PyramidNet과 DenseNet 모델에 적합한 하이퍼 파라미터의 설계 실험

Team08 (김국진, 김장언, 진주성)

*초록— 이번 competition의 목적은 200만 개 이하의 파라미터를 갖는 모델의 정확도를 최대화하는 것이다. 모델의 성능을 높이기 위해서 데이터 증강, 모델 하이퍼 파라미터 조정, 학습 하이퍼 파라미터 조정, 옵티마이저, 활성화 함수 변경 등 다양한 실험을 시도했다. 다양한 실험을 한 모델은 PyramidNet, DenseNet을 두 가지이다. 모델 layer 개수, block 개수, growth rate에 집중하였고 학습 파라미터는 세 가지 스케줄러를 통해 학습률을 변경하면서 성능을 높였다. 그 결과, 테스트 데이터셋에서 약 90%의 정확도로 데이터를 분류할 수 있었다.*

*Keyword* — Data Augumentation, Depth, PyramidNet, DenseNet, Cosannealing, Learning rate , Hyper Parameter

# **소개**

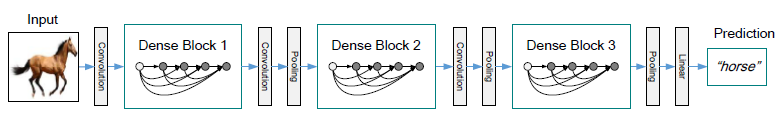
이미지 분류 기술은 CNN 기반의 딥러닝을 활용하여 5개의 합성곱 Layer와 3개의 Fully-connected Layer를 가진 AlexNet, 3\*3 커널 및 ReLU를 추가한 VGGNet, skip connection을 활용한 더 깊은 구조의 ResNet 으로 발전해왔다. 최근에는 비용 및 오류율 감소를 목표로 한 간소화된 구조의 SpinalNet, PyramidNet 등이 연구되고 있다. 또한 동일한 구조에서 더 나은 성능을 위한 최적화 기법 연구 역시 활발하다. 단순하고 구현이 쉬운 SGD에서 효율적인 Gradient갱신을 위한 모멘텀, Adam 등으로 발전해왔다. 최근에는 역전파 과정에 변화를 주는 방식으로 균일한 손실의 이웃 파라미터를 추적하는 SAM이 연구되고 있다. 최신의 모델과 기술들은 이미지 분류 문제에서 대부분 95% 이상의 정확도를 보이며, 이제는 기존 모델을 활용하여 Object Detection, Object Segmentation 분야들이 연구되고 있다. Region Proposal 알고리즘이 CNN 모델과 결합하여 R-CNN이 탄생하였고 최신의 논문들은 Region Proposal 영역을 network로 구성하여 end-to-end 방식으로 학습 가능한 Faster R-CNN 모델이 나왔다.

이번 competition의 목적은 제한된 조건에서 최상의 성능을 얻어내는 것이다. 파라미터 2백만 개는 많은 수가 아니다. ResNet과 DenseNet의 경우에서 층을 하는 경우 적게는 1.7백만 개, 많게 하는 경우 25.6백만 개의 파라미터를 가진다. 이러한 제한된 조건에서 성능을 높일 수 있는 방법은 하이퍼 파라미터 조정, 제시된 모델의 변형, 데이터 증강이다. 여러가지 실험을 통해 최적의 모델을 찾아 내야 한다.

데이터는 32x32의 RGB 이미지이다. Train, valid, test 이미지는 각각 9만 개가 존재한다. 분류는 10가지이다. 비행기, 트럭, 배, 말 등으로 구성된다. Dataloader의 Shuffle 특성을 이용하여 무작위로 사진을 확인을 해봤다. 대부분 사진이라는 특징 때문에 피사체가 중심에 있지만 중심에서 벗어난 사진들도 여러 있었다. 그뿐만 아니라 고양이를 안고 있는 사람과 같이 분류의 난이도가 높은 사진들도 확인할 수 있었다. 이러한 상황에서 모델의 성능을 높이기 위해서는 데이터를 전처리, 어떤 변형을 사용해서 좋은 특징을 찾아 일반화 능력을 높일 것인지도 중요한 요소라고 판단했다.

## 딥러닝 모델

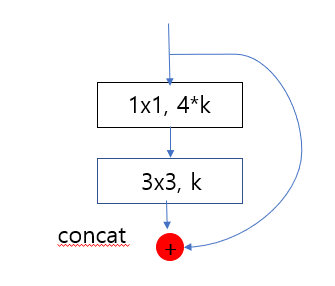
1) DenseNet: Densely connected 된 CNN 구조를 의미한다. 다음 그림을 통해서 설명해본다.



Dense Block이라고 부르는 하나의 connection 블록을 만든다. Convolution, Pooling layer와 함께 순차적으로 Dense Block들을 거쳐서 마지막에 Linear layer를 거쳐서 결과를 도출한다.

네트워크가 Densely 연결되어 있기 때문에 자칫 feature map의 수가 과도하게 많아지는 문제가 발생한다. 이를 해결하기 위해 Growth rate라는 하이퍼 파라미터를 만들어 feature map의 수를 조정해준다. 이 Growth rate를 활용해 Bottleneck Layer를 구현하여 feature map의 수를

조정할 수 있다.



Bottleneck Layer를 논문에 있는 방식으로 도식화. 1x1 convolution에서 4\*k만큼 채널을 늘려주는 것이 특징이다.

Transition Layer를 통해서 feature map의 사이즈와 개수를 줄여준다. Dense block 뒤에 적용되며 Batch Normalization, ReLU, 1x1 convolution, 2x2 average pooling으로 구성된다. Theta를 정의하여 1x1 conv를 줄어드는 feature map의 개수를 정할 수 있다.

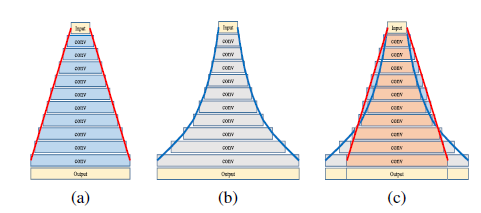
(2) PyramidNet : PyramidNet는 기존의 ResNet의 down sampling으로 인해 발생하는 급격한 feature map의 차원 증가를 개선하기 위해 제안되었다. ResNet은 down sampling을 진행하지 않는 unit들에서는 convolution 필터 수가 동일하지만 PyramidNet에서는 모든 residual unit에서 convolution filter의 수가 깊이가 깊어질수록 점점 증가하게 되고, 결과적으로 모델 구조가 피라미드와 유사하게 생성된다. 이러한 구조를 통해서 PyramidNet은 기존 ResNet에 비해 더 좋은 일반화 성능을 가지게 된다.

그림 : 피라미드 형태의 모델 구조

따라서 기존 모델의 bottleneck 구조도 아래로 갈수록 convolution filter를 늘리도록 설계되었다.

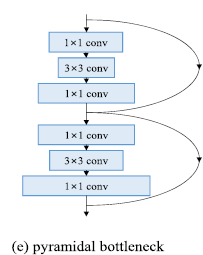
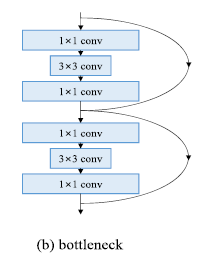


그림 : Resnet에 사용되는 bottleneck구조 (a), PyramidNet에 사용되는 bottleneck 구조 (e)

PyramidNet의 Residual Block은 기존 Block과는 다른 구조로 설계되었다. Residual Block에서 ReLU의 잦은 사용이 모델 성능에 악영향을 끼친다고 판단되어 ReLU를 하나씩 제거하고 Batch Normalization Layer를 대신

추가하였다.

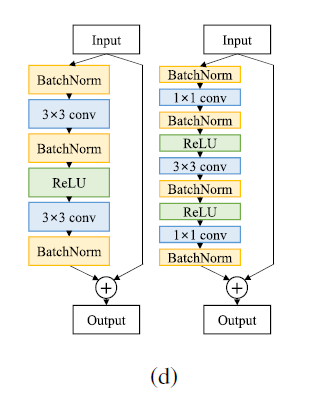
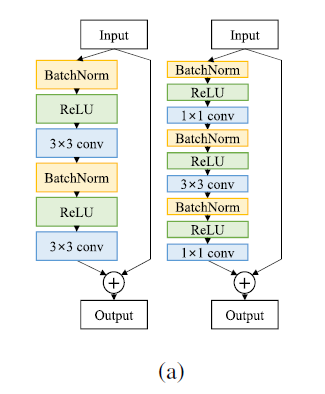
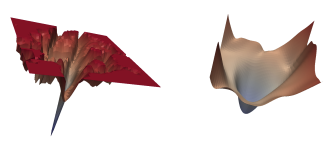


그림 : Resnet에 사용되는 Residual Block구조 (a), PyramidNet에 사용되는 Residual Block 구조 (d)

## 데이터 증강 기법

데이터를 증강시키기 위해 Random Crop, Rotate, Horizontal Flip. Color Jitter, Random Perspective네 가지 방법을 활용했다. Random Crop에 padding을 적용하는 것은 이미지 shift와 같은 효과를 가지고 있다. 해당 실험에서는 무작위 위치를 중심으로 28\*28로 자르고 padding을 4로 적용해준다. 이렇게 되면 원본과 비교하여 위치가 이동되고 pad된 부분은 검은색으로 0으로 취급된다. Rotate의 경우 이미지를 랜덤하게 45도 회전을 시켰고, Flip의 경우는 50%의 확률로 이미지를 반전시켰다. Color Jitter에서 반전, 채도, 포화는 0에서 3사이의 값으로 설정한다. 이 값은 무작위로 선정되기 한다. 다른 색감의 이미지를 생성하려고 시도했다. 또한 마지막으로 Random Perspective는 왜곡의 정도를 0.5로 설정하고 50%의 확률로 적용되도록 했다.

## 옵티마이저

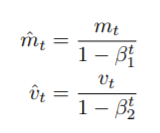
(1) SAM : Overparametered model의 training loss는 일반화 성능에 유효한 설명력을 상실한다. 이에 반해 SAM은 Training loss와 loss sharpness를 동시에 최적화함으로써 모델 전반에 걸쳐 일반화 성능을 개선하는 최적화 기법이다. 모델 학습의 목적이 낮은 population loss임에도 불구하고 단순히 최소 training loss의 파라미터를 찾기 때문에 비슷한 training loss에도 불구하고 각기 다른 population loss가 발생한다. 따라서 단순하게 낮은 training loss를 가진 파라미터를 찾는 것이 아닌 균일하게 낮은 training loss 및 곡률을 가진 이웃 파라미터를 찾아 더 개선된 일반화 성능을 갖는다. 본 실험에서는 두 번의 최적화 과정을 실시한다. 먼저 Sharpness-aware update를 진행하는 기저 optimizer을 설정하고, 하이퍼 파라미터인 rho 값을 지정해 그 범위 내에서 가장 높은 loss를 갖는 weights를 찾는다. 이를 통해 population loss의 upper bound를 최소화한다. 다음으로 loss landscape에서 가장 높은 지점으로부터 gradient를 가진 original weights들을 갱신한다. 그 이후 기저 optimizer을 통해 실질적인 sharpness-aware 갱신한다.

(left) A sharp minimum to which a ResNet trained with SGD (right) A wide minimum to which the same ResNet trained with SAM

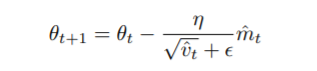


Sharpness-Aware Minimization(SAM)

(2) Adam: Adaptive Moment Estimation(Adam)은 파라미터마다 다른 크기의 update를 적용하는 방법이다. 모평균과 모분산을 얻기 위해 표본평균과 표본제곱의 평균으로 1,2 차 moment를 추정하기 때문에 Moment Estimation이라 불린다. 1,2차 moment를 추정할 때, 초깃값을 0으로 설정할 경우 학습 초기 가중치들이 0으로 편향되기 때문에 bias-corrected 계산을 통해 이를 해결한다. 그 후 Adam update rule을 사용해 파라미터별로 갱신한다.



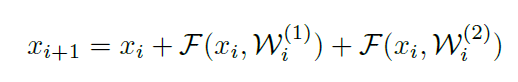
Bias-corrected

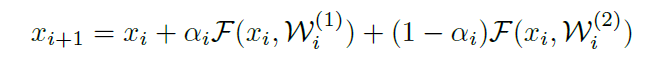


Adam update rule

## 규제

(1) Shake-Shake regularization :





Shake-Shake regularization은 기존의 Residual Function의 출력에 0에서 1사이의 Random한 noise를 곱해줘서 Data augmentation을 한 효과를 주는 방법이다. 학습의 forward에서도 noise를 곱하고 backward에서도 곱해준다. 이렇게 noise를 곱하는 방법은 residual branch간의 관계성을 줄여주어 overfitting이 일어날 확률을 낮추어 주게 된다.

# **제안 방법**

제안 방법에 대한 모델의 형태, 모델의 성능 향상을 위해 적용된 기술들을 그림을 포함하여 구체적으로 기술하시오. (그림은 가져오지 말고 직접 그리세요)

**학습 순서도**

80%에 빠르게 도달하는 **Epoch, Learning Rate, Mini Batch Size**의 설정

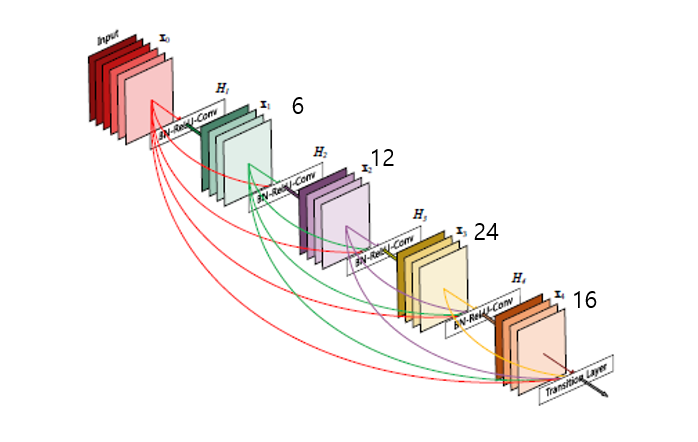
모델 깊이와 너비, 블록 수 등 내부 파라미터의 조정

80%에 빠르게 도달하는 **Epoch, Learning Rate, Mini Batch Size**의 설정

스케줄러 조정, 데이터 증강 기법 사용을 통한 성능 극대화

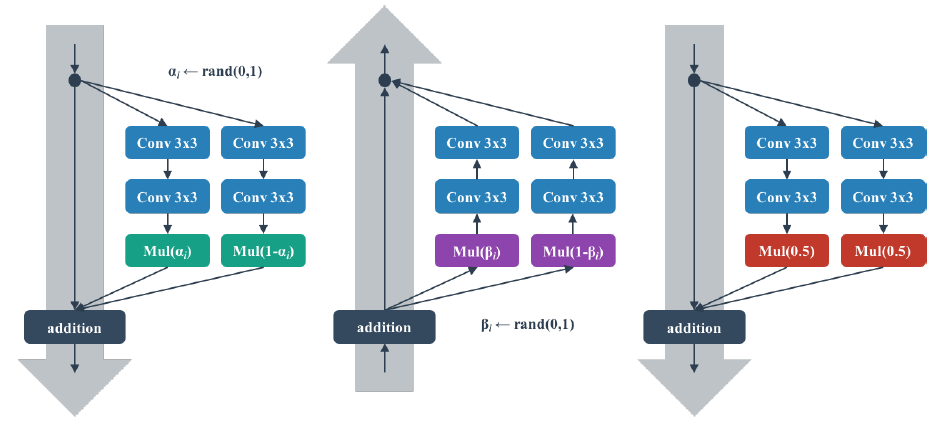
모델의 성능을 향상시키기 위해 가장 우선적으로 고려한 것은 80% 이상의 정확도에 빠르게 도달하기 위해 학습 하이퍼 파라미터인Mini Batch Size, Learning Rate와 Epoch를 조정하는 것이었다. 80% 이상의 모델 성능을 확보한 후, 모델 하이퍼 파라미터를 조정하는 방향으로 성능을 최적화시켰다. 그후 스케줄러를 조정해보고 데이터를 증강하면서 성능 향상을 시도했다.

Densenet)



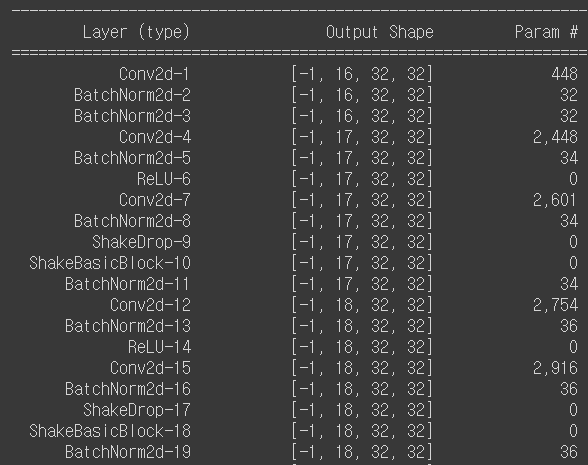
DenseNet의 기본적인 구조는 다음과 같다. 각 단계에서 표시된 숫자는 Block의 개수를 의미한다. 추후 조정하는Block의 수에 따라 레이어의 수는 변하게 된다. 이러한 기본적인 모델에 여러가지 Hyper Parameter를 수정해보며 실험을 진행한다.

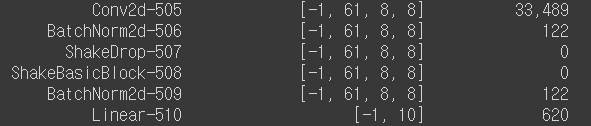
PyramidNet)



모델의 forward와 backward, test 과정에서 Shake Regularization이 적용되는 모습은 위의 그림과 같다. test에서는 alpha값이 0.5로 사용된다.

Torch summary를 통해 모델의 정보를 출력해보면 다음과 같다.





즉, 모델의 구조는 다음과 같은 사다리꼴 형태가 되며, 모델 내부는 ShakeBlock, BatchNormalization, Conv2d, BatchNorm, ReLU 를 포함하는 그룹이 한 unit이 된다.

ShakeBlock

BatchNorm

Conv2d

BatchNorm

ReLU

그림 : 피라미드와 유사한 모델의 구조

모델에서 layer 수를 증가시키면 피라미드의 높이가 높아지고 alpha의 값을 증가시키면 모델 내부 unit의 convolution filter 수가 증가하면서, 피라미드의 너비가 넓어진다.

# **실험**

## 실험 조건

실험은Shake PyramidNet, DenseNet두 가지의 모델에 대해서 진행했다. 학습 하이퍼 파라미터 중 Batch Size, Leaning Rate, Epoch를 변경해보고 모델 하이퍼 파라미터는 Layer 수 Growth Rate, Block 수, alpha를 변경하면서 실험을 진행했다. 데이터의 특성은 32x32x3의 RGB 이미지이다. Cifar-10의 이미지와 이미지 데이터셋이 혼합되어 있다. GPU환경은 Tesla V100 16GB의 Colab이다.

## 하이퍼 파라미터에 대한 실험

다양한 하이퍼 파라미터(예: layer의 개수, filter의 개수, 학습률, 가중치 감쇄 값 등)에 대해 값들을 변경해 가면서 실험을 수행했다면 실험 결과값을 표로 보이고 이를 분석하시오.

##### (1) DenseNet : DenseNet에서는 모델 Layer 수, Growth Rate, Dense Block의 수를 조절해보면서 다양한 실험을 시도했다.

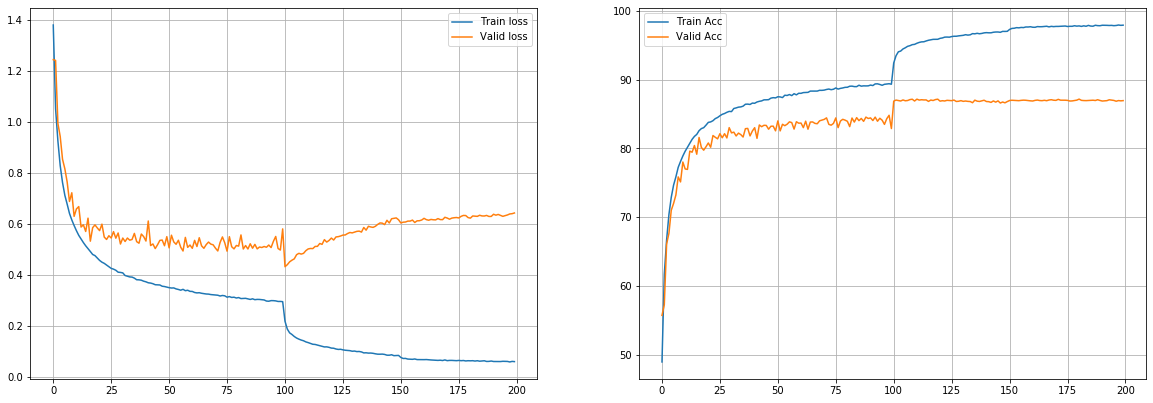
특별한 언급이 없는 한 대부분 Base Line에 명시된 Parameter를 사용한다. 중요한 것을 설명하자면 다음과 같다. Transformation은 기본적으로 주어진 것을 이용한다. Learning rate의 초기 값은 0.001이다. Growth rate는 16이다. Test data는 17만 개, Validation data도 1만 개를 사용했다. Batch Size는 256이다. DenseNet의 layer는 121이다. 단, Learning Scheduler에 대한 실험을 할 때만 train data만을 사용했다. Validation data는 Random하게 뽑아서 한 라벨의 데이터만 Validation data로 사용되는 일이 없도록 한다.

1. Learning scheduler

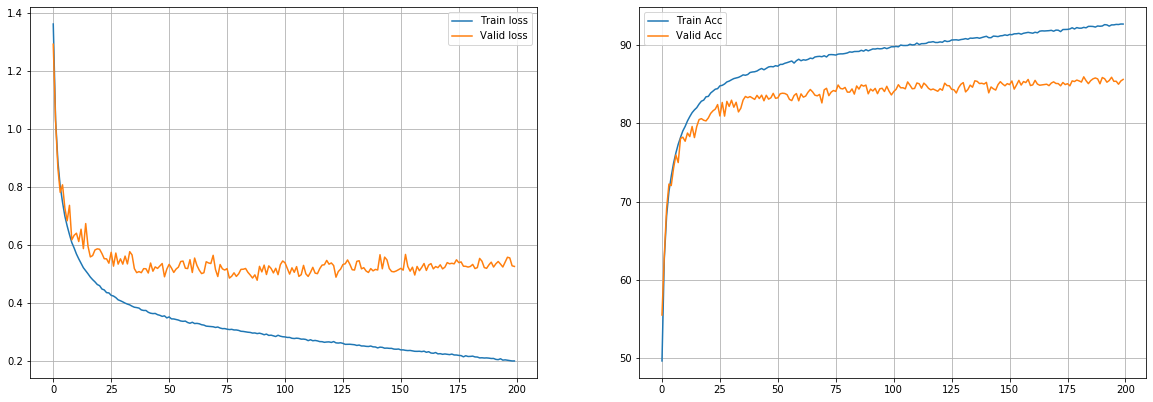
모델을 학습하는 것에 있어서 학습률은 매우 중요하다. 처음에는 학습률을 0.1을 시작으로 했다. Epoch를 아무리 높여도 나아지지 않았다. 경험을 통해서 학습률을 0.001을 초기 세팅으로 하는 것이 가장 좋은 것임을 알았다. Learning scheduler는 Multi step Learning rate scheduler를 사용한다. 이는 사용자가 설정한 epoch의 도달 시 gamma 값을 곱해서 learning rate에 변화를 준다. 일반적으로 큰 학습율로 학습을 하다가 이 값을 줄여주는 방향으로 한다. 이를 더 개선해보고자 Learning rate scheduler를 실험해봤다. 실험을 해본 Scheduler는 Cosine Annealing(이하 COS), ReduceLearingrateOnPlateau Sheduler(이하 RLP)이다. COS는 우리가 익히 아는 코사인 그래프처럼 Learning rate를 최대 epoch에 맞춰서 서서히 감소시키면서 학습을 진행한다. RLP는 기준으로 삼는 것이 개선이 없다면 gamma 값을 곱해서 학습율을 변화시킨다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Densenet Net / Scheduler | Train Accuracy | Valid Accuracy | Epoch | Gamma |
| Multistep | 97.96% | 87.18% | 200 | 0.1 |
| COS | 92.71% | 85.95% | 200 | 0.1 |
| RLP | 95.56% | 87.15% | 200 | 0.1 |

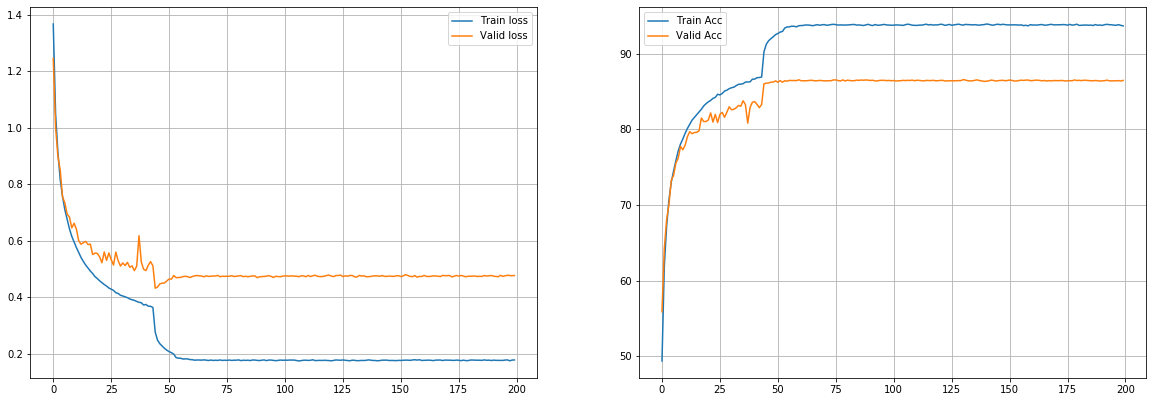
이를 통해 유추할 수 있는 것은 다음과 같다. COS 스케줄러는 Valid Accuracy와 Train Accuracy와의 격차가 좁다. Epoch가 증가하면 더 개선될 여지가 있다. Valid Accuracy의 경우 Multistep이 가장 높지만 Train data에 상당히 과적합이 됐다. RLP는 Multistep과 COS의 중간의 성격으로 보인다. 확실한 우위를 가지는 스케줄러를 확정할 수 없다. 앞으로의 실험에서 여러가지를 계속해서 사용해본다.



Multi Step Scheduler Graph



COS Scheduler Graph



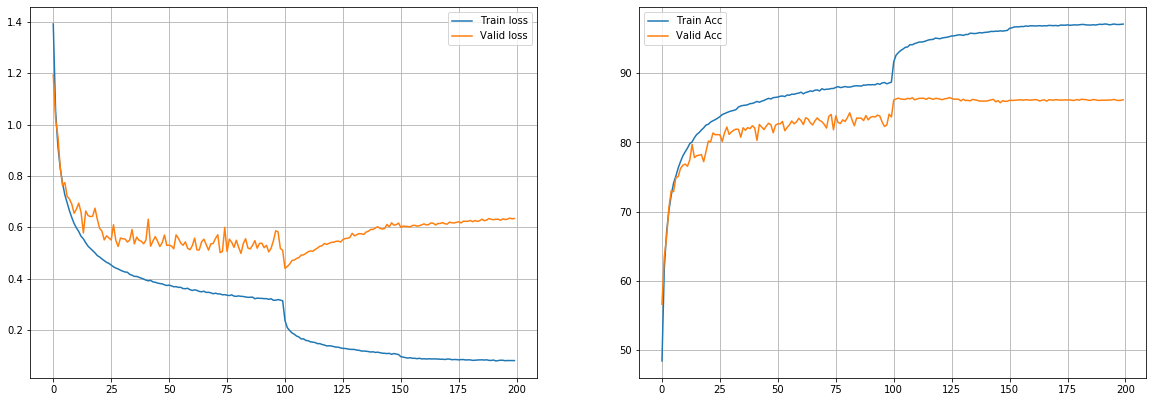
RLP Scheduler

2.Growth rate

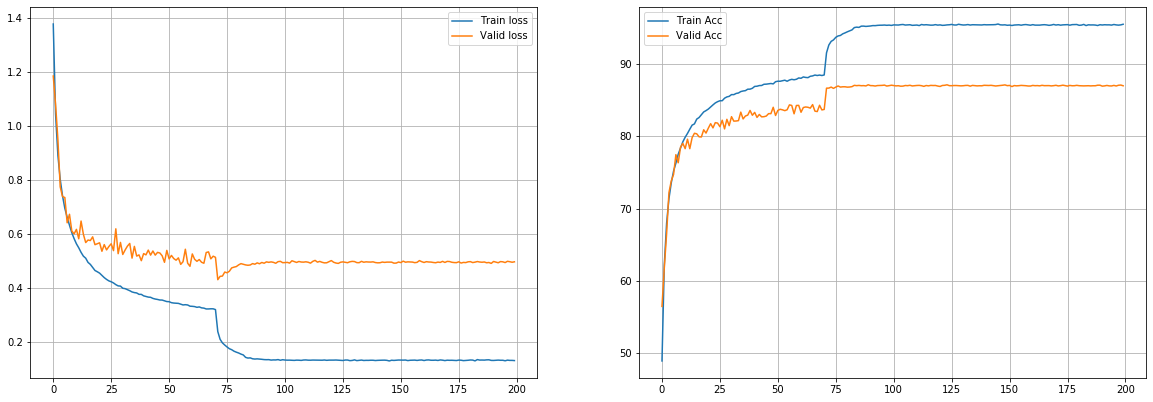
Growth rate는 입력 받은 input을 몇 채널로 늘릴지를 결정한다. 즉 Growth rate를 크게 하면 Parameter 수를 증가시킬 수 있다. Parameter 수의 증가에 따른 성능 개선을 확인해보고자 Growth rate를 12, 16, 17을 적용해본다. 모두 Multistep Scheduler를 사용한다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Densenet Net / Growth rate | Params | Train Accuracy | Valid Accuracy | Epoch |
| 12 | 1.2M | 97.12% | 86.49% | 200 |
| 16 | 1.92M | 97.96% | 87.18% | 200 |
| 17 | 1.98M | 98.17% | 87.64% | 200 |

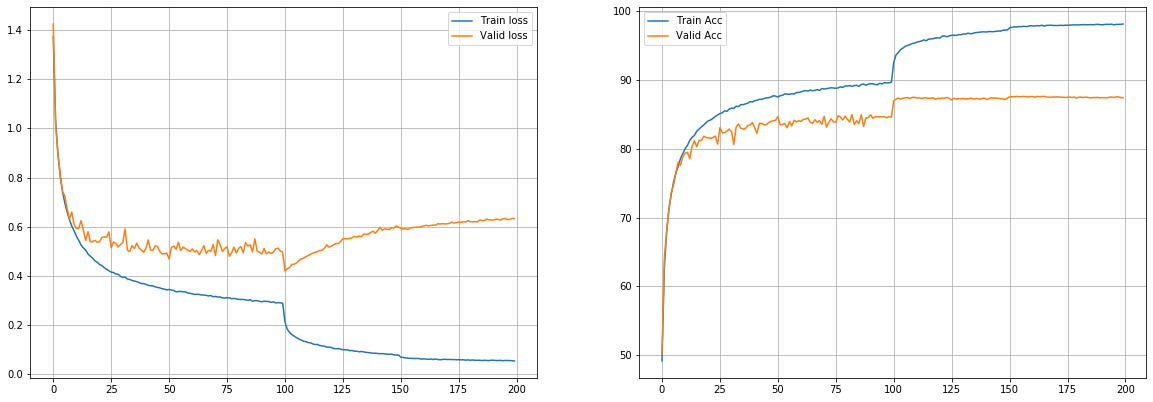
이를 통해서 파라미터가 많을수록 확실히 더 개선이 있음을 알 수 있다. Growth rate가 1 증가했음에도 성능이 0.5% 개선됐다. 성능 향상을 위해선 Parameter를 환경이 허락하는 선에서 최대한 사용하는 것이 바람직함을 확인했다. 앞으로의 실험에서는 대부분 Growth rate를 17로 한다.



Growth rate 12



Growth rate 16

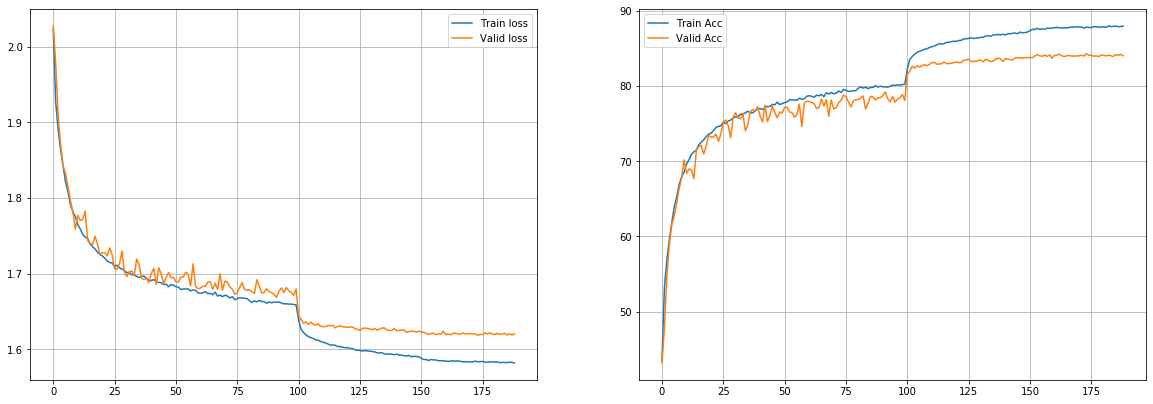


Growth rate 17

3. Softmax

일반적으로 Softmax는 CrossEntropy Loss function에 적용하기에 알맞은 방향으로 변환해준다. 그렇기 때문에 모델의 Gradient를 개선하는 것에 있어 분명히 좋은 영향을 주는 것으로 예상했다. 그러나 실험을 직접 해본 결과 Linear로 계산된 값을 입력한 것보다 성능이 훨씬 좋지 않았다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Densenet Net / output | Growth  Rate | Train Accuracy | Valid Accuracy | Epoch |
| Linear | 17 | 98.17% | 86.49% | 200 |
| Softmax | 17 | 87.98% | 84.3% | 200 |

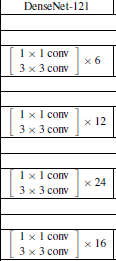


[그림]Softmax 적용시 성능 그래프

우선 train과 valid Accuracy의 차이가 적다는 점은 굉장히 바람직하다. 그러나 같은 조건에서 output을 linear로 처리한 것보다 valid accuracy가 굉장히 작다. 또한 150 epoch에서 learning rate를 변화를 줬음에도 큰 변화가 없다. 그러므로 epoch를 크게 늘린다고 해도 성능 상승이 보장되지 않는다. 또한 colab을 사용하는 환경에서 epoch가 과도하게 크면 커널이 종료되는 문제가 있기 때문에 softmax보다는 linear를 사용해서 output을 사용하기로 했다.

4. Depth 조절

모델의 파라미터를 조절하는 방법에 Block의 개수를 조절하는 방법이 있다.

[그림]Densenet Block 구성

현재 블록을 [6,12,24,16]으로 설정했기에 DenseNet의 layer수가 121층이 되는데 이를 조정하는 것은 곧 Layer 수를 조정하는 것이라고 볼 수 있다. 여기서 모델을 깊게 해보는 것을 집중적으로 실험해보고자 한다.

Case0-> Block[6,12,24,16]

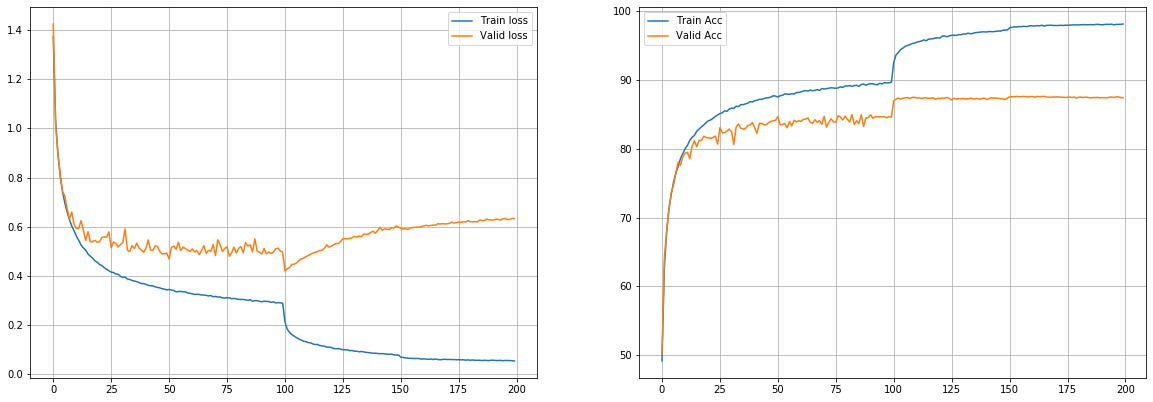
Case1-> Block[3,6,12,24,16,8]

Case2-> Block[6,12,15,18,21,24]

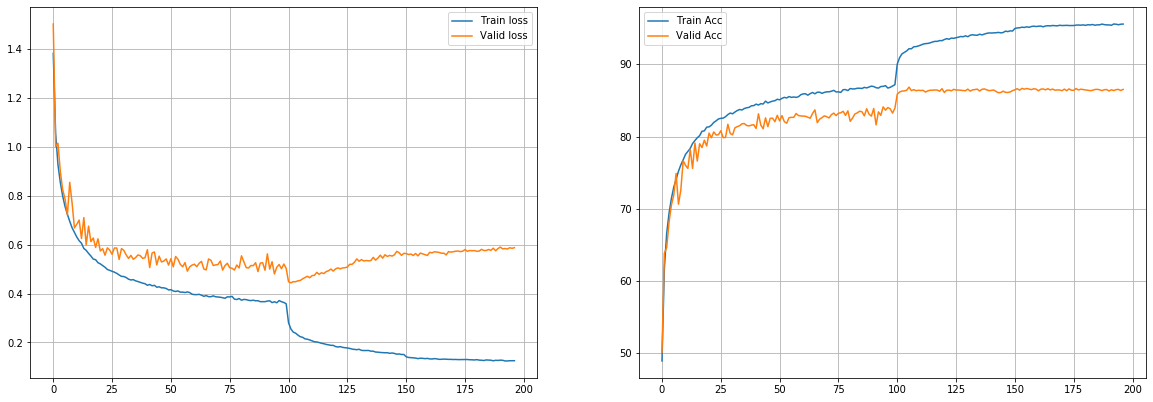
Case3-> Block[6,9,12,15,18,21]

|  |  |  |  |
| --- | --- | --- | --- |
| DenseNet / Block case | Growth rate | Train Accuracy | Valid Accuracy |
| Block[6,12,24,16] | 17 | 98.17% | 87.68% |
| Block[3,6,12,24,16,8] | 17 | 86.1% | 82.48% |
| Block[6,12,15,18,21,24] | 12 | 96.27% | 87.09% |
| Block[6,9,12,15,18,21] | 12 | 95.484% | 86.79% |

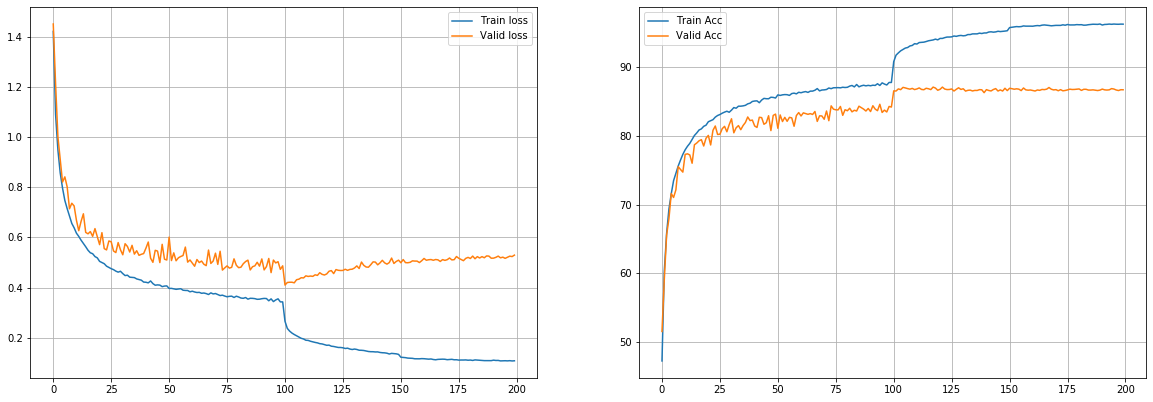
Scheduler는 공통적으로 결과를 보면 알 수 있듯이 layer를 깊게 쌓는다고 해서 결과가 무조건 좋아지진 않았다. 특히 Case1의 경우 성능이 급격하게 떨어졌다. 이에 대한 명확한 근거는 제시하지 못하겠으나 논문의 저자들이 무수한 실험을 거쳐 얻은 최적의 블록 구성이 [6,12,24,16] 인 것으로 추정된다. 또한 층이 깊어질수록 학습 시간은 길어지는데 성능은 좋지 못한 부분은 상당히 아쉬운 대목이다. 그러나 이를 직접 실험해서 확인해보았다는 것에 의의가 있다.



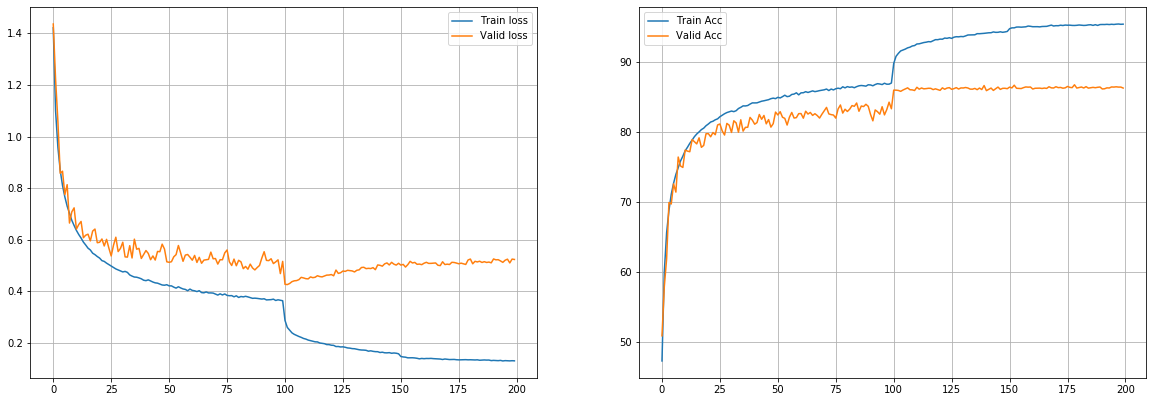
[그림]Case0



[그림]Case1



[그림]Case2



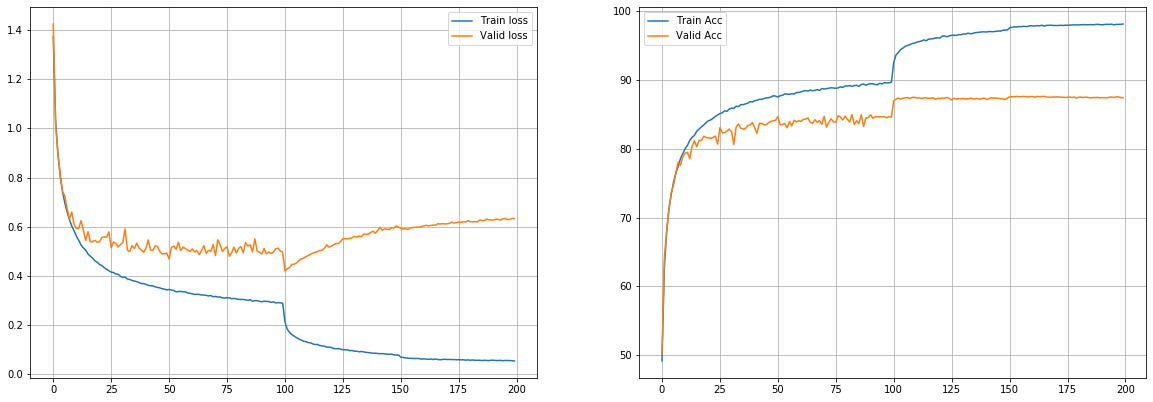
[그림]Case3

5. 활성함수 선택

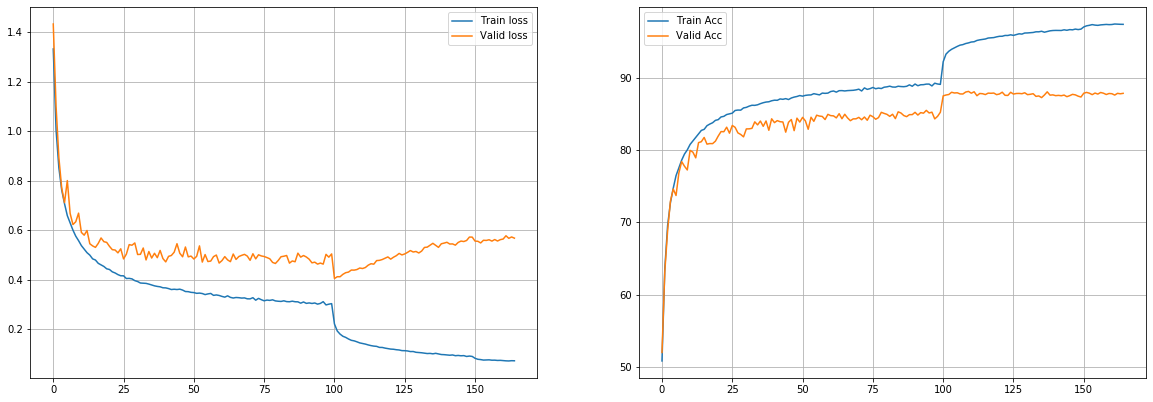
현재까지는 활성함수 모두 ReLU를 사용했다. 다른 활성 함수를 적용해보기 위해서 LeakyReLU 적용을 시도했다. LeakyReLU는 ReLU의 단점을 보완한다고 볼 수 있다. ReLU는 음의 값을 아예 무시하지만 LeakyReLU의 경우 음의 값을 어느 정도 고려할 수 있다. 모든 조건을 동일하게 하고 활성함수만 차이를 뒀다. Scheduler는 Multi Step을 사용했으며 나머지 조건은 동일하다.

|  |  |  |  |
| --- | --- | --- | --- |
| DenseNet / Activation | Growth rate | Train Accuracy | Valid Accuracy |
| ReLU | 17 | 98.17% | 87.68% |
| Leaky ReLU | 17 | 97.5% | 88.16% |

Leaky ReLU를 사용한 것이 상당히 큰 효과가 있었다. 많은 실험을 했지만 최적은 거의 BaseLine의 파라미터들을 사용하는 것이었다. 그런데 LeakyReLU를 사용한 것만으로 성능이 거의 0.5% 상승했다. 이 실험을 가장 마지막에 시도한 것이 상당히 아쉬웠다.



[그림] ReLU사용



[그림]Leaky ReLU

(2) Shake PyramidNet: PyramidNet에서는 모델 Layer 수, 피라미드의 너비를 결정하는 alpha 값을 조정해보면서 실험을 진행했다. 모델의 깊이와 너비 중에서 어떤 요소가 더 중요한지를 밝히기 위해 (layer=110, alpha=48)인 모델과 (layer=48, alpha=96)인 두 모델의 성능을 분석했다

|  |  |  |  |
| --- | --- | --- | --- |
| PyramidNet / Acc | Params | Train Accuracy | Valid Accuracy |
| (layer=110, =48) | 1.8M | 93.96% | 87.68% |
| (layer=48, =96) | 1.92M | 89.86% | 86.81% |

학습결과 깊이가 깊을수록 모델의 표현력이 좋아져서 학습 정확도는 높아지지만 오버피팅이 발생했다. PyramidNet 논문의 내용처럼 너비가 더 넓을수록 모델의 일반화 성능은 더욱 좋아졌다. 아래의 그림은 EarlyStopping을 적용했을 때, 모델의 학습상황을 기록한 그림이다

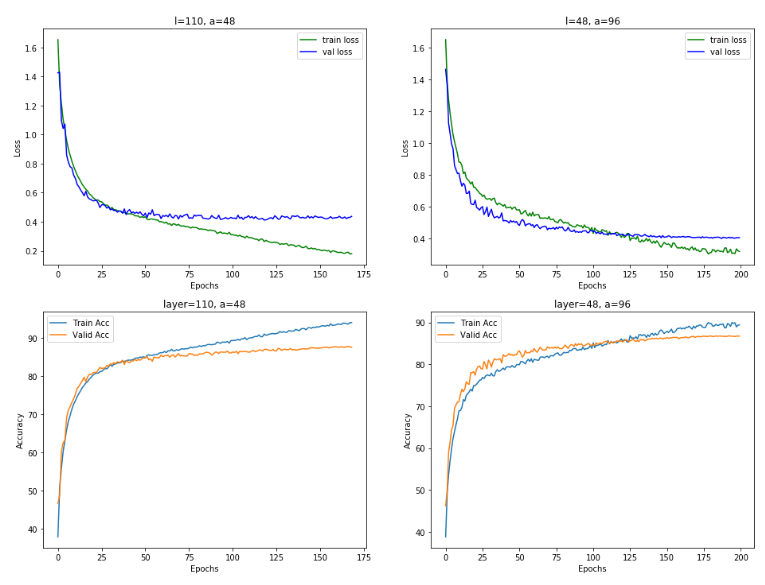
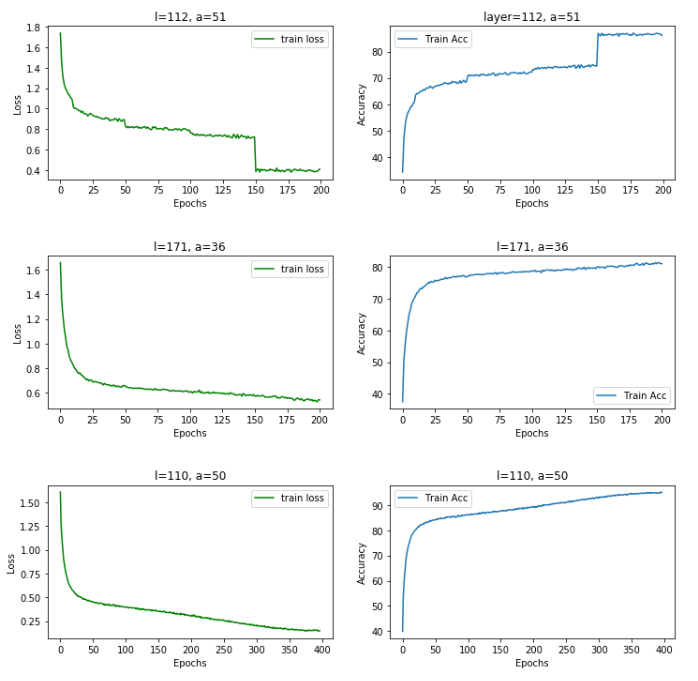
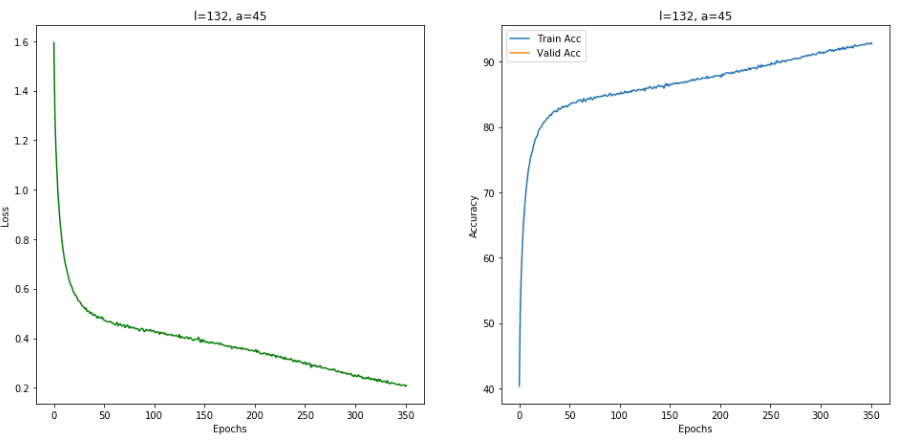


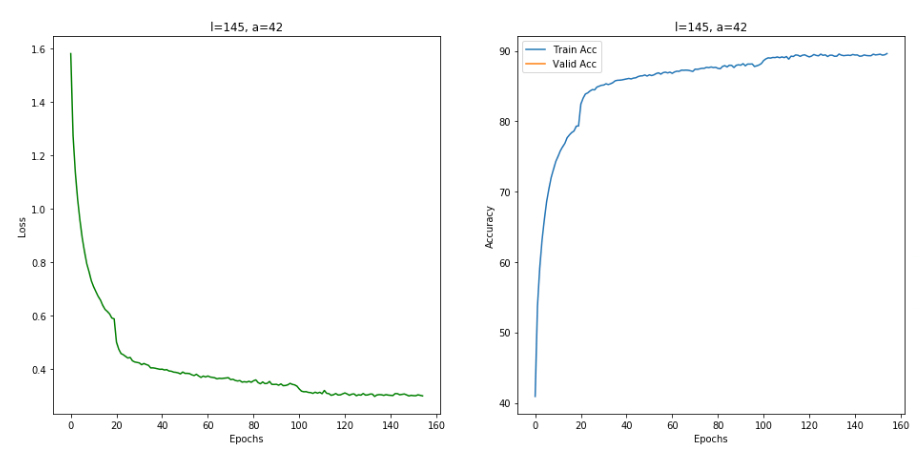
그림 : layer와 alpha값에 따른 모델의 학습기록

다음으로 모델의 layer, alpha 수에 적합한 Epoch와 Scheduler를 알아보기 위해 세 가지 실험을 추가적으로 진행했다. 모델의 성능을 극대화하기 위해 실험에서 train dataset과 valid dataset을 모두 활용하였다. 각 모델의 layer 수와 alpha를 모두Epoch와 Scheduler를 다르게 해서 실험을 진행했다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PyramidNet / Acc | Params | Train Accuracy | Epoch | Scheduler |
| (layer=112, =51) | 1.95M | 87.07% | 200 | Multi Step |
| (layer=171, =36) | 1.98M | 81.46% | 200 | Cos |
| (layer=110, =50) | 1.92M | 95.21% | 400 | Cos |



경험적으로 Epoch를 400 이상으로 설정하고 Cos Annealing Scheduler를 사용하는 것이 좋다는 것을 알 수 있었다. 가장 학습이 잘 된 표의 세 번째 모델의 경우 testset에 대해서 88.95 %의 정확도를 보였다. 실험을 통해서 모델의 layer 수와 alpha는 너무 크지도, 작지도 않은 적당한 값으로 결정해야함을 파악할 수 있었다. 마지막으로 layer 수를 132, 145 alpha를 45 ,42로 설정해서 모델을 구성하고 실험을 진행했다. 



Epoch를 400으로 설정하고, Cos Annealing Scheduler를 이용하여 가장 성능이 좋은 모델을 학습시킬 수 있었다. 이 모델은 train dataset에 대해 92%의 정확도를 보였다. test dataset에 대해서는 약 90%의 정확도를 보여서 모델의 성능이 가장 좋게 나왔다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PyramidNet | Params | Train Accuracy | Test Accuracy | Scheduler |
| **(layer=132, =45)** | **1.95M** | **92.01%** | **90.74%** | **Cos** |
| (layer=110, =50) | 1.92M | 95.21% | 88,95% | Cos |
| (layer=145, =42) | 1.92M | 89.61% | 88.37% | Multi Step |

## 데이터 증강 기법에 대한 비교 실험

(1) DenseNet에서 기본적인 이미지 메소드를 이용해 Random Crop, Horizontal Flip, Center Crop, Color Jitter에 대해 실험을 진행했다. Random Crop에 Padding을 적용하는 것은 이미지 Shift와 동일한 효과를 지닌다. 이는 일반적으로 아주 효과적인 변환 방법으로 알려져 있다. 좌우 대칭도 마찬가지로 효과적인 변환 방법으로 알려져 있다. Center Crop의 경우 보통 이미지는 피사체를 중심에 두고 촬영을 하기 때문에 불필요한 배경을 제거하고 피사체에 집중하게 한다면 성능 향상이 분명 있을 것이라고 예상했다. Color Jitter의 경우 이미지의 명도, 채도, 포화를 조정해서 아예 색다른 이미지를 만든다. 이 기법은 데이터를 늘리는 것에 효과적일 것으로 생각했다. 전혀 다른 이미지가 생성되기 때문이다. 원본 데이터 데이터는 그대로 사용하고 이를 복사한 이미지에 이미지 변환을 적용하여 데이터를 2배로 만든다.

Competition에서 항상 train data만 학습에 사용하다가 valid data의 상당 부분을 train으로 사용하니 3~4% 성능이 향상될 정도로 큰 변화가 있었다. 이 점에 착안해서 데이터를 의미 있게 증가시킨다면 분명 모델 성능 향상에 도움이 될 것이라고 생각했다. 또한 데이터 자체를 증가시키지 않고 일부분에 대해서 이미지 변환을 시도하면 데이터가 더욱 더 일반화가 돼서 성능이 향상될 것으로 예측했다.

데이터 개수를 늘리는 것에 대한 실험이다. 총 케이스는 3가지가 있다. 먼저 모두 validation을 위해 1만 개의 데이터는 valid 폴더에 있는 data에서 random split을 통해서 선정한다. 즉 train data 9만 개와 valid data 8만 개의 원본을 사용한다. 여기에는 확률적으로 RandomCrop과 RandomHorizontal Flip을 적용한다. 그 후 train 원본 데이터에서 절반인 4만5천 개와 원본 valid data의 절반인 4만 개, 총 23.5만 개의 데이터를 train에 사용한다. 적용한 transformation은 Case는 다음과 같다.

Case 0 =  
{원본 데이터-> CenterCrop(24), Pad(4), RandomHorizontalFlip()  
복사 데이터 ->  
ColorJitter((0,3)), CenterCrop(24), Pad(4), RandomHorizontalFlip()  
}  
  
ColorJitter 안의 파라미터의 경우 명도, 채도, 포화도의 값을 랜덤하게 (0,3)을 적용해주는 것이다. 낮은 확률로 원본과 거의 차이가 없는 이미지가 생성될 것을 우려해서 최솟값을 0이 아닌 것으로 하는 Case 1을 생성해보았다.

Case 1=

{원본 데이터-> RandomCrop(32, padding=4), RandomHorizontalFlip()

복사 데이터 ->

ColorJitter((0.2,3)) ,CenterCrop(24), Pad(4), RandomHorizontalFlip()

}

Case 2는 복사 데이터에도 RandomCrop을 적용한다.

Case 2 = {

원본 데이터-> RandomCrop(32, padding=4), RandomHorizontalFlip()

복사 데이터 ->

ColorJitter((0.2,3)), RandomCrop(32, padding=4), RandomHorizontalFlip()

}

나머지 조건은 모두 동일하다.

Block: [6,12,24,16]

Epoch: 300

Optim: Adam

Batch size: 256

Scheduler: MultistepLR[150,220,250]

Dropout: 0.2

Growth rate: 17

}

나머지 조건은 모두 동일하다.

Block: [6,12,24,16]

Epoch: 300

Optim: Adam

Batch size: 256

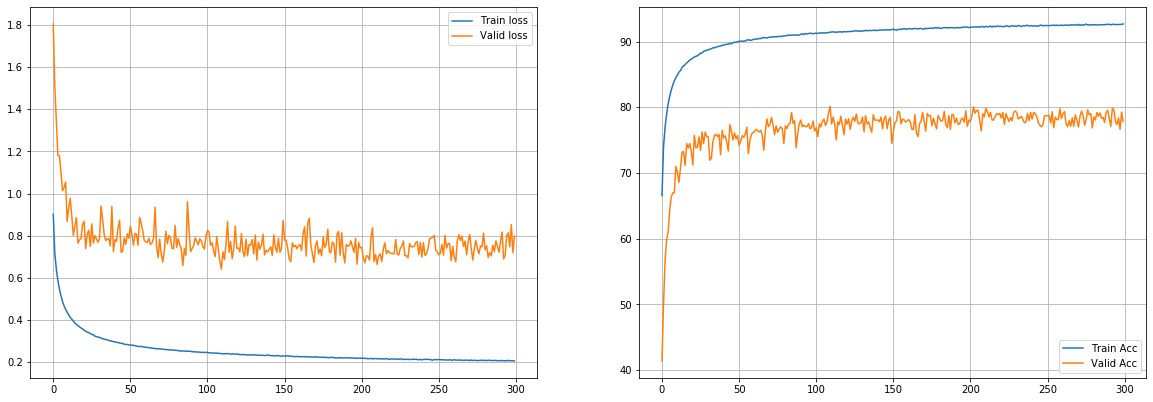
Scheduler: MultistepLR[150,220,250]

Dropout: 0.2

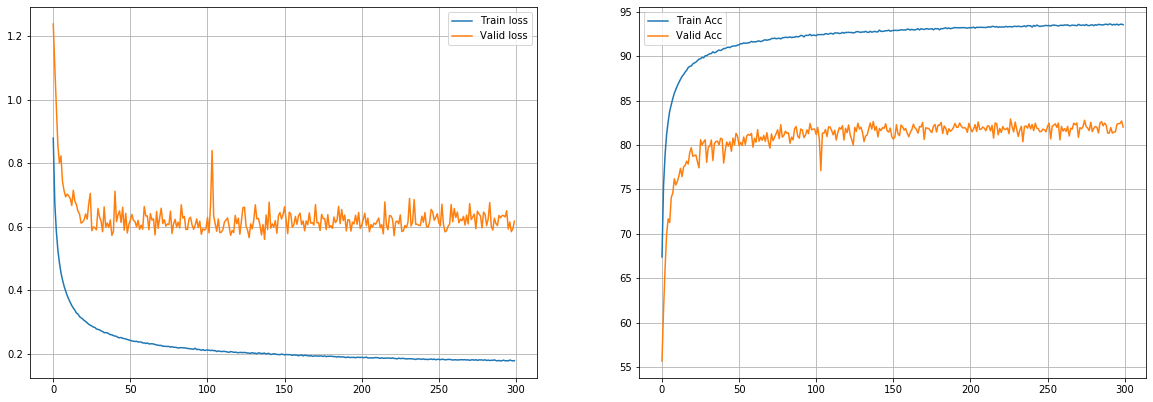
Growthrate: 17

|  |  |  |
| --- | --- | --- |
| Densenet /Case | Train Accuracy | Valid Accuracy |
| Case0 | 92.67% | 80.14% |
| Case1 | 93.21% | 82.92% |
| Case2 | 93.67% | 82.92% |

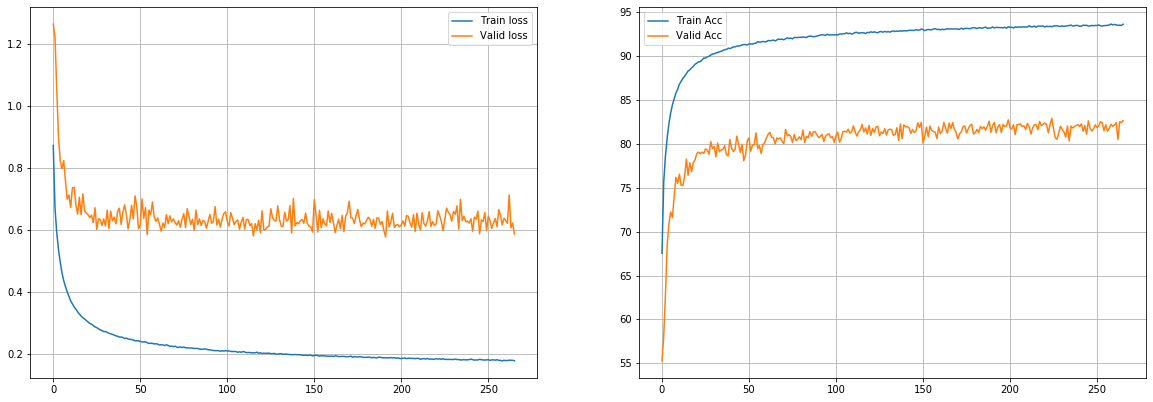
기대와 달리 매우 실망스러운 결과가 나타났다. 데이터의 과적합을 방지하기 위해서 데이터 증가가 굉장히 효과적일 것으로 예측했으나 기대와는 전혀 달랐다. 아무래도 데이터가 원본과는 많이 다른 상태이면서도 인식을 제대로 할 수 있도록 데이터를 증가시켜야 하는데 실험에서 적용한 기법들은 이를 만족하지 못하는 듯하다. 또한 피사체가 대부분 중심에 있는 것에 착안해 적용해본 Center Crop 효과가 크지 않았다. 그러나 ColorJitter의 하한을 0으로 두는 것보다는 조금이라도 작은 값을 할당하는 것이 더 낫다는 것은 실험을 통해서 알 수 있었다.



[그림]Case0



[그림]Case1

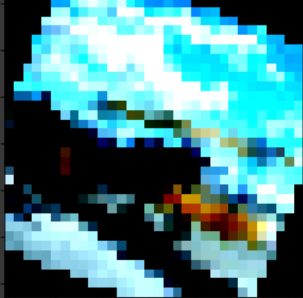
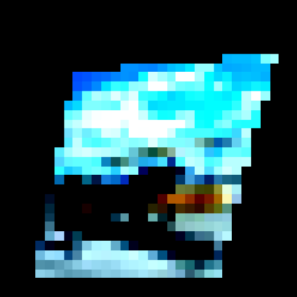


[그림]Case2

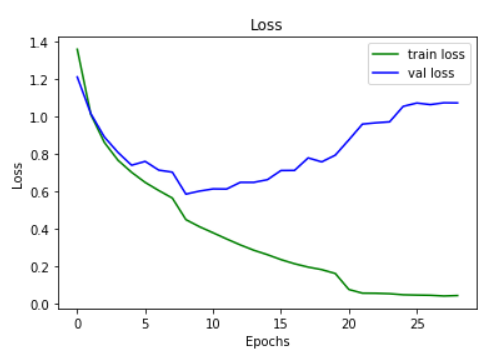
(2) Shake PyramidNet이 DenseNet보다 모델 성능이 더 좋게 나와서 추가적으로 base transform과 Rotate와 Random Perspective세 가지에 대해 PyramidNet으로만 실험을 진행했다. 여기서 base transform은 Random Crop, Horizontal Flip을 말한다.

|  |  |  |
| --- | --- | --- |
| Transform / Acc | Train Accuracy | Valid Accuracy |
| Base | 93.58% | 78.73% |
| Rotate | 77.36% | 74.11% |
| Random Perspective | 80.94% | 72.76% |

표. n. 각각의 변환에 대한 Train Accuracy와 Valid Accuracy

앞서 DenseNet에서 실험했던 것처럼, 데이터를 증강했을 때, 학습 초기에 성능이 좋지 않다면 장기적으로 학습을 진행해도 학습이 잘되지 않는다는 사실을 경험적으로 파악할 수 있었다. 따라서 30 Epochs 이내로 학습을 진행했을 때, 데이터의 변형이 학습에 어떻게 영향을 미치는지 분석했다. 실험 결과, 데이터의 변형은 오히려 모델의 학습에 방해가 되는 요인이었다.    
그림n : 왼쪽부터 비행기를 Base, Rotate, Random로 순서대로 변환한 모습

일반적으로 데이터의 증강은 모델의 일반화 성능을 높인다고 알려졌지만 실험에서는 그렇지 않았다. 그 원인을 분석해보면 학습 데이터의 이미지 사이즈가 32x32로 너무 작아서 transform이 매우 큰 영향을 주기 때문으로 추측된다. 각각의 transform을 모아서 데이터를 3배 늘린 후, 모델을 학습시킨 결과 오히려 오버피팅이 심해지는 결과가 나타났다.



## 최신 모델간 비교 실험

이번 연구에서는 DenseNet 과PyramidNet에 대해 중점적으로 실험했다. 각 모델의 최고 성능을 비교해본다. 단 모델이 다른 만큼 모든 조건을 동일하게 할 수 없다는 것을 기억해야 한다.

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Params | Train Accuracy | Valid Accuracy |
| Densenet | 1.98M | 97.5% | 88.16% |
| Pyramidnet | 1.92M | 92.01% | 90.74% |

PyramidNet은 층이 깊어질수록 Convolution filter의 개수가 많아진다. 즉 더 많은 특징을 추출하고 표현하게 된다. 이러한 구조에 의해 PyramidNet은 좋은 일반화 성능을 가진다. 실제로 값을 보면 DenseNet의 경우 Train과 Valid Accuracy의 차이가 상당히 두드러진다. 그러나 PyramidNet은 그 차이가 DenseNet보다 훨씬 작다. 적어도 이번 Competition에서 제공된 환경에서는 PyramidNet의 성능이 DenseNet보다 좋다고 할 수 있다.

# 결론

이번 competetion을 통해 기본 CNN부터 ResNet, DenseNet, PyramidNet 네 가지 모델에 대해서 파라미터를 조정해봤다. 특히 DenseNet과 PyramidNet에 집중적으로 다양한 실험을 진행했다.

DenseNet의 경우 기본적인 Baseline을 만들기 위한 실험을 많이 진행했다. 그 결과 초기 학습률은 0.001과 같이 작은 값으로 설정하는 것이 좋고 Scheduler를 활용해서 학습률을 학습 도중에 변경하는 것이 중요함을 알았다. 또한 Softmax를 적용하는 것이 기대와 달리 우수한 성능을 보이지 않았다. 또한 무작정 층을 깊게 하는 것은 효과가 없었다. 오히려 학습 시간만 길어지는 효과가 있었다. 층을 깊게 하려면 여러가지 다른 Hyper Parameter를 고려한 층의 깊이를 정해야 하는 것으로 예상된다. LeakyReLU를 사용하는 것이 ReLU를 사용하는 것보다 조금 더 나은 성능을 보여준다.

PyramidNet의 경우Squeeze and Excitation, Shake-Shake Regularization, Sharpness aware minimization 등의 기술을 활용하여 실험을 진행했다. 그 결과, Shake-Shake Regularization을 적용한 PyramidNet이 가장 성능이 좋았다.

공통적으로 Data augmentation을 적용해본 결과 성능은 오히려 더 나빠졌다. 여러가지 이유가 있겠지만 데이터의 증가 기법이 학습에 도움이 되는 유의미한 데이터 증가가 아닌 단순히 이미지 개수만 많아진 것으로 추정된다는 점과 이미지 사이즈가 너무 작기 때문에 데이터 변형이 효과적으로 일어나지 않았다는 점이 있다.

이러한 모델과 기술들은 앞으로 NPU같은 하드웨어의 발전과 함께 자율주행 로봇, 자동차에 사용되는 영상 데이터를 처리하는데 크게 활용될 것이다. 또는 강화학습 과 결합하여 게임 인공지능을 개발하는데 사용될 수 있다. 마지막으로 자연어처리 분야와 결합하여 Multi Modal Learning에 적용되어 음성과 영상 데이터를 함께 처리하는 모델을 설계하는 데 활용할 수 있을 것이다.

매일 인공지능 분야에서는 새로운 기술이 쏟아져 나온다. 혁신은 기존의 것들을 새롭게 연결시킴으로써 탄생한다. 때문에 앞으로는 한 가지 task만을 잘 해결하는 사람이 아닌 여러 task를 해결해보고 다양한 머신러닝 기술들을 통합하고 활용하는 사람이 미래의 인공지능 시장의 선두주자가 될 것이다.

# 참고문헌

참고문헌은 정확한 format 을 지켜서 기술하시오.

*Basic format for conference proceedings(published):*

1. Dongyoon Han , Jiwhan Kim, Junmo Kim “Deep Pyramidal Residual Networks”, Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on, pp.1-6
2. Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, Quoc V.Le “AutoAugment: Learning Augmentation Strategies from Data”, Google Brain,pp.113-117
3. Xavier Gastaldi ,“Shake-Shake regularization”, Workshop track - ICLR 2017, pp.1-6
4. Gao Huang, Zhuang Liu, Laurens van der Maaten , “Densely Connected Convolutional Networks”, Kilian Q. Weinberger Cornell University,pp.1-4
5. Pierre Foret, Ariel Kleiner, Hossein Mobahi, Behnam Neyshabur “Sharpness-Aware Minimization for Efficiently Improving Generalization”, Google Research, Mountain View, CA, USA,pp. 1-6
6. Sebastian Ruder, “An overview of gradient descent optimization algorithms”, Insight Centre for Data Analytics, NUI Galway Aylien Ltd., Dublin, pp. 7-8

탐원들의 Biography도 반드시 아래의 format에 맞게 넣고 본인이 지금껏 이룬 업적, 학문적 관심사, 미래의 포부 등을 포함시키시오.

**진주성.** 경희대학교 산업경영공학과, 컴퓨터 공학을 복수전공 하고 있다. 현재 산업경영공학과 금융공학 연구실에서 포트폴리오 최적화에 적용하기 위해 패널 데이터를 활용하여 유의미한 군집을 구성할 수 있는지를 연구하고 있다. 주된 관심 분야는 금융, 포트폴리오 최적화이다. CFA(Charted Financial Analysis) Level1 Passed, 2019이며 2021년에 Level2 시험 응시를 계획하고 있다. 월급 만으로 노후까지 설계하기 어렵다는 점과 한국 시장이 불안정하다는 점에서 최소한의 리스크를 가지고 안정적인 수익을 얻는 방법을 연구하는 퀀트가 되기를 희망한다. 강화학습을 기반으로 사람이 생각하는 방식으로 투자하는 알고리즘을 개발하고 싶다.

김국진. 경희대학교에서 산업경영공학과와 소프트웨어융합학과를 복수전공하고 있다.

현재, 산업경영공학과 Smart Factory LAB에서 학부연구생으로 연구에 참여하고 있다, 주된 관심분야는 머신러닝이다. 딥러닝의 컴퓨터비전, 자연어처리 task에 관심이 있고 최근에는 강화학습을 집중적으로 연구하고 있다. 현재, DQN을 활용한 마인크래프트 에이전트 개발(2020.09~), PPO, Behavior cloning을 활용한 택배상자 적재 에이전트 개발 프로젝트에 참여하고 있다. 2021년에는 역강화학습, 그래프 뉴럴 네트워크에 대한 연구를 진행할 예정이다. 먼 미래에는 GNN, 컴퓨터비전, 자연어처리, 강화학습 분야의 여러 이론들을 활용하여 사람과 상호작용 가능한 인공지능을 개발하는 것을 꿈꾸고 있다.

**김장언.** 경희대학교 산업경영공학과, 소프트웨 어융합학과를 복수전공하고 있다. 막연한 금융업 계 Career를 희망해왔으나 직감이 아닌 프로그래밍을 이용해 데이터에 기반한 투자 의사결정을 하는 시스템 트레이더에 매력을 느껴 뒤늦은 프로그래밍 공부를 시작했다. 투자자산운용사 자격을 보유했고 2021.2 CFA Level 1 시험을 앞두고 있는 등 금융 도메인 공부와 함께 이를 구현하는 능력을 위해 파이썬을 활용한 알고리즘 및 데이터분석 공부를 병행하고 있다. 한국은 금융업의 선진 국가인 미국, 싱가포르, 홍콩 등에 큰 영향을 받으며 금융 선진화를 위해 노력하고 있지만 제도적 한계로 인해 그 효과가 미미하다. 이 선진화의 핵심이 바로 AI 및 빅데이터에 있다 믿으며, 이 변화를 이끌어가는 사람이 되고 싶다.