최적의 모델을 찾아서: PyramidNet

김민경, 김성수, 이찬

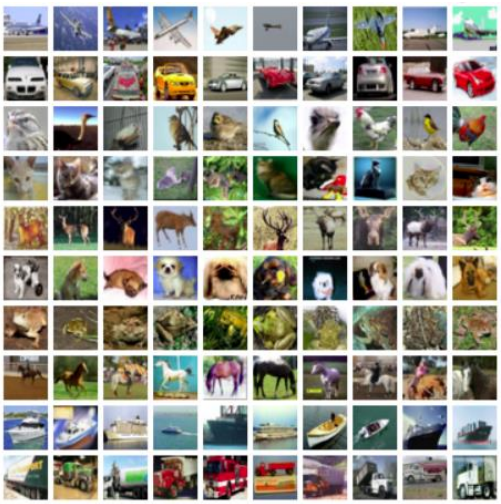
초록: layer 가 196, alpha 가 48 인 PyramidNet 에 CosineAnnealing Scheduler, SGD optimizer, LeakyReLU Activation Function 을 기반으로 진행했고 Smoothlabeling 과 Shakedrop 등을 활용하여 분석을 진행하였다. 특히 Shakedrop 을 활용하여 Validation Data 를 Train Data 와 합쳐 학습함으로써 Data Augmentation 효과를 누리는 동시에 Shakedrop 이 Validation Data 역할을 해주어 Test data 에서 91.418% 정확도를 보여주었다.

키워드: PyramidNet, CosineAnnealingScheduler, SAM, Shakedrop, LeakyReLU, BatchNormalization

Ⅰ. 소개

최근 연구동향을 살펴보면 다양한 방식으로 레이어를 구성하거나 이미지를 변형하기도 하고 기존의 방식을 활용하고 변형하여 좋은 성능을 내기도 한다. 이번 Competition의 목적은 더 좋은 성능을 내기 위해 기존 연구되었던 기술들을 사용해보고 다양한 방식으로 변형해보는 것이 이라고 볼 수 있다.

이번 Competition에서 사용되는 Dataset은 Cifar-10과 같이 10개의 라벨로 구성되어 있고 학습에 사용되는 이미지의 양이 50000장, 테스트에 사용되는 이미지의 양이 10000장인 기존 Cifar-10과 다르게 학습, 검증, 테스트에 각각 90000개의 이미지를 사용할 수 있어 좀 더 많은 양의 데이터를 사용할 수 있다는 점이 특징이다.



[그림 1: CIFAR-10]

이번 Competition을 통해 어떻게 이미지 분류의 정확도를 향상시켜야 할지 연구하고자 한다. 성능을 향상시키기 위해 다양한 모델들을 사용해 직접 학습시켜보고, ReLU 등의 기본적인 활성 함수와 최신기술인 Hardswish, Cosine Annealing 등의 스케줄러, SAM 등의 최적화 방식, 가우시안 노이즈 등의 데이터 변형 등을

사용하고 성능을 비교하면서 최적의 모델을 찾도록 한다.

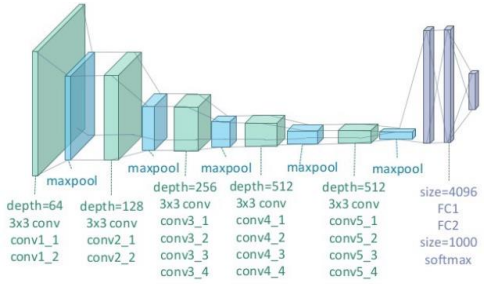
Ⅱ. 관련연구

실험을 진행하기 전에 CIFAR-10으로 정확도를 비교한 기존의 논문들을 살펴보았다. 중점적으로 살펴본 모델은 ResNet, VGGNet, DenseNet, PyramidNet이다. 추가로 모델의 성능을 높이기 위한 규제 방법(Shake Drop), 활성함수, 학습률 조정(scheduler), 최적화 함수(optimizer), 정규화에 대한 연구 논문들을 살펴보았다. 그 외에 Data Augmentation, Label Smoothing, Gaussian Noise와 관련된 연구 내용도 살펴보았다.

A. 딥러닝 모델 (논문의 모델 비교 내용)

A-1) VGGNet

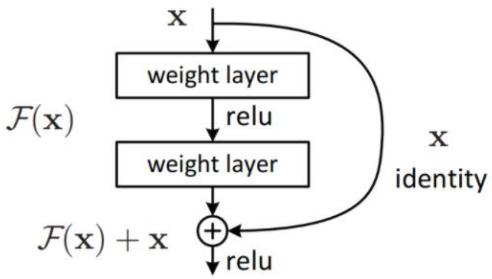
VGGNet은 옥스포드 대학의 연구팀 VGG에 의해 개발된 모델로 사용하기 쉬운 구조와 좋은 성능을 가진 것이 특징이다. 네트워크의 깊이를 깊게 만들어 성능을 최대로 한 것이므로 16개 또는 19개의 레이어로 이루어져 있다. 각 컨볼루션 필터 커널의 사이즈는 가장 작은 3x3을 사용하여 네트워크의 깊이를 깊게 만들었다. (Karen Sinmonyan, 2015)



[그림 2: VGGNet]

A-2) ResNet

ResNet의 핵심 아이디어는 아래 그림과 같은 Residual Block이다. 이는 그래디언트가 잘 흐를 수 있도록 일종의 지름길을 만들어 주자는 생각이다. 이는 Vanishing Gradient 문제를 해결해 주기 때문에 ResNet은 일반적으로 모델의 층이 아주 깊다. ResNet이 좋은 또다른 이유는 residual block이 앙상블 모델을 구축한 것과 비슷한 효과를 내기 때문인데, 이는 Residual Block의 Skip Connection 덕분에 입력 데이터와 그래디언트가 오갈 수 있는 통로가 크게 늘어나기 때문이다. (Kaiming He, 2016)

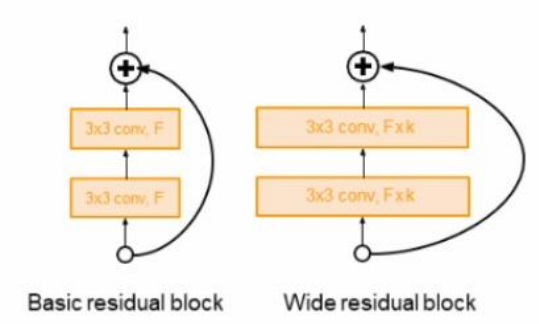


[그림 3: ResNet]

A-3) Wide_ResNet

Wide ResNet은 말 그대로 기존의 ResNet에서 층을 더 넓히는 방법이다. 즉, 기존의 ResNet

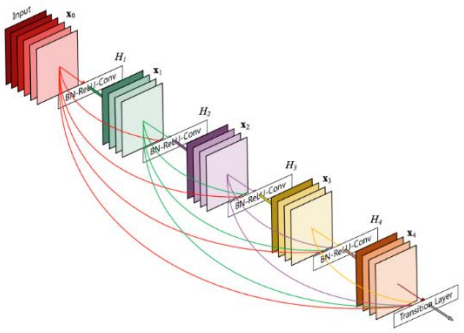
은 깊게 쌓는 것에 집중했다면 Wide ResNet은 Residual에 집중한다. 아래 그림처럼 기존의 ResNet에는 F개의 필터만 있었다면 Wide ResNet은 F*K개의 필터로 구성한다. 50층의 Wide ResNet이 152층의 ResNet보다 성능이 더 좋게 나왔다고 한다. (Sergey Zagoruyko, 2016))



[그림 4: Wide ResNet]

A-4) DenseNet

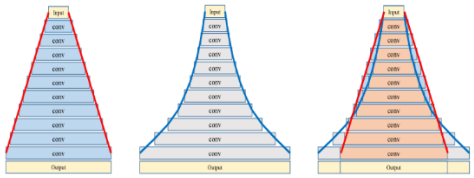
DenseNet은 이전의 많은 Layer의 출력을 한꺼번에 받는 방식이다. 입력이 들어오면 Convolution을 하나 거친 이후에 DenseBlock을 통과한다. Transition Layer(BN + Conv 1x1 + Average Pooling)를 두어 Average Pooling을 사용해 Down Sampling을 한다. (Gao Huang, 2017)DenseBlock 내부에서 이전 Layer의 입력을 모두 받아 Concatenation하기 때문에 Feature Map의 Channel 수가 점점 늘어난다는 특징이 있다. Layer의 입력으로는 이전 입력이 모두 Concatenated되서 들어오기 때문에 Channel 수가 계속 달라지지만 Layer 내부에서는 K의 Channel로 유지된다.



[그림 5: DenseNet]

A-5) PyramidNet

모든 Layer에서 Channel 수가 변하도록 해서 특정 Layer에 집중되어 있던 Width의 변화를 전체 네트워크로 분산시킨다. PyramidNet에서 Layer의 Width를 늘리는 방법에는 두 가지가 있다. 먼저, Additive 방식은 전체 Layer 동안 얼마나 Width를 늘릴지에 대한 알파 값을 정한 후 이전 Width에 비해서 Alpha / N만큼 늘리는 것이다. 두번째는 Multiplicative 방식으로 지수배로 늘리는 방식이다. 성능은 Additive 방식이 좋은 편인데 초기 Layer 들의 Width가 Multiplicative 방식보다 더 큰 경향이 있기 때문이다. 다음 그림에서 가운데가 Multiplicative 방식이며 오른쪽이 Additive 방식이다. (Dongyoon Han, 2017)



[그림 6: PyramidNet]

B. 데이터 증강 기법(Data Augmentation)

데이터 증강 기법은 기존의 Train DataSet을 약간 변형하여 학습 데이터 수를 늘리는 방법이다. 대표적으로 Scaling, Flip, Rotate이 있다. 이 외에도 이미지 밝기 조절, 노이즈 넣기 등 다양한 방법이 있는데 앞의 세 가지 방법만 소개하고 Gaussian Noise에 대한 설명은 '제안 방법'에서 자세히 다루도록 한다. Scaling은 이미지를 확대하는 방법이고, Flip은 이미지를 반전시키는 방법으로 Vertical Flip과 Horizon Flip이 있다. Rotate는 이미지를 회전시키는 방법이다. 일반적으로 Data Augmentation은 한 가지 기법만 쓰기 보다 여러 가지 기법을 혼합해서 쓴다.(Ryo Takahashi, 2020)

이렇게 동일한 이미지를 약간만 변형시키는 이유는 컴퓨터가 다른 사진으로 인식하기 때문이다. 컴퓨터 입장에서는 같은 이미지라도 약간만 회전이 되거나 확대되어도 픽셀 값이 달라지기 때문에 다른 사진으로 분류한다. 따라서 기존의 Train Data를 데이터 증강 기법으로 몇 배로 부풀릴 수가 있다. 우리는 Scaling, Rotate를 적용하여 Train Set을 두 배로 부풀려서 학습시킨 결과 성능이 0.01% 정도 향상됨을 확인했다. 하지만 학습 시간이 n배로 길어져서 데이터 증강 기법은 사용하지 않기로 했다. 아래 [그림 7]은 데이터 증강 기법의 예시다.

[그림 7: Data Augmentation]

C. 규제 방법 (Shakedrop, Early stopping)

C-1) ShakeDrop

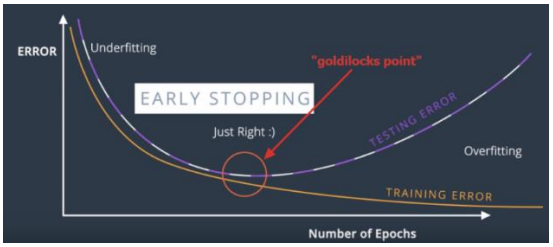
ShakeDrop은 학습을 방해하여 오류율을 줄이는 Shake-Shake 정규화에서 영감을 받았으며 ResNeXt에만 적용할 수 있는 Shake-Shake와 달리 ShakeDrop은 ResNet, Wide ResNet, PyramidNet에도 적용할 수 있다.

학습하는 과정에서 컨볼루션 레이어의 출력에 음의 요소를 곱하여 학습을 강력하게 방해한다. 이러한 정규화 과정을 통해 과적합을 개선하는 효과를 얻을 수 있다.

C-2) Early Stopping

학습횟수가 많을수록 학습 데이터에 대한 오차는 작아지나 이는 오버피팅을 초래하여 모델의 일반화 성능을 떨어지게 한다. 우리는 이를 해결하기 위하여 Early stopping기법을 활용한다. 이는 일정 기간의 Epoch때와 비교해서 오차가 줄어들지 않았다면 학습을 중단시키는 기법이다. 이를 통해 규제(Regularization)효과를 얻을 수 있으며 모델의 일반화 성능을 개선할 수 있다.

본 Competition에는 Early Stopping을 별도로 설정하지 않았다. 왜냐하면 Shakedrop 기법을 활용하여 오버피팅을 방지할 수 있다는 것이 기존 논문을 통해 밝혀졌는데 직접 실험해본 결과 오버피팅 개선에 효과를 보여 두번의 규제를 넣는 것은 불필요하다고 생각했기 때문이다.



[그림 8: Early Stopping]

D. 스케쥴러

D-1) Exponential LR

매번마다 현재 Step의 비율을 지수(Exponential)로 취해주는 Learning Scheduler 이다. Stage_Length를 정해주고 현재 Step이 Stage_Length 이하일 때까지는 1이하의 값을 지수로 취해주고, 현재 Step이 Stage_Length를 초과할 경우는 1 이상의 값을 지수로 취한다. floor 함수를 취해서 Step의 비율 값을 정수로 설정할 수도 있다. (Zhiyuan Li, 2019)

D-2) Loss Plateau Decay

Warmup 이전까지는 선형적으로 증가시키다가 오차가 더 이상 줄어들지 않고 일정 값에 정체(Plateau)가 생기면 학습률을 줄인다. 오차가 줄어들지 않기 시작한 후 몇 번의 Epoch부터 학습률을 줄일지 결정하는 Patience값을 조절할 수 있다.

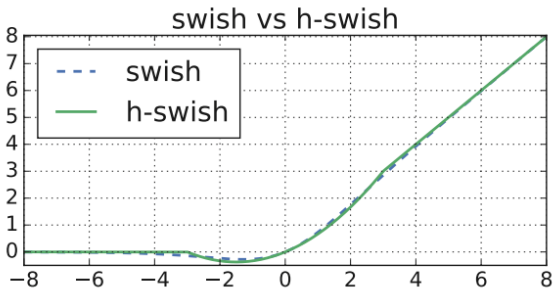
D-3) Cosine Annealing Warm Restarts

이는 코사인 형태로 학습률을 차츰 조정하면서 정확도를 향상시킬 수 있는 방법이다. 이는 오차와 정답의 차이를 100% 반영해서 가중치를 수정하는 것이 아닌 학습률이 지정되어 있는 정도로 훈련한다. 이는 각각 Restart지점마다 다른 모델이 튀어나올수도 있는데 이때 앙상블 기법을 활용하여 정확도를 더 높일 수 있다.(Akhilesh Gotmare, 2018)

E. 활성함수

E-1) Hard swish

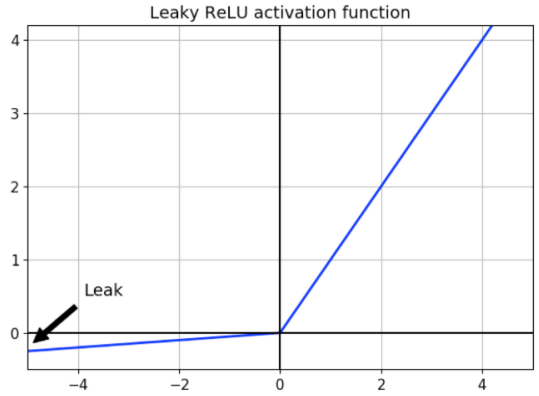
Hard Swish란 Swish함수를 기반으로 한 활성함수이다. 계산 비용이 많이 드는 시그모이드를 부분 선형 아날로그로 대체한다. (Prajit Ramachandran, 2017)



[그림 9: hardswish]

E-2) LeakyReLU

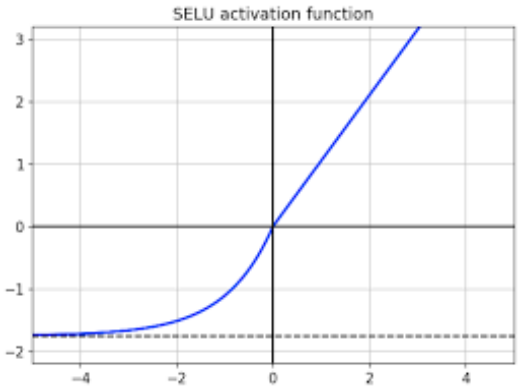
양수일 때는 x값을, 음수일 때는 a * x 값을 반환한다. ReLU와 거의 비슷한 형태를 가지며 입력 값이 음수일 때 완만한 선형 함수를 그려주는 것이 특징이다. 일반적으로 알파를 0.01로 설정한다. (Chigozie Enyinna Nwankpa, 2018))



[그림 10: LeakyReLU]

E-3) SELU

ReLU와 거의 비슷한 형태를 갖는다. 양수일 경우 x값을 반환하고 음수일 경우 지수 함수를 이용하여 a(e^x-1)의 값을 반환한다. 미분 함수가 끊어지지 않고 이어져있는 형태이다. (Chigozie Enyinna Nwankpa, 2018)



[그림 11: SELU]

F. Optimizer

F-1) SAM

Sharpness-Aware Minimization의 약자이며 손실 값과 손실 선명도를 동시에 최소화한다. 특히, 균일하게 손실이 적은 이웃에 있는 매개변수를 찾는다. SAM은 모델 일반화를 개선하고 여러 데이터 세트에 대한 SoTA 성능을 제공한다. 또한, 노이즈 레이블을 사용하는 학습을 특별히 목표로 하는 SoTA 절차에서 제공하는 것과 동등한 레이블 노이즈에 대한 견고성을 제공한다. Gradient를 구하기 위해 Forward-Backward과정을 2회반복하므로 다소 학습이 느리다는 단점이 존재한다. (Foret. P., A. Kleiner, 2020)

F-2) Adam

Momentum 과 AdaGrad를 융합한 방법으로 학습률을 줄여나가고 속도를 계산하여 학습의 갱신강도를 적응적으로 조정해나가는 방법이다. 학습률을 L2 norm을 기반으로 조절한다. Momentum term에 RMSProp에서 등장했던 지수이동평균을 적용하고 RMSProp과 같은 방식으로 변수들이 Update하는 방향과 크기는 Adaptive 벡터에 의해서 결정된다. (Diederik, 2015)

F-3) Adadelta

Adagrad가 학습률을 자동으로 조정할 수 있게 되었으나 학습이 진행될수록 기울기의 값이 급격하게 작아져 학습이 원활하게 진행되지 않는 문제를 극복한 것이다. Adagrad의 대각행렬은 경사 제곱의 누적합이기에 단조증가하기 때문이다. Adadelta는 0번째 단계부터 제곱합을 누적해가는 것이 아닌 누적해가는 단계의 수를 정수w로 제한하여 직전 단계까지의 모든 경사의 제곱합을 감쇠평균시켜 재귀식으로 계산한

다. 이를 통해 단계가 거듭될수록 기울기의 합이 지수함수적으로 작아지게 되어 학습을 개선하였다. (Matthew D, 2012)

G. GaussianNoise

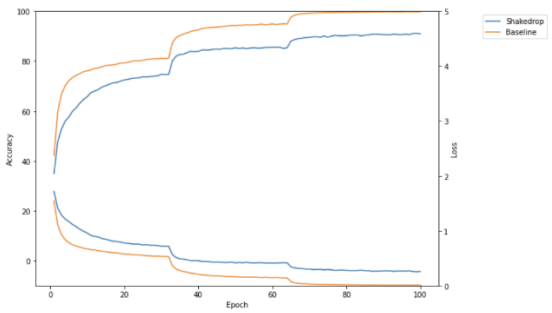
모델의 학습 및 성능 개선에 있어 우리는 Data를 추가적으로 수집한다. 하지만 데이터의 추가수집은 시간과 비용에 있어 적지 않은 소모가 있다. GaussianNoise는 별도의 비용없이 data augmentation의 효과를 누릴 수 있는 방법 중 하나다. GaussianNoise란 정규분포를 가지는 잡음으로 어느정도 랜덤하면서 자연계에서 쉽게 볼 수 있는 분포를 말한다. 오버피팅은 일반적으로 신경망이 유용하지 않을 수 있는 고주파 특징을 학습할 때 발생한다. Gaussian Noise는 기본적으로 모든 주파수에 데이터 포인트를 포함하여 고주파 기능을 효과적으로 왜곡시킨다. 이는 낮은 주파수의 특징 또한 왜곡시키는데 적절한 노이즈는 이를 무시하고 학습능력을 개선하여 Regularization 효과를 얻을 수 있다. Gaussian Noise는 Input뿐만 아니라 가중치, 기울기 및 활성함수에도 추가되어 활용될 수 있으며 네트워크의 견고함을 향상시키고 더 나은 일반화 성능을 개선할 뿐만 아니라 학습속도를 빠르게 할 수 있다. (Boyat, 2015))

[그림 12: Gaussian Noise]

Ⅲ. 제안방법

A. ShakeDrop으로 과적합 방지

학습을 함에 있어 Train Dataset의 수가 많을수록 학습이 잘 되는 것은 명확하다. 기존 Baseline의 Train Set과 Validation Set은 각각 9만장이다. 우리는 Validation Set을 Train Set에 포함시키면 학습 데이터 수가 2배가 되기 때문에 성능이 높아질 것이라 생각했다. 실제로 아래 실험에서 제시한 것처럼 Validation set을 추가하여 학습했을 때 일반화 성능이 3% 증가하였다. 하지만 Validation set이 없으면 학습이 과적합 되는지 알 방법이 없기 때문에 과적합 방지를 위한 다른 대안을 모색해야 했다. 우리는 ShakeDrop 기법을 적용시켜 과적합 문제를 해결하였다. 기존 Baseline에서 피라미드넷 모델에 Shakedrop 기법을 적용시킨 모델과 적용하지 않은 모델의 결과를 그래프로 정리해 보았다. Shakedrop 기법을 적용시킨 모델의 정확도는 Test 데이터의 정확도와 큰 차이가 없는 반면에 Shakedrop 기법을 적용시키지 않은 모델의 정확도 99%를 넘어간 것을 확인할 수 있다. 이를 근거로 우리는 학습을 할 때 Validation Set을 Train Set에 포함시켰다.



[그림 13: Shakedrop VS Baseline]

B. Label Smoothing

Label Smoothing은 데이터 정규화 기법 중 하나로 Hard Label을 Soft Label로 Smoothing 하는 것을 뜻한다. Hard Label이란 원핫 인코딩 된 벡터를 의미하고, Soft Label이란 라벨이 0과 1 사이의 값으로 구성되어 있는 것을 말한다. (Tong He, 2018) 여기서 핵심은 기존에 0과 1의 경우 Label을 1임을 100%확신하는 개념이었으나, Label Smoothing에서는 반드시 그렇지 않을 수 있다는 것이 핵심이다. Label Smoothing은 0또는 1이 아닌 Smooth하게 부여하여 라벨링이 잘못된 데이터에 Regularization을 실시한다. 이는 모델이 Target을 정확하게 예측하지 않아도 되도록 만들며 기존엔 최대한 0또는 1로 예측하도록 학습되는데, Label Smoothing를 통해 0.9정도로 예측했다면 그에 해당하는 loss값을 산출하게 하는 것이다. 이는 모델의 오버피팅을 방지해주어 모델의 일반화 성능 개선에 도움을 준다. (Muller R, Simon, 2019)

C. Batch Normalization

Batch Normalization은 각 미니배치 별로 사전 정규화 처리를 수행하는 기법이다. 데이터 셋 전체를 정규화하는 것보다 학습에 사용하는 미니배치별로 정규화하기 때문에 학습하는 과정 전체에서 효과가 나타난다. 대부분의 논문에서 거의 다 사용한다고 보면 될 정도로 많이 쓰이고 있다. BN을 해주면 그레디언트의 스케일이나 초기 값에 대한 의존성이 줄어들어 큰 학습률을 설정할 수 있기 때문에 결과적으로 빠른 학습이 가능하게 된다. 즉, 기존 방법에서 학습률을 높게 잡을 경우 그레디언트가 소멸하거나 Local minima에 빠지는 경향이 있었는데 이는 스케일 때문이었으며, 배치 정규화를 사용할 경우 오류역전파를 할 때 파라미터의 스케일에 영향을 받지 않게 되기 때문에 학습률을 높게 설정할 수 있는 것이다. 또한 규제 효과가 있기 때문에 드롭아웃 등의 기법을 사용하지 않아도 된다. (Loffe S, 2015)

D. 제안 방법의 형태

본 Competition에 있어 PyramidNet을 기반 모델링을 진행하였다. 파라미터의 개수를 200만개 이내로 하기 위해 Layer를 196개, Alpha를 48로 지정하여 190만개의 파라미터로 조절하여 진행하였다.

하이퍼 파라미터의 튜닝은 다음과 같이 진행하였다. CosineAnnealing Scheduler를 적용하여 learning rate가 코사인 형태로 변화하도록 설정하였고 batch size는 256으로 진행하였다. 활성함수로는 LeakyReLU를, Optimzer로 SGD를 활용하였으며 epoch는 230으로 진행하였다. 기타 기술로는 Shakedrop 및 Labelsmoothing을 활용하여 성능 개선에 활용하였다. 특히 Shakedrop을 활용하여 기존 Validation Data를 Train Data에 합쳐 2배의 데이터로 Augmentation해도 과적합을 방지하는 효과를 얻을 수 있었고 기존 Validation Data의 역할을 Shakedrop이 수행하면서 오버피팅을 효과적으로 방지할 수 있었다.

Ⅳ. 실험

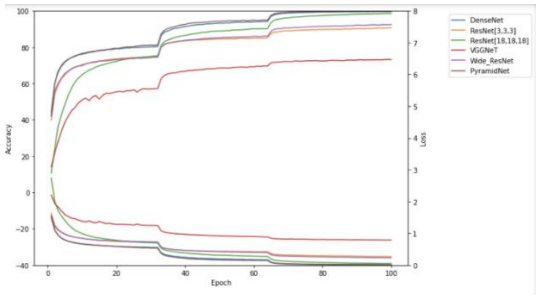
A. 실험 조건

본 실험에 있어 우선적으로 CNN내 최신모델간 비교실험을 실시하였다. 그 중 PyramidNet이 학습에 있어 가장 좋은 성능을 보였고 PyramidNet을 기준모델로 정하여 하이퍼 파라미터를 튜닝하였다. 크게 Layer와 Alpha, Scheduler, Optimizer, Activation Function을 바꿔가며 실험했고 이들 중 가장 좋은 성능을 낸 하이퍼 파라미터를 조합하여 최종 모델을 구성하였다. 하이퍼 파라미터 비교에 있어 Epoch는 100으로, Batch Size는 256 등으로 각 해당사항이외에는 모두 고정시켰다.

이를 파악하기 위하여 CIFAR-10 데이터의 일부를 활용하였다. CIFAR-10은 32x32 픽셀의 10개 Class로 이루어진 데이터이며 비행기 등의 사물 분류문제 해결을 위해 활용되는 데이터로 Train, Validation, Test 데이터 각각 9만장을 활용하여 분석하였다.

GPU 환경으로 Google Colab의 GPU를 활용하였고 PyramidNet의 경우 Baseline으로 삼은 모델에 대해 1Epoch 당 200초의 시간이 소모되었으며 최종 결합한 모델은 400초가량이 소모되었다.

B. 최신 모델간 비교 실험



[그림 14: 모델 별 성능 비교]

실험한 모델은 VGGNeT, DenseNet, PyramidNet, ResNet이다. 특히 ResNet의 경우는 Wide_ResNet과 ResNet[3,3,3] 그리고 ResNet[18,18,18]을 추가 비교하였다. Accuracy와 Loss는 위 [그림 14]와 같다. 상대적으로 가장 최신모델인 PyramidNet과 DenseNet이 Train Accuracy 및 Loss에서 가장 좋은 성능을 보였다. VGGNet의 경우 확연하게 학습이 잘 안된 듯한 안 좋은 성능을 보여주었고 ResNet의 경우 [18,18,18], Wide ResNet, [3,3,3]순으로 좋은 Train Accuracy를 보여주었다. 특히 ResNet[18,18,18]의 경우 다른 두 모델보다는 좋은 퍼포먼스를 보여주었다. 이들에 대한 Test Accuracy 결과는 [표 1]과 같다.

Model	Test accuracy
VGGNet	71.299%
DenseNet	85.839%
ResNet[3,3,3]	81.571%
ResNet[18,18,18]	81.672%
Wide ResNet	79.080%
PyramidNet	87.520%

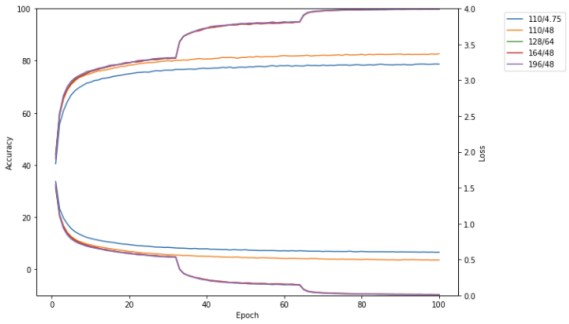
[표 1: 모델 별 일반화 성능]

VGGNeT의 경우 학습이 잘 안된 것이 그대로 반영되어 가장 낮은 Test Accuracy를 보여주었다. 오히려 Train Accuracy에 비해서

Wide_ResNet의 경우는 일반화 성능이 안 좋은 것을 볼 수 있다. 이는 학습데이터에 과적합된 것으로 해석할 수 있었다. 이는 VGGNet과 함께 Test Accuracy가 80%가 되지 않는 모델로 뽑혔다. 이를 토대로 최신 모델들 중 PyramidNet이 DenseNet에 비해 2%가 높고, 가장 낮은 정확도 모델인 VGGNet보다 약 16% 더 좋은 성능을 보였기에 가장 유의미하다고 판단하여 본 연구에 있어 주모델로 활용하였다.

C. 하이퍼 파라미터에 대한 비교 실험

C-1) Layer와 Alpha조절



[그림 15: Layer와 Alpha비교]

PyramidNet의 성능을 결정하는 핵심 파라미터인 Layer와 Alpha(Widen Factor)값을 조절해 주었다. 논문에서 제시한 (Layer, Alpha)값 (110, 48)을 기준으로 (110, 4.75), (128, 64), (164, 48), (196, 48)을 비교하였다. 파라미터를 다음과 같이 설정한 이유는 Layer에 따른 영향 그리고 Alpha에 따른 영향을 우선적으로 확인하고 이들의 교호작용 효과의 유무에 대해 확인하여 최적 하이퍼 파라미터 값을 도출하기 위함이다. 결과는 위 그래프와 같았다. Train Accuracy의 경우 Baseline 값인 (110, 48)을 기준으로 보았을 때 Layer와 Alpha가 커지면 커질수록 성능이 향상되는 것을 볼 수 있었다. 특히, Layer가 128이상, Alpha가 48이상일 때 Train Accuracy는 거의 완벽에 가까운 모습을 보여주었다. 이에 대한 Test Accuracy는 아래 [표 2]와 같다.

(Layer, Alpha)	Test accuracy
(110, 4.75)	71.709%
(110, 48)	76.229%
(128, 64)	87.260%
(164, 48)	87.407%
(196, 48)	87.485%

[표 2: Layer와 Alpha 일반화 성능]

가장 Layer와 Alpha값이 작은 (110, 4.75)에 대해서 성능이 가장 좋지 못했다. 가장 성능이 좋았던 모델과 비교했을 때 16%정도의 차이가 나며 이는 VGGNet과 비슷한 성능을 보였다. Alpha를 증가시켜 (110, 48)일 때 약 5% 개선된 Test Accuracy를 보여주었으며 이를 통해 Alpha를 증가시킨 것이 효과가 있다고 해석할 수 있었다.

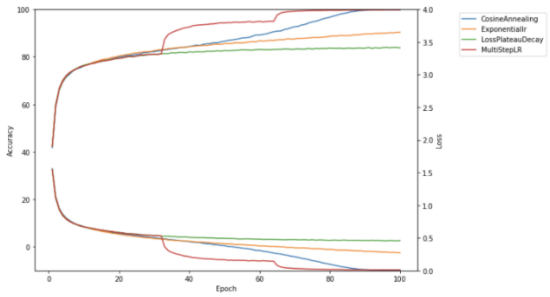
(110, 48)과 (164, 48) 및 (196, 48)과 비교했을 때 약 11% 성능이 증가한 것을 볼 수 있었다. 이는 Alpha값을 증가시켰을 때 5%정도 증가한 것에 비해 16%가 증가하여 Layer가 Alpha에 비해 모델 퍼포먼스 측면에서 더 유의미하게 작용한다고 해석할 수 있다. 특히 (164, 48)과 (196, 48)을 비교했을 때 Layer가 32개 더 많은 (196, 48)이 좋은 성능을 보여 Layer는 많을수록 성능이 좋아진다고 판단하였다.

(128, 64)를 (110, 48)과 비교했을 때 성능은 16%증가한 수치를 볼 수 있었다. 이는 Alpha와 Layer를 동시에 약간씩 증가시켰을 때 효과

를 보기 위하여 진행하였다. 이는 파라미터가 190만개에 가까운 (196, 48)에 버금가는 성능을 보여주었고 Layer 그 자체와 Alpha 그 자체가 의미가 있지만 Layer와 Alpha의 교호작용도 무시할 수 없다고 해석할 수 있었다.

최종적으로 (196, 48)이 가장 유의미한 Test Accuracy를 보여 일반화성능이 가장 좋다고 판단하였다. 이를 기반으로 Final Modeling에 있어 Layer=196, Alpha=48을 모델에 적용하였다.

C-2) Scheduler 조절



[그림 16: Scheduler 비교]

학습률을 일정하게 가져가는 대신 Scheduler를 통해 효과적으로 학습할 수 있도록 다양한 Scheduler를 테스트하였다. Baseline으로 기존 학습률 0.1에서 100epoch에 대해 MultiStepLR을 활용하였고, epoch 32, 64마다 0.1씩 곱하도록 설정하였다. 비교 대안으로 CosineAnnealing, ExponentialLR, LossPlateauDecay를 활용하여 가장 최적의 scheduler를 찾고자 하였다. 결과는 위 [그림 16]과 같다. 오히려 눈에 띄는 것은 MultiStepLR이었다. Epoch가 40이 넘어가면 가장 좋은 성능을 보이기 시작했으며 다른 Scheduler에 비해 바뀌는 구간에서 확연한 성능차이를 보여주었다. 그 뒤로 CosineAnnealing과 ExponentialLR, LossPlateauDecay순으로 유의미한 성능을 보여주었다. 이에 대한 Test Accuracy는 아래 [표 3]과 같다.

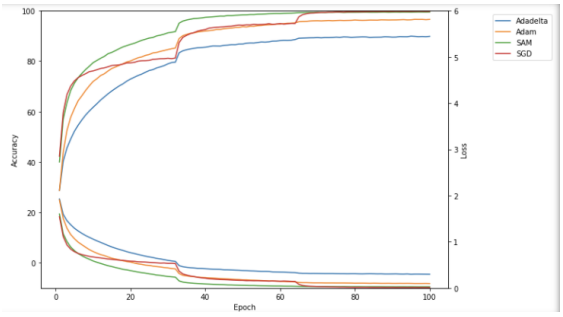
Scheduler	Test accuracy
MultiStepLR(Baseline)	87.485%
CosineAnnealing	88.563%
ExponentialLR	81.633%
LossPlateauDecay	78.183%

[표 3: Scheduler 비교]

Baseline으로 활용한 MultiStepLR의 경우 꽤 좋은 축의 성능에 속하였다. ExponentialLR과 LossPlateauDecay의 경우 Train Accuracy가 2가지에 비해 뒤쳐졌던 것이 그대로 반영되었다. 하지만 CosineAnnealing의 경우 MultiStepLR에 비해 학습되는 속도는 느렸지만 오히려 가장 좋은 성능을 보여주었다. MultiStepLR보다 일반화 성능이 1%정도 높은 것을 볼 수 있었고 이는 Cosine그래프처럼 학습률이 올라갔다 내려갔다 주기적으로 반복하는 것이 효율적으로 활용됐다고 해석할 수 있다.

위 결과에서 나타난 것처럼 CosineAnnealing이 가장 좋은 퍼포먼스를 보여주어 최종 결합한 모델에서는 Scheduler로 CosineAnnealing을 활용하였다.

C-3) Optimizer 조절



[그림 17: Optimzer 비교]

기존 Baseline인 SGD Optimizer 이외에 다양한 Optimizer를 통해 성능을 향상시키고자 하였다. 위 [그림 17]에서 보는 것처럼 학습을 가장 잘한 Optimizer는 SGD와 SAM이다. 특히 SGD의 경우는 4가지 중 가장 높은 Train Accuracy를 보이며 가장 뛰어난 학습능력을 기대하게 했다. 하지만 Adam과 Adadelta는 기존에 알려진 것과 다르게 Train Accuracy가 크게 높지 않았다. "무엇을 사용할지 모를 때는 Adam을 사용하라"라는 속설과 다르게 엄청난 성능을 보여주진 못했다.

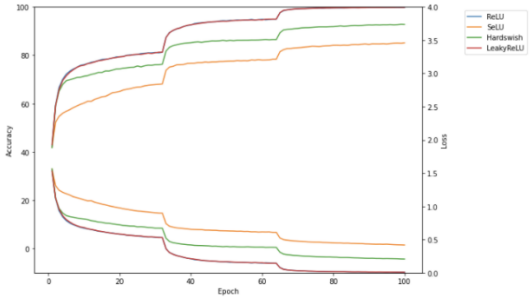
4가지 모두 동일한 실험조건을 위해 32, 64에 MultiStepLR을 설정하여 각각 32와 64에서 수직적인 발전양상을 띄고 있으며 이를 통한 Test Accuracy는 아래 [표 4]와 같다.

Optimizer	Test accuracy
SGD(Baseline)	87.485%
Adadelta	75.136%
Adam	82.912%
SAM	87.377%

[표 4: Optimzer 일반화 성능]

위 [표 4]에서 본 것처럼 SAM과 SGD가 거의 유사한 높은 성능을 보여주었다. 오히려 높은 Train Accuracy로 과적합에 의해 학습이 잘 안 되었을 수도 있을 것이라는 생각을 했지만 가장 좋은 성능을 보여주었다. 이는 가장 성능이 낮은 Adadelta에 비해 약 12%가 높은 성능을 보여주어 괜찮은 일반화 성능을 보여주었다. 이를 통해 기존에 알려진 사실은 Adam이 우월하다는 것이 정설이지만 학습하는 데이터 그리고 모델별로 이는 바뀔 수 있음을 실감할 수 있었다. 이를 통해 최종 모델에는 SGD와 SAM 중 SGD를 선택하여 모델링을 실시하였다.

C-4) Activation Function 조절



[그림 18: Activation Function 비교]

Baseline 활성함수인 ReLU와 비교하여 SeLU, Hardswish, LeakyReLU에 대하여 실험을 실시했다. Adam과 SeLU의 조합이 성능향상에 좋았다는 선행연구를 통해 SeLU를 선정하였고, ReLU의 변형형태인 LeakyReLU, 그리고 최신기술인 Hardswish를 비교하고자 하였다. 우선적으로 가장 학습이 잘 된 활성함수는 ReLU와 LeakyReLU다. 기존 연구에서 ReLU가 많이 쓰이는 대목을 알 수 있었던 부분이었고 SeLU와 Hardswish에 비해 눈에 띄게 높은 성능을 보여주었다.

SeLU와 Hardswish의 경우는 기대 이하의 성능을 보여주었다. 본 연구에서는 다른 실험환경을 맞추어 주기 위하여 활성함수 이외에 다

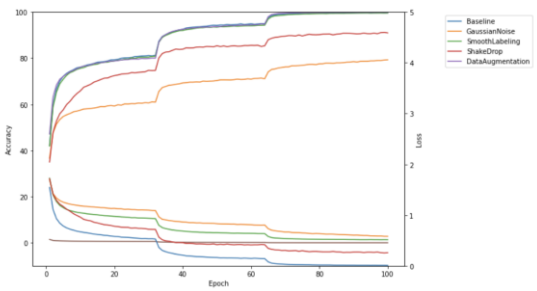
른 조건을 동일하게 하였고, Optimizer가 Adam이 아닌 SGD를 사용하여 기존 논문에서 제시한 조합과 벗어나 유의미한 성능이 안 나왔다고 판단하였다. 더불어 Hardswish의 경우 기대 이하의 학습을 보여주었다. 약 90%정도 훈련 데이터에 적합하였고 최신 기술이라는 말이 무색하게 Baseline의 수준을 따라가지도 못하였다. 이들의 Test Accuracy는 아래 [표 5]와 같다.

Activation Function	Test accuracy
ReLU(Baseline)	87.485%
SeLU	79.275%
Hardswish	83.787%
LeakyReLU	87.542%

[표 5: Activation Function 일반화 성능]

Epoch에 따른 Train Accuracy 그래프에서 보는 것처럼 test accuracy는 LeakyReLU와 ReLU가 가장 높았다. 비교적 학습과정에서 수월하게 학습한 것이 Test Data에도 뛰어난 성능을 보여주었다. 그 이하 순위도 마찬가지로 Hardswish와 SeLU가 차지하였다. SeLU의 경우는 Test Accuracy가 80%에 미쳐 도달하지 못하여 저조한 성능을 보였다. 이를 토대로 본 연구의 최종 모델에서는 LeakyReLU를 활성함수로 사용하여 모델링하였다.

D. 기타 기법에 대한 비교 실험



[그림 19: 기타 기법 비교]

기존 Baseline에 성능향상을 위해 다양한 테크닉을 추가하여 실험을 실시하였다. 실시한 것은 GasussianNoise, labelsmoothing, Shakedrop, DataAugmentation이다. 이들의 train accuracy는 위 [그림 19]와 같다. 눈에 띄는 테크닉은 LabelSmoothing과 Data Augmentation이었다. 학습에 있어 가장 효과적인 학습을 보여주었다. 더불어 Shakedrop의 경우 Validation Data를 활용하지 않고 오버피팅을 효과적으로 막아주는 테크닉으로 Train Accuracy상에는 완전히 Train Data에 적합되지 않아서 성능이 높지 않을 수 있겠다고 판단하였다. Gaussian Noise 같은 경우는 Data Augmentation의 효과가 있음에도 불구하고 더 낮은 Train Accuracy를 보여주었다. 약 70%정도 되는 수치로 Train Accuracy가 Baseline 및 LabelSmoothing 및 Data Augmentation과 30%정도 차이 났다. 이들에 대한 Test Accuracy는 아래 [표 6]과 같다.

Technic	Test accuracy
Baseline	87.485%
GaussianNoise	62.909%
SmoothLabeling	87.529%
Shakedrop	87.380%
DataAugmentation	88.134%

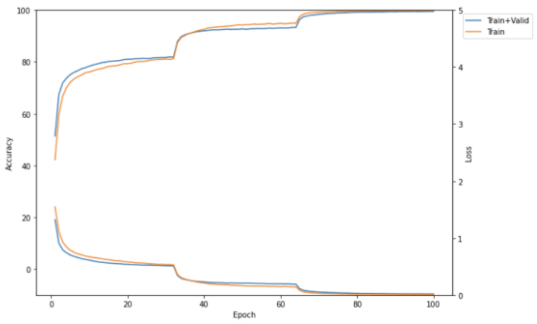
[표 6: 기타기법 일반화 성능]

labelSmoothing과 Data Augmentation은 Train Accuracy Graph에서 보듯 가장 뛰어난 학습능력을 보여준 것처럼 Test Accuracy도 이에 상응하는 결과를 보여주었다. 특히 Data Augmentation의 경우 데이터가 늘어나 학습시간이 약간 증가했으나 이에 따른 가장 좋은 일반화 성능을 보여주었다. 더불어 Shakedrop의 경우도 이에 버금가는 효과를 보여주었는데 Train Accuracy는 0.1%낮지만 Train Accuracy와 큰 차이가 없어 오버피팅을 효과적으로 방지할 수 있다는 사실을 입증하였다.

하지만 Gaussian Noise의 데이터 증강효과를 실감할 수는 없었다. 오히려 15% 많이 하락한 성능을 보여주었다. 이는 기존 선행연구에서 제시한 데이터의 추가로 오버피팅을 방지하는 것은 물론 일반화 성능을 개선해준다는 사실을 반박하였고 Gaussian Noise는 사용하는 데이터 및 모델에 따라 제공하는 효과가 다를 수 있다고 판단하였다.

E. Train VS Train+Validation 일반화 성능 비교

위 D의 결과에 착안하여 Data Augmentation이 효과가 있다는 것을 입증하였다. 본 Competition에서는 기존의 Crop, Rotate의 방식을 활용하는 것이 아닌 Validation Set을 학습데이터에 합쳤을 때 일반화 성능이 개선되는 정도를 비교하고자 하였다. 학습 그래프는 아래 [그림 20]과 같다.



[그림 20: Train VS Train+Validation 비교]

학습 그래프에서는 눈에 띄는 차이를 볼 수는 없었으며 거의 비슷하게 Train Accuracy를 보인 것을 볼 수 있다. 아래 [표 7]은 일반화 성능 즉 Test Accuracy를 나타낸 표다.

Dataset	Test accuracy
Train(Baseline)	87.485%
Train+Validation	90.323%

[표 7: Train VS Train+Validation 일반화 성능]

위 [표 7]에서 드러난 것처럼 약 3%의 성능 개선 효과를 볼 수 있었으며 위 실험 결과에 따라 최종 모델링 할 때 Data Augmentation에 있어 Train Data와 Validation Data를 합치는 것이 옳다고 판단하였다.

F. 최종 모델링

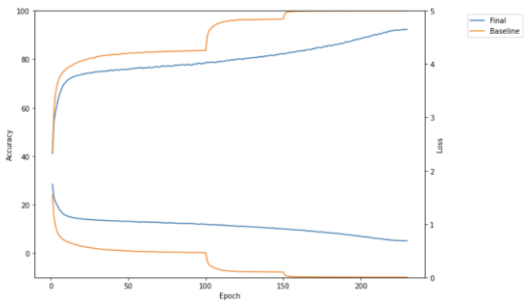
위 B~D의 실험을 기반으로 성능이 좋았던 것들을 모아 모델링을 실시하였다. 우선적으로 모델의 아키텍처는 PyramidNet으로 선정하였다. 비교대상 5가지 모델 중 가장 준수한 성능을 보였기에 CIFAR-10에 적합하다고 판단하였다. 더불어 주 파라미터인 Layer와 Alpha에 대

해 (196, 48)이 최적 조합이란 것을 실험을 통해 획득하여 Layer를 196으로, Alpha를 48로 지정하였다.

우선적으로 학습률의 효율적 조정에 있어 Scheduler로 CosineAnnealingWarmRestart를 활용했고 Optimizer로 SGD 그리고 활성함수로 LeakyReLU를 활용하였다.

더불어 Gaussian Noise는 배제하고 LabelSmoothing과 Shakedrop을 채택하였다. Gaussian Noise의 효과가 미미하여 Data Augmentation 효과를 입증한 것에 기반하여 Train과 Validation Data를 합치기로 하였고 이에 대한 Validation Data의 역할을 Shakedrop이 함으로써 효율적인 학습을 진행하도록 설정하였다. Shakedrop에 대한 효과는 아래 그래프에서 확인 가능하다. Train Accuracy는 비교적 높지 않지만 Test Accuracy 즉, 일반화 성능은 향상되었다. 이는 데이터가 학습하는데 있어 과적합을 방지해주었다고 해석할 수 있었다.

이에 대한 성능은 기존 Baseline이 87.485%의 Test Accuracy를 제시했으나 위와 같은 방법으로 딥러닝 최적화 결과 91.418%로 약 4%정도 Test Accuracy를 증가시킬 수 있어 약 8%대 오류율을 200만개의 파라미터 내에서 달성하였다.



[그림 21: Final Model VS Baseline]

Model	Test accuracy
Baseline	87.485%
Final modeling	91.485%

[표 8: Final 모델 일반화 성능]

Ⅴ. 결론

본 Competition에서 활용한 모델은 PyramidNet이다. 파라미터 수는 190만개로 Layer 196개, Alpha를 48로 진행하여 모델링을 진행하였다. 특히 Layer와 Alpha는 PyramidNet의 주요 파라미터로 선행연구에서 제시한 "Layer 110, Alpha 48"에서 Layer와 Alpha를 각각 늘리고 줄여가며 최적의 조합을 찾은 결과 Layer는 증가할수록 좋은 성능을, Alpha는 무조건 증가시키기보단 Layer와 비례하여 증가 시 좋은 퍼포먼스를 낸다고 판단하여 (196, 48)로 설정하였다.

활성함수 LeakyReLU, 스케줄러는 CosineAnnealing, Optimizer는 SAM을 활용하였다. CosineAnnealingScheduler로 지속적으로 감소하거나 증가하는 Scheduler가 아닌 증감을 반복하는 Scheduler로 효율적인 학습을 할 수 있었고 LeakyReLU를 활용하여 음수인 부분을 부분적으로 살려 성능 개선에 효과가 있었다. 기타 테크닉으로 Shakedrop과 Labelsmoothing을 활용하였다. Shakedrop의 사용으로 Validation Data와 Train Data를 합쳐 Data Augmentation 효과를 얻을 수 있었고 기존의 Validation Data의 역할을 Shakedrop이 대체하여 데이터를 효율적으로 사용할 수 있었다. 더불어 Labelsmoothing을 활용해 데이터 정규화에 있어 0또는 1보다 더욱 유연한 수치를 제공하여 일반화 성능 개선에 효과를 얻을 수 있었다.

본 Competition에서 활용한 자료는 이미지 분류 및 검출에 있어 훌륭하게 쓰일 것으로 판단된다. 정해진 학습 데이터의 수, 200만개의 파라미터 개수에도 불구하고 Error Rate 8%대 성능을 보여주었다. 이는 주어진 환경속에서 이뤄낸 최적화의 효과라고 볼 수 있다. 딥러닝을 모델링함에 있어 데이터가 많고 파라미터 수에 제약 받지 않을 정도로 Computing Resource가 풍부하다면 이러한 최적화 기법은 무의미하다. 하지만 현실세계는 무제한적으로 누릴 수 없으며 한계가 존재하기 마련이다.

연구에 하다보면 가진 자원을 최대한 활용한 후에 성능을 높이고자 고민을 할 것이다. 시간은 주어져 있고 RAM 등의 성능도 제한적이다. 데이터가 풍부하다면 파라미터 수 최적화를 통해 기존보다 더 많은 데이터를 투입할 수 있을 것이고 데이터가 적다면 데이터를 의도적으로 부풀려 학습을 개선할 수 있을 것이다.

현실에는 CNN 이외에도 다양한 딥러닝 분야가 존재한다. 시간은 한정적이며 우리는 더 좋은 모델의 탄생에 있어 딥러닝 최적화는 필수불가결한 존재다. 각 모델별로 적합한 최적화 방법론은 다를 수 있지만 본 연구는 기타 사물 및 사람인식을 넘어 이와 유사한 물체를 생성해내는 GAN 등에도 더욱 효과적으로 활용될 수 있는 방향성을 제시한다.

Ⅳ. 참고문헌

[1] Very Deep Convolutional Networks for Large Scale Image Recognition, arXiv: 1409.1556, Karen Sinmonyan, UK, 2015

[2] Deep Residual Learning for Image Recognition, CVPR, Kaiming He, 2016, pp770-778

[3] Wide Residual Networks, arXiv: 1604.07146, Sergey Zagoruyko, France, 2016

[4] Densely Connected Convolutional Networks, CVPR, Gao Huang, 2017, pp. 4700-4708

[5] Deep Pyramid Residual Networks, CVPR, Dongyoon Han, 2017, pp. 5927-5935

[6] Data Augmentation Using Random Image Cropping and Patching for Deep CNNs, IEEE Transactions on Circuits and Systems for Video Technology vol 30, n9, Ryo Takahashi, pp 2917-2931, 2020

[7] Bag of Tricks for Image Classification with Convolutional Neural Networks, arXiv:1812.01187, Tong He, 2018

[8] A Closer Look at Deep Learning Heuristics: Learning rate restarts, Warmup and Distillation. arXiv:1810.13243, Akhilesh Gotmare, 2018

[9] Early Stopping — But When? Neural Networks: Tricks of the Trade, Lecture Notes in Computer Science, vol 7700. Springer, Berlin, Heidelberg, Prechelt L, 2012

[10] An Exponential Learning Rate Schedule for Deep Learning, arXiv:1910.07454, Zhiyuan Li, 2019

[11] SWISH: A SELF-GATED ACTIVATION FUNCTION, arXiv:1710.05941v1, Prajit Ramachandran, 2017

[12] Activation Functions: Comparison of Trends in Practice and Research for Deep Learning, arXiv:1811.03378v1, Chigozie Enyinna Nwankpa, 2018

[13] ShakeDrop Regularization for Deep Residual Learning. arXiv:1802.02375v3, YOSHIHIRO YAMADA, 2019

[14] Self-Normalizing Neural Networks, Günter Klambauer, Thomas Unterthiner, Andreas Mayr, Sepp Hochreiter, NIPS, 2017.

[15] Sharpness-Aware Minimization for Efficiently Improving Generalization, Forêt, P., A. Kleiner, H. Mobahi and Behnam Neyshabur, 2020.

[16] Adam: A Method for Stochastic Optimization, Kingma, Diederik P. and Jimmy Ba, 2015.

[17] ADADELTA: An Adaptive Learning Rate Method, Zeiler, Matthew D, 2012.

[18] When Does Label Smoothing Help?, Müller, R., Simon Kornblith and Geoffrey E. Hinton, NeurIPS, 2019.

[19] Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, Ioffe, S. and Christian Szegedy, *ICML,* 2015.

[20] A Review Paper: Noise Models in Digital Image Processing, Boyat, Ajay & Joshi, Brijendra. Signal & Image Processing: An International Journal, 2015.

저자1 김민경은 2018년 경희대학교 글로벌커뮤니케이션학부로 입학하여 SW교양과 웹/파이썬 프로그래밍을 듣던 중 프로그래밍에 흥미가 생겨 컴퓨

터공학과로 전과하게 되었다. 관심있는 분야는 임베디드, 영상 처리이다. 현재 머신러닝과 컴퓨터비젼을 배우며 인공지능의 기초를 공부하고 있으며 앞으로 영상처리, 임베디드와 연관된 AI 기술들을 공부하는 것이 목표이다.

저**자2 김성수**는 1997년 서울에서 태어났다. 2016년 3월, 경희대학교 산업경영공학과에 입학하여 현재 2020년 12월 학부생으로 3학년으로 재학중이다. 육군을 만기 전역했고 2학년 과대표를 수행하였다. 2017년 3학년 2학기부터 경희대학교 산업경영공학과 Data Mining 연구실에서 학부연구생으로 연구를 진행하고 있으며 2020 KHUTHON에서 연관성 분석을 통한 모델링을 통해 우수상을 받은 경력이 있다. DataMining 및 데이터 사이언스 분야에 관심이 있으며 추후 해당 분야에 전문적 지식을 쌓기 위하여 현재 대학원 진학을 생각하고 있다. 이를 위해 최근 머신러닝 및 딥러닝에 대해 학습하고 있으며 미래에 데이터 사이언스나 딥러닝 분야에 한 획을 긋는 엔지니어가 되는 것이 최종목표로 하고 있다.

저자3 이찬은 2016년에 경희대학교에 입학하여 산업경영공학과를 전공하였다. 2016~2018년 육군 만기 전역을 한 뒤에 컴퓨터공학과에 흥미를 느껴 2019년부터 컴퓨터공학과 복수전공을 시작하였다. 2019년 하반기에 데이터마이닝 관련 수업을 들으면서 데이터 분석에 관심을 가지게 되었다. 2020년 1분기 동안 데이터마이닝 연구실에서 교수님과 단기 연구를 진행하였다. 2020년 9~11월에는 (주)테서에서 국가에서 진행하는 데이터 바우처 사업을 보조하면서 데이터 분석가의 자질을 다졌다. 데이터 바우처 사업 중에 머신러닝에 관심을 가지게 되었으며, 현재 머신러닝 수업을 들으면서 AI를 공부중이다.