



IMAGENET PREDICTION

8조

조장 : 박성우
민예린 박정민
전수연 최경준

목차

CONTENTS

01

IMAGENET
DATASET
설명

02

DATA 전처리
및 환경설정

03

모델 구축
및 비교

04

XAI
및 마무리

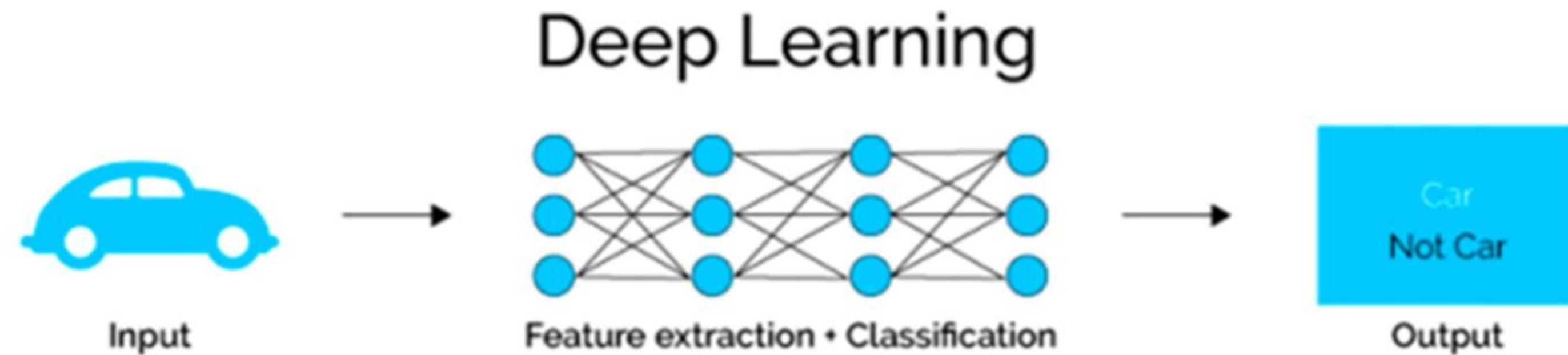
00 PROJECT FORWARD

프로젝트 방향

PROJECT 목표

1000개의 class에 대한 이미지 분류

좋은 input 데이터를 사용하여, 이 데이터에 맞는 좋은 모델을 사용하면 좋은 output이 나올 것이라고 기대



01

IMAGENET DATASET

01

IMAGENET DATASET?

IMAGENET DATASET 설명

IMAGENET DATASET

1000개의 class로 구성된 이미지 데이터로 사람이 직접 labeling 작업을 함



01

IMAGENET DATASET?

IMAGENET DATASET 설명

IMAGENET DATASET

주어진 데이터 - 무작위로 위치한 이미지들



01

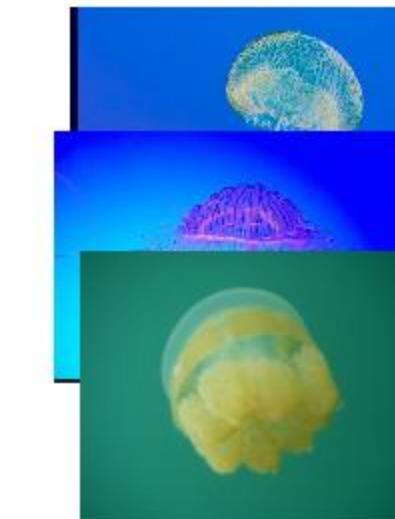
IMAGENET DATASET?

IMAGENET DATASET 설명

IMAGENET DATASET

각 클래스 별로 폴더 생성 - 각 클래스 당 50개의 이미지

50개의 이미지



감자튀김

해파리

...

삼고양이

야구공

1000개의 CLASS

02

DATA PROCESSING & SETTING

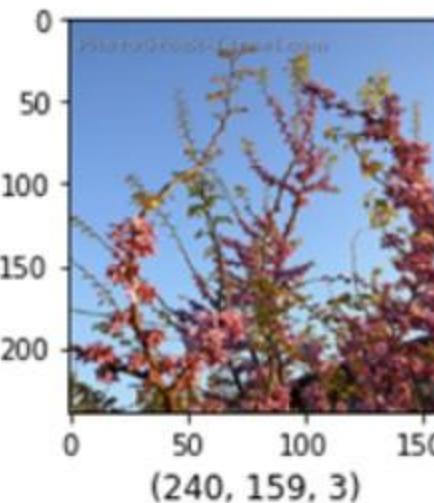
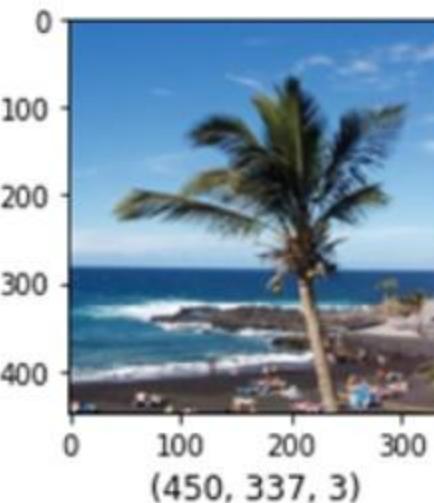
02

DATA PROCESSING & SETTING

데이터 전처리

01

이미지 SIZE

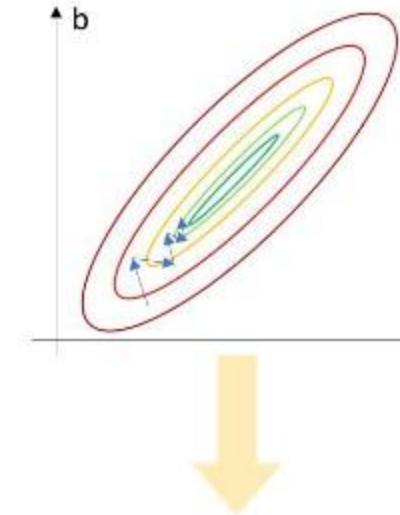


=> 이미지 RESIZE 필요

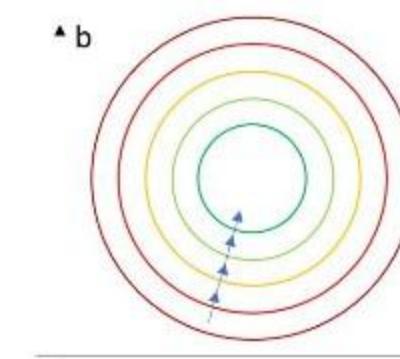
02

NORMALIZE

Unnormalized:



Normalized:



=> RGB 데이터에 대해
train, valid, test에 대해
정규화 필요

03

이미지 증강



=> 과적합 방지를 위해 이미지 데이터 증강
(왜곡, 회전, 확대 등)

02 DATA PROCESSING & SETTING

데이터 전처리



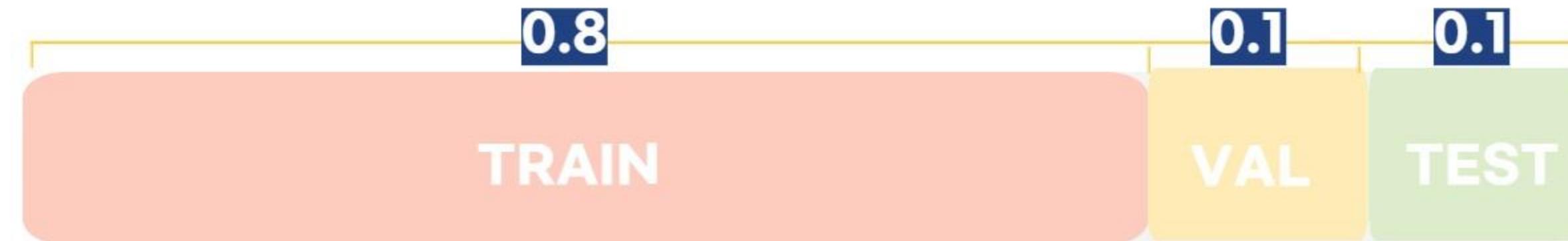
원본 IMAGE

PROCESSING



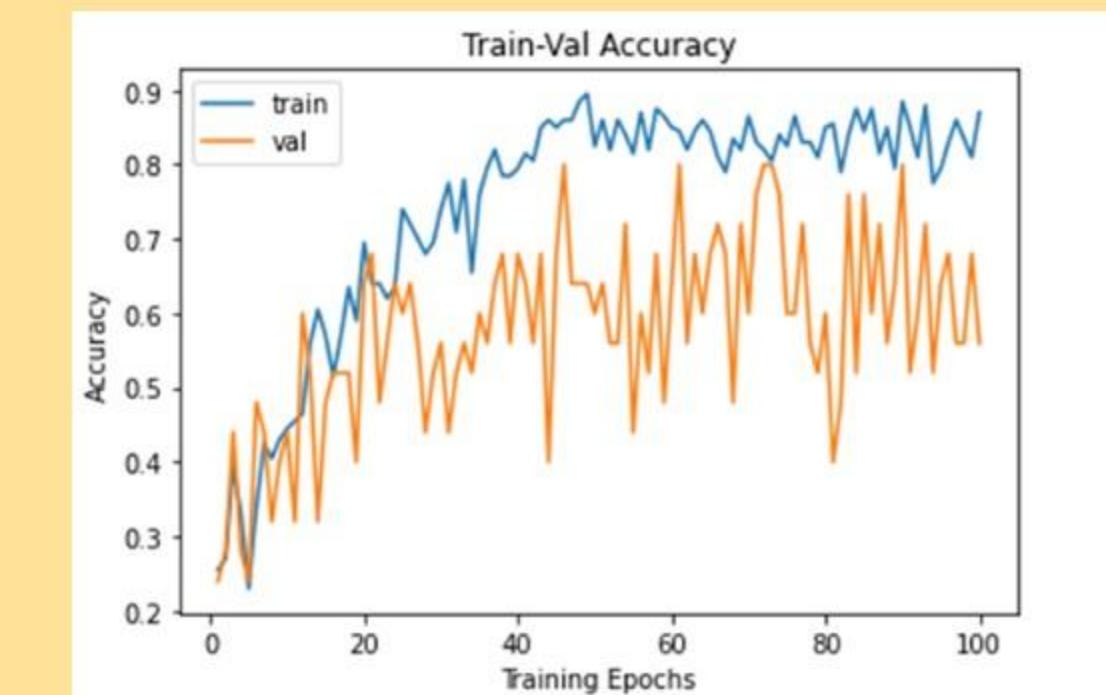
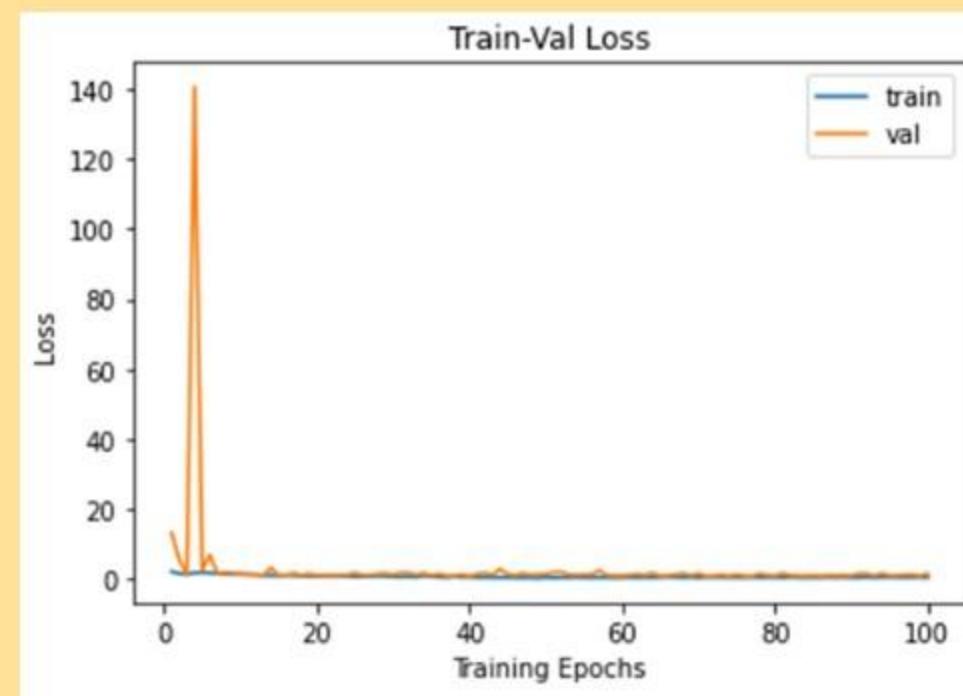
02 DATA PROCESSING & SETTING

환경 구축



초기 모델 생성

5개의 CLASS에 대해 모델 구축



=> 좋은 INPUT이 아니라고 생각

입력 데이터의 문제점

1. 너무 많은 class의 수

해결. 1000개의 class가 아닌 10개의 class에 대해 분류
(저장공간 부족과 성능 저하로 인하여)

2. class 당 너무 적은 데이터의 양(50개)

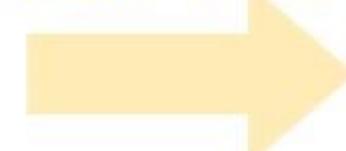
해결1. 이미지 증강하여 인위적으로 데이터 증식
해결2. 데이터 양 보충

해결1. 이미지 증강하여 인위적으로 데이터 증식

-> test를 제외한, train과 valid 데이터를 이미지 증강을 이용해 데이터 증식하여 개수 늘림



이미지 증강



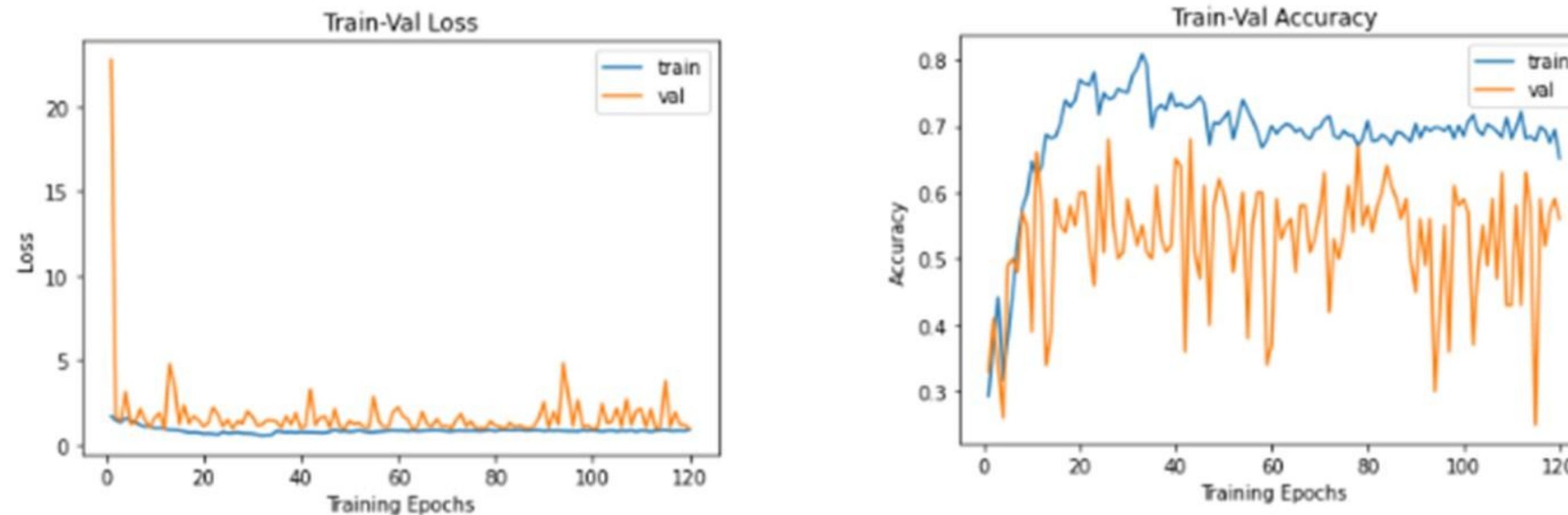
(왜곡, 확대, 늘림 등)

train : 500 → 1189

valid : 25 → 100



해결1. 결과



=> valid 데이터에 대한 정확도의 편차가 커, 좋은 방법이 아닐 수 있다고 생각

해결2. 데이터 양 보충

- => 이미지 넷의 실제 train 데이터를 불러옴
- => 한 class 당 약 9000~17000개의 이미지를 가짐



-> 충분한 양의 데이터 + 이상치 데이터 발견X

03

MODEL 구축

03 MODEL

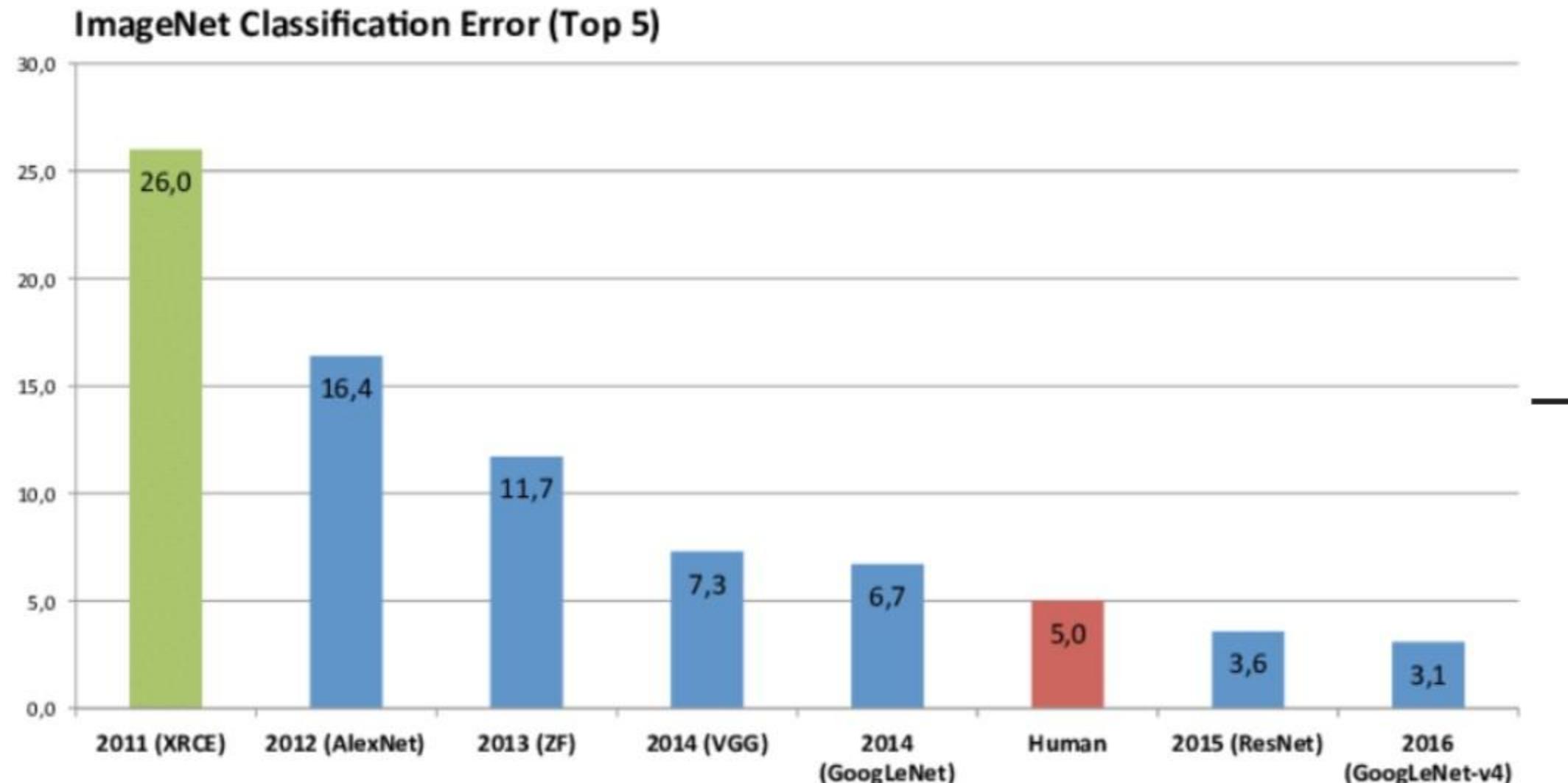
ALEXNET

VGGNET

GOOGLENET

RESNET

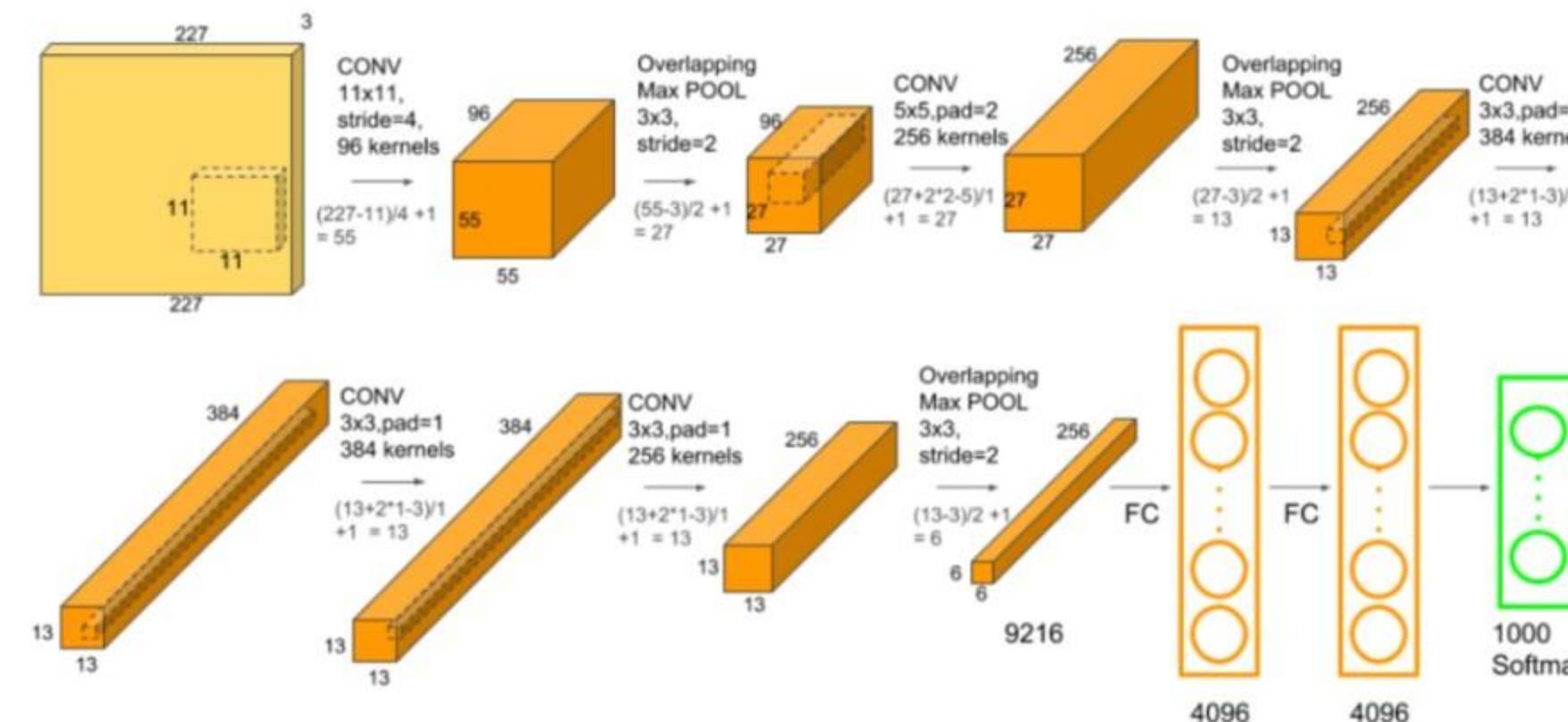
IMAGENET 분류 연도별 TOP-5 ERROR



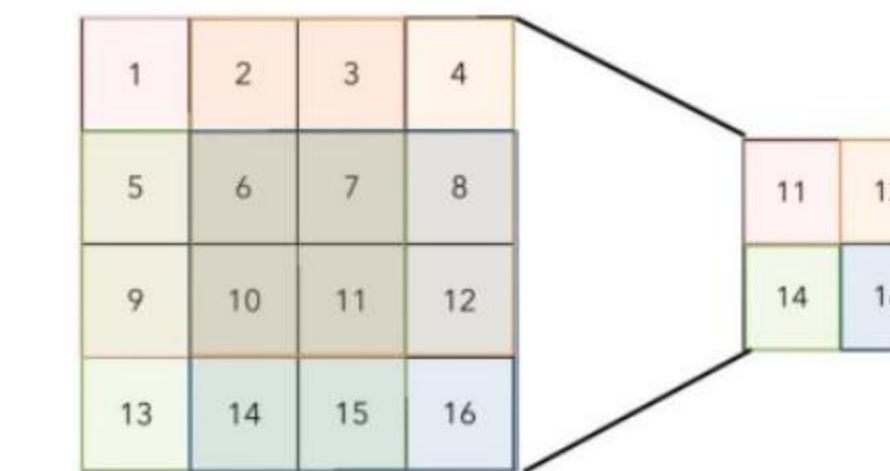
EFFICIENTNET
MOBILENET

ALEXNET

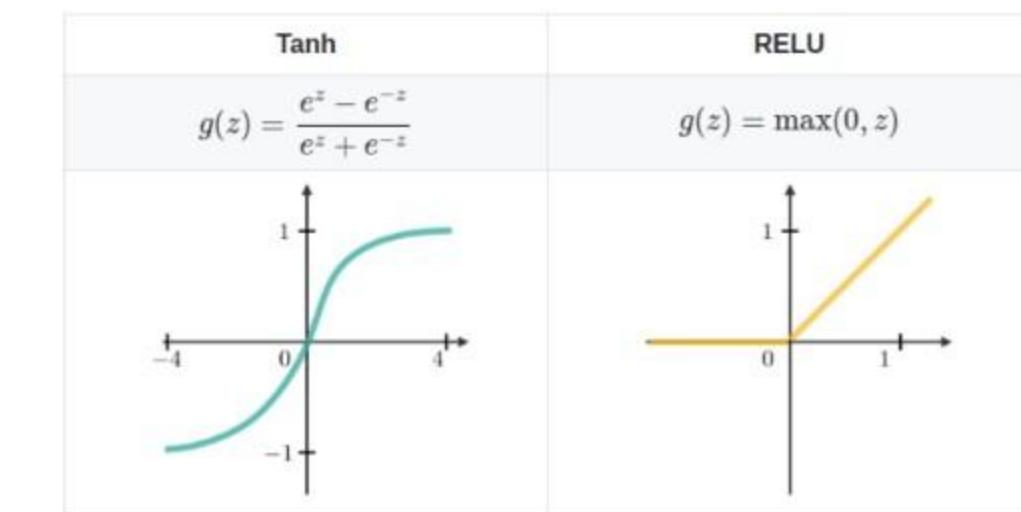
ILSVRC 의 2012년 대회에서 1위를 차지한 모델



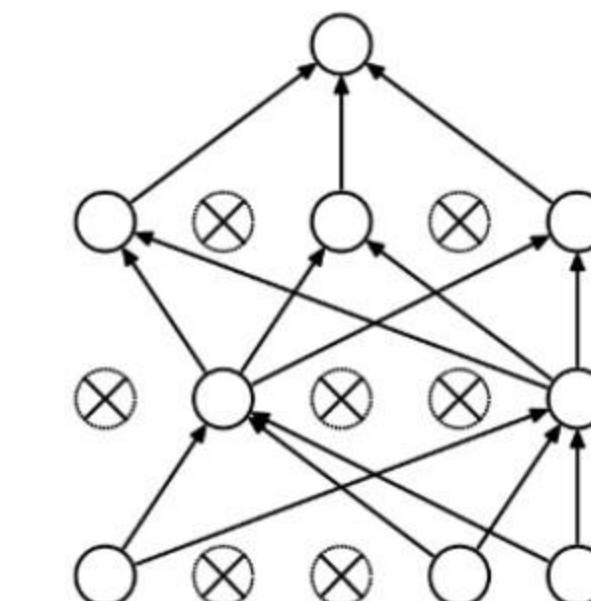
ALEXNET ARCHITECTURE



OVERLAPPING POOLING



ACTIVATION FUNCTION



DROPOUT

03 MODEL

ALEXNET 모델 구현 과정

01

ALEXNET 모델 정의 및 PARAMETER 설정

8개의 LAYERS

5개의 CONVOLUTION LAYER

+ 3개의 FULL-CONNECTED LAYER

INPUT SIZE = 227x227x3

NORMALIZE DATA

epochs = 90

lrate = 0.01

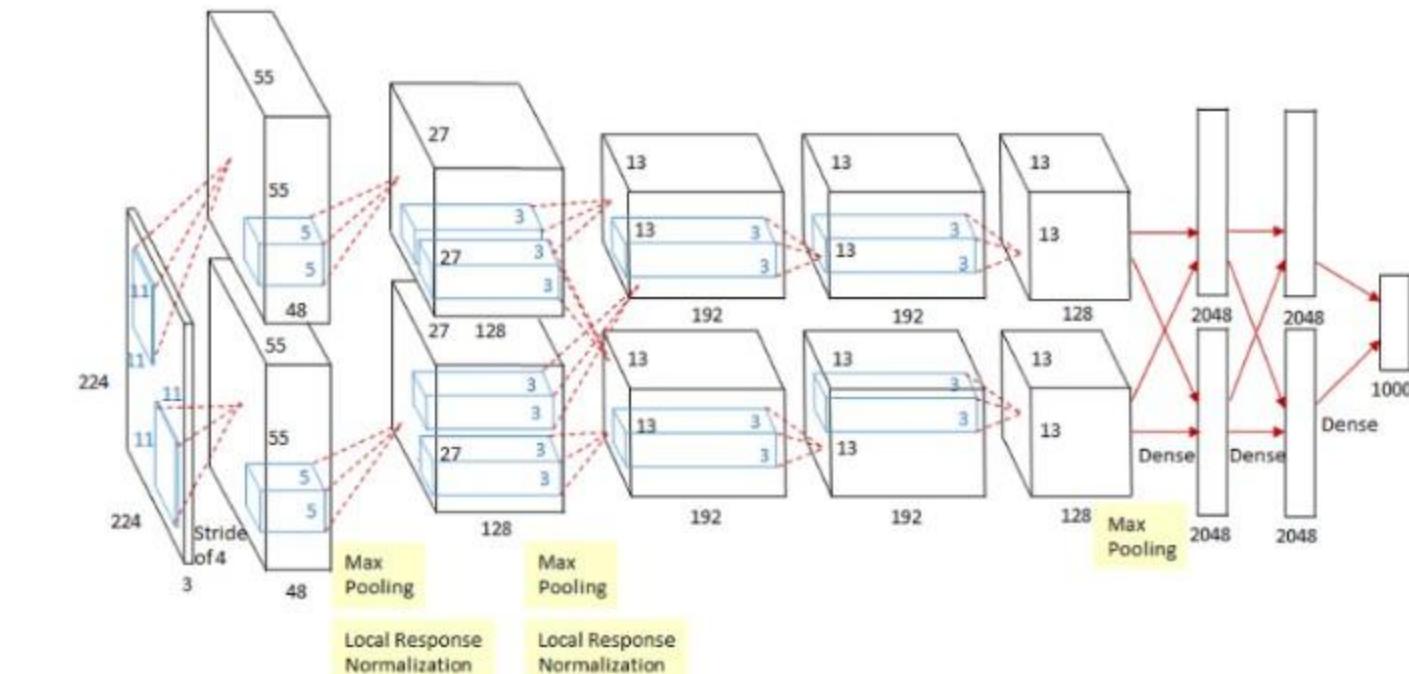
decay = 0.0005

batch_size = 128

optimizer = Adam

learning rate = 0.001

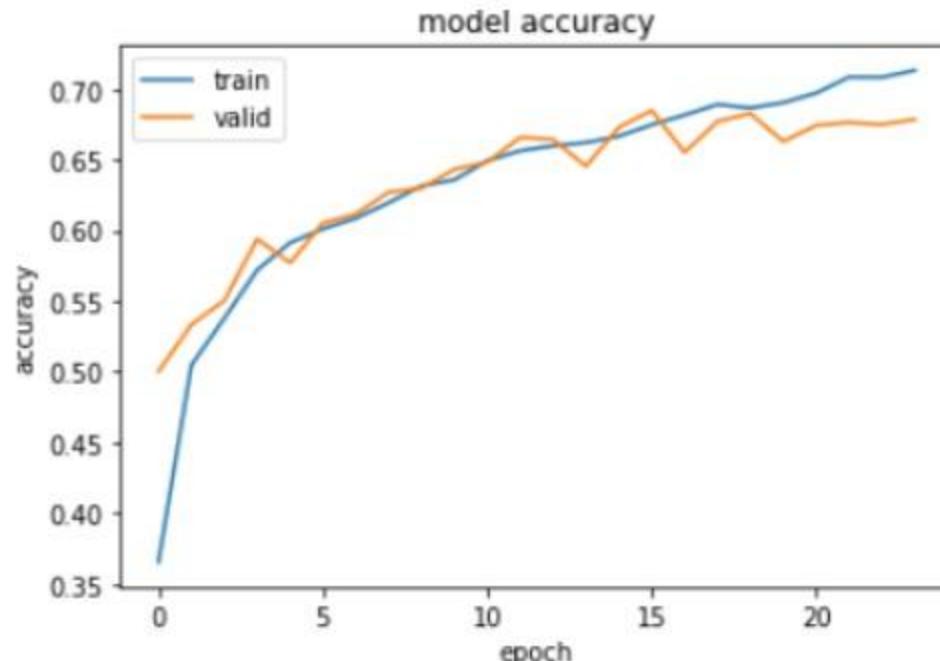
loss= 'categorical_
crossentropy'



AlexNet

03 MODEL

ALEXNET

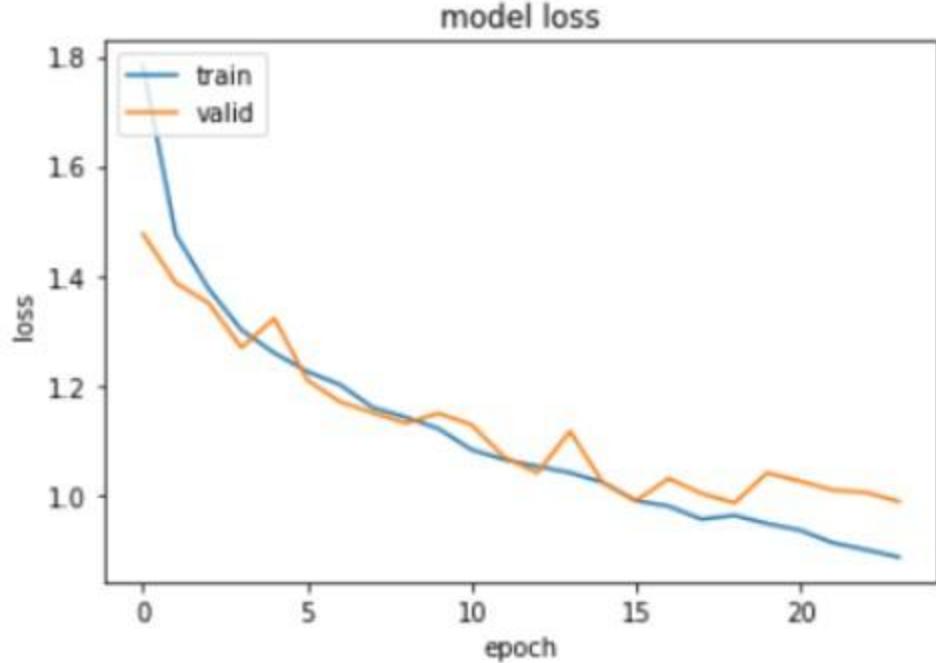


- Valid Accuracy

```
▶ print('Accuracy Score = ',np.max(history.history['val_accuracy']))  
⇒ Accuracy Score =  0.6847290396690369
```

- macro f1-score

```
[ ] import sklearn  
from sklearn.metrics import f1_score  
  
sklearn.metrics.f1_score(y_test1, pred_test, average='macro')  
  
0.45739020975166534
```



- Top2 Accuracy

```
[ ] # top2 정확도  
n = 2  
top_n_pre = np.argsort(pred, axis=1)[:,-n:]  
correct = 0  
acc = []  
for i,data in enumerate(top_n_pre):  
    corr = y_test1[i] in data  
    correct += corr  
  
correct/len(pred)  
  
0.604
```

결론

plot : 우상향하는 accuracy 그래프와
우하향하는 loss 그래프를 보면 모델이 비
교적 fitting이 잘 됐음을 볼 수 있음

accuracy : validation accuracy로
다 evaluate를 통해 모델을 평가한 acc
uracy가 훨씬 낮아 overfitting의 가능
성이 높다.

03 MODEL

VGGNET

ILSVRC 2014에서 준우승을 차지한 모델

19 layers

input size : 224X224X3

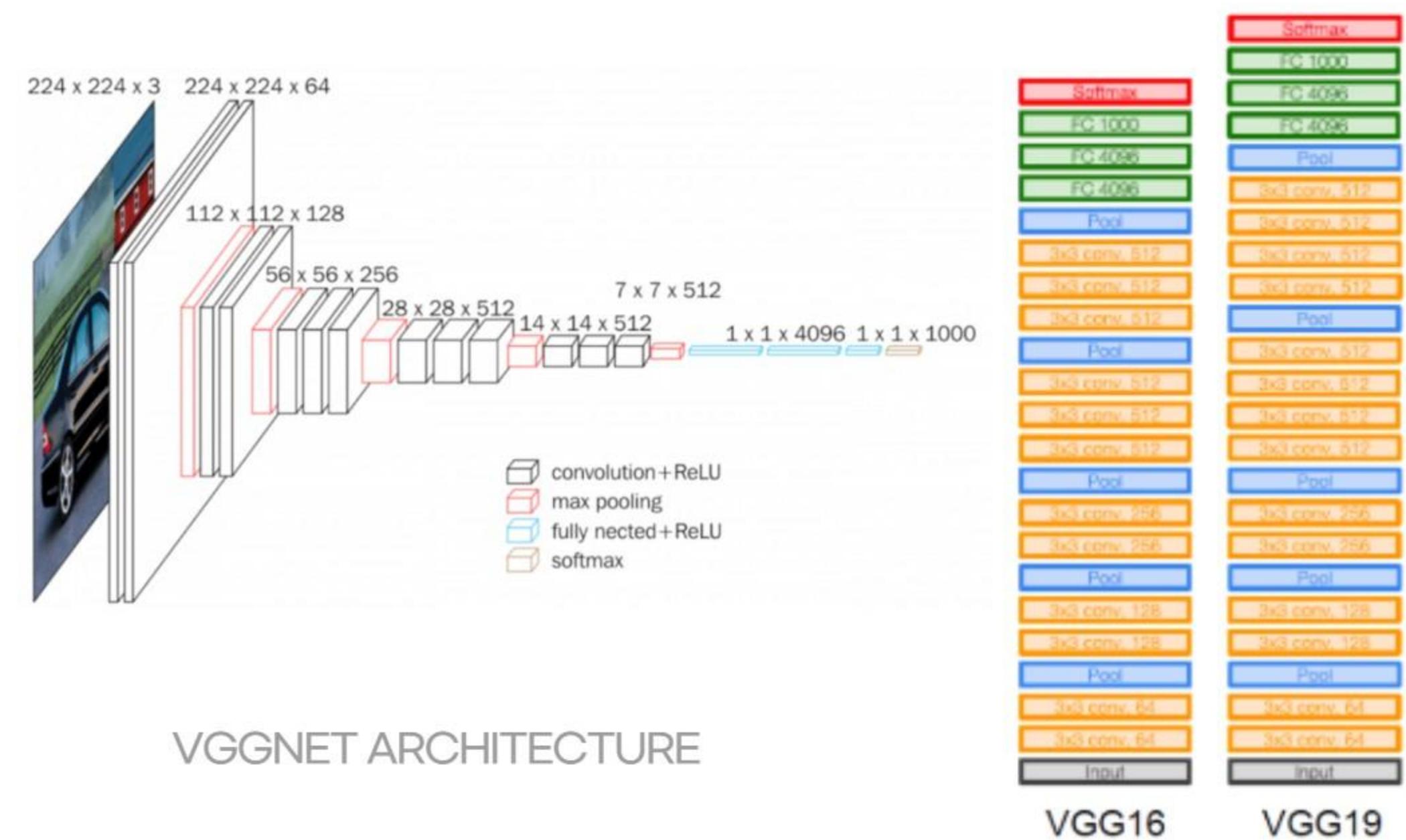
=> Alexnet 과 달리 Convolution

layer, Pooling, Fully-Connecte

d layer로만 이루어짐 (단순한 구조)

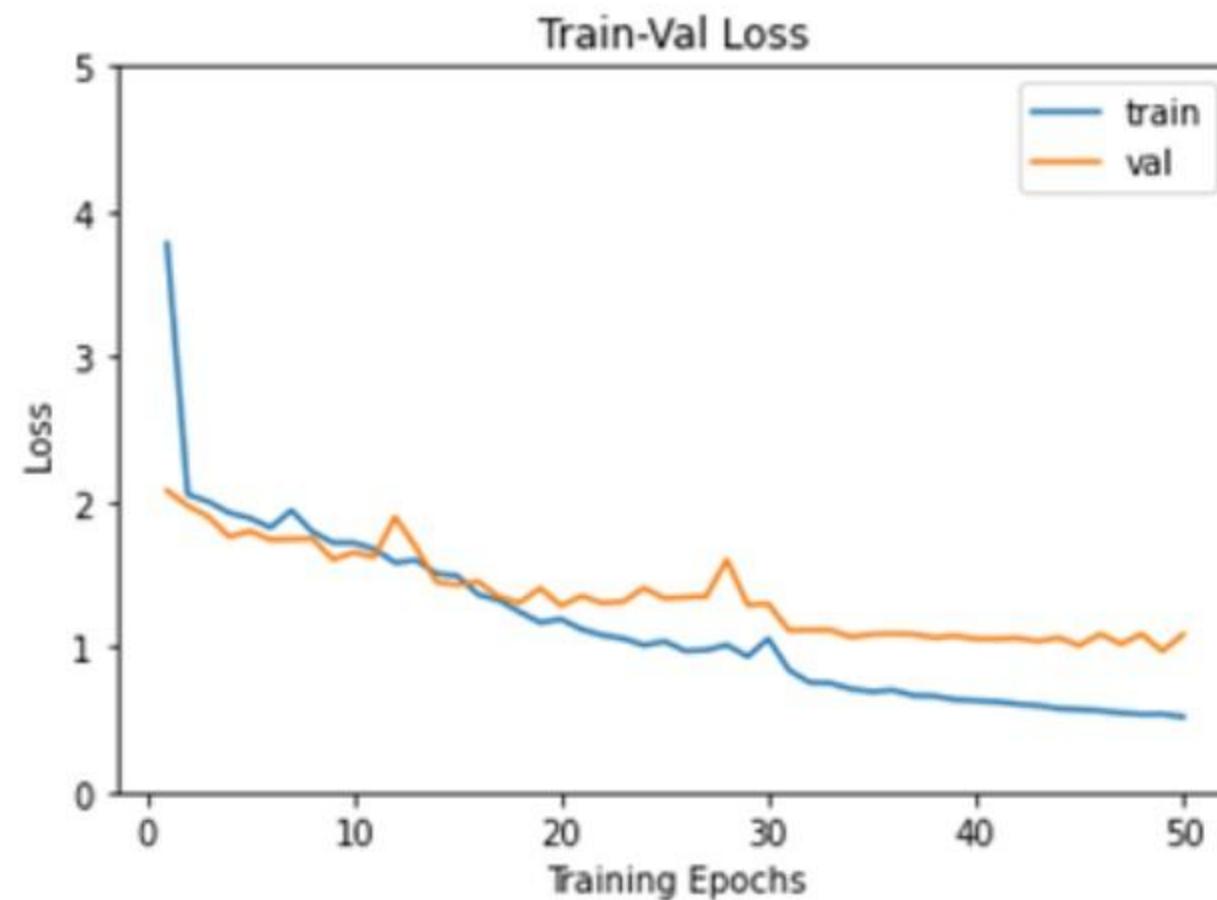
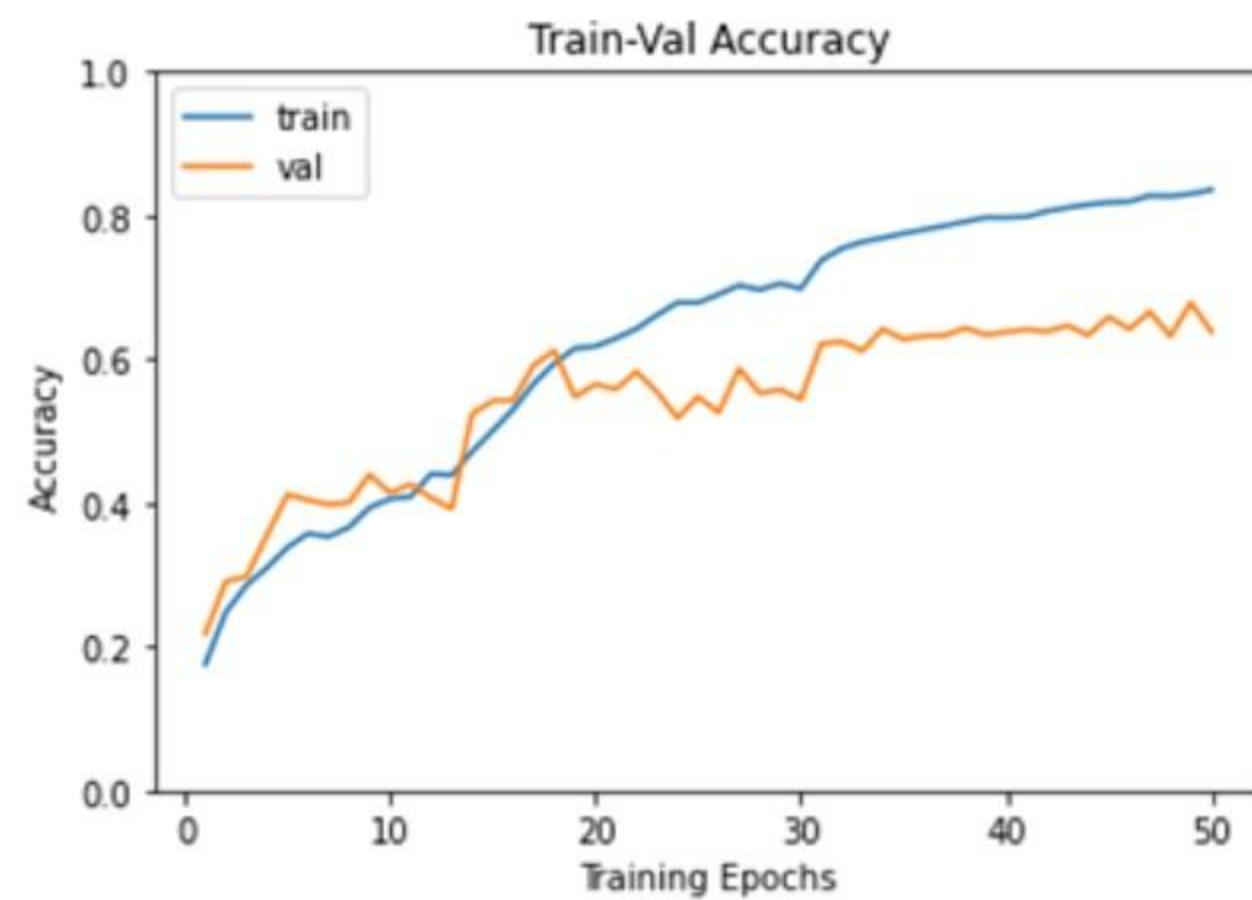
=> 깊이가 매우 깊다, 학습이 매우 느

리다, 용량이 엄청나게 크다



03 MODEL

VGGNET

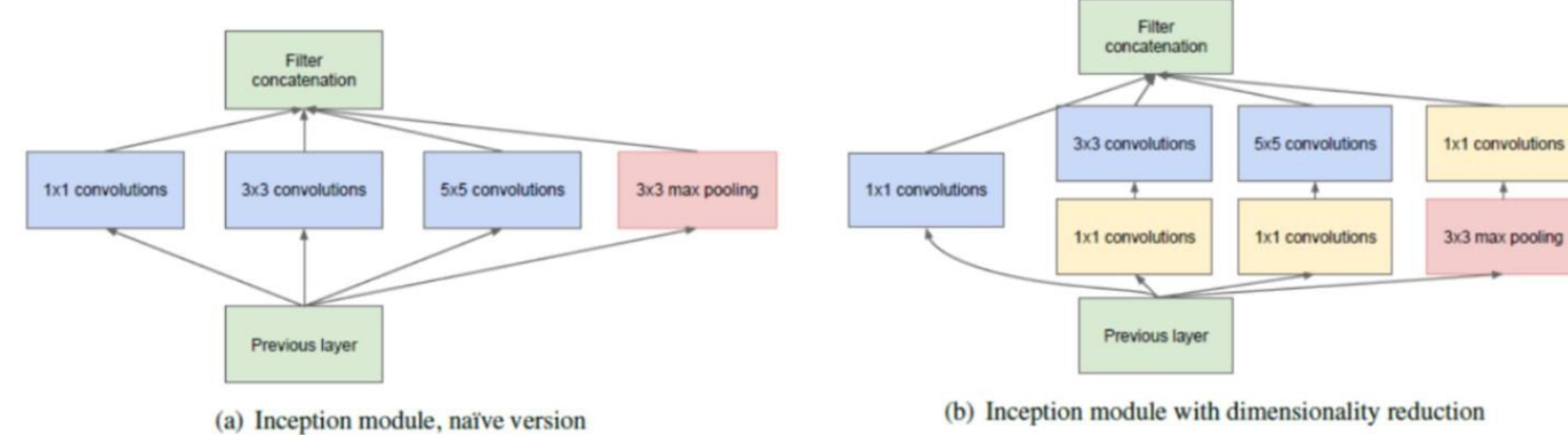


결론

오른쪽으로 갈수록 train data와 valid data의 accuracy의 차이가 커짐
epoch=25로 하였기 때문에 epoch을 늘려 overfitting 해결 가능

GOOGLENET

ILSVRC 2014에서 VGG를 제치고 우승을 차지한 모델



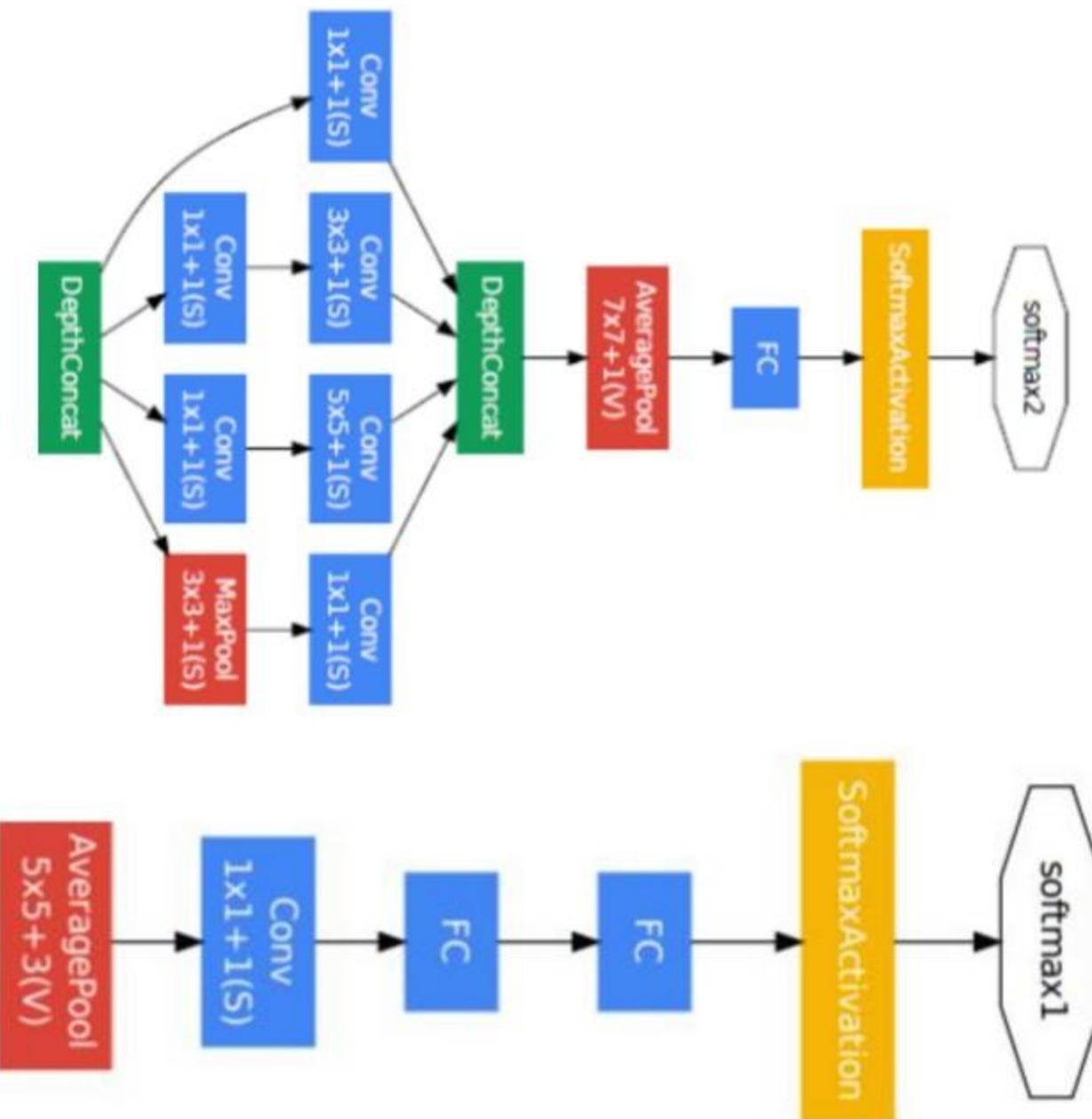
출처: 논문

Inception Module

한번에 한종류의 필터만 사용하는 CNN과는 다르게 구글넷은 한번에 (1,1), (3,3), (5,5)의 크기가 다른 세가지 필터를 병렬로 사용

03 MODEL

GOOGLENET



평균 풀링

=> Googlenet에서는 완전 연결 층 대신에 평균 풀링을 사용하여
피처맵을 벡터화 하여 소프트맥스 처리

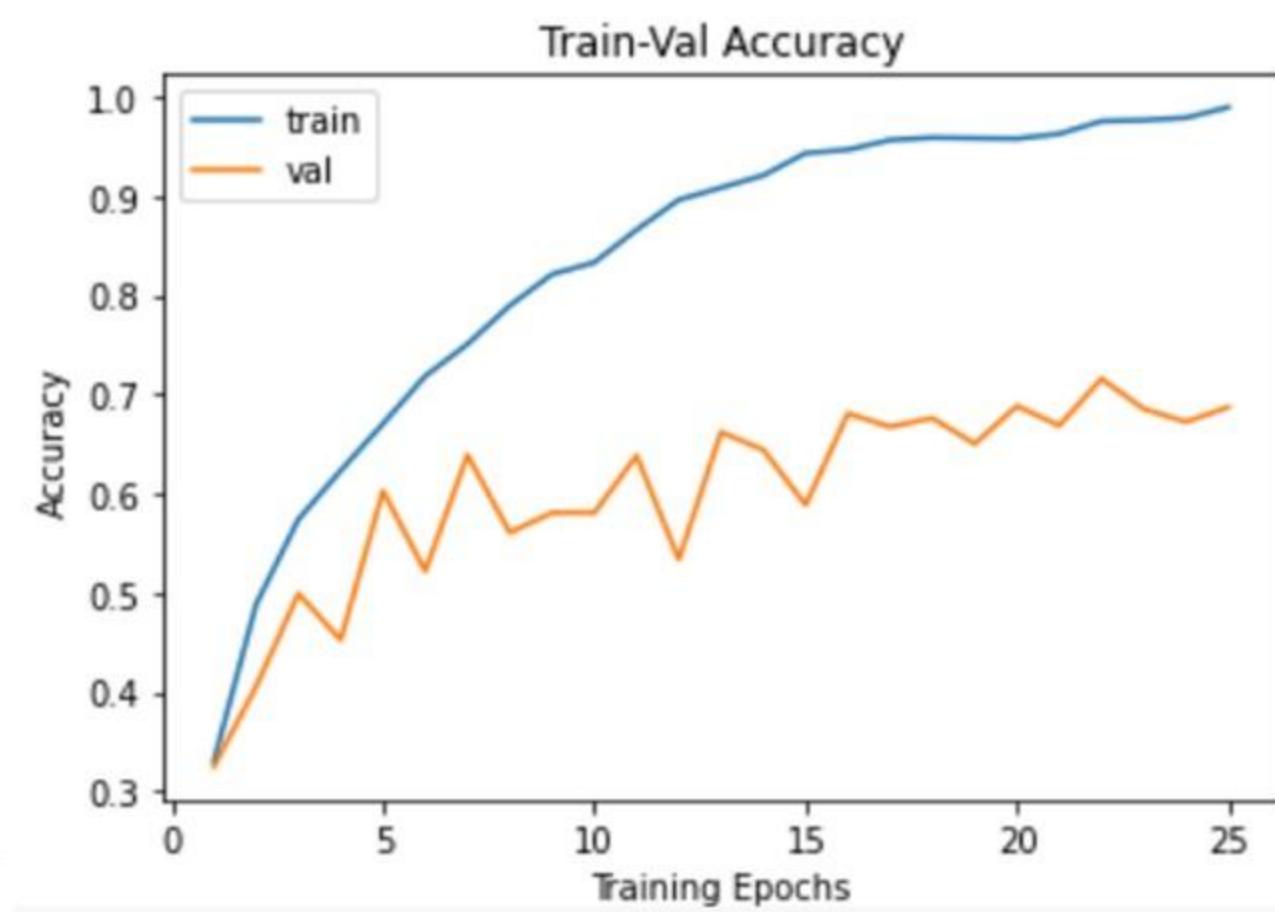
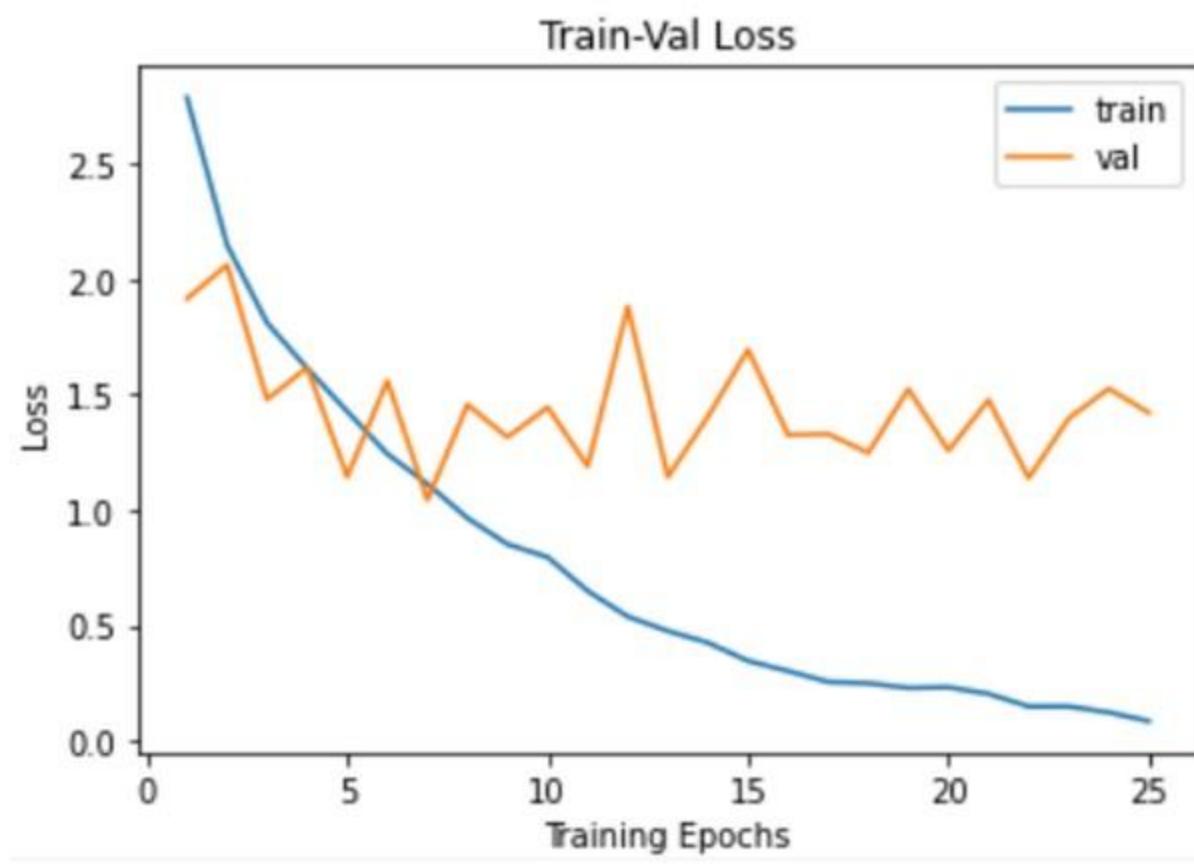
보조 분류기

=> Googlenet의 보조 분류기(auxiliary classifier)는 중간
에 softmax를 두어 중간에서도 backpropagation 가능

03 MODEL

GOOGLENET

[예측 그래프]



```
1 lr = 1e-5
2 epochs = np.arange(1,26)
3 criterion = torch.nn.CrossEntropyLoss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=lr)
5 batch_size = 16
6 total_batch = len(train_dl)
```

RESNET

Residual Block의 구조

RESNET은 기본적으로 VGG-19의 구조를 빼대

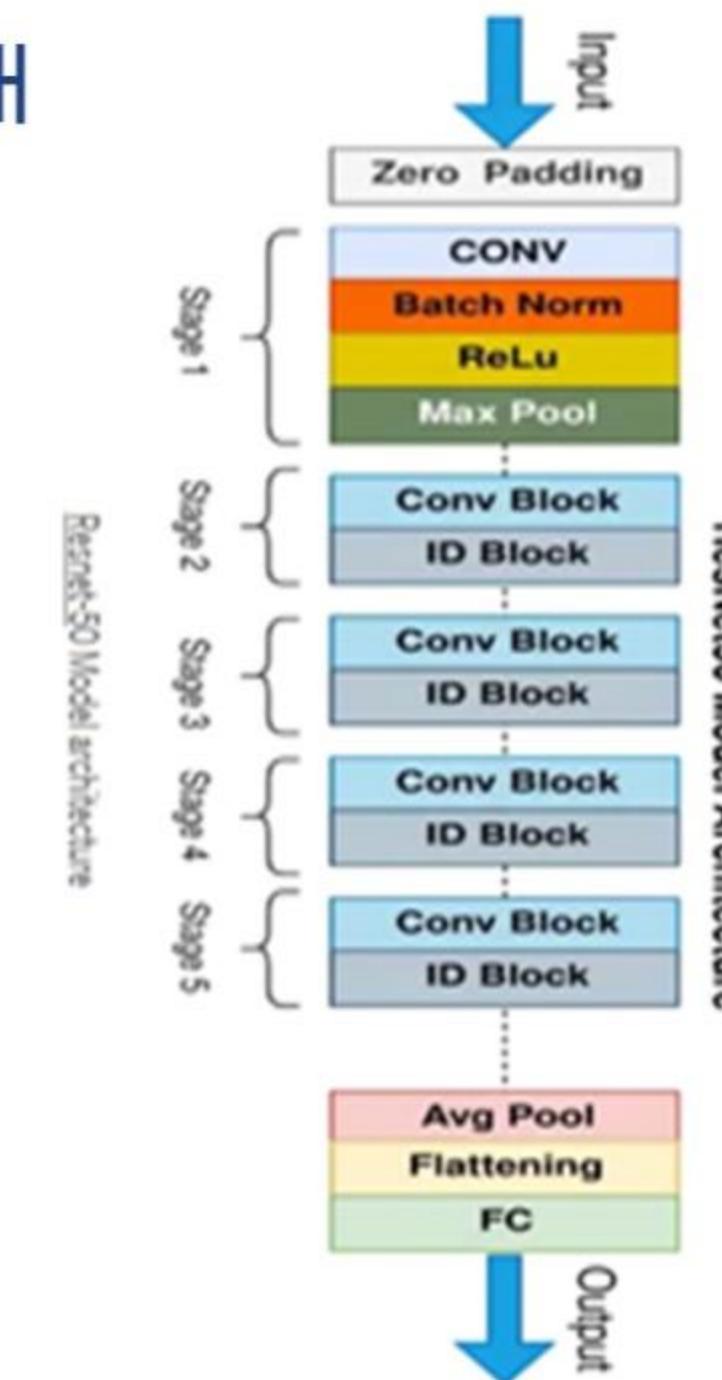
망이 깊어지면 기울기 소실/폭주로 인해 depth가 증
가할지라도 어느정도 선에 다다르면 성능이 떨어짐.

→ Vanishing gradient 문제 해결

RESIDUAL BLOCK

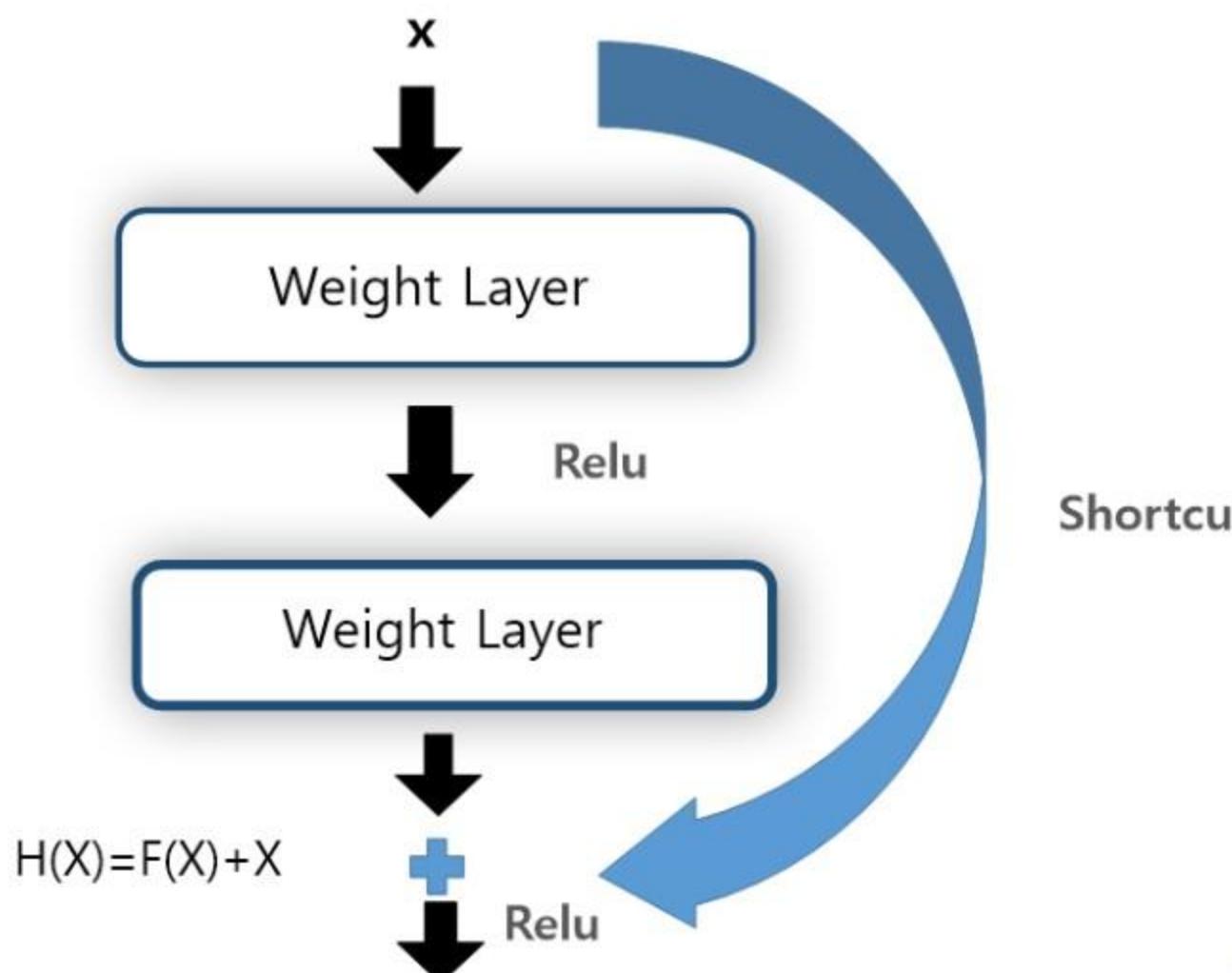
전체를 학습하는 것보다 학습이 오히려 쉬워지고, 수렴
을 더 잘할 수 있게 된다.

층이 깊어져도 gradient가 잘 흐를 수 있도록 일종의 지름길



RESNET

기울기 소실(GRADIENT VANISHING)
문제 해결



BOTTLENECK CLASS

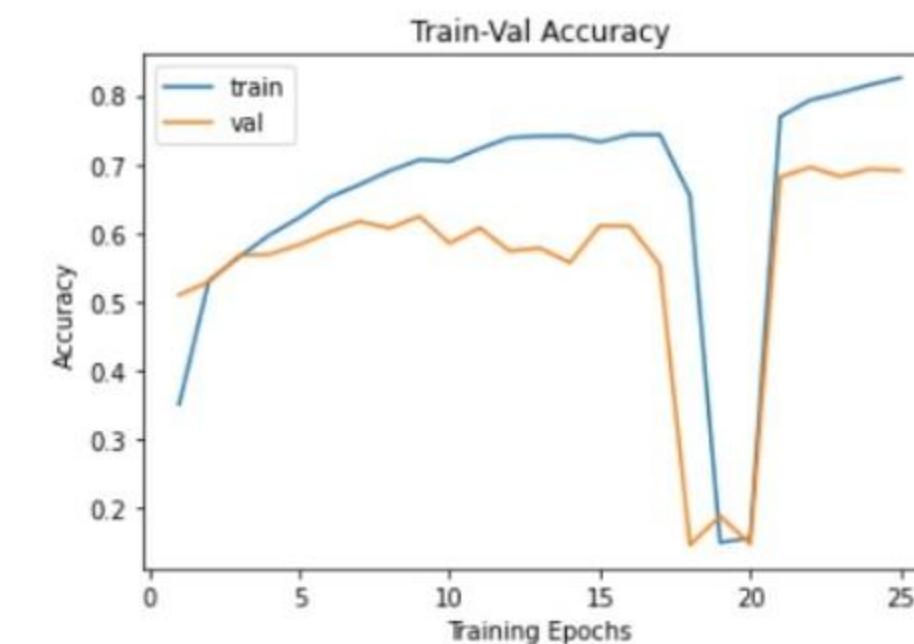
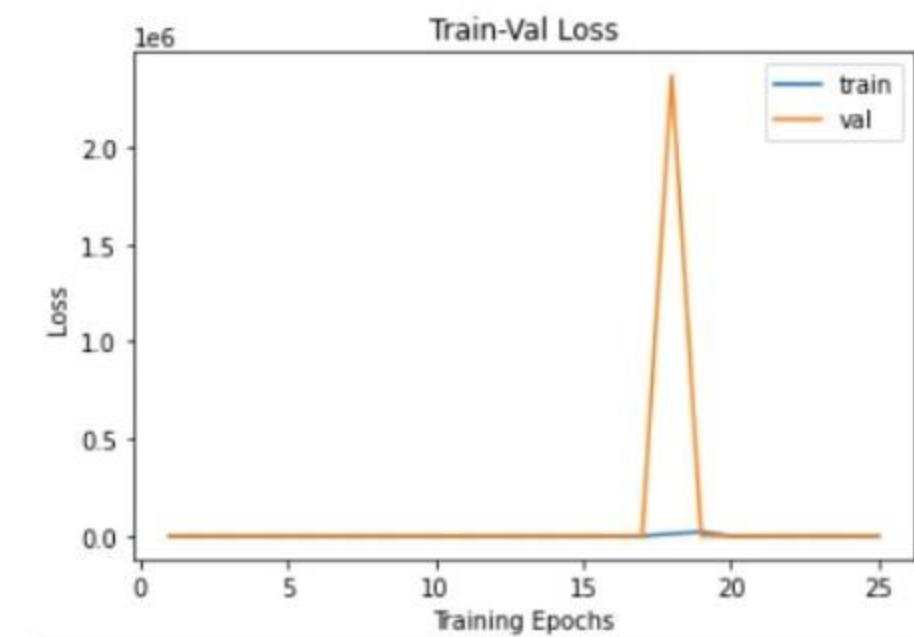
```
class Bottleneck(nn.Module):
    expansion = 4
    Cardinality = 32 # group 수
    Basewidth = 64 # bottleneck 채널이 64이면 group convolution의 채널은 depth가 됩니다.
    Depth = 4 # basewidth일 때, group convolution의 채널 수
    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()
        C = Bottleneck.Cardinality
        D = int(Bottleneck.Depth * out_channels / Bottleneck.Basewidth)

        self.conv_residual = nn.Sequential(
            nn.Conv2d(in_channels, C * D, 1, stride=1, padding=0, bias=False),
            nn.BatchNorm2d(C*D),
            nn.ReLU(),
            nn.Conv2d(C*D, C*D, 3, stride=stride, padding=1, groups=Bottleneck.Cardinality, bias=False),
            nn.BatchNorm2d(C*D),
            nn.ReLU(),
            nn.Conv2d(C*D, out_channels * Bottleneck.expansion, 1, stride=1, padding=0, bias=False),
            nn.BatchNorm2d(out_channels * Bottleneck.expansion)
        )
```

03 MODEL

RESNET 50

- => 50개 계층으로 구성된 컨벌루션 신경망
- => 훈련된 신경망의 사전 훈련된 버전을 불러옴
- => 기존의 VGG 네트워크보다 더 깊음
- => residual block을 활용해 복잡도와 성능은 더 개선구현이 간단
- => 학습 난이도가 매우 낮아짐
- => 깊이가 깊어질수록 높은 정확도 향상

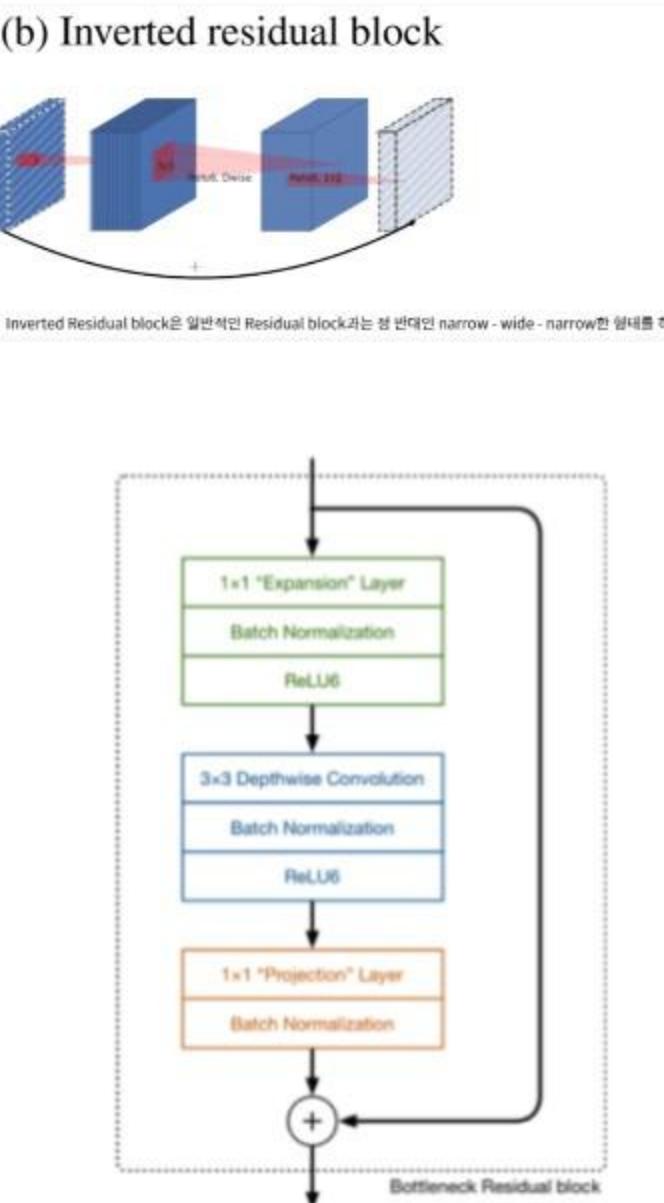


03 MODEL

MOBILENET VS 다른 MODEL

특징 1: 다른 모델보다 더 적은 매개변수를 사용하여 깊이별 분리 가능 컨볼루션을 사용

특징 2: 다른 모델보다 크기와 계싼 비용 (속도)를 줄이기 위해 선형 병목 현상 및 배치 정규화 기술 사용



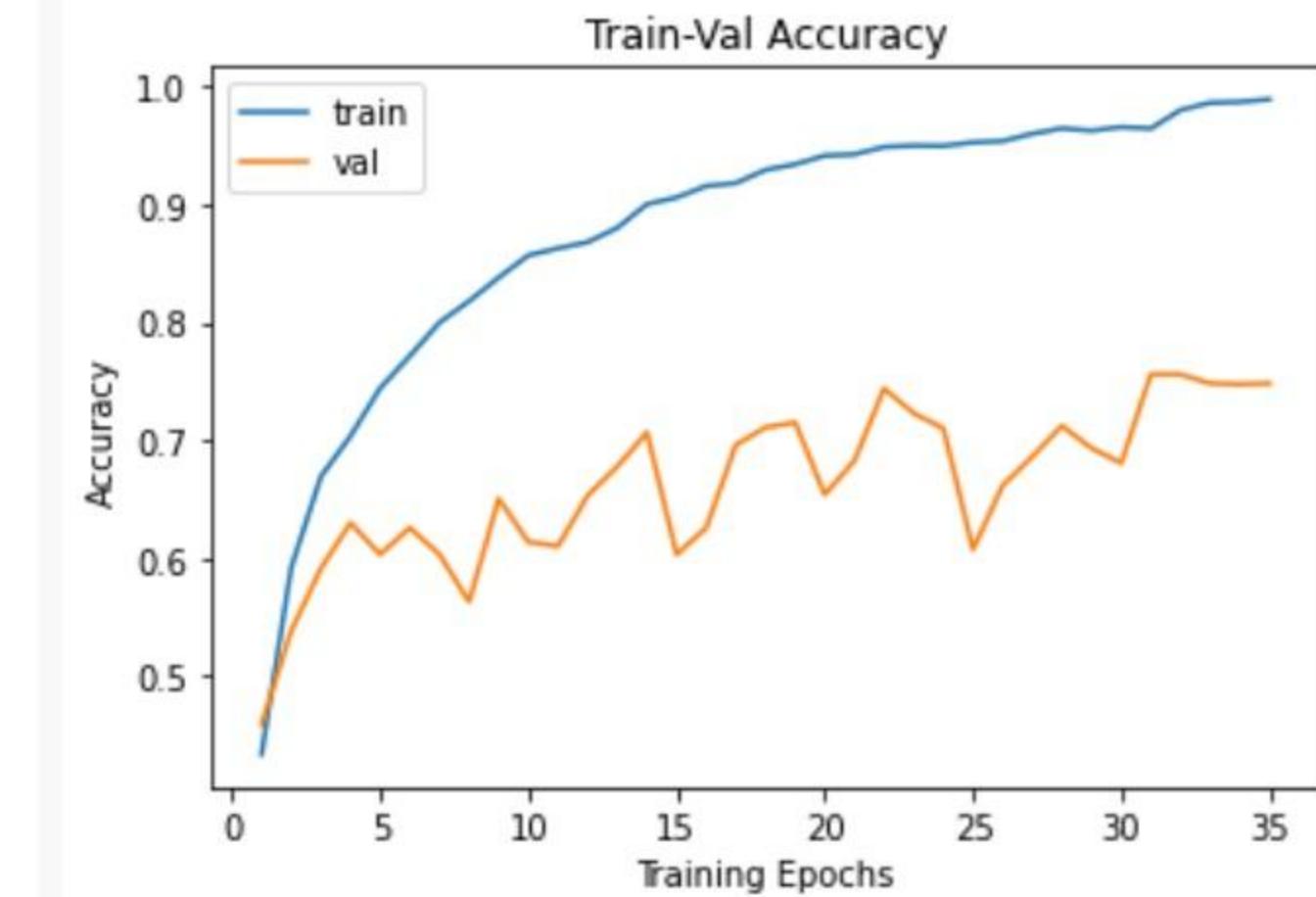
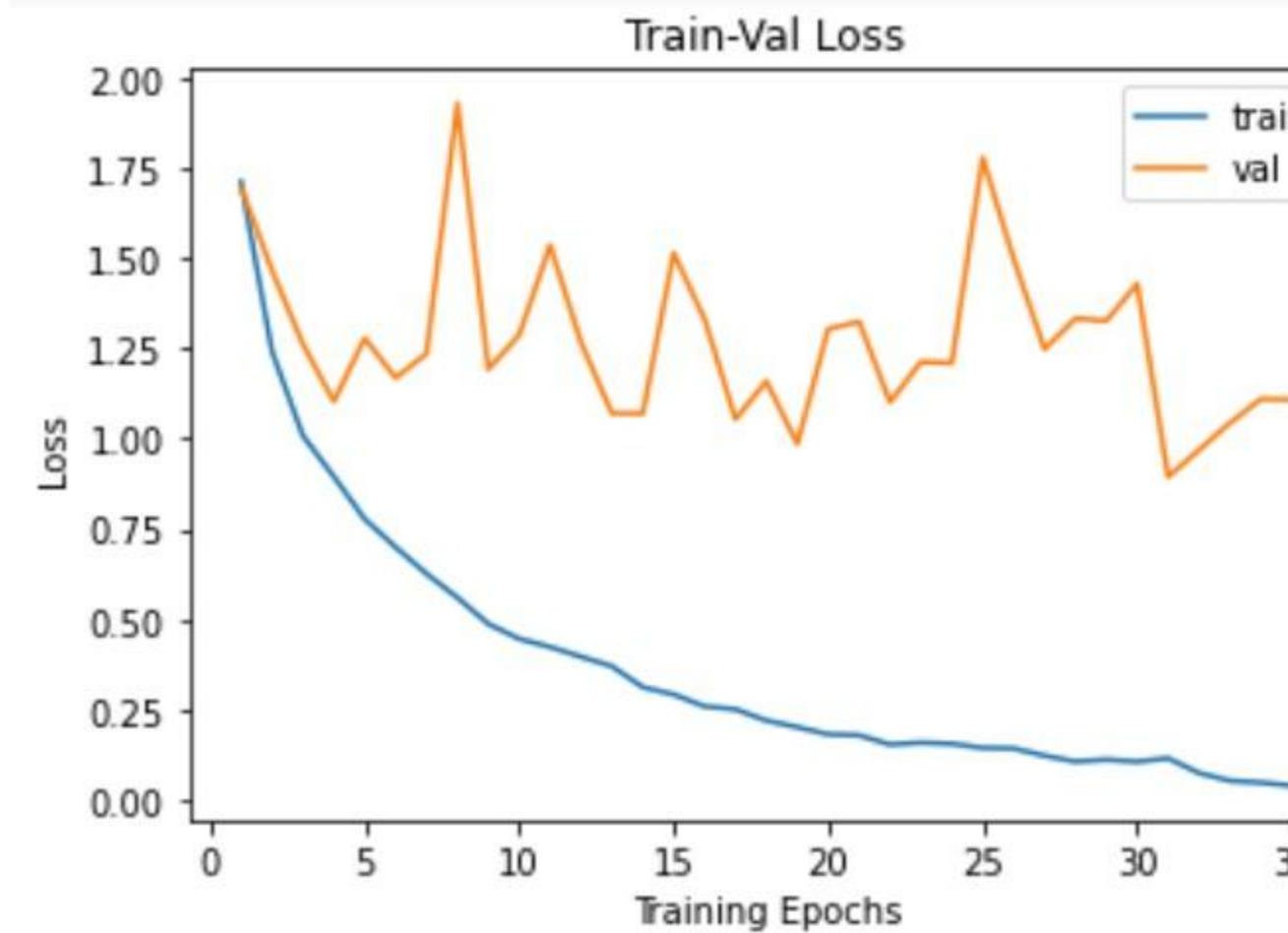
MobileNet-V1

2017년 Google에서 개발한 MobileNet의 원본 버전
깊이별 분리 가능 컨볼루션을 사용하는 것이 특징
다양한 크기로 제공되며 각 크기는 서로 다른 절충에 최적화되어있음

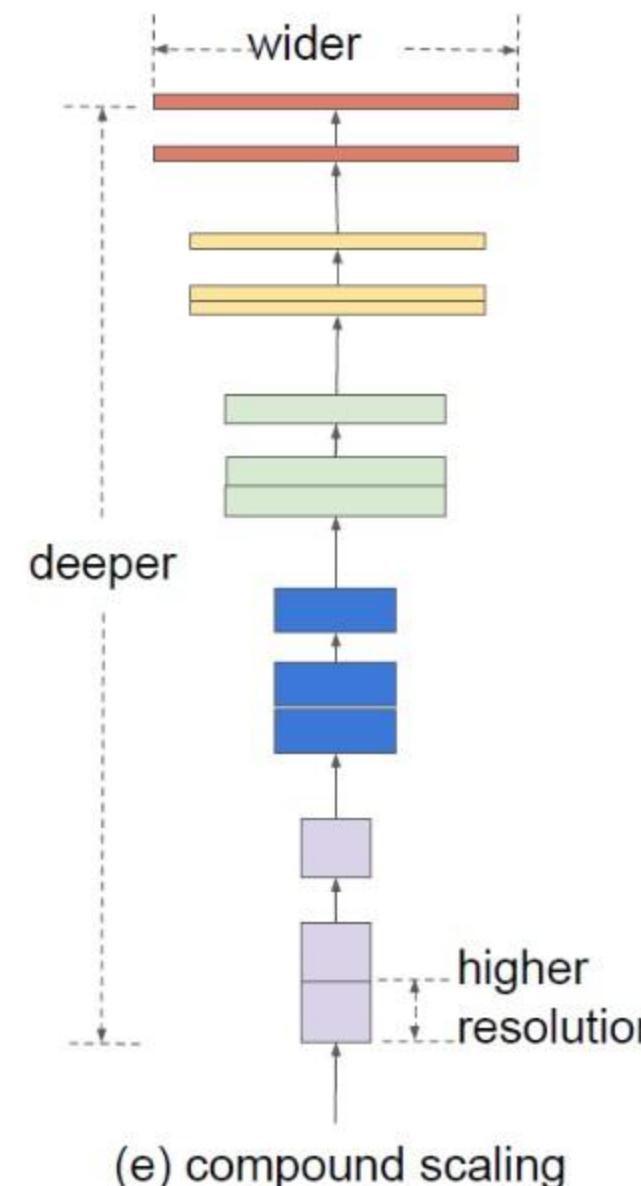
MobileNet-V2

2018년 Google에서 개발한 MobileNet의 수정된 버전
역 잔차 블록 및 선형 병목 현상을 사용하는 등 MobileNet-V1에 비해 여러 가지 개선 사항이 포함
모델의 정확도와 능률 개선

MOBILENET



EFFICIENTNET



모델의 크게 만드는 3가지 방법

1. network의 depth를 깊게 만드는 것
2. channel width(filter 개수)를 늘리는 것
(width가 넓을수록 미세한 정보가 많이 담아짐)
3. input image의 해상도를 올리는 것

=> 3가지의 최적의 조합을 효율적으로 만들 수 있도록 하는
compound scaling 방법을 제안

03 MODEL

EFFICIENTNET

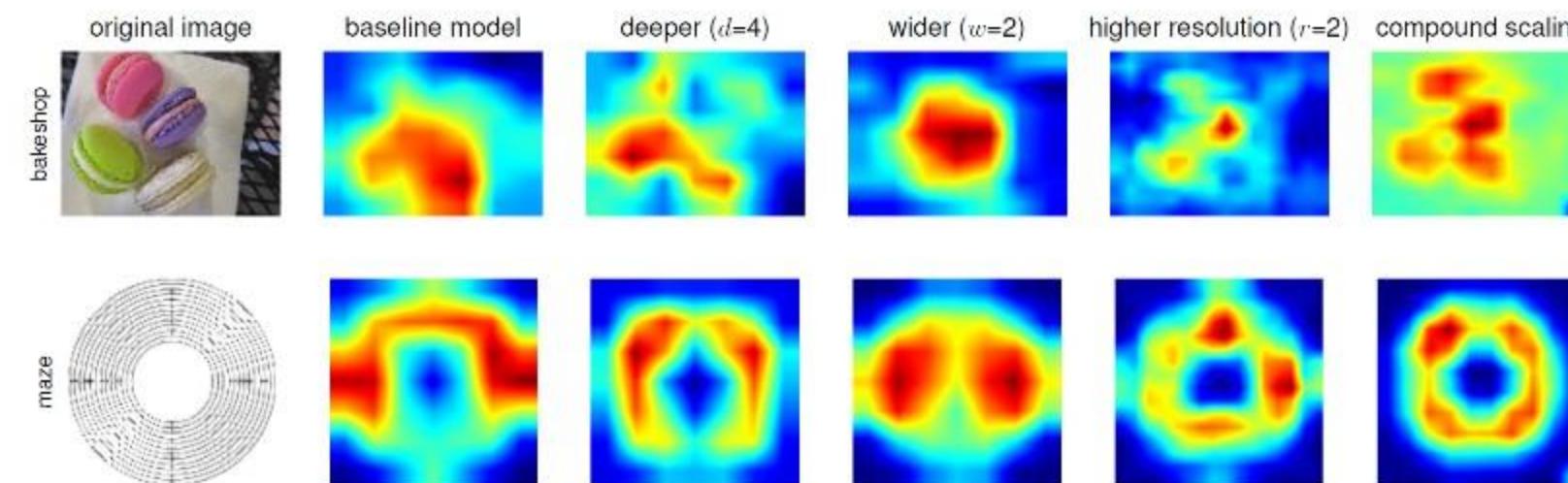
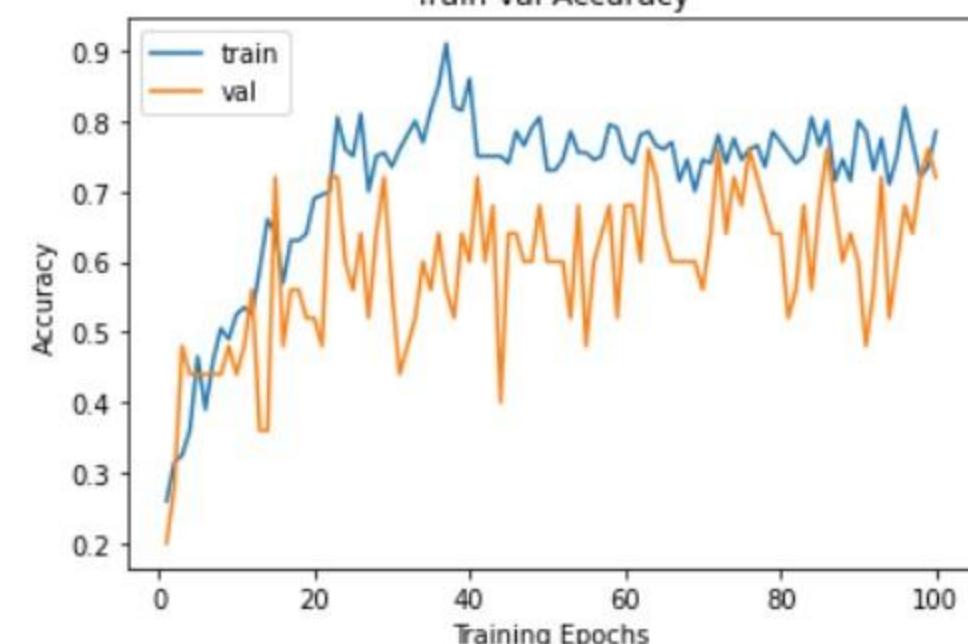
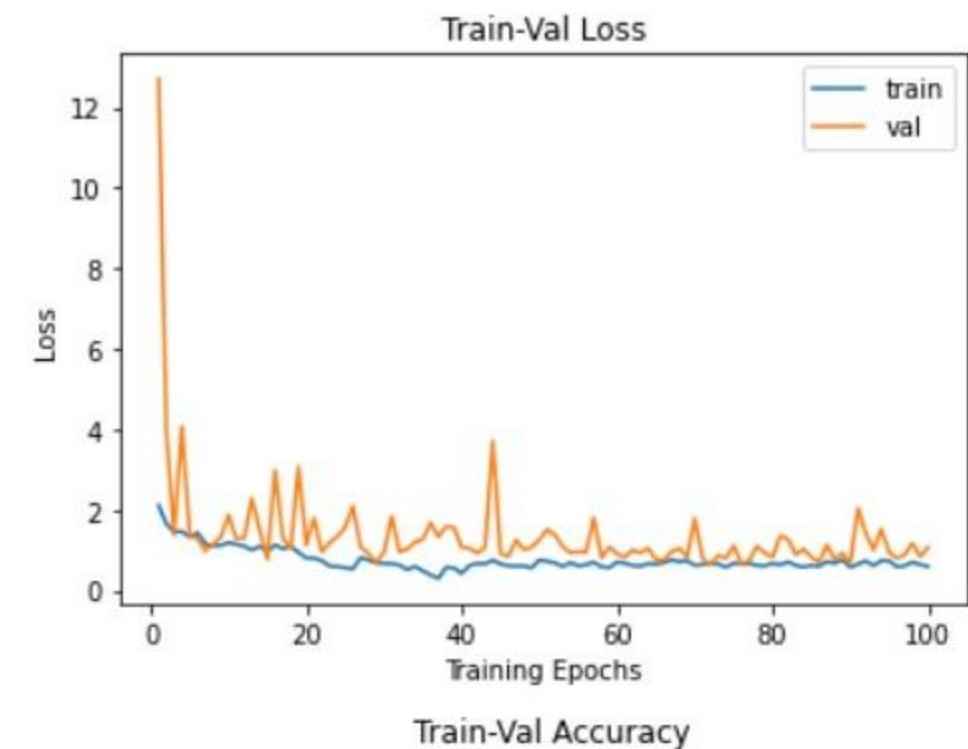


Figure 7. Class Activation Map (CAM) (Zhou et al., 2016) for Models with different scaling methods. Our compound scaling method allows the scaled model (last column) to focus on more relevant regions with more object details. Model details are in Table 7.

위의 그림을 통해 모델이 이미지의 어디에 집중하고 있는지 알 수 있으며, compound scaling method가 다른 방법에 비해 더 좋은 것을 알 수 있음



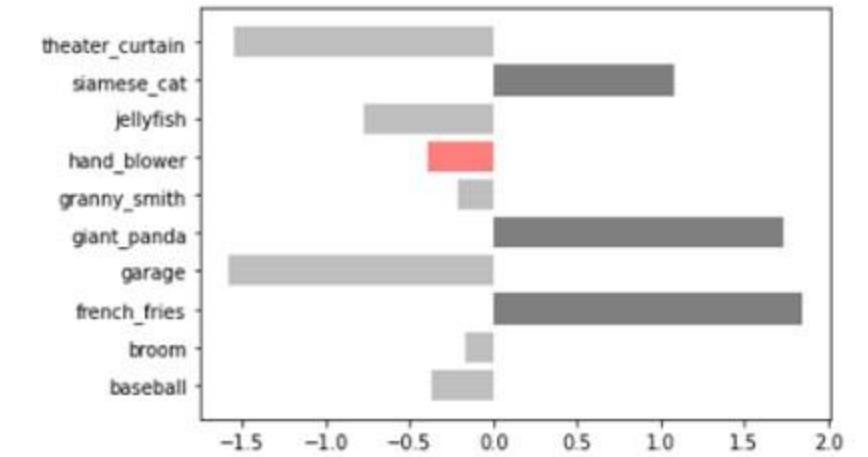
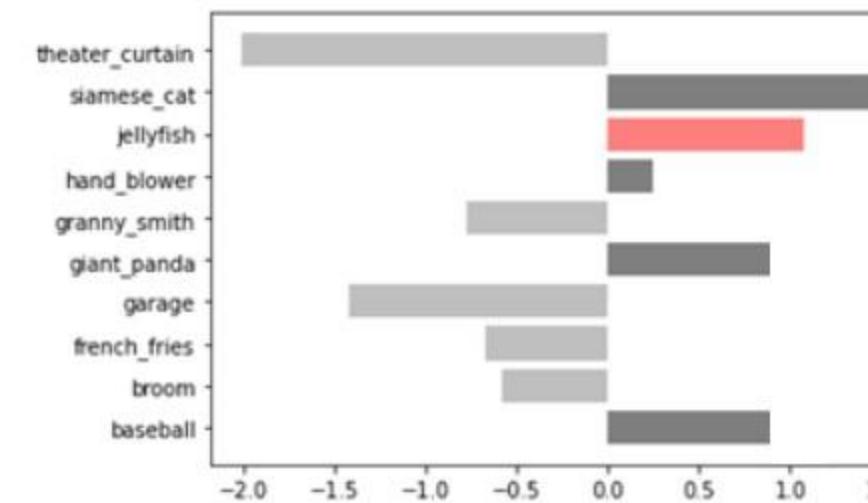
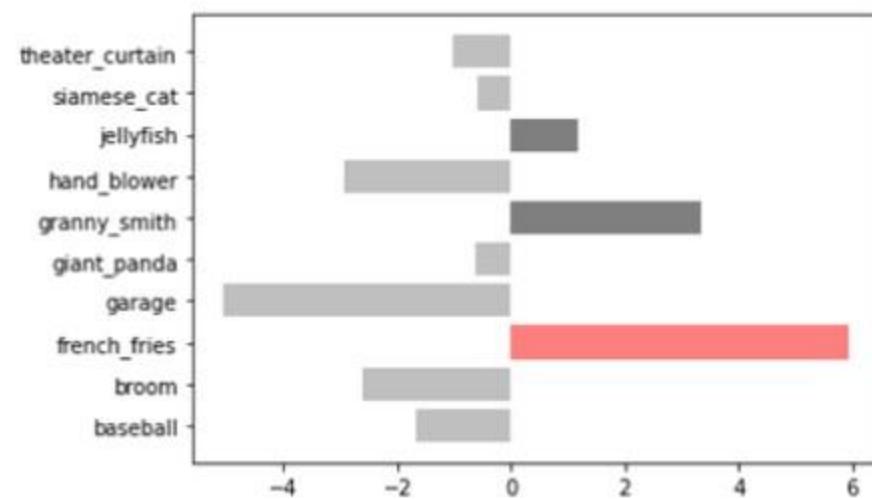
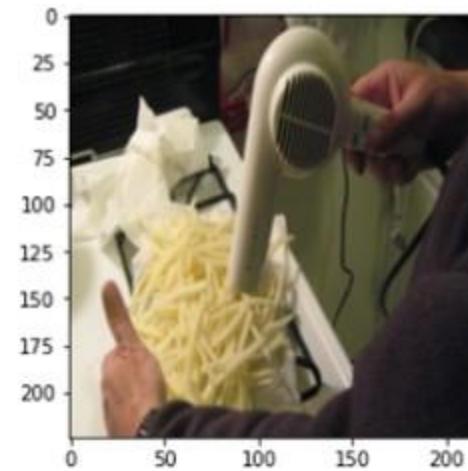
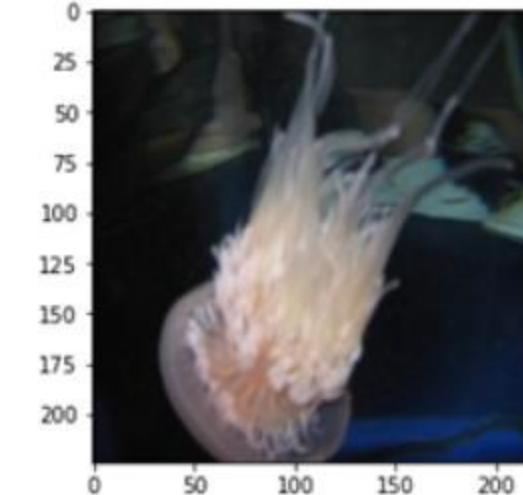
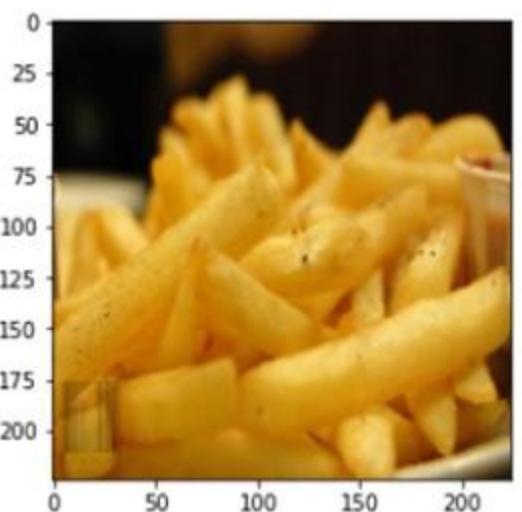
03

MODEL COMPARISION

03

MODEL COMPARISON

성능평가지표



01

TOP-1 ACC

02

TOP-2 ACC

03

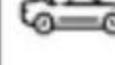
MODEL COMPARISON

성능평가지표

03

MACRO F1- SCORE

: 클래스별/레이블별 F1-SCORE의 평균

Label	True Positive (TP)	False Positive (FP)	False Negative (FN)	Precision	Recall	F1 Score
 Airplane	2	1	1	0.67	0.67	$2 * (0.67 * 0.67) / (0.67 + 0.67) = \mathbf{0.67}$
 Boat	1	3	0	0.25	1.00	$2 * (0.25 * 1.00) / (0.25 + 1.00) = \mathbf{0.40}$
 Car	3	0	3	1.00	0.50	$2 * (1.00 * 0.50) / (1.00 + 0.50) = \mathbf{0.67}$

=> 각 클래스 별 F1 SCORE의 평균
0~1 사이의 값으로,
1에 가까울수록 성능이 좋음

Model	ALEXNET	VGGNET	GOOGLENET	RESNET	MOBILENET	EFFICIENTNET
train ACC	71.3 %	86.3 %	-	83 %	98 %	78 %
train LOSS	0.886	0.536	2.59	0.534	0.115	0.81
valid ACC	68.4 %	67.8 %	38 %	69%	75.5 %	67 %
valid LOSS	98.5	0.97	1.627	0.917	0.894	0.923
test LOSS	3.974	0.961	1.44	0.895	-	0.936
top-1 ACC	44.4 %	69.6 %	36 %	-	-	71.2 %
top-2 ACC	60.4 %	85.2 %	60 %	-	-	85.6 %
macro F1-score	0.457	0.7	0.287	0.7	-	0.707

Model	ALEXNET	VGGNET	GOOGLENET	RESNET	MOBILENET	EFFICIENTNET
train ACC	71.3 %	86.3 %	-	83 %	98 %	78 %
train LOSS	0.886	0.536	2.59	0.534	0.115	0.81
valid ACC	68.4 %	67.8 %	38 %	69%	75.5 %	67 %
valid LOSS	98.5	0.97	1.627	0.917	0.894	0.923
test LOSS	3.974	0.961	1.44	0.895	-	0.936
top-1 ACC	44.4 %	69.6 %	36 %	-	-	71.2 %
top-2 ACC	60.4 %	85.2 %	60 %	-	-	85.6 %
macro F1-score	0.457	0.7	0.287	0.7	-	0.707

Model	ALEXNET	VGGNET	GOOGLENET	RESNET	MOBILENET	EFFICIENTNET
train ACC	71.3 %	86.3 %	-	83 %	98 %	78 %
train LOSS	0.886	0.536	2.59	0.534	0.115	0.81
valid ACC	68.4 %	67.8 %	38 %	69%	75.5 %	67 %
valid LOSS	98.5	0.97	1.627	0.917	0.894	0.923
test LOSS	3.974	0.961	1.44	0.895	-	0.936
top-1 ACC	44.4 %	69.6 %	36 %	-	-	71.2 %
top-2 ACC	60.4 %	85.2 %	60 %	-	-	85.6 %
macro F1-score	0.457	0.7	0.287	0.7	-	0.707

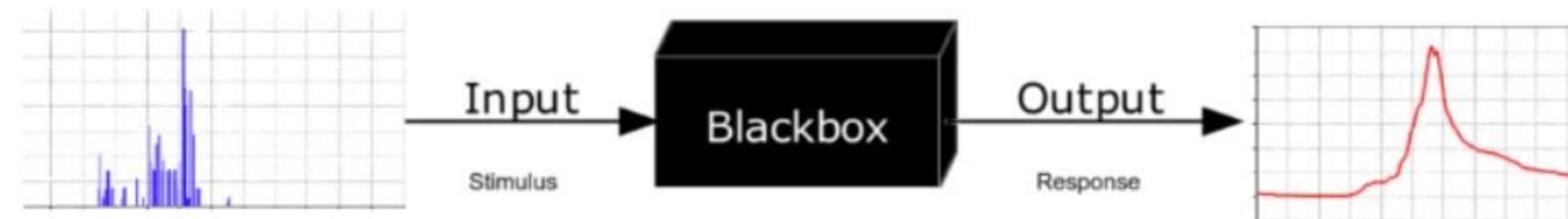
04

XAI

XAI (EXPLAINABLE ARTIFICIAL INTELLIGENCE)

- 딥러닝의 단점

: 입, 출력만 나올 뿐 그 내부 작업은 명확하지 않은 것



설명 가능한 AI인 XAI 를 통해 인공신경망의 한계를 극복

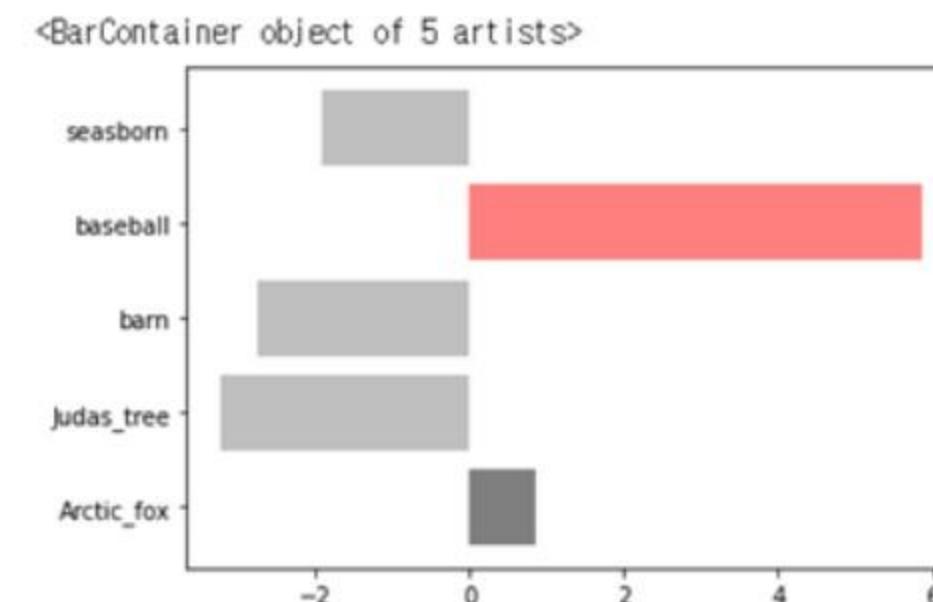
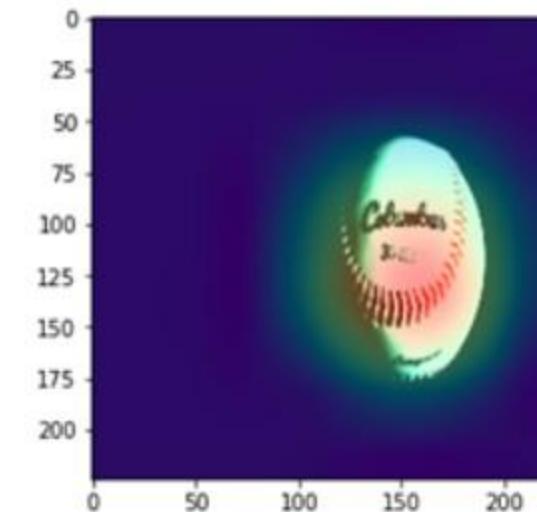
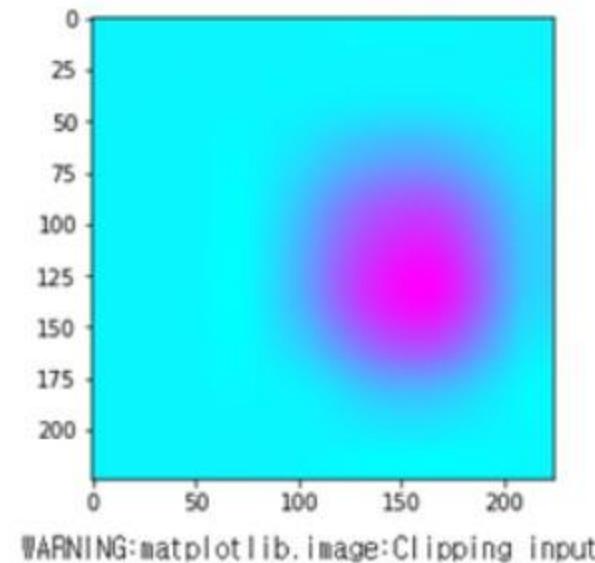
- CAM (CLASS ACTIVATION MAP)

: CNN이 입력으로 들어온 이미지를 분류할 때 "어떤 부분을 보고" 예측을 했는지를 알려주는 XAI 모델

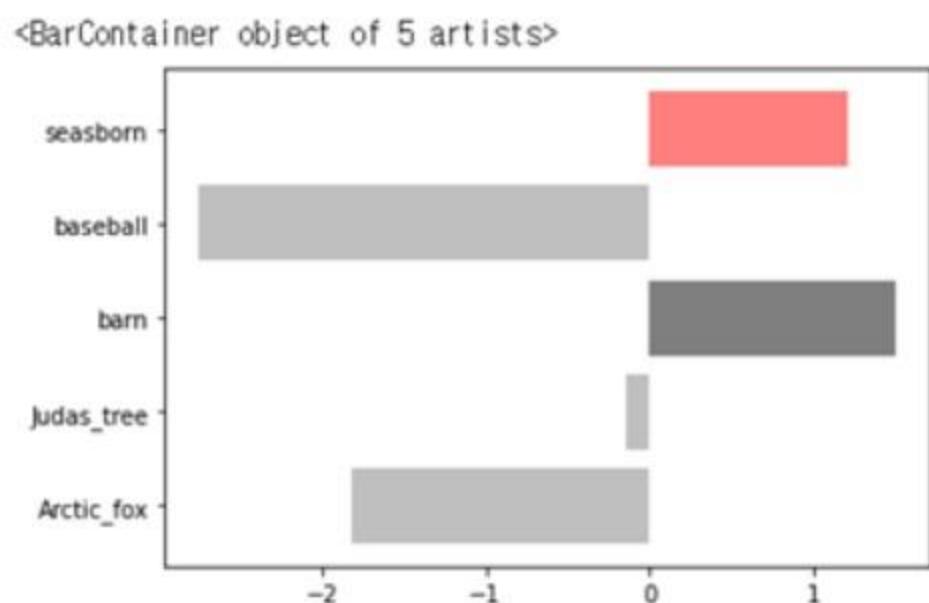
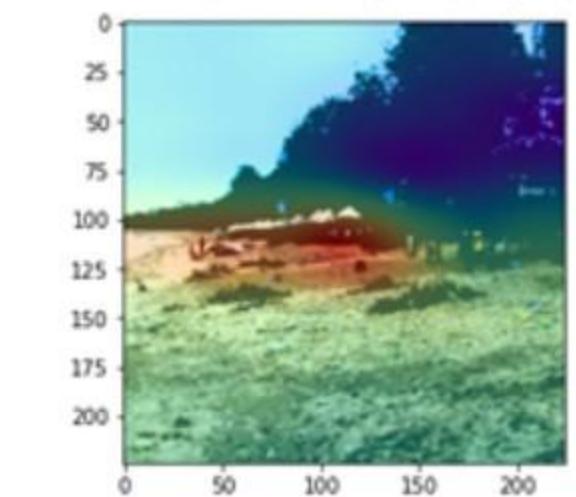
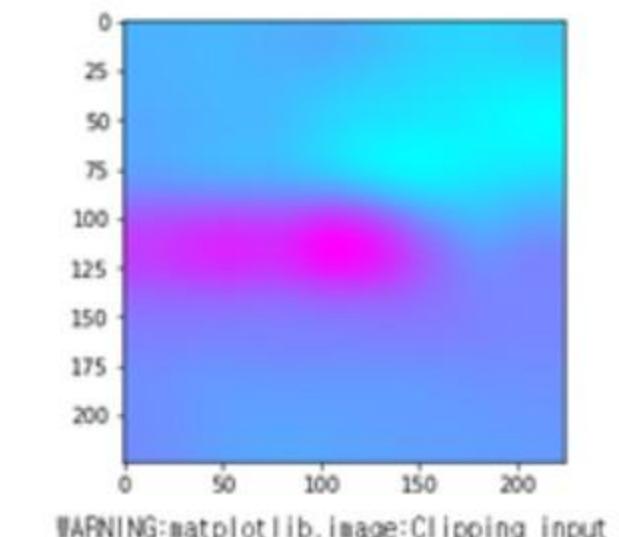
04 XAI

CAM

이미지를 분류하는 데에 있어 중요한 부분에 가까워질수록 파란색에서 빨간색으로 표현된 히트맵



IMAGENET_BASEBALL



IMAGENET_SEABORN

04 마무리

박성우

모델을 직접 구현해 보면서 딥러닝 네트워크에 대한 이해도가 크게 상승한 것을 느꼈고 다음에 기회가 된다면 옵티마이저 별 성능 차이를 비교해보고 싶습니다.

민예린

파이썬 코드에 대한 이해와 딥러닝 API 혹은 메소드에 대해 좀 더 공부해 보고 싶었는데 시간적으로 여유가 없어서 좀 아쉬웠습니다. 제가 구현하려 했던 모델을 확실하게 구현해보고 싶어졌습니다.

박정민

GOOGLE COLAB으로 프로젝트를 진행하였는데 IMAGENET이 대용량 데이터라 논문을 통해 알게 된 지식을 온전히 활용하기에는 RAM의 용량이나 GPU 사용량의 제한이 있어서 아쉬웠다. 시간이 더 있다면 이러한 제한도 극복할 수 있는 지식을 얻어 모델을 완벽하게 구현해보고 싶다.

04 마무리

전수연

위에서는 PYTORCH 공식 도큐먼트의 VGG 시리즈에 있는 모델을 이용하여 CAM모델의 결과를 살펴보았다.
시간적으로 여유가 있었다면 직접 모델 성능 평가를 확인한 모델에 대한 CAM모델을 살펴보는 시간을 가졌으면 하는 아쉬움이 남는다.

이를 통해 모델이 틀리게 예측한 원인을 분석하여 보고 그에 대해 보완하는 작업을 할 수 있을 것이다.

최경준

SEMENTICE SEGMENTATION 픽셀별 분류를 실행하는 개념
추출된 특징을 분석 하늘, 숲, 동물을 각각 다른 분류를 해낸다.
각 클래스별로 다른 수치 데이터 반환해야 해서 연산량이 많다.
동물 두마리가 겹쳐있는 사진이 있다면 같은 클래스로 보는데 각각의 클래스로 나눠 픽셀 하나하나를 분류해보고 싶다

TABA

THANK
YOU

