: 8000, Loss: 565.806030, Accuracy: 0.981600
Step: 8500, Loss: 405.790283, Accuracy: 0.986400
Step: 9000, Loss: 416.924744, Accuracy: 0.985900
Step: 9500, Loss: 434.931213, Accuracy: 0.985200
Step: 10000, Loss: 421.056946, Accuracy: 0.985800
Step: 10500, Loss: 410.155701, Accuracy: 0.986700
Step: 11000, Loss: 384.129822, Accuracy: 0.987500
Step: 11500, Loss: 367.053986, Accuracy: 0.986900
Step: 12000, Loss: 403.732025, Accuracy: 0.987300
Step: 12500, Loss: 365.863800, Accuracy: 0.987900
Step: 13000, Loss: 349.518402, Accuracy: 0.988300
Step: 13500, Loss: 341.059845, Accuracy: 0.989200
Step: 14000, Loss: 323.472778, Accuracy: 0.989500
Step: 14500, Loss: 320.564240, Accuracy: 0.988800
Step: 15000, Loss: 322.682922, Accuracy: 0.989000
Step: 15500, Loss: 339.287292, Accuracy: 0.987900
Step: 16000, Loss: 336.724304, Accuracy: 0.989300
Step: 16500, Loss: 299.451813, Accuracy: 0.990700
Step: 17000, Loss: 297.317505, Accuracy: 0.989900
Step: 17500, Loss: 270.348724, Accuracy: 0.990100
Step: 18000, Loss: 295.117096, Accuracy: 0.990300
Step: 18500, Loss: 276.089966, Accuracy: 0.990000
Step: 19000, Loss: 269.409424, Accuracy: 0.990700
Step: 19500, Loss: 258.122192, Accuracy: 0.991300
Step: 20000, Loss: 260.647522, Accuracy: 0.990300
```

L2 NORM POOL

```
Step: 500, Loss: 2665.283203, Accuracy: 0.922600
Step: 1000, Loss: 1576.662720, Accuracy: 0.951800
Step: 1500, Loss: 1249.111938, Accuracy: 0.960700
Step: 2000, Loss: 913.077271, Accuracy: 0.972100
Step: 2500, Loss: 807.068848, Accuracy: 0.973200
Step: 3000, Loss: 727.119751, Accuracy: 0.975700
Step: 3500, Loss: 630.249329, Accuracy: 0.978900
Step: 4000, Loss: 577.198792, Accuracy: 0.981600
Step: 4500, Loss: 515.691101, Accuracy: 0.982900
Step: 5000, Loss: 462.003784, Accuracy: 0.985500
Step: 5500, Loss: 477.145508, Accuracy: 0.985300
Step: 6000, Loss: 394.364136, Accuracy: 0.986900
Step: 6500, Loss: 402.150635, Accuracy: 0.986600
Step: 7000, Loss: 370.281799, Accuracy: 0.987000
Step: 7500, Loss: 414.265503, Accuracy: 0.985400
Step: 8000, Loss: 390.994293, Accuracy: 0.987600
Step: 8500, Loss: 313.756836, Accuracy: 0.988600
Step: 9000, Loss: 318.868530, Accuracy: 0.988200
Step: 9500, Loss: 372.927765, Accuracy: 0.988000
Step: 10000, Loss: 338.005280, Accuracy: 0.988300
Step: 10500, Loss: 320.689087, Accuracy: 0.989000
Step: 11000, Loss: 295.808044, Accuracy: 0.989900
Step: 11500, Loss: 289.067108, Accuracy: 0.989900
Step: 12000, Loss: 290.368134, Accuracy: 0.989700
Step: 12500, Loss: 298.563904, Accuracy: 0.989900
Step: 13000, Loss: 296.895081, Accuracy: 0.990500
Step: 13500, Loss: 270.001129, Accuracy: 0.990000
Step: 14000, Loss: 259.749329, Accuracy: 0.990800
Step: 14500, Loss: 301.586670, Accuracy: 0.989700
Step: 15000, Loss: 263.703552, Accuracy: 0.990300
Step: 15500, Loss: 268.698303, Accuracy: 0.990300
Step: 16000, Loss: 273.294342, Accuracy: 0.990500
Step: 16500, Loss: 267.153412, Accuracy: 0.990800
Step: 17000, Loss: 228.915222, Accuracy: 0.992100
Step: 17500, Loss: 233.481140, Accuracy: 0.992000
Step: 18000, Loss: 240.943008, Accuracy: 0.990600
Step: 18500, Loss: 262.143707, Accuracy: 0.990600
Step: 19000, Loss: 242.880219, Accuracy: 0.991300
Step: 19500, Loss: 240.149673, Accuracy: 0.991100
Step: 20000, Loss: 236.596848, Accuracy: 0.991800
```

VggNet, GoogLeNet 참고

```
Step: 500, Loss: 23063.441406, Accuracy: 0.113500
Step: 1000, Loss: 23093.640625, Accuracy: 0.097400
Step: 1500, Loss: 22939.089844, Accuracy: 0.172300
Step: 2000, Loss: 7695.068359, Accuracy: 0.779900
Step: 2500, Loss: 5026.540527, Accuracy: 0.851000
Step: 3000, Loss: 3825.431396, Accuracy: 0.885800
Step: 3500, Loss: 3018.037598, Accuracy: 0.911500
Step: 4000, Loss: 2569.554199, Accuracy: 0.922100
Step: 4500, Loss: 2213.478760, Accuracy: 0.933700
Step: 5000, Loss: 1959.145142, Accuracy: 0.940200
Step: 5500, Loss: 1794.832642, Accuracy: 0.946100
Step: 6000, Loss: 1570.968384, Accuracy: 0.953600
Step: 6500, Loss: 1442.686890, Accuracy: 0.957400
Step: 7000, Loss: 1383.531982, Accuracy: 0.958100
Step: 7500, Loss: 1291.664917, Accuracy: 0.960100
Step: 8000, Loss: 1223.900757, Accuracy: 0.962100
Step: 8500, Loss: 1141.617676, Accuracy: 0.965600
Step: 9000, Loss: 1050.357666, Accuracy: 0.966300
Step: 9500, Loss: 1027.063965, Accuracy: 0.967900
Step: 10000, Loss: 981.267883, Accuracy: 0.969000
Step: 10500, Loss: 927.156616, Accuracy: 0.970400
Step: 11000, Loss: 909.020752, Accuracy: 0.969500
Step: 11500, Loss: 880.240112, Accuracy: 0.970200
Step: 12000, Loss: 838.720642, Accuracy: 0.972700
Step: 12500, Loss: 822.900024, Accuracy: 0.972600
Step: 13000, Loss: 787.302795, Accuracy: 0.975200
Step: 13500, Loss: 813.353271, Accuracy: 0.972300
Step: 14000, Loss: 758.819580, Accuracy: 0.975600
Step: 14500, Loss: 742.752197, Accuracy: 0.976700
Step: 15000, Loss: 722.477661, Accuracy: 0.976100
Step: 15500, Loss: 722.373169, Accuracy: 0.976200
Step: 16000, Loss: 725.320740, Accuracy: 0.975300
Step: 16500, Loss: 704.670532, Accuracy: 0.977500
Step: 17000, Loss: 662.177429, Accuracy: 0.978100
Step: 17500, Loss: 663.459595, Accuracy: 0.977400
Step: 18000, Loss: 678.219543, Accuracy: 0.977600
Step: 18500, Loss: 636.862915, Accuracy: 0.978500
Step: 19000, Loss: 623.875610, Accuracy: 0.979100
Step: 19500, Loss: 617.312744, Accuracy: 0.979300
Step: 20000, Loss: 615.094727, Accuracy: 0.979700
```

All Convolution Net

```
Step: 500, Loss: 2485.467529, Accuracy: 0.924000
Step: 1000, Loss: 1741.250732, Accuracy: 0.948600
Step: 1500, Loss: 1358.299316, Accuracy: 0.959400
Step: 2000, Loss: 1162.624023, Accuracy: 0.963900
Step: 2500, Loss: 1096.795410, Accuracy: 0.967200
Step: 3000, Loss: 981.356812, Accuracy: 0.968600
Step: 3500, Loss: 834.316772, Accuracy: 0.974100
Step: 4000, Loss: 795.587402, Accuracy: 0.975500
Step: 4500, Loss: 722.895264, Accuracy: 0.977000
Step: 5000, Loss: 706.344238, Accuracy: 0.976500
Step: 5500, Loss: 699.070007, Accuracy: 0.977000
Step: 6000, Loss: 615.580994, Accuracy: 0.980500
Step: 6500, Loss: 598.287354, Accuracy: 0.980600
Step: 7000, Loss: 628.060364, Accuracy: 0.980900
Step: 7500, Loss: 584.958496, Accuracy: 0.981100
Step: 8000, Loss: 603.274231, Accuracy: 0.980900
Step: 8500, Loss: 506.202881, Accuracy: 0.983600
Step: 9000, Loss: 484.459778, Accuracy: 0.984400
Step: 9500, Loss: 552.302979, Accuracy: 0.982300
Step: 10000, Loss: 482.846588, Accuracy: 0.983700
Step: 10500, Loss: 513.089172, Accuracy: 0.983800
Step: 11000, Loss: 451.384369, Accuracy: 0.985500
Step: 11500, Loss: 466.903931, Accuracy: 0.984300
Step: 12000, Loss: 470.313599, Accuracy: 0.985300
Step: 12500, Loss: 466.324005, Accuracy: 0.984500
Step: 13000, Loss: 461.448517, Accuracy: 0.985500
Step: 13500, Loss: 437.905518, Accuracy: 0.985600
Step: 14000, Loss: 443.259766, Accuracy: 0.985400
Step: 14500, Loss: 421.411987, Accuracy: 0.986100
Step: 15000, Loss: 417.355408, Accuracy: 0.986500
Step: 15500, Loss: 406.815582, Accuracy: 0.985600
Step: 16000, Loss: 437.808838, Accuracy: 0.985100
Step: 16500, Loss: 401.181305, Accuracy: 0.987100
Step: 17000, Loss: 460.554901, Accuracy: 0.985200
Step: 17500, Loss: 381.053406, Accuracy: 0.986300
Step: 18000, Loss: 379.854187, Accuracy: 0.986900
Step: 18500, Loss: 418.111115, Accuracy: 0.986000
Step: 19000, Loss: 398.692871, Accuracy: 0.986800
Step: 19500, Loss: 380.208282, Accuracy: 0.985800
Step: 20000, Loss: 416.816406, Accuracy: 0.986500
```

이름	Accuracy
예제코드	0.991800
단순 변형 코드	0.991200
풀링 필터 크기 변경 코드	0.992100
AVG 풀링 사용 코드	0.990300
L2 Norm 풀링 사용 코드	0.991800
VggNet, GoogLeNet 참고 코드	0.979700
'All Convolution Net' 논문 참고 코드	0.986500

결론적으로, 가장 높았던 Accuracy 는 풀링 필터의 크기를 3x3 으로 변화시킨 코드의 99.2%의 정확도였다.

첫째로는, 확실히 Max 풀링이, AVG 풀링이나 L2 Norm 풀링 보다 성능이 좋았던 것을 볼 수 있다.

둘째로는, VggNet, GoogLeNet 참고 코드와, 논문 참고 코드의 성능이 안좋았던 것이 아쉽다. 이는 이러한 CNN 디자인이 지금의 3 층 보다 훨씬 깊은 층을 고려하여 만든 것이기 때문에, 현재 3 층짜리 CNN 에서는 성능이 안좋았던 것이라 예상한다.

5. 참고 및 출처

차례대로

1. CNN 참고 글(CS231n 번역)
2. Tensorflow 에서 L2 Norm Pooling 하는 코드 참고
3. Tensorflow 의 AVG pooling 내장함수 레퍼런스
4. VGGNet(VGG16) 참고 블로그
5. GoogleNet 참고 블로그
6. GoogleNet 참고 블로그 2
7. All Convolution Net 논문
8. Padding='SAME' 옵션의 출력 크기 계산 참고 블로그

http://aikorea.org/cs231n/convolutional-networks/

https://stackoverflow.com/questions/36620995/how-to-use-l2-pooling-in-tensorflow

https://www.tensorflow.org/api_docs/python/tf/nn/avg_pool

https://bskyvision.com/504

https://ikkison.tistory.com/86

https://datascienceschool.net/view-notebook/8d34d65bcced42ef84996b5d56321ba9/

http://arxiv.org/abs/1412.6806

https://coderkoo.tistory.com/13