HOTDOG

Hotdog or Not ? / CNN classifier

Play-Data Deeplearning Bootcamp team 3
: 양효준, 정제경, 전현준

Table Of Contents

Hotdog-Not-Hotdog

Data to create the hotdog-not-hotdog app from HBO's Silicon Valley

Data Explorer

174.4 MB

- hotdog-nothotdog
  - test
  - train

Summary

- 3644 files

3644 개의 데이터 ---------- 과대 적합(Overfitting) 방지

이미지 증식 활용

```
transform = transforms.Compose([transforms.Resize([299, 299]),
                                transforms.RandomHorizontalFlip(0.2),
                                transforms.RandomRotation(degrees = (0, 20)),
                                transforms.ToTensor()
                               ])
```

Data Augmentation Before

Data Augmentation After

설정해둔 함수로 인해 Lotation, HorizontalFlip된 몇몇 데이터들를 볼 수 있음

Data Seperaton

```
print(type(trainset), len(trainset)) # 3000
print(type(validset), len(validset))
print(type(testset), len(testset)) # 644
```

```
<class 'torch.utils.data.dataset.Subset'> 2400
<class 'torch.utils.data.dataset.Subset'> 600
<class 'torchvision.datasets.folder.ImageFolder'> 644
```

```
trainset, validset = random_split(trainset, [2400, 600])
```

원래 trainset - 3000,
testset - 644 으로 할당

데이터셋이 더 큰 trainset
에서 검증 데이터를 들고옴

Dataset/DataLoader

데이터셋 생성

```
trainset = datasets.ImageFolder(root= data_dir + '/hotdog-nothotdog/train', transform=t
testset = datasets.ImageFolder(root= data_dir + '/hotdog-nothotdog/test', transform=tra
```

데이터로더 생성

```
batch_size = 16
trainloader = DataLoader(trainset, batch_size=batch_size, shuffle=True)
validloader = DataLoader(validset, batch_size=batch_size, shuffle=False)
testloader = DataLoader(testset, batch_size=batch_size, shuffle=False)
```

Why batch_size = 16 ?

Test AUC

Batch size	Adam LR = 0.0001	Adam LR = 0.001
16	**0.9677**	0.9144
32	0.9636	0.9332
64	0.9616	0.9381
128	0.9567	0.9432
256	0.9585	**0.9652**

Test AUC

Batch size	Adam LR = 0.0001	Adam LR = 0.001
16	**0.9677**	0.9144
32	0.9636	0.9332
64	0.9616	0.9381
128	0.9567	0.9432
256	0.9585	**0.9652**

한 논문에서 Batch size 를 16으로 설정했을 때 가장 좋게 성능이 나왔기에 먼저 16을 설정해보았고,
그 이후 18, 22 정도로 조금 높은 값 또한 설정해보았으나 더 낮은 성능을 보여줘 해당 값을 택함

Learning Rate/Loss/Optimizer

Test AUC

Batch size	Adam LR = 0.0001	Adam LR = 0.001
16	**0.9677**	0.9144
32	0.9636	0.9332
64	0.9616	0.9381
128	0.9567	0.9432
256	0.9585	**0.9652**

Test AUC

Batch size	Adam LR = 0.0001	Adam LR = 0.001
16	**0.9677**	0.9144
32	0.9636	0.9332
64	0.9616	0.9381
128	0.9567	0.9432
256	0.9585	**0.9652**

잘모르겠으면 Adam?!
Momentum + AdaGrad
-> 하이퍼파라미터

Why Learning Rate = 0.00001?

참고 논문에서는 batch size가 16일 때 lr = 0.0001로 설정하는 것이 가장 높은 성능을 보여주었으나,

실제로 실행하였을때 0.00001일 때보다 낮은 성능을 보여주었고 (Resnet 기준 89%)

후반부로 갈수록 과적합 또한 조금 더 발생하였기에 해당 값으로 설정 (lr =0.00001 일때는 Resnet 기준 93% 성능)

Lr 값만으로도 약 4%정도의 높은 차이가 나타남

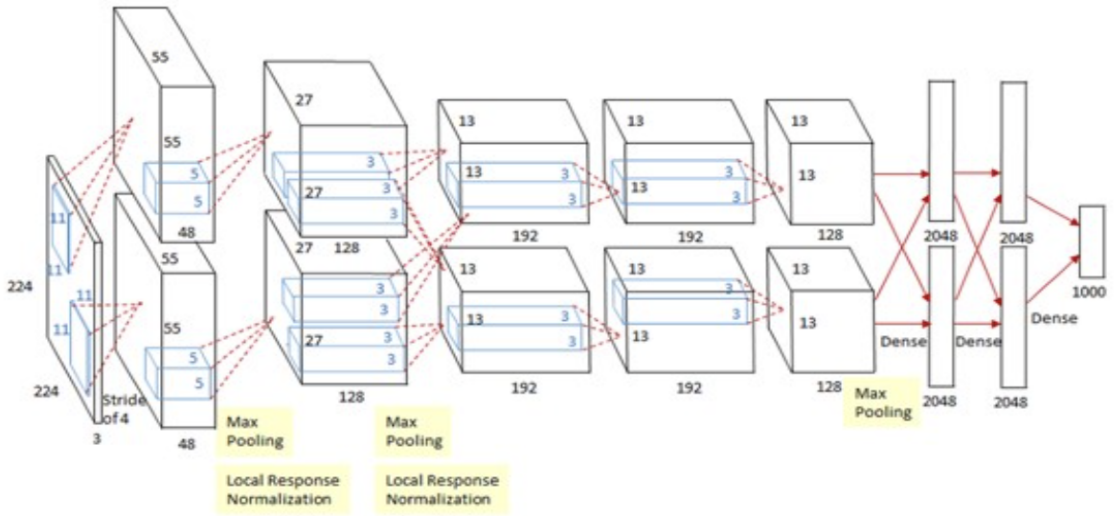
손실 함수 - CrossEntropyLoss
Optimizer = Adam 사용

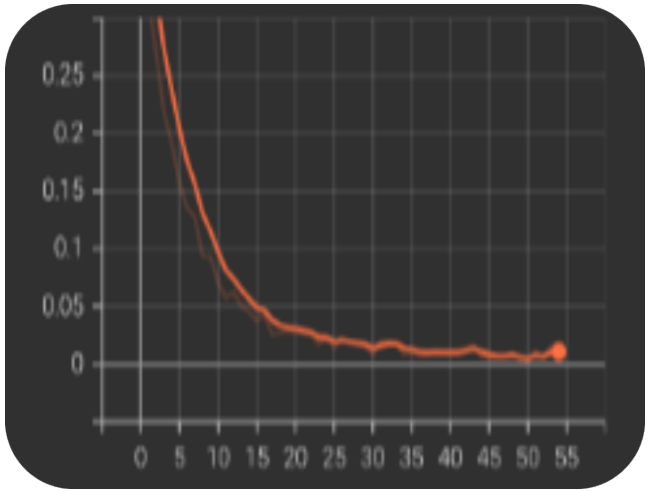
```
learning_rate = 0.00001
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr = learning_rate)
```

AlexNet

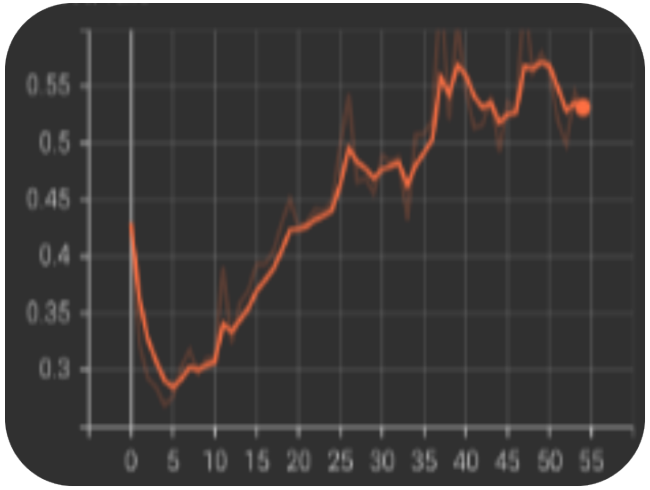
딥러닝의 '가능성'을 찾다

8개의 Layer (Conv2D Layer * 5, Full-connection Layer * 3)

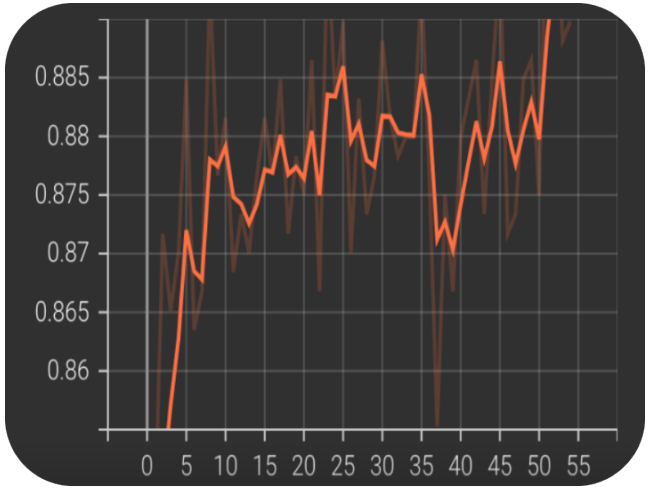




Train Loss



Valid Loss



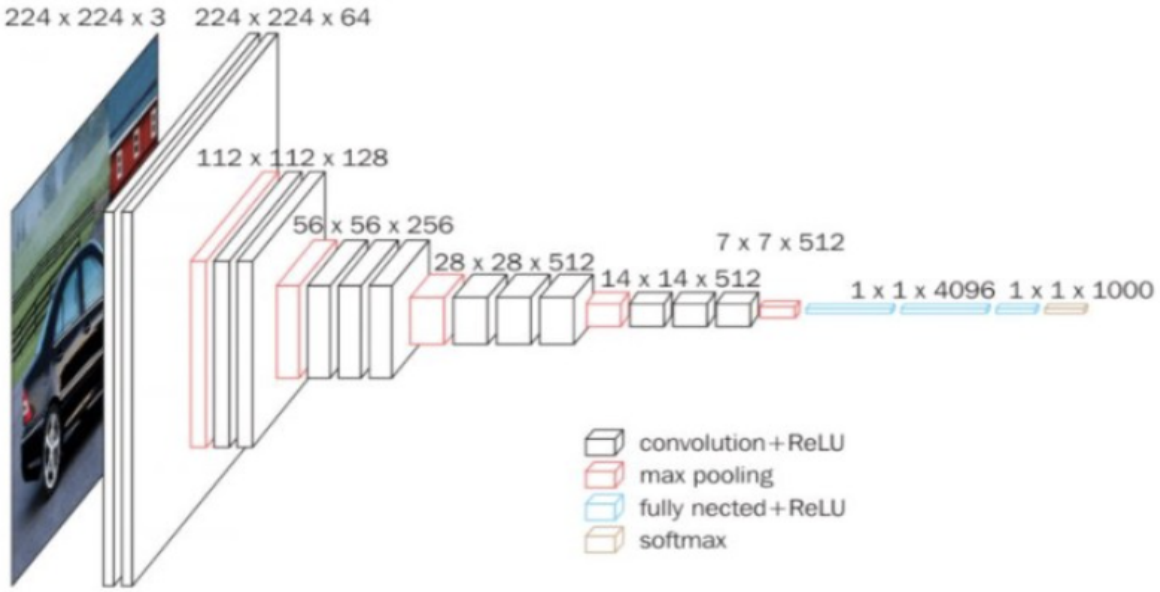
Valid Accuracy

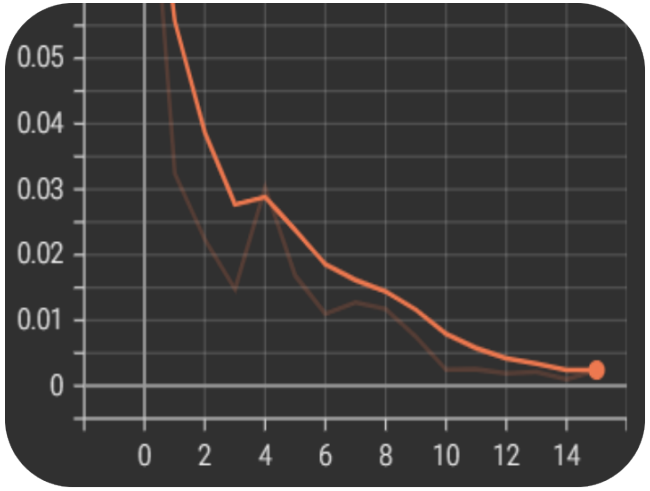
과적합 발생 - 0.25 정도의 격차

VGGNet

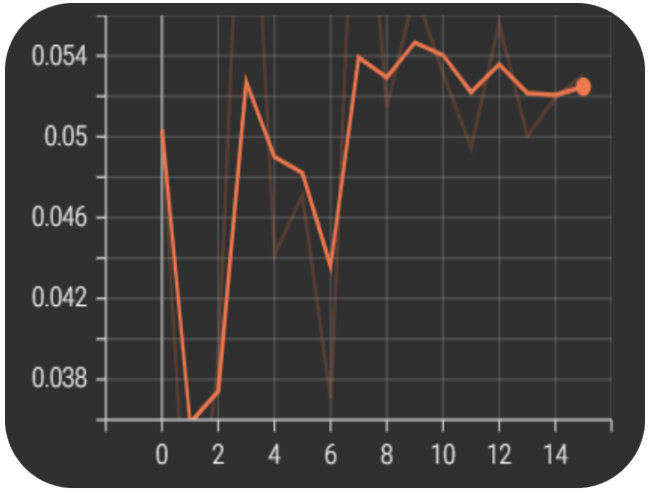
16개의 Layer
3X3 Kernel, Stride 1, padding = 1
MaxPool2D

VGG16, VGG19가 존재하는데 시간효율 대비 성능 차이는 크게 차이나지 않기에 VGG16을 선택

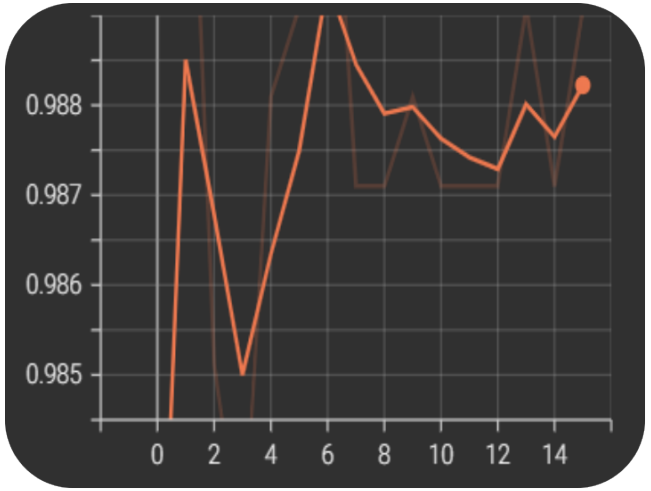




Train Loss



Valid Loss



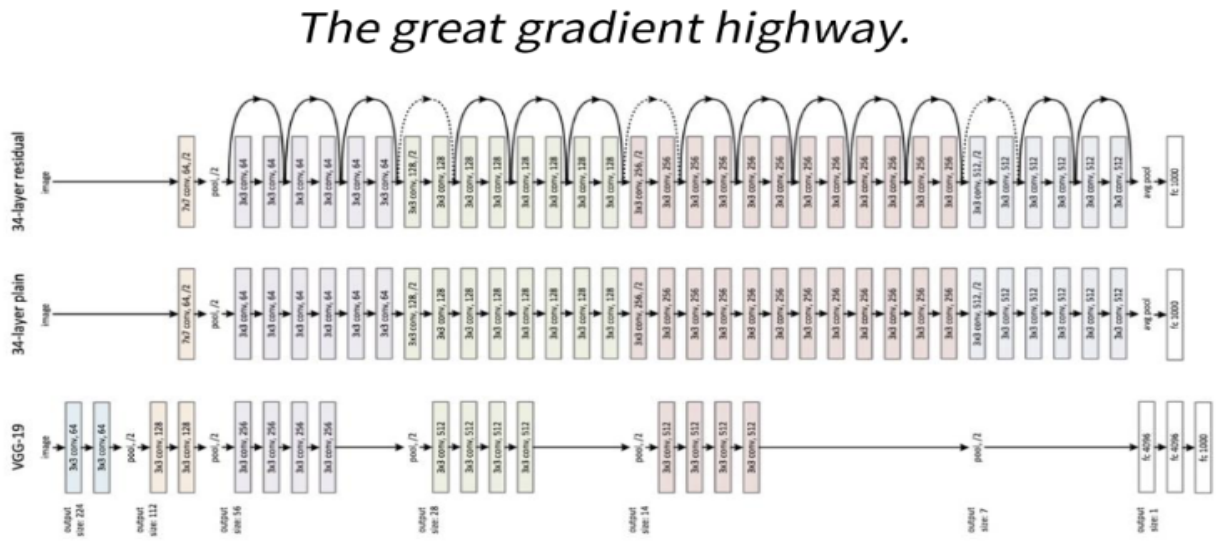
Valid Accuracy

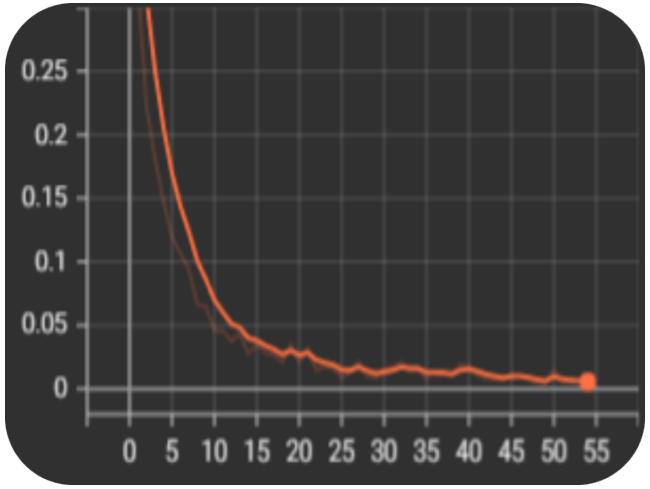
과적합 발생했으나 0.016 정도의 격차로 줄어들었음

ResNet

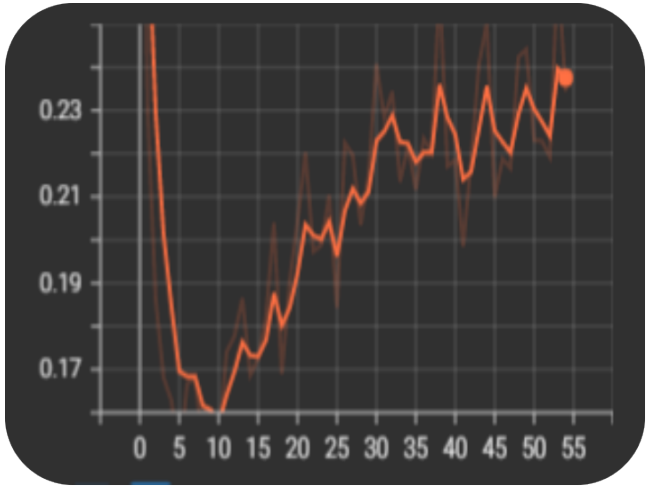
‘지름길’ - Skip Connection

Residual 구조를 통해 작은 변화가 생기더라도 더 민감하게 반응해서 학습 가능
- 152 Layer

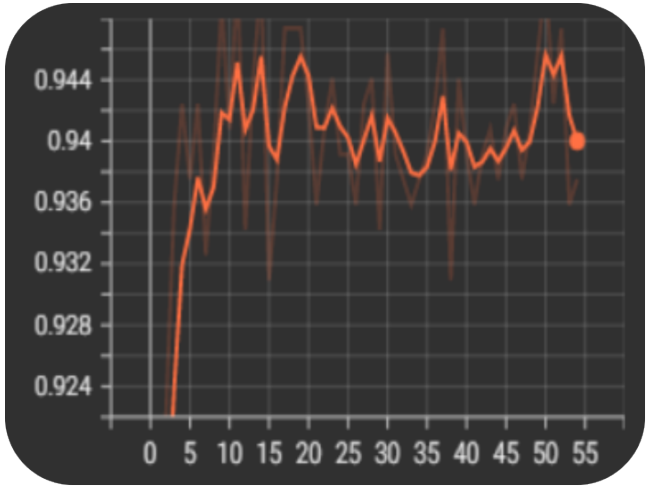




Train Loss



Valid Loss



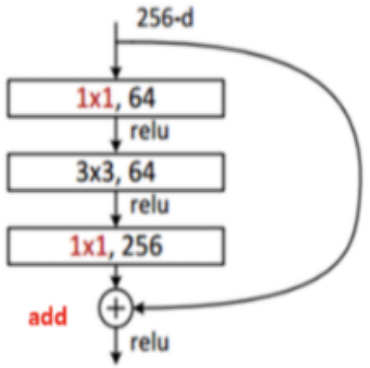
Valid Accuracy

과적합 발생 - 0.06 정도의 격차

Resnet vs Densenet

Resnet

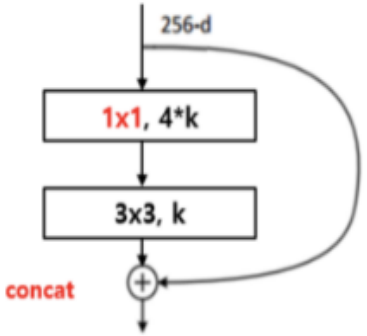
$$x_l = H_l(x_{l-1}) + x_{l-1}$$



bottleneck
(for ResNet)

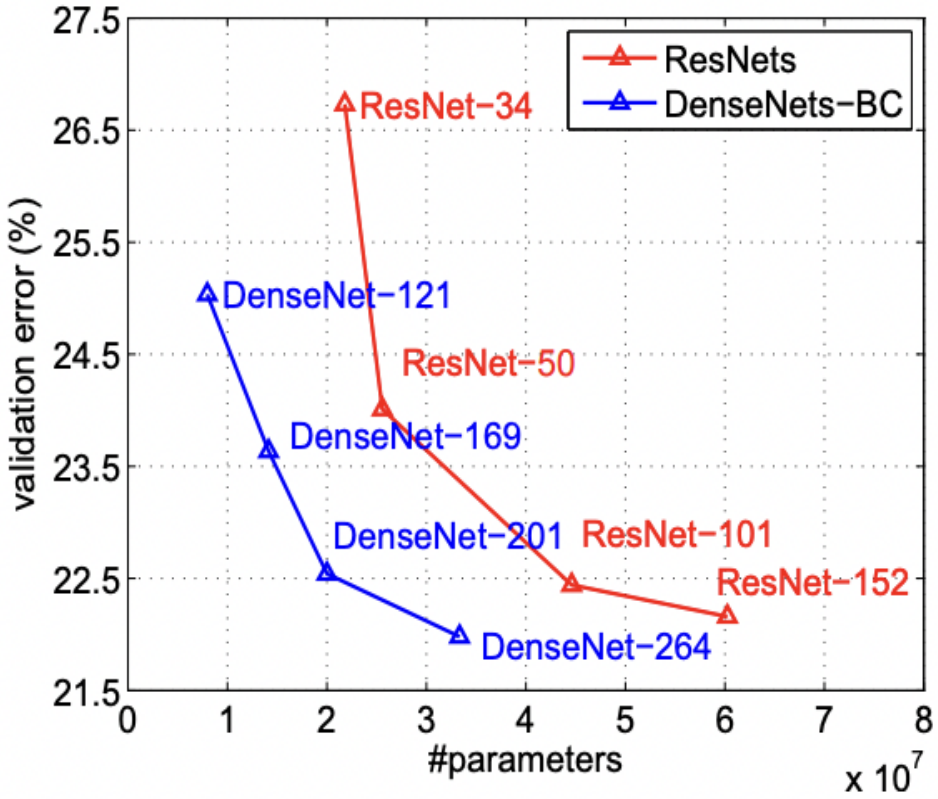
Densenet

$$x_l = H_l([x_0, x_1, \ldots, x_{l-1}])$$



bottleneck
(for DenseNet)

Residual Block을 계속 쌓다보니
파라미터가 계속 누적되어 계산량 증폭 문제점

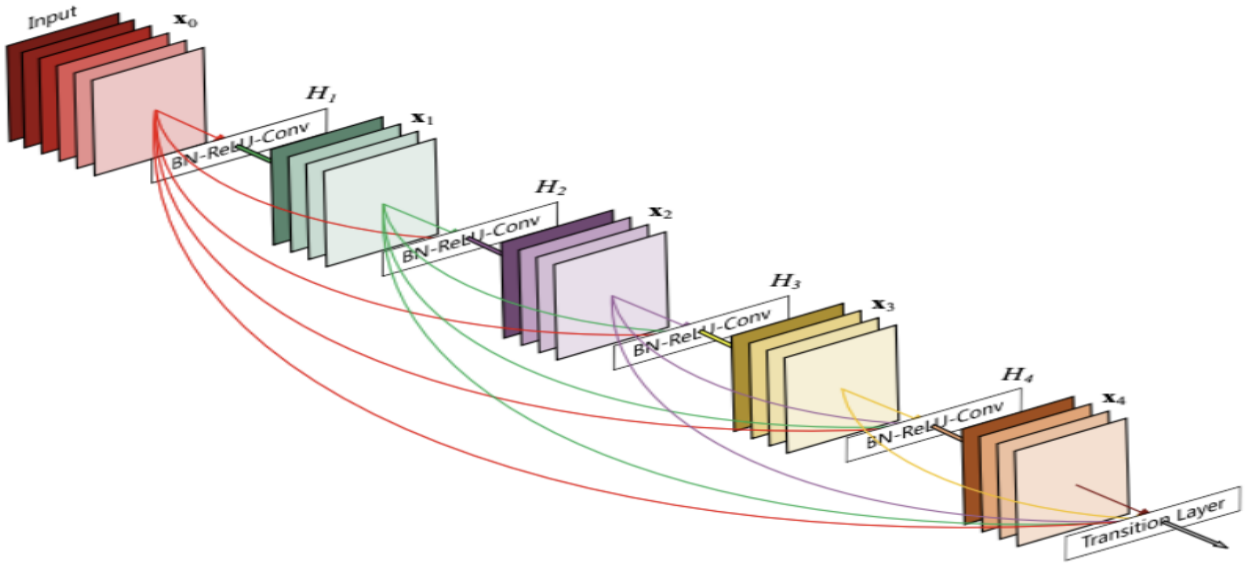


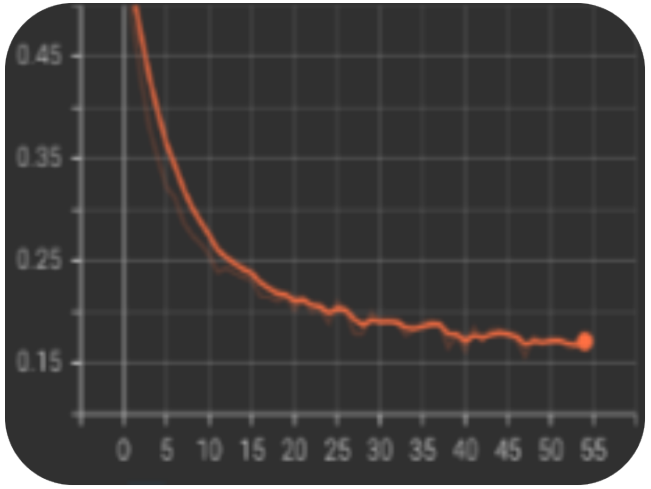
Learning Parameters 개수는 적지만 보다 좋은 성능 구현 가능

DenseNet

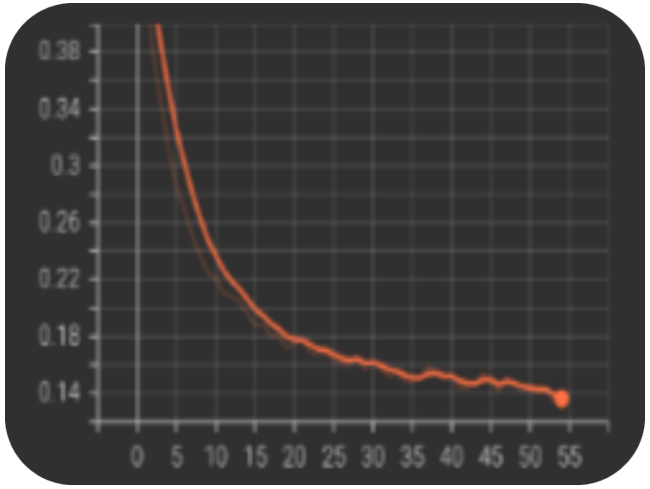
적어진 파라미터

- Vanishing Gradient 문제 감소
- 과적합에 강건 (그래서 data augmetation 시 성능 증진이 미미할 수 있다)

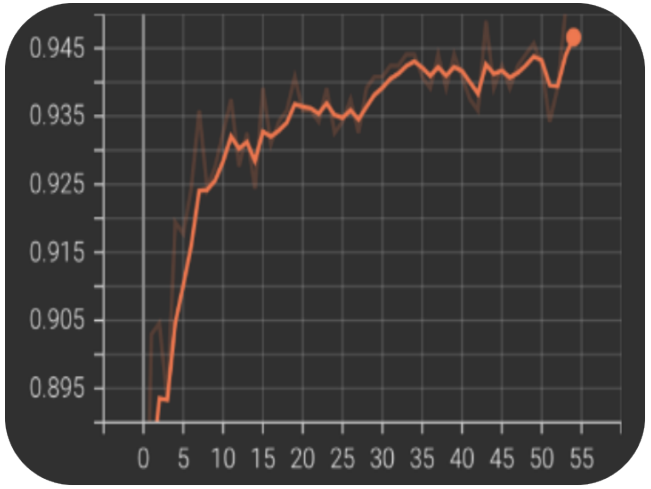




Train Loss



Valid Loss



Valid Accuracy

Performance Comparison

- AlexNet -> VGGNet -> ResNet -> DenseNet 순으로, 점차 손실은 감소하고 정확성이 높아지는 모습
- 이미지 증식이 결과 예측 성능감소에 영향을 끼침 (AlexNet 기준: 손실 0.117 증가, 정확도 0.006감소)

AlexNet

이미지 증식 전

Test Loss 0.526

Test Accuracy 0.882

이미지 증식 후 - 개선된 결과

Test Loss 0.643

Test Accuracy 0.876

VGGNet

이미지 증식 후

Test Loss 0.444

Test Accuracy 0.914

ResNet

이미지 증식 후

Test Loss 0.223

Test Accuracy 0.937

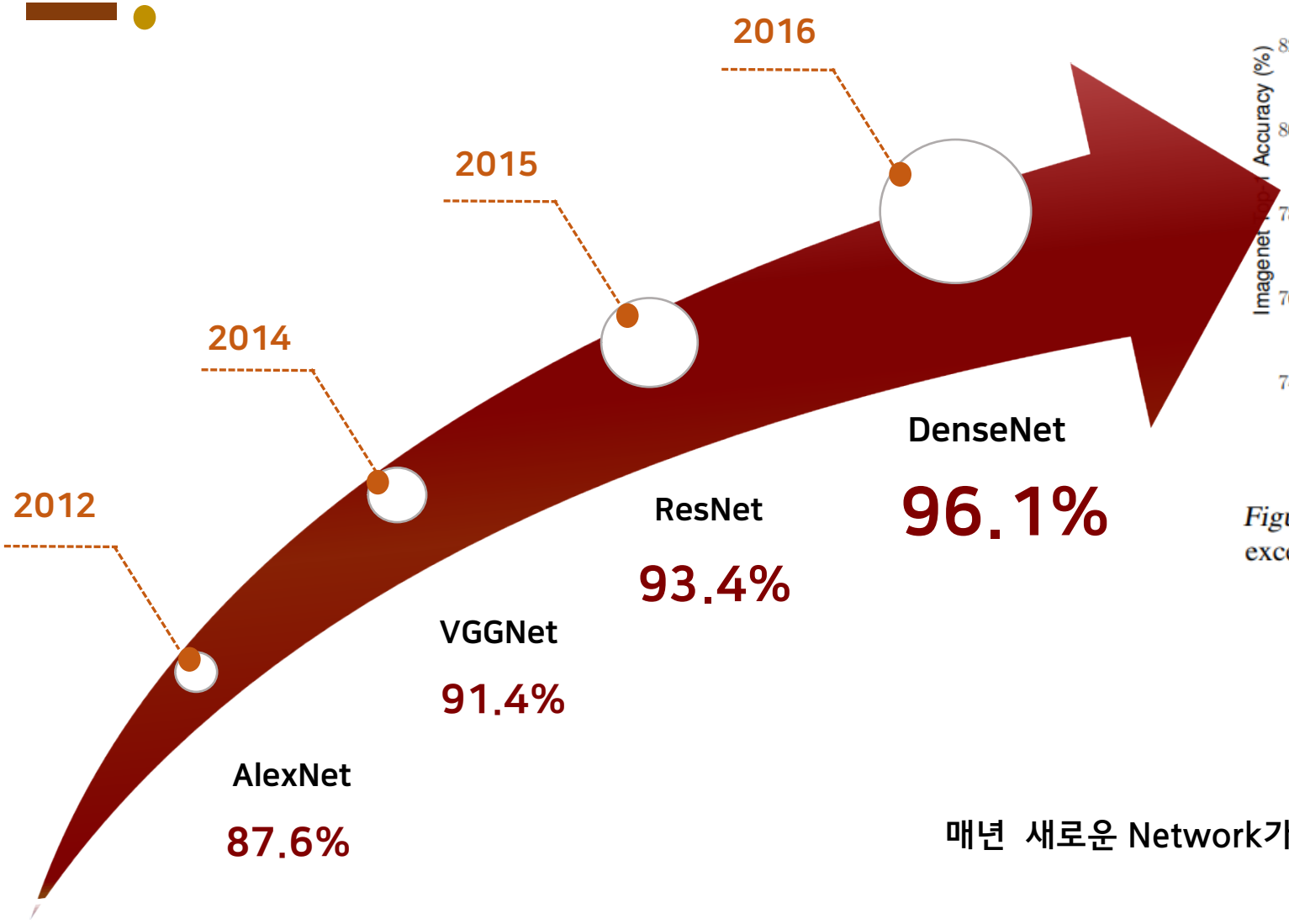
DenseNet

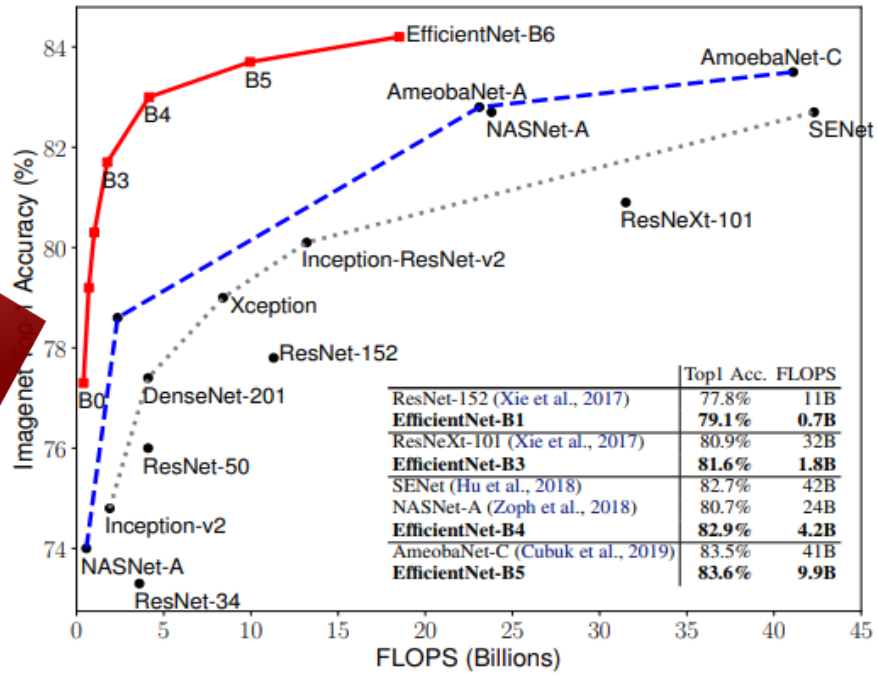
이미지 증식 후

Test Loss 0.128

Test Accuracy 0.963

Infographic





	Top1 Acc.	FLOPS
ResNet-152 (Xie et al., 2017)	77.8%	11B
EfficientNet-B1	**79.1%**	**0.7B**
ResNeXt-101 (Xie et al., 2017)	80.9%	32B
EfficientNet-B3	**81.6%**	**1.8B**
SENet (Hu et al., 2018)	82.7%	42B
NASNet-A (Zoph et al., 2018)	80.7%	24B
EfficientNet-B4	**82.9%**	**4.2B**
AmeobaNet-C (Cubuk et al., 2019)	83.5%	41B
EfficientNet-B5	**83.6%**	**9.9B**

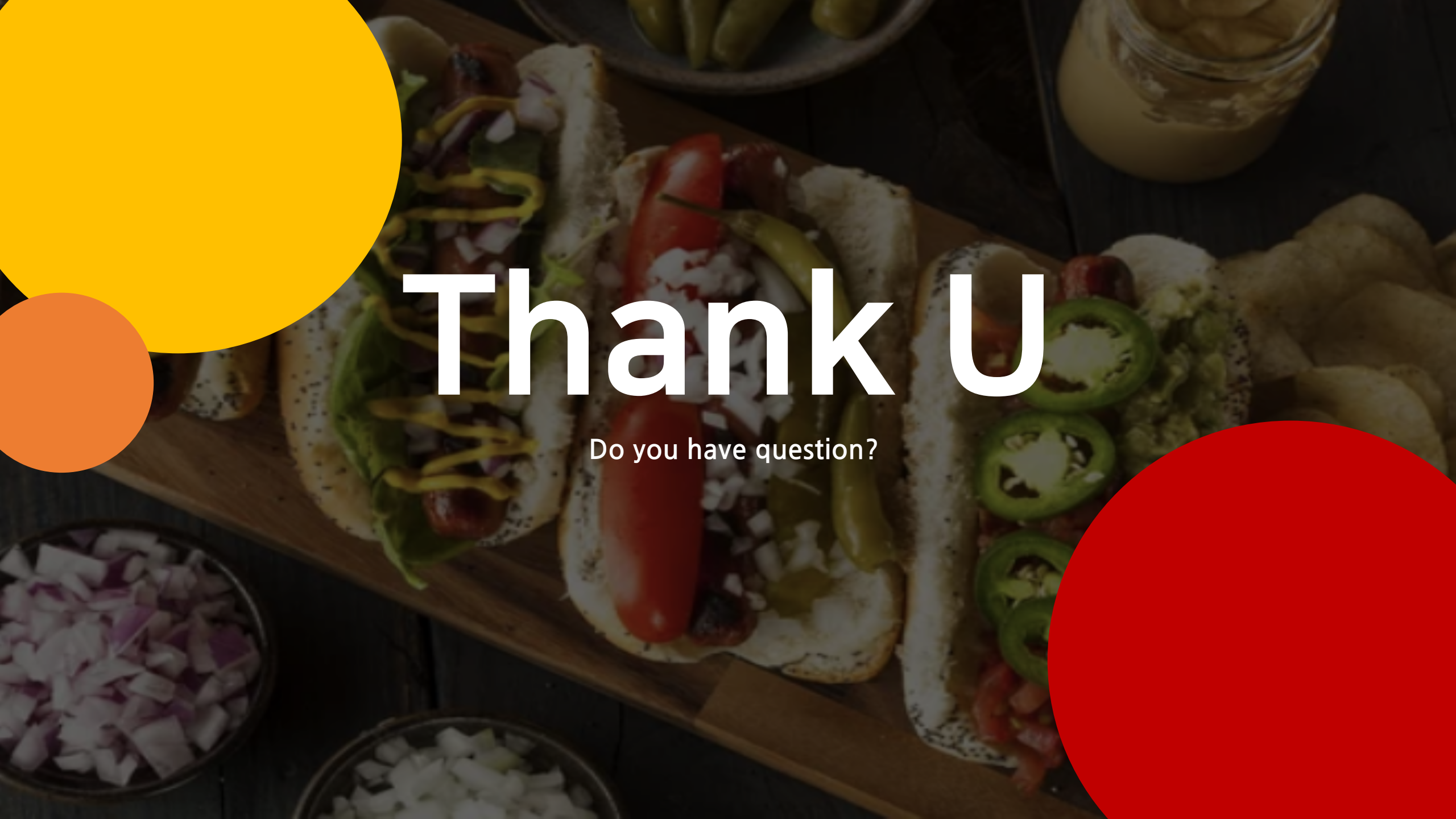
Figure 5. **FLOPS vs. ImageNet Accuracy** – Similar to Figure 1 except it compares FLOPS rather than model size.

우리가 실행한 Network Architecture 이외에도
매년 새로운 Network가 나오고 있고, 매번 더 발전된 성능을 보여주고 있음

Reference

PanellbrahemKandel,「 The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset」, 2020

https://www.sciencedirect.com/science/article/pii/S2405959519303455#fig2



Thank U

Do you have question?