안녕하세요 Deep Residual Networks에 대한 발제를 맡은 DS팀 박수연이라고 합니다.

여기 계신 여러분 모두 Deep Residual Networks, 다른 말로 ResNet이라는 단어는 한 번쯤 들어 보셨을 것이라고 생각합니다. 2015년에 ResNet을 처음 소개한 Deep Residual Learning for Image Recognition 논문의 인용 횟수만 보더라도, 이 개념이 CNN 분야에 얼마나 큰 영향을 미쳤는지 짐작할 수 있습니다. 오늘은 이 ResNet이 어떻게 나오게 되었고, 어떤 장점들이 있는지, 어떤 구조를 가지고 있는지에 대해 알아보려고 합니다.

먼저 ResNet이 나오기 이전에 사람들은 layer를 더 많이 쌓는 방법으로 이미지 인식의 정확도를 높이려고 애썼습니다. layer를 더 많이 쌓아 neural network의 깊이가 깊어질수록 일반적으로 성능이 좋아졌기 때문입니다. 그러나 일정 수준 이상으로 layer를 깊게 쌓게 되면 두 가지 문제가 발생하였습니다. 첫번째는 vanishing gradients problem이고, 두번째는 degradation problem입니다. vanishing gradients problem은 layer를 많이 거칠수록 전달되는 기울기 값이 크게 줄어들어 초기 layer로 갈수록 학습이 잘 되지 않는다는 문제입니다. degradation problem은 test accuracy가 떨어지는 문제입니다. 일반적으로 train accuracy가 너무 높고 test accuracy가 낮은 원인인 overfitting과 다르게, layer가 깊어지면 test accuracy 뿐만 아니라 train accuracy도 매우 낮아지는 문제를 발견하였습니다. 따라서 이 논문의 저자들은 vanishing/exploding gradients 문제보다는 degradation problem에 집중하여 어떻게 하면 layer가 깊어져도 정확도를 높일 수 있는지 고민하였습니다.

여러가지 시도를 통해 깊은 neural network에서도 정확도가 높은 방법인 Residual Learning이 나오게 되었습니다. Residual Learning은 쉽게 말해서, 이전 layer의 결과를 그대로 다시 이용하는 것을 말합니다. 기존의 neural network와 ResNet을 비교하여 어떤 점들이 달라졌는지 알아보겠습니다. 먼저 기존의 neural network는 입력 값 x가 주어졌을 때 여러 layer를 거쳐 우리가 원하는 결과 값으로 mapping 하는 함수 H(x)를 구하는 것이 목표였습니다. 즉, 그동안은 input x에 따라 우리가 의도한 대로 값을 출력하는 함수 H(x)를 구하여 모델에 학습시키는 과정이 필요했습니다.

그러나 ResNet의 경우 앞선 layer에서 학습된 정보 x를 그대로 가져오고, 입력 값 x가 weight layer를 거쳐서 나온 결과인 F(x)를 더한 F(x) + x 값이 우리가 의도한 mapping function H(x)와 같다고 생각했습니다. 그리고 이때 F(x)를 Residual Function이라고 합니다. 이렇게 H(x)를 F(x) + x로 보게 되면, x는 입력 값이므로 우리가 구할 필요가 없기 때문에 F(x) 값만 구하여 모델에 학습시켜주면 됩니다. F(x)만 추가적으로 학습하는 것은 H(x)를 전부 학습하는 것보다 훨씬 쉽습니다. 기존의 neural network 구조는 각각의 layer에 대해 weight 값을 개별적으로 모두 학습을 진행하여 수렴하기가 어려운 반면, F(x)는 H(x)가 x가 되도록 identity mapping을 수행할 경우 F(x) 값이 0이 나오기 때문에 최적화하기 매우 쉽습니다. 따라서 ResNet의 구조를 보면, 입력에서 출력으로 연결되는 shortcut path가 추가된 것을 확인할 수 있습니다.

이렇게 입력 값을 출력 값으로 연결시킨 구조인 ResNet은 기존의 Neural Network가 가지고 있던 문제들을 많이 해결하였습니다. 첫번째로, 단순히 입력 값과 같은 x가 출력 값에 더해지는 형태기 때문에, parameter 수에 영향이 없고 덧셈을 제외하면 연산량 증가가 없습니다. 이때 F(x) + x 연산을 위해서 F(x)와 x의 차원이 같아야 하는데, 차원이 다를 경우 이를 같게 맞추기 위하여 linear projection인 Ws를 x와 곱하여 덧셈을 수행합니다. 이 그림을 보시면 첫번째 weight layer의 weight 값 W1과 x를 곱한 값을 activation function ReLU에 통과시키고, 이 값에다 다시 두번째 weight layer의 weight 값 W2를 곱한 값이 우리가 원하는 F(x)입니다. 여기다 x를 그냥 더하는 것이 아니라, linear projection Ws를 x에 곱하여 차원을 맞춰주고 더하는 것을 보실 수 있습니다.

두번째 장점으로, 발제 초반에 언급했던 기존 깊은 layer를 가진 neural network의 문제인 vanishing gradient problem도 어느 정도 해결할 수 있습니다. 앞서 보았던 식을 다시 한번 가져오겠습니다. 이 식의 양변에 미분을 적용하여 Back Propagation을 수행할 때 weight가 어떻게 되는지 관찰해봅시다. 왼쪽 항의 값이 0이 되면 weight 값이 사라져 학습이 제대로 이루어지지 않는 문제인 vanishing gradient problem이 발생합니다. 그러나 우변을 보면, 초기 입력 값 x가 미분을 해도 1로 남아있는 것을 확인할 수 있습니다. 따라서 입력 값을 출력 값으로 연결시켜주는 ResNet 구조는 gradient descent 문제도 해결할 수 있다는 장점이 있습니다.

이렇게 parameter 수가 변하지 않고, vanishing gradient problem을 해결할 수 있다는 장점들도 물론 훌륭하지만, ResNet을 이렇게 빛나게 한 건 뭐니뭐니 해도 역시 깊은 layer에서 굉장히 뛰어난 성능을 보여준다는 점입니다. 초기에 Image Classification을 위해 나온 모델인 AlexNet은 겨우 8개의 layer였습니다. 저희가 지난 시간에 배웠던 VGG와 GoogleNet은 각각 19개, 22개의 layer로 AlexNet보다는 많이 깊어진 것을 볼 수 있습니다. 그러나 ResNet의 경우 이와 비교도 안 될 정도인 152개의 layer를 쌓아 굉장히 깊은 모델을 만들었다는 사실을 알 수 있습니다. ResNet을 test error 측면에서 보더라도 3.57 퍼센트의 error로 다른 모델들보다 훨씬 낮은 error 값을 기록한 것을 볼 수 있습니다. layer가 깊어질수록 정확도가 떨어지는 문제인 degradation problem을 해결하고, 이렇게 다른 모델들보다 깊은 구조임에도 불구하고 가장 정확한 Image Classification을 수행할 수 있다는 사실을 알 수 있습니다. 논문을 보시면 Image Classification 말고도 Object Detection, Segmentation 등의 분야에서도 ResNet이 굉장히 높은 정확도를 보이고 있다는 점을 확인할 수 있습니다.

그럼 본격적으로 ResNet의 Architecture에 대해 자세히 알아보겠습니다. 일단 ResNet의 기본 구조는 저희가 지난 시간에 배웠던 VGG net입니다. convolution filter의 크기는 대부분 3 곱하기 3이고, output feature map 크기를 모두 같게 만들기 위해 layer들이 모두 같은 수의 convolution filter를 사용하도록 구성되어 있습니다. Output feature map의 크기가 절반으로 줄어드는 경우에는 time complexity를 동일하게 유지해주기 위해서 convolution filter의 수를 2배로 늘려주었습니다. 또 특이하게 Downsampling을 수행할 때는 pooling이 아니라 stride가 2인 convolution filter를 사용하였습니다. 모델 마지막 부분에는 Global Average Pooling인 GAP을 사용하여 차원을 줄이고 벡터로 만든 뒤에, 사이즈가 1000인 Fully connected layer와 Softmax를 사용하여 classification을 합니다. 이런 Base VGG net 구조에 Short cut connection을 추가한 형태가 바로 ResNet의 Architecture입니다. 이때, layer의 input과 output의 차원이 같다면 identity short cut을 바로 연결해주면 되지만, 차원이 맞지 않는 경우 input과 output의 차원을 같게 만드는 과정이 꼭 필요합니다. input과 output의 차원을 맞추는 방법은 두 가지가 있습니다. 첫번째는 zero padding을 적용하여 차원의 크기를 증가시켜 맞추는 방법입니다. 두번째 방법은 1 곱하기 1 convolution filter를 사용하여 앞서 말씀드린 linear projection Ws를 곱하여 차원을 맞추는 것입니다. 이렇게 차원을 맞춰 연결한 short cut path들은 feature map을 2칸씩 건너 뛰므로 stride가 2인 것도 확인하실 수 있습니다.

이제 기본적인 ResNet의 구조를 바탕으로, 변형된 ResNet Architecture들은 어떤 것들이 있는지, 그리고 어떤 점들이 변형되었는지 알아보겠습니다.

먼저 ResNet 구조에서 training 시간을 고려하여 Bottleneck design을 적용한 Deeper Bottleneck Architecture가 있습니다. 기본 ResNet이 3 곱하기 3 convolution layer을 건너뛰는 short cut path를 연결한 구조라면, Deeper Bottleneck 구조는 1 곱하기 1 convolution layer가 추가되는 것이 핵심입니다. 1 곱하기 1 Convolution layer는 연산량이 작고, 주로 Feature map의 크기를 줄이거나 키울 때 사용합니다. Deeper Bottleneck Architecture는 기존 ResNet 구조와 복잡성은 비슷하게 가지되, input과 output의 차원을 줄여 연산량을 감소시키는 특징이 있습니다. 참고로 이 그림에서 보면 왼쪽에는 64-dimension이 오른쪽에는 256-dimension으로 증가한 것을 확인할 수 있는데, 이는 identity short cut을 유지하기 위해, zero padding을 사용하여 차원을 맞춰준 것입니다.

다음으로는 Pre-Activation ResNet이라고 하는 사전 활성화 ResNet 구조가 있습니다. 기존의 ResNet이 skip connection이 적용된 후 activation function을 거치는 형태라면, Pre-Activation ResNet은 순서를 바꿔 activation function을 먼저 거친 이후에 skip connection이 적용되는 구조입니다. 왜 이런 구조를 만들게 되었을까요? 일반적인 CNN의 경우 최종 결과 값은 수많은 행렬들의 곱셈으로 이루어집니다. 반면, ResNet의 경우에는 최종 결과 값을 Residual Function인 F의 합들로 표현할 수 있습니다. 이를 이해하기 위해, 먼저 Residual Unit의 수식을 살펴봅시다. 이 식을 보기 전에 몇 가지 기호들에 대해 설명해드리겠습니다. 먼저 x는 Residual unit의 입력 값, y는 Residual unit의 출력 값을 의미합니다. 아래 첨자로 나와있는 l은 각 layer의 번호이고, W는 가중치 행렬 즉 weight입니다. 대문자 F는 Residual Function F(x)를 의미하고, 소문자 f는 activation function을 의미합니다. 마지막으로 h는 mapping function H(x)인데, 우리는 이 H(x)가 x와 같도록 하는 Identity Mapping을 원하기 때문에 그냥 차원을 같도록 만들어주는 identity function이라고 생각하시면 됩니다. 이 식에서 activation function f를 ReLU가 아니라 Identity mapping이라고 생각한다면, x l+1은 f(yl)이라는 식을 x l+1은 yl이다. 로 바꿀 수 있습니다. 그리고 이렇게 만들어진 식을 1번 식의 좌변에 대입하면 다음과 같은 식의 형태가 구해집니다. 이 x l+1의 형태는 x l이 Residual Block을 한 번 거쳐서 나온 결과입니다. 같은 방법으로 Residual Block을 한 번 더 거치게 되면 다음과 같은 식이 구해집니다. 여러 Residual Block을 거친 결과는 어떻게 될까요? 이 식은 앞에서 저희가 했던 과정들을 반복하여 일반화한 결과입니다. 이때 대문자 L은 전체 layer의 인덱스를, 소문자 l은 부분 layer의 인덱스를 나타냅니다. 이 식에서 볼 수 있듯이, Feed Forwarding을 할 때 ResNet은 값들이 Residual Function인 F의 합들로 표현 가능합니다. 또한, 초기 입력 값 xl이 최종 layer large L까지 남아있는 것을 알 수 있습니다. 다음 슬라이드를 보시면, 곱셈으로 이루어진 기존의 CNN들과 다르게 forward path가 덧셈으로 단순해져 계산이 쉬워지는 장점을 확인할 수 있습니다.

activation function을 ReLU에서 identity mapping으로 바꾸면 Feed Fowarding이 쉬워질 뿐만 아니라, Back Propagation에서도 vanishing gradient problem을 방지할 수 있습니다. 앞에 나온 식을 다시 가져와 양변을 미분하면, 다음과 같은 식이 얻어집니다. 이 식에서 E는 loss function을 나타내고, 좌변은 현재 layer인 xl에 적용되는 weight 값을 의미합니다. 이 값이 만약 0이 되버린다면, weight가 제대로 업데이트 되지 않는 vanishing gradient problem이 발생합니다. 그러나, 남아있던 xl이 chain rule에 의해 미분해도 1로 남아있기 때문에, gradient가 소실되지 않습니다. 따라서 activation function을 ReLU에서 identity mapping으로 만든다면 forward propagation과 back propagation 단계에서 모두 큰 장점이 있습니다.

그러나 일반적인 ResNet의 구조는 activation function으로 Identity mapping이 아니라 ReLU 함수를 사용하고 있습니다. 그렇다면 ResNet의 구조를 바꿔서 activation function을 ReLU가 아니라 Identity mapping으로 만드는 방법이 없을까요? 그 질문에 대한 답으로 나온 구조가 바로 Pre-Activation ResNet입니다. 이 그림을 보시면, 논문의 저자들이 시도한 여러가지 ResNet 구조의 변형들이 있습니다. 모두 Deep Bottleneck Architecture 구조이지만, layer의 순서들이 다릅니다. 먼저 우리가 기본적으로 알고 있는 ResNet은 short cut path를 거친 값이 합쳐진 이후에 ReLU를 지나는 것을 볼 수 있습니다. (b)는 어떤 점이 바뀌었을까요? 바로 마지막 Batch Normalization을 addition 연산 뒤로 옮겼습니다. 잠깐 Batch Normalization에 대해 설명하자면, Batch normalization은 평균과 표준편차를 조정하는 정규화 과정의 일종으로 모델이 학습할 때 각 Batch마다 평균과 분산을 조정하여 학습의 속도를 높이고 입력 값이 특정 방향으로 쏠리는 것을 방지하는 역할을 합니다. Batch는 모델이 한번 parameter를 업데이트할 때 사용할 데이터의 개수를 의미합니다. 다시 돌아와서, 마지막 Batch normalization을 addition 연산 뒤로 옮긴 (b)는 원래 ResNet 구조보다 error 값이 더 커진 것을 알 수 있습니다. (c)의 경우에는 ReLU를 shortcut path 안으로 넣어 ReLU를 통과한 뒤에 addition 연산을 수행하는 것을 볼 수 있습니다. 언뜻 보면 (a)보다 성능이 좋아야 할 것 같지만, 실제로는 (a)보다 error 값이 증가한 것을 확인할 수 있습니다. 이는 ReLU를 addition 연산 직전에 통과시켜 ReLU 함수의 그래프 모양 상, 양수 값만 남아있고 음수 값은 모두 0으로 바뀌기 때문입니다. 따라서 forward propagation을 수행할 때 값이 양수로 편향되는 문제 때문에 성능이 나빠진 것입니다. (d)와 (e)는 weight layer를 통과하기 이전에 ReLU를 통과하는 pre-activation 구조입니다. (d)를 보면 ReLU만 앞에 놓은 구조인 반면, (e)를 보면 ReLu와 Batch Normalization 모두 weight layer 앞에 위치한 구조입니다. (d)와 (e) 중 결과적으로 더 낮은 error 값을 기록한 (e)를 full pre-activation architecture라고 부릅니다.

그렇다면 Pre-Activation 구조는 왜 ResNet보다 성능이 좋은 걸까요? 첫번째 이유는 바로 activation function f가 identity mapping 형태가 되어 최적화하기 쉬워졌기 때문입니다. Pre-Activation 구조를 만들게 된 이유기도 하죠. ReLU를 activation function으로 사용하면 skip connection을 거쳐 더해진 값이 음수면 0이 되어 전부 사라지기 때문에, 음의 손실이 많이 발생하여 초기에 학습할 때 오차가 많이 발생하게 됩니다. 그러나 pre-activation 구조에서는 음수 값들도 그대로 사용할 수 있어, 초반에도 학습이 잘 되는 것을 확인할 수 있습니다. 굵은 선이 test error를 나타내고, 점선이 training loss를 나타냅니다. 두번째 이유는 Batch Normalization의 영향으로 정규화가 잘 되어 overfitting을 방지하기 때문입니다. Pre-Activation 구조는 Batch Normalization을 거쳐 정규화된 신호가 weight layer의 input으로 들어가기 때문에 ResNet보다 일반화 성능이 높습니다. 따라서 그래프를 보면 overfitting을 방지하여 origin ResNet보다 pre-activation model이 training loss는 높지만, test error는 더 낮은 것을 확인하실 수 있습니다.

마지막으로 ResNext에 대한 설명으로 넘어가겠습니다. ResNeXt는 Aggregated Residual Transformations for Deep Neural Networks라는 논문에서 2016년에 소개된 모델입니다. ResNeXt는 앞서 소개드렸던 ResNet의 Deeper Bottleneck Architecture를 수정한 구조입니다. Deeper Bottleneck Architecture와 비교해봤을 때, 입력의 차원은 256 dimension으로 같지만, 첫번째 layer에서 1 곱하기 1 convolution filter를 사용하여 channel의 수를 입력 값의 절반인 128로 만듭니다. 그 다음으로, 128개의 channel을 32개의 그룹으로 나눠 각 그룹 당 128 나누기 32 값인 4개의 채널이 되도록 합니다. 이때 32개의 group convolution은 각각 4개의 입력 값에 대해 연산을 수행하여 4개의 feature map을 만들게 됩니다. 그리고 마지막 layer에서는 32개의 그룹에서 생성한 4개의 feature map들을 연결하여 총 128개의 channel을 만들고, 이를 다시 1 곱하기 1 convolution filter를 거쳐 output이 256 dimension이 되도록 만듭니다. 이런 아이디어는 VGG net에서 GPU의 한계로 인해 neural network의 channel 수를 두개로 쪼개서 각각 2개의 GPU로 연산하였을 때 모델의 성능이 증가했다는 사실에서 출발하게 되었습니다. 이렇게 그룹을 나눠 학습하시키는 방법을 Grouped convolution이라고 하고, 이런 Grouped convolution에서는 서로 다른 그룹에 포함된 feature map들이 서로 다른 특징에 집중하여 학습되기 때문에 좋은 성능을 보입니다.

ResNeXt 모델의 대표적인 hyperparameter에는 Cardinality와 Width가 있습니다. Cardinality는 전체 channel 수를 몇 개의 그룹으로 나눌지를 조정하는 hyperparameter고, Width는 하나의 그룹에 포함된 channel 수를 의미하는 hyperparameter입니다. 이전에 설명했던 구조에서는 128개의 channel을 32개의 그룹으로 나눠 하나의 그룹당 4개의 channel이 되도록 만들었으므로, cardinality가 32이고 width가 4인 것을 알 수 있습니다. 그리고 이 논문이 나오기 이전에 Wide Residual Network에서 모델의 깊이를 증가시키는 것보단 Width를 증가시키는 것이 모델의 성능에 더욱 효과적이라는 연구가 밝혀졌습니다. 여기서 한 걸음 더 나아가 ResNext의 논문 저자들은 width를 증가시키는 것보다 Cardinality를 증가시키는 것이 모델의 성능에 더욱 효과적이라는 사실을 밝혀냈습니다. 동일한 parameter 수를 가지는 세 모델이 있을 때, Cardinality를 증가시킨 모델이 가장 효과적이고, 그 다음이 width를 증가시킨 모델, 마지막이 layer를 깊게 쌓은 모델 순이라는 것입니다. 논문에 포함된 실험 결과를 보시면, 정말로 ResNet-200, ResNet-101, ResNext-101 중 ResNext-101의 성능이 가장 높은 것을 확인할 수 있습니다.

지금까지 많은 내용을 전달해드려서 마지막으로 정리하고 발표를 마무리하겠습니다. ResNet은 기존 VGG model에서 short cut path를 추가한 Residual block 구조를 통해 layer를 깊게 쌓았을 때 정확도가 떨어지는 문제를 해결하였습니다. 이런 ResNet의 구조를 변형시킨 모델로 Deeper Bottleneck Architecture, Pre-Activation, ResNext 각각의 구조와 특징 등을 소개해드렸습니다.

이상으로 발제를 마치겠습니다. 감사합니다.