ResNet

Deep Residual Learning for Image Recognition

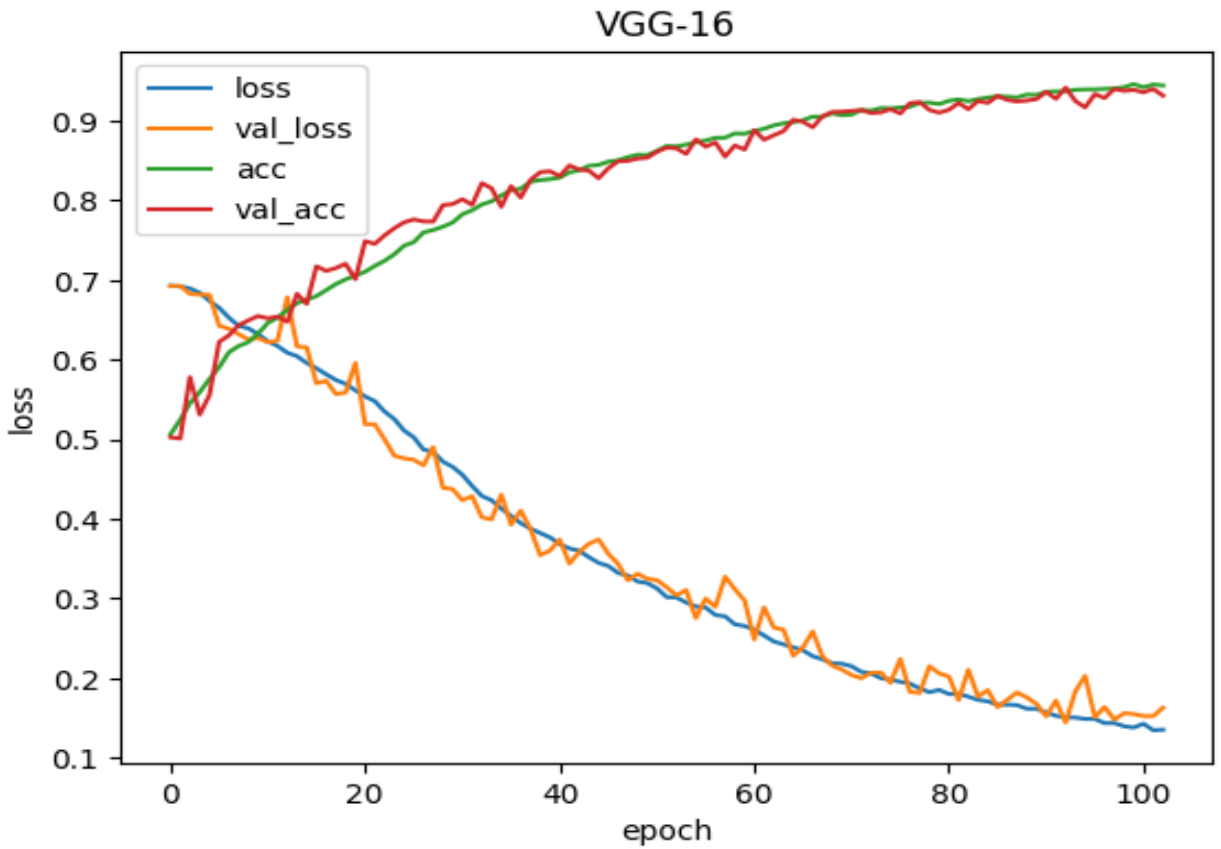
서울과학기술대학교 국방인공지능응용학과
이찬호

Part 1

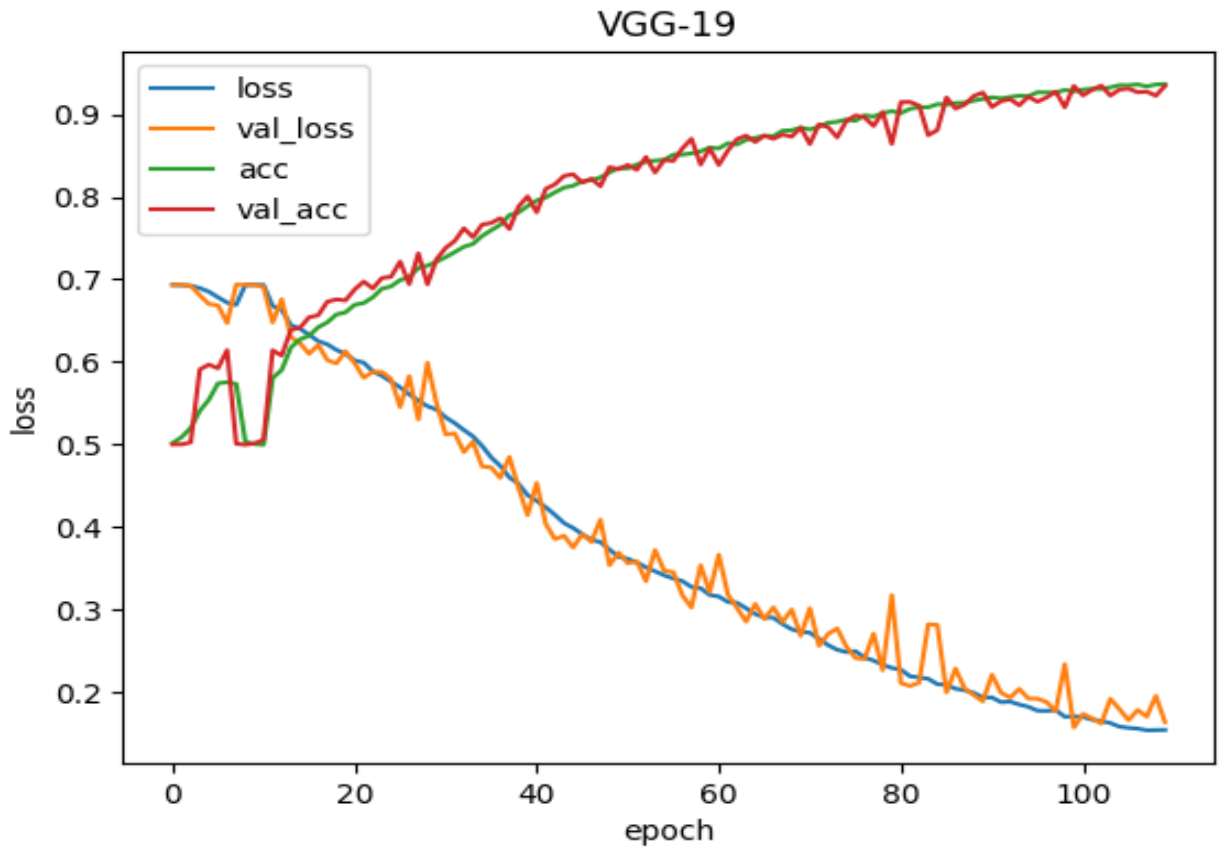
Previous –VGG-16&VGG-19

Part 1

Previous –VGG-16&VGG-19



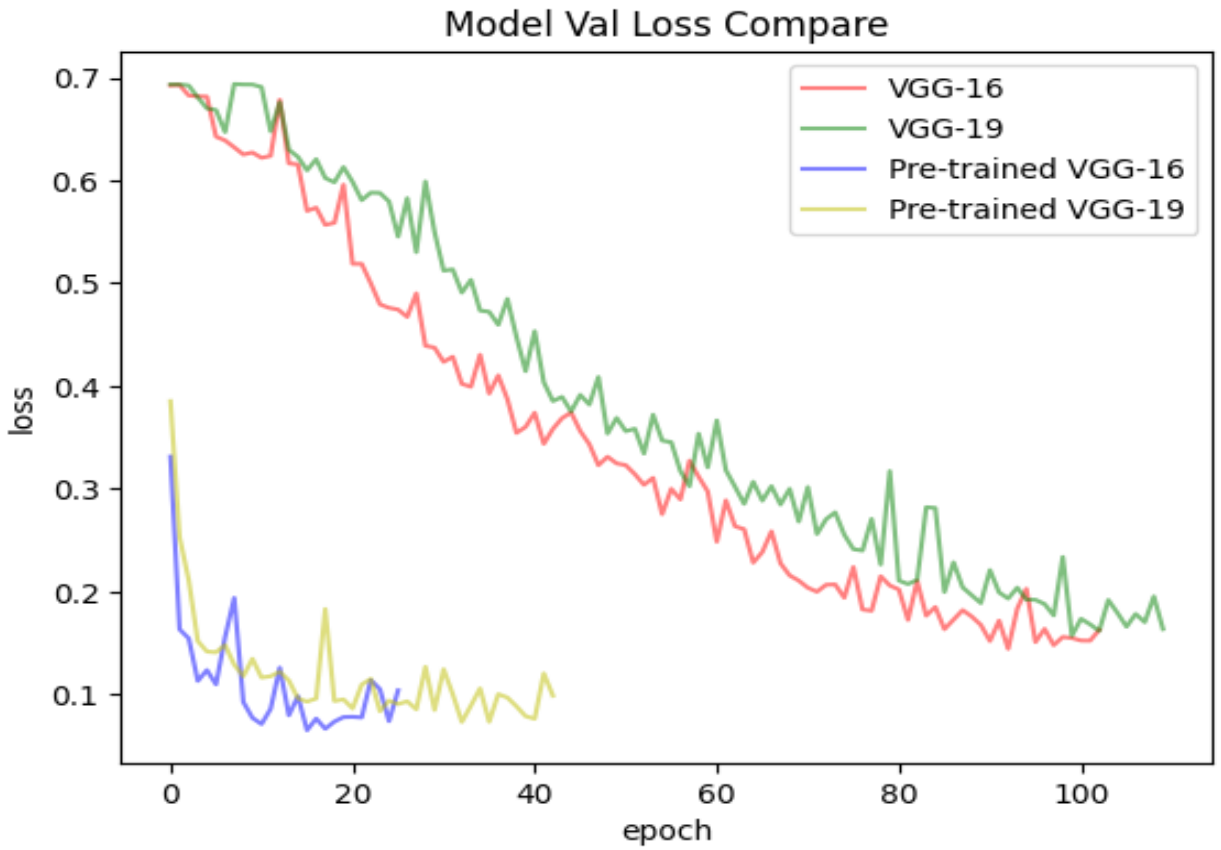
Activation: softmax
Optimizer: SGD
Learning Rate: 0.01
Loss: categorical_crossentropy
Epochs: 200 (103회에 조기 종료)
Best Val Acc: 0.9416
Best Val Loss: 0.1442

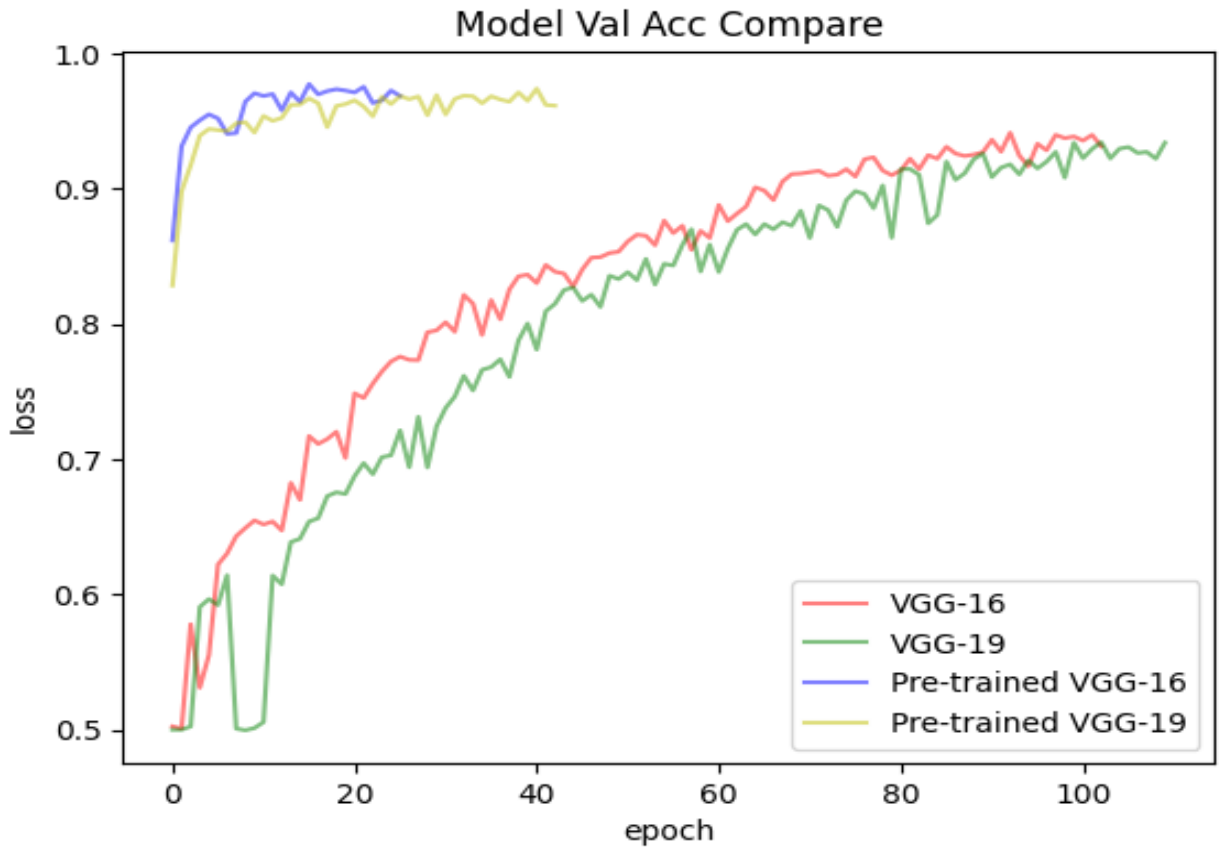


Activation: softmax
Optimizer: SGD
Learning Rate: 0.01
Loss: categorical_crossentropy
Epochs: 200 (110회에 조기 종료)
Best Val Acc: 0.9346
Best Val Loss: 0.1571

Part 1

Previous





VGG-16 Val Loss: 0.1442
VGG-19 Val Loss: 0.1571
Pre-trained VGG-16 Val Loss: 0.065
Pre-trained VGG-19 Val Loss: 0.073

VGG-16 Val Acc: 0.9416
VGG-19 Val Acc : 0.9346
Pre-trained VGG-16 Val Acc : 0.9776
Pre-trained VGG-19 Val Acc : 0.9742

Part 2

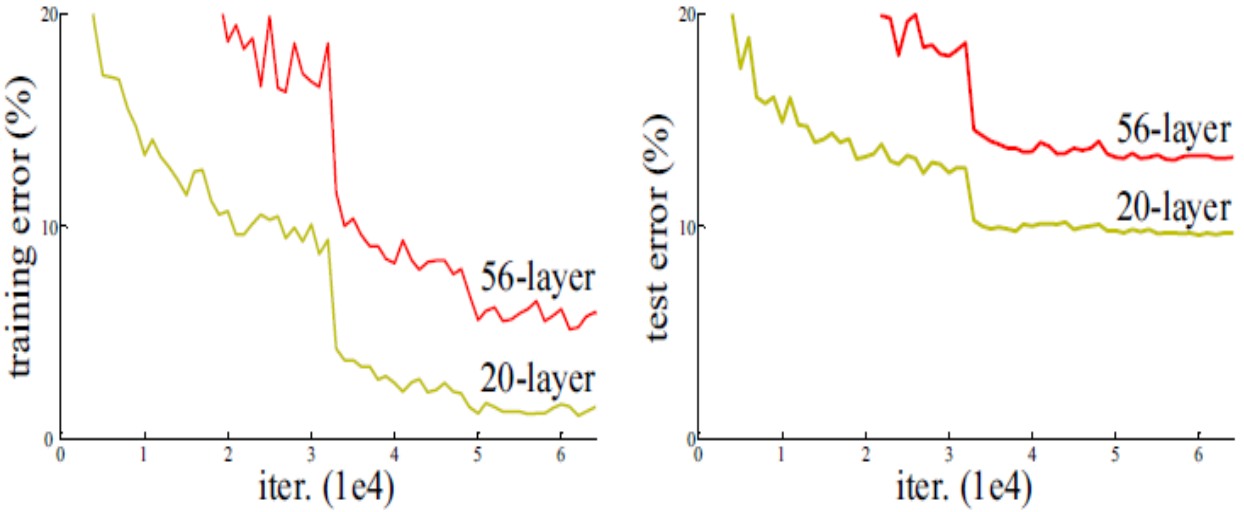
ResNet

Part 2

ResNet

Introduction

- ResNet은 MS에서 residual* learning을 이용해 개발한 알고리즘으로 ILSVRC 2015에서 우승
- Deep layer를 가진 모델들은 vanishing/exploding gradient 문제를 겪으며 optimize가 쉽지 않은데, normalization layer나 normalized initialization* 같은 방법을 사용
- 하지만, 더 깊어지면 vanishing gradient 문제가 발생하면서 degradation* 문제가 발생
- 본 논문에서는 이 문제가 optimize 자체에 대한 문제라 보고 identity mapping*을 사용하는 구조적 변화를 해결하고자 deep residual learning framework를 제안



<Figure 1. degradation>

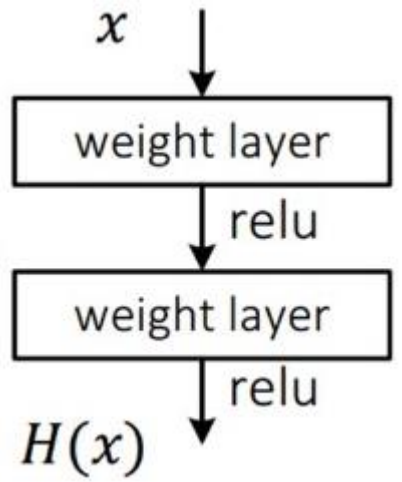
- **Residual:** 결과의 오류(y에서 x를 뺀 나머지) **(ResNet** 이전엔 평가의 기준으로만 삼음**)**
- **normalized initialization:** 가중치 초기화 기법
- **degradation:** 모델이 커지면서 **train acc, test acc**가 낮아지는 것 **(overfitting**은 **test acc**만 낮아짐**)**
- **Identity mapping:** 어떤 입력 값**(**x**)**이 주어졌을 때, 해당 입력 값이 그대로 출력 값으로 매핑 되는 것 **(**$f(x) = x$**)**

Part 2

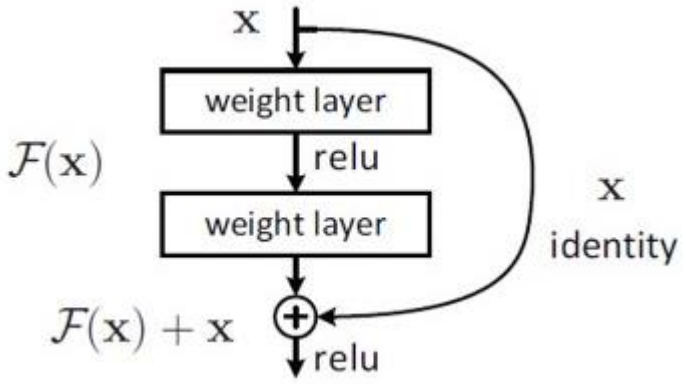
ResNet

Deep Residual Learning Framework

- 기존 identity mapping은 입력 x를 받고 layer를 거쳐 $\mathrm{H}(x)$를 얻는 것이 목적
- Residual Learning은 $\mathrm{H}(x)$가 아닌 출력과 입력의 차인 $\mathrm{H}(x) - x$를 얻는 것이 목표
- 따라서 Residual Function인 $\mathrm{F}(\mathrm{x}) = \mathrm{H}(x) - x$를 최소화하며 이는 즉, 출력과 입력의 차를 줄인다는 의미
- x 값은 입력 값이므로 도중에 변경하지 못해서 $\mathrm{F}(x) = 0$이 최적의 해이고, 결국 0 = $\mathrm{H}(x) - x$로 $\mathrm{H}(x) = x$가 됨
- 즉, $\mathrm{H}(x)$를 x로 mapping하는 것이 학습 목표
- $\mathrm{H}(x) = x$라는 최적의 목표 값이 pre-conditioning으로 제공되어 identity mapping인 $\mathrm{F}(x)$가 학습이 더 쉬워짐
- 결과적으로 $\mathrm{H}(x) = F(x) + x$이므로 입력에서 출력으로 바로 연결되는 shortcut만 추가하면 네트워크 구조 또한 크게 변경할 필요가 없음
- 또한, 입력과 같은 x가 그대로 출력에 연결되기 때문에 파라미터 수에 영향이 없고 덧셈이 늘어나는 것만 제외하면 shortcut 연결을 통한 연산량 증가는 없음
- 곱셈 연산에서 덧셈 연산으로 변경되어 몇 개의 layer를 건너 뛰는 효과를 얻어서 forward와 backward가 단순해지는 효과가 있고 gradient 소멸 문제도 해결



<Figure2. 기존 네트워크**>**



<Figure3. ResNet 네트워크**>**

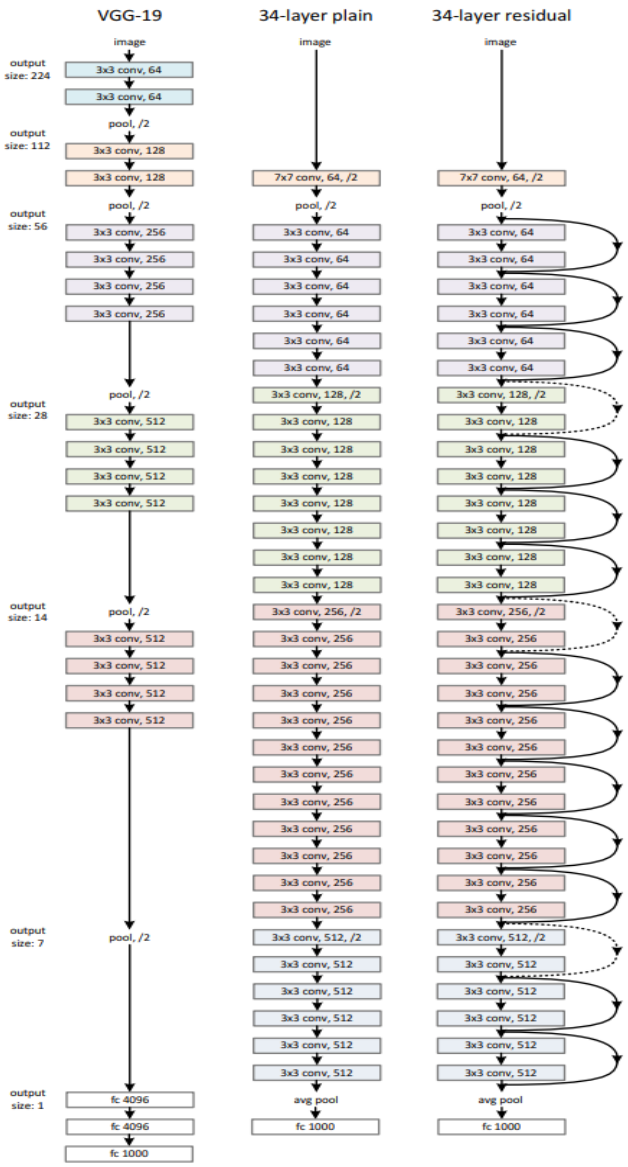
Part 2

ResNet

Architecture

- Plain Network
  - baseline 모델로 VGGNet 영향을 받음
  - Conv filter 사이즈 : 3 x 3 (VGGNet 영향)
  - Output feature map 사이즈가 같은 layer들은 모두 같은 수의 conv filter를 사용
  - Output feature map 사이즈가 반으로 줄 시 time complexity를 동일하게 유지하기 위해 필터 수를 2배로 늘림
  - Down sampling 수행 시 pooling을 수행 하는 것이 아닌 stride가 2인 conv filter를 사용
  - 끝에 Global Average Pooling과 1000 사이즈인 FC layer와 Softmax를 사용 (VGGNet 영향)
- Residual Network
  - Plain을 기반으로 Shortcut connection을 추가
  - Input과 output 차원이 같은 경우 identity shortcut 사용
  - 차원 증가 시 2가지 방법 사용
    - zero padding 적용 → 차원 증가
    - projection shortcut 사용 (1 x 1 conv)
      (shortcut이 feature map을 2 사이즈씩 건너 뛰어서 stride 2로 설정)

<Figure4. Architecture>



Part 2

ResNet

Implementation

- 이미지 짧은 쪽이 [256, 480] 사이가 되도록 랜덤하게 resize
- Horizontal flip* 부분적으로 적용
- Per-pixel mean*을 빼줌
- Standard color augmentation* 적용
- Batch Normalization 적용
- Optimizer: SGD (mini-batch size: 256)
- Learning rate: 0.1 (학습 정체 시 0.1씩 늘어남)
- Weight decay*: 0.0001
- Momentum: 0.9
- 60×10^4 반복 수행
- Dropout 미사용

<Figure4. Architecture>

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

- **Horizontal flip*:** 데이터 증강 기법
- **Per-pixel mean*:** 이미지 데이터 정규화 기법
- **Standard color augmentation*:** 이미지 색상을 변형해 데이터 다양성을 높이는 데이터 증강 기법
- **Weight decay*:** 정규화 기법으로 모델의 가중치가 너무 커지지 않도록 제한해 **overfitting**을 방지

Part 3

ResNet50 구현

Part 3

ResNet50

```
def conv1(x):
    x = ZeroPadding2D(padding=(3, 3))(x)
    x = Conv2D(64, (7, 7), strides=(2, 2))(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = ZeroPadding2D(padding=(1,1))(x)
    return x

def conv2(x):
    x = MaxPooling2D((3, 3), 2)(x)

    shortcut = x

    for i in range(3):
        if (i == 0):
            x = Conv2D(64, (1, 1), strides=(1, 1), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(64, (3, 3), strides=(1, 1), padding='same')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(256, (1, 1), strides=(1, 1), padding='valid')(x)
            shortcut = Conv2D(256, (1, 1), strides=(1, 1), padding='valid')(shortcut)
            x = BatchNormalization()(x)
            shortcut = BatchNormalization()(shortcut)
            x = Add()([x, shortcut])
            x = Activation('relu')(x)
            shortcut = x
        else:
            x = Conv2D(64, (1, 1), strides=(1, 1), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(64, (3, 3), strides=(1, 1), padding='same')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(256, (1, 1), strides=(1, 1), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Add()([x, shortcut])
            x = Activation('relu')(x)
            shortcut = x
    return x
```

```
def conv3(x):
    shortcut = x

    for i in range(4):
        if(i == 0):
            x = Conv2D(128, (1, 1), strides=(2, 2), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(128, (3, 3), strides=(1, 1), padding='same')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(512, (1, 1), strides=(1, 1), padding='valid')(x)
            shortcut = Conv2D(512, (1, 1), strides=(2, 2), padding='valid')(shortcut)
            x = BatchNormalization()(x)
            shortcut = BatchNormalization()(shortcut)
            x = Add()([x, shortcut])
            x = Activation('relu')(x)
            shortcut = x
        else:
            x = Conv2D(128, (1, 1), strides=(1, 1), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(128, (3, 3), strides=(1, 1), padding='same')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(512, (1, 1), strides=(1, 1), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Add()([x, shortcut])
            x = Activation('relu')(x)
            shortcut = x
    return x
```

Part 3

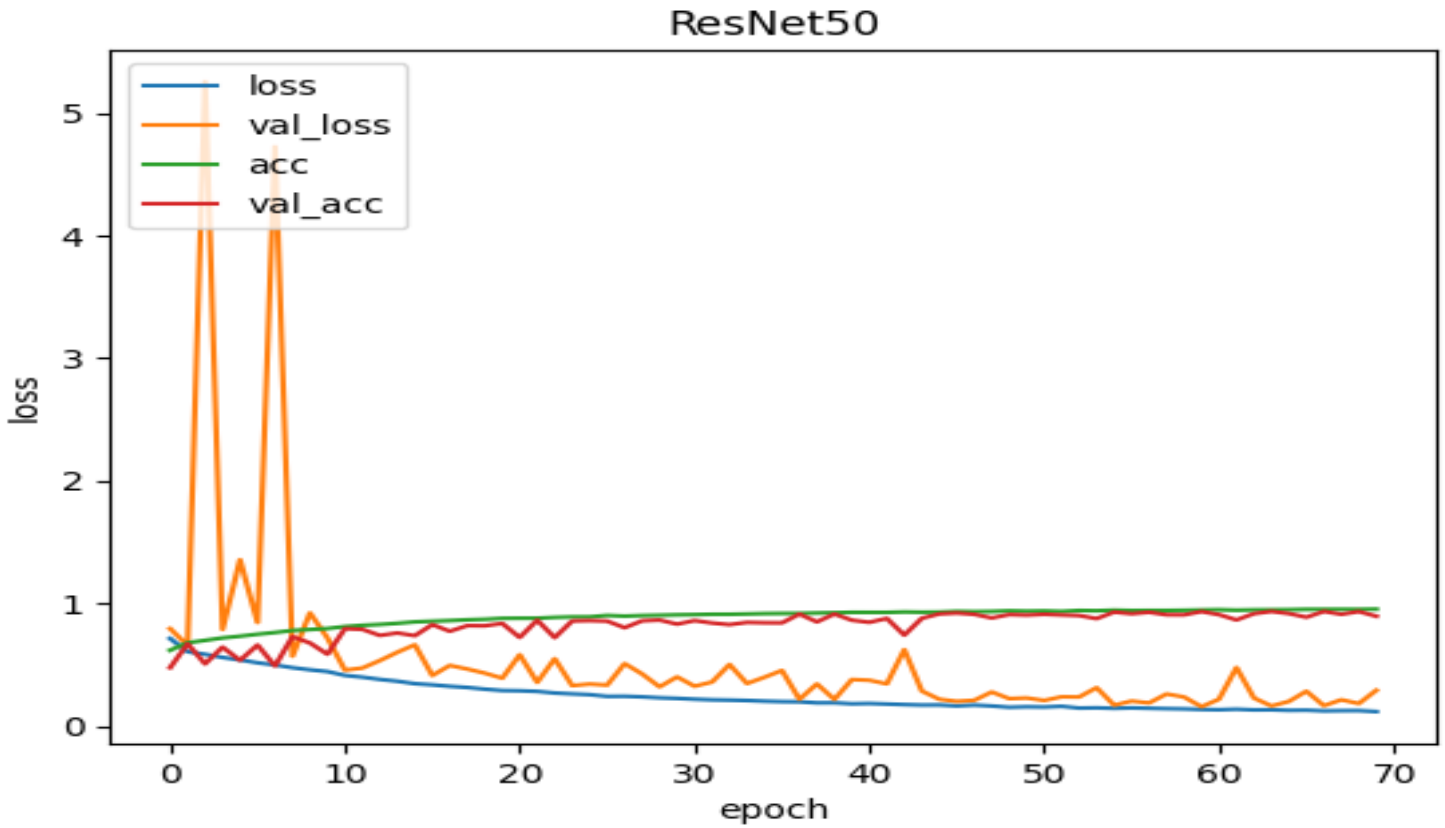
ResNet50

```
def conv4(x):
    shortcut = x

    for i in range(6):
        if(i == 0):
            x = Conv2D(256, (1, 1), strides=(2, 2), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(256, (3, 3), strides=(1, 1), padding='same')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(1024, (1, 1), strides=(1, 1), padding='valid')(x)
            shortcut = Conv2D(1024, (1, 1), strides=(2, 2), padding='valid')(shortcut)
            x = BatchNormalization()(x)
            shortcut = BatchNormalization()(shortcut)
            x = Add()([x, shortcut])
            x = Activation('relu')(x)
            shortcut = x
        else:
            x = Conv2D(256, (1, 1), strides=(1, 1), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(256, (3, 3), strides=(1, 1), padding='same')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(1024, (1, 1), strides=(1, 1), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Add()([x, shortcut])
            x = Activation('relu')(x)
            shortcut = x
    return x
```

```
def conv5(x):
    shortcut = x

    for i in range(3):
        if(i == 0):
            x = Conv2D(512, (1, 1), strides=(2, 2), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(512, (3, 3), strides=(1, 1), padding='same')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(2048, (1, 1), strides=(1, 1), padding='valid')(x)
            shortcut = Conv2D(2048, (1, 1), strides=(2, 2), padding='valid')(shortcut)
            x = BatchNormalization()(x)
            shortcut = BatchNormalization()(shortcut)
            x = Add()([x, shortcut])
            x = Activation('relu')(x)
            shortcut = x
        else:
            x = Conv2D(512, (1, 1), strides=(1, 1), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(512, (3, 3), strides=(1, 1), padding='same')(x)
            x = BatchNormalization()(x)
            x = Activation('relu')(x)
            x = Conv2D(2048, (1, 1), strides=(1, 1), padding='valid')(x)
            x = BatchNormalization()(x)
            x = Add()([x, shortcut])
            x = Activation('relu')(x)
            shortcut = x
    return x
```

Part 3

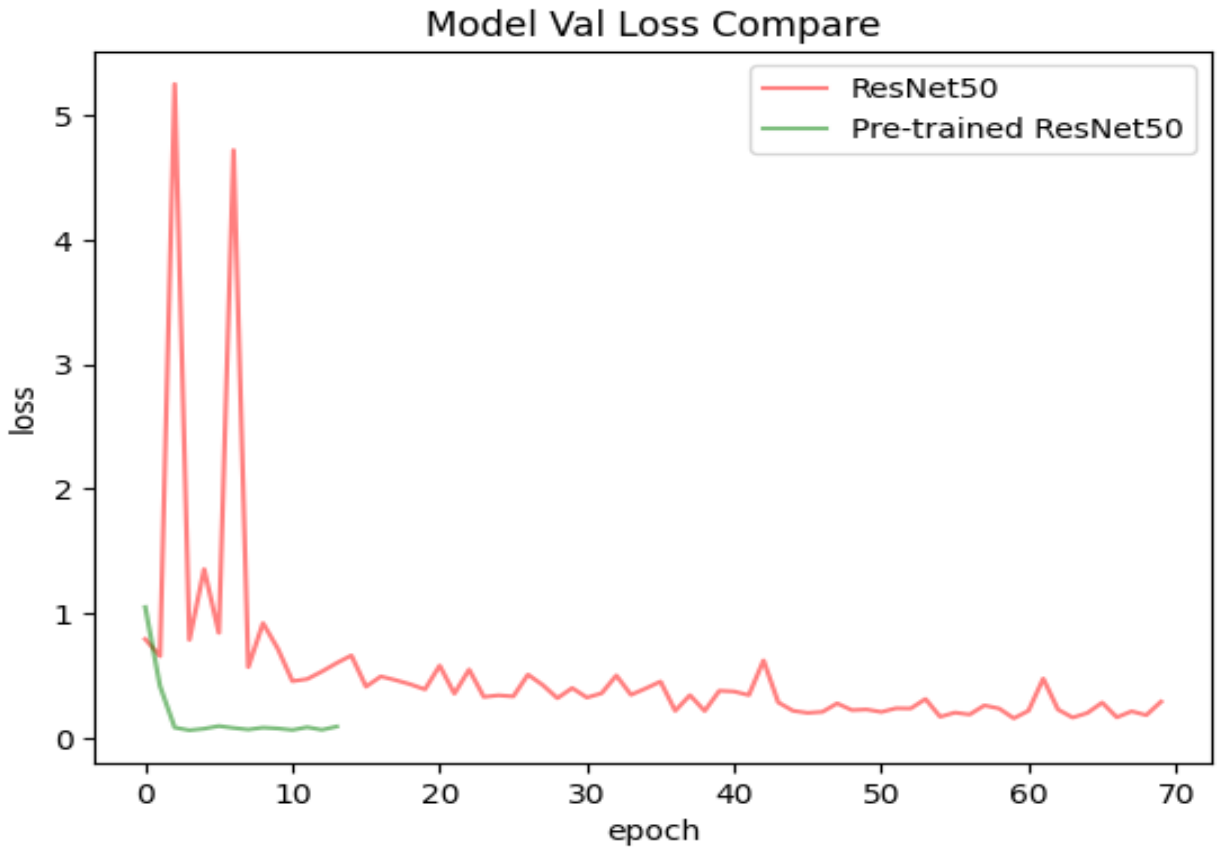
ResNet50

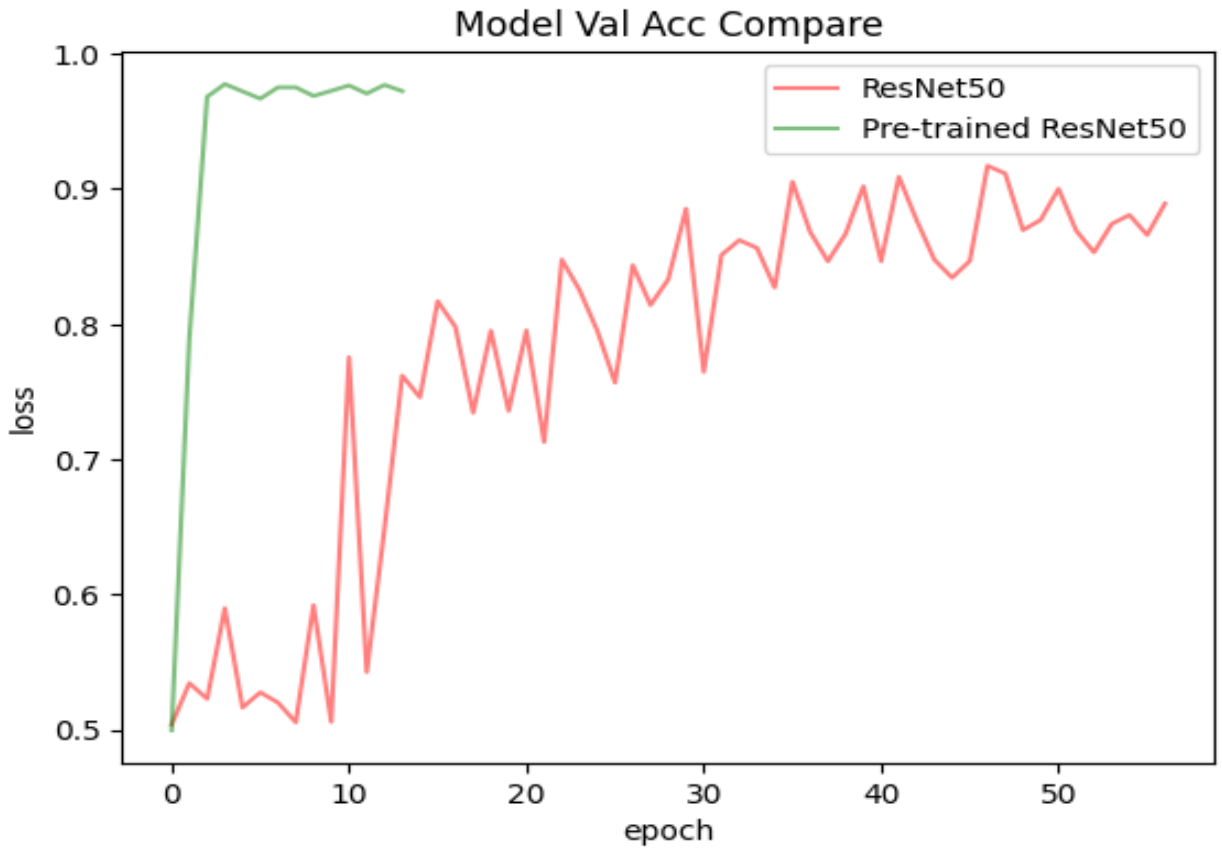


Activation: Sigmoid
Optimizer: Adam
Learning Rate: 0.001
Loss: binary_crossentropy
Epochs: 200 (69회 조기 종료)
Best Val Acc: 0.9338
Best Val Loss: 0.1565

Part 3

ResNet50





ResNet50 Best Val Loss: 0.1565
Pre-trained ResNet50 Best Val Loss: 0.061

ResNet50 Best Val Acc: 0.9338
Pre-trained ResNet50 Best Val Acc: 0.9776