Going Deeper with Convolutions

- Reading the paper in terms of dimensionality reduction -

1. Introduction

Convolution Neural Network(CNN)은 더 많은 convolution layer를 쌓아서 층을 깊게 만들수록 더욱 좋은 성능을 기대할 수 있습니다. 대표적으로 2014년에 발표된 VGG(VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION)에서 필터의 크기를 3x3으로 최대한 작게 설정한 상태에서 convolution 층을 더 많이 쌓을수록 좋은 성능을 보였다고 증명하였습니다

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**

[그림1] VGG네트워크 유형 별 구조, weight layers의 층수에 따라 네트워크 정의(예: VGG16)

ConvNet config. (Table 1)	top-1 val. error (%)	top-5 val. error (%)
A	29.6	10.4
A-LRN	29.7	10.5
B	28.7	9.9
C	28.1	9.4
	28.1	9.3
	27.3	8.8
D	27.0	8.8
	26.8	8.7
	25.6	8.1
E	27.3	9.0
	26.9	8.7
	25.5	**8.0**

[그림2] VGG네트워크 별 ImageNet 데이터 정확도, 층수가 가장 많은 VGG19가 가장 높은 정확도를 보임

천 개의 클래스와 약 210만장의 학습 이미지로 구성된 ImageNet 데이터를 이용하여 계층 수가 다른 6개의 VGG 모델을 실험한 결과, 계층이 깊어지면 깊어질수록 분류 성능이 올라간다는 것을 보여주고 있습니다. 논문에서는 convolution 계층의 중첩이 더 많은 이미지의 특징을 추출하고 non-linear 활성화 함수인 ReLU 계층이 증가하면서 decision boundary의 설정을 더 명확하게 할 수 있기 때문이라고 말합니다. 그렇다면 "convolution 계층 수 백 개를 쌓아서 모델을 만들면 되지 않을까?" 라는 의문이 들지만 단순히 convolution 계층을 쌓는 것은 치명적인 결함을 가지고 있습니다.

2. Background

치명적인 결함은 바로 네트워크가 깊어질수록 학습해야 하는 파라미터의 수와 연산량이 기하급수적으로 늘어난다는 것입니다. 이 부분은 convolution 연산에 대한 쉬운 예시로 설명할 수 있습니다.

만약 이미지가 128 x 112 x 112(채널, 가로, 세로)이고, 128개의 필터가 3 x 3사이즈, 패딩 1, 스트라이드 1의 조건으로 convolution 연산을 실행한다면 결과는 128 x 112 x 112로 입력이미지와 같은 형태가 됩니다. 따라서 총 필요한 연산은 필터의 크기와 출력 이미지의 곱인 128 x 3 x 3 x 128 x 112 x 112 = 1,849,668,064번이 됩니다. 파라미터 수는 ((64 x 3 x 3) + 1) x 128 = 73856개가 됩니다. 한 개의 convolution 계층의 연산량과 파라미터의 수만 해도 엄청난 연산을 필요로 하므로, 많은 층을 쌓는다는 것은 굉장히 부담이 큰 일입니다.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 224, 224]	1,792
ReLU-2	[-1, 64, 224, 224]	0
Conv2d-3	[-1, 64, 224, 224]	36,928
ReLU-4	[-1, 64, 224, 224]	0
MaxPool2d-5	[-1, 64, 112, 112]	0
Conv2d-6	[-1, 128, 112, 112]	73,856
ReLU-7	[-1, 128, 112, 112]	0
	⋮(중략)	
Conv2d-29	[-1, 512, 14, 14]	2,359,808
ReLU-30	[-1, 512, 14, 14]	0
MaxPool2d-31	[-1, 512, 7, 7]	0
AdaptiveAvgPool2d-32	[-1, 512, 7, 7]	0
Linear-33	[-1, 4096]	102,764,544
ReLU-34	[-1, 4096]	0
Dropout-35	[-1, 4096]	0
Linear-36	[-1, 4096]	16,781,312
ReLU-37	[-1, 4096]	0
Dropout-38	[-1, 4096]	0
Linear-39	[-1, 1000]	4,097,000

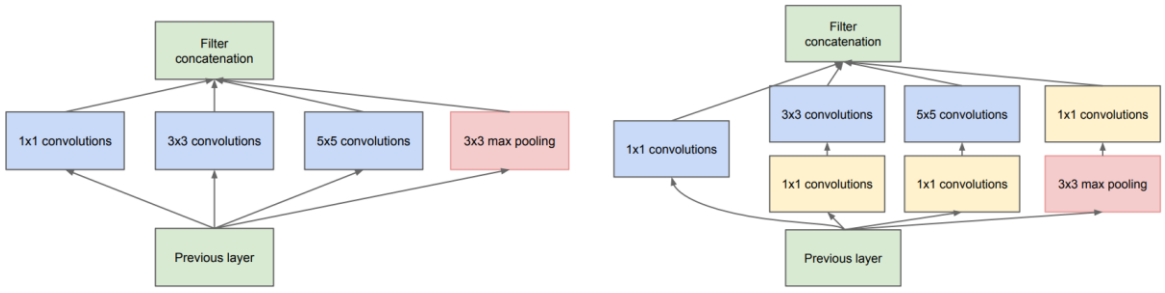
[그림3] VGG16 네트워크의 계층 별 학습 파라미터 수

위의 그림의 'Param #'은 VGG16 네트워크의 계층 별 학습해야 할 파라미터의 개수를 의미합니다. 기본적으로 파라미터의 개수와 연산량은 비례하며, convolution 연산의 경우 파라미터의 개수에 따라 연산량은 기하급수적으로 증가합니다. 따라서 파라미터의 개수를 줄이는 것이 연산량을 줄이기 위한 핵심이 됩니다. 'Output Shape'은 각 계층의 출력 이미지의 크기를 나타내는데, 점점 이미지의 크기가 작아지지만, 출력 이미지의 채널 수의 증가에 따라 파라미터의 개수가 기하급수적으로 증가하

는 것을 볼 수 있습니다. 즉 파라미터의 개수를 줄이기 위해서는 채널을 감소시키는 것이 핵심이라고 할 수 있습니다.

3. Proposed Method

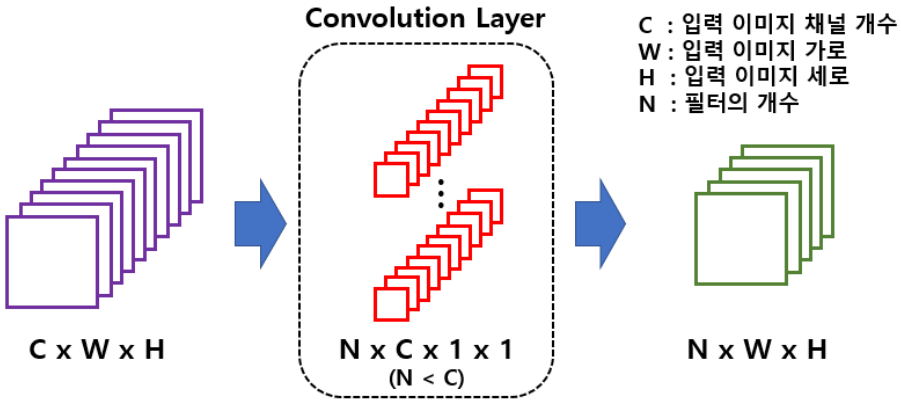
이미지의 채널을 감소시키는 문제는 결국 차원 축소의 문제로 볼 수 있습니다. GoogLeNet은 기존 CNN 네트워크들의 convolution 계층의 연산량과 파라미터의 수를 감소시키는 문제에 대해서 해결책을 제시하는 것에 중점을 두고 있습니다. GoogLeNet에서 제시한 방법은 바로 인셉션(inception)모듈입니다. Inception 모듈의 핵심은 1 x 1 convolution 필터로 설명할 수 있습니다.



(a) Inception module, naïve version (b) Inception module with dimension reductions

[그림4] (a) 1 x 1 convolution 필터 적용 전, (b) 적용 후

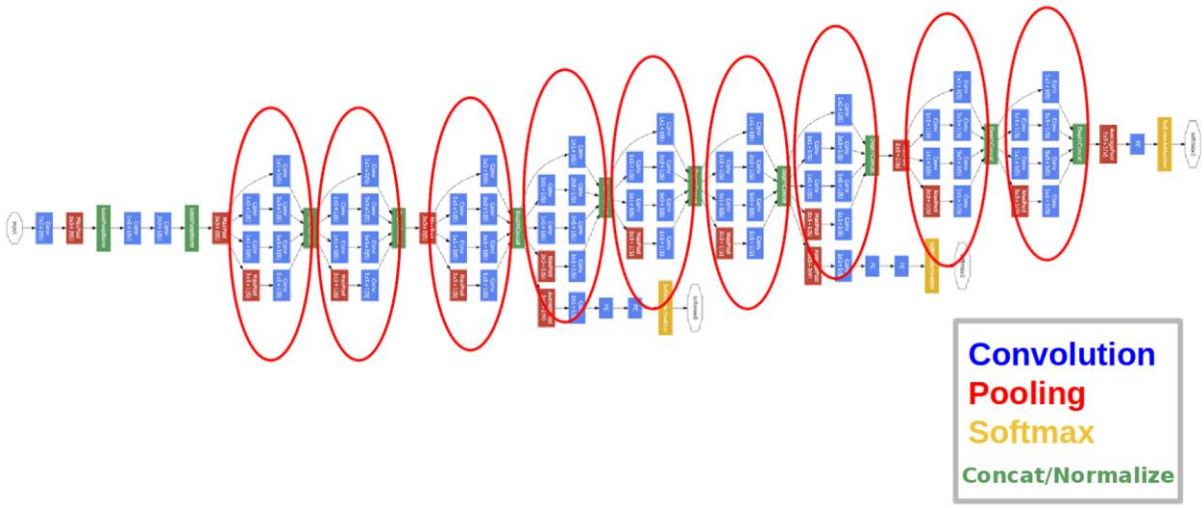
Inception 모듈의 원래 목적은 다양한 크기의 convolution 필터를 통해 feature를 효율적으로 뽑아내는 것입니다. 하지만 이 경우, convolution 연산이 병렬적으로 3번 진행되기 때문에 연산량이 너무 많다는 문제를 가지고 있었습니다. 따라서 입력 이미지의 convolution 연산 전에 1x1 필터를 배치하여, 차원 축소를 통해 연산량을 감소시킵니다. 즉, convolution 계층의 입력 이미지의 차원을 축소시켜, 자연스럽게 필터의 차원을 축소시키는 원리입니다. 그림 4-(a)는 맨 처음 제안된 inception 모듈이며, 그림 4-(b)는 차원 축소를 위한 1x1 filter가 추가된 inception 모듈입니다. Max-pooling은 계층 이전이 아닌 계층 이후에 차원 축소가 진행됩니다.



[그림5] 1 x 1 convolution 필터 원리

1 x 1 convolution의 그림은 위의 그림과 같습니다. 1 x 1 convolution의 목적은 차원 축소이므로, 필터의 개수는 입력 이미지의 채널 수보다 항상 작게 설정됩니다. 이로 인해 입력 이미지와 출력 이미지의 가로, 세로 길이는 같지만, 채널이 감소된 이미지가 생성되는 것입니다.

이 1 x 1 convolution의 의미에 대해 좀 더 깊게 생각해보면, 1 x 1 필터는 오로지 채널 간의 관계에 대한 연산을 수행한다고 볼 수 있습니다. 즉, 채널 간의 특징을 추출하는 것입니다. 이는 결국 차원을 축소시키는 대신에 차원 간의 정보를 요약하는 것으로 볼 수 있습니다. PCA가 다차원의 데이터에서 분산이 최소화되는 방향으로 차원을 조정하는 것이라면, 1 x 1 convolution 필터는 채널 간의 관계를 최소화하여, 각 이미지 채널이 가진 정보의 중첩을 최소화하는 것으로 볼 수 있습니다.



[그림6] GooLeNet 구조

GooLeNet은 이러한 inception 모듈을 사용하여, VGG 네트워크보다 더 깊은 층을 쌓으면서, 연산량을 대폭 감소시켰습니다. 위의 그림은 GooLeNet의 아키텍쳐로서, 총 22개의 convolution 계층과 fully-connected 계층으로 구성되어 있습니다. Pytorch에서 제공하는 pre-trained 된 모델들을 사용하여 파라미터 개수의 차이도 쉽게 확인할 수 있습니다.

```
import torch
import torchvision
from torchsummary import summary

USE_CUDA = torch.cuda.is_available()
DEVICE = torch.device("cuda" if USE_CUDA else "cpu")

model_VGG16 = torchvision.models.vgg16().to(DEVICE)
model_googlenet = torchvision.models.googlenet().to(DEVICE)

summary(model_VGG16.to(DEVICE), input_size = (3,224,224), device = "CUDA")
summary(model_googlenet.to(DEVICE), input_size = (3,224,224), device = "CUDA")
```

VGG16

```
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 218.78
Params size (MB): 527.79
Estimated Total Size (MB): 747.15
----------------------------------------------------------------
```

GoogLeNet

```
Total params: 13,004,888
Trainable params: 13,004,888
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 94.25
Params size (MB): 49.61
Estimated Total Size (MB): 144.43
----------------------------------------------------------------
```

[그림7] VGG16과 GooLeNet의 총 파라미터 수 비교

VGG16네트워크는 총 파라미터의 개수가 약 1억 3800만 개인데 반해, GoogLeNet은 약 1300만 개로서 10분의 1 수준인 것을 알 수 있습니다. GoogLeNet은 약 1000개의 클래스에 대하여 학습 데이터 210만장, 테스트 데이터 6만장으로 구성된 ImageNet 데이터를 사용하여 정확도를 평가하는 대회인 ILSVRC 2014대회에서 VGG를 제치고 우승하여, 공식적인 성능을 인정받았습니다.

4. Experiments(Code Review)

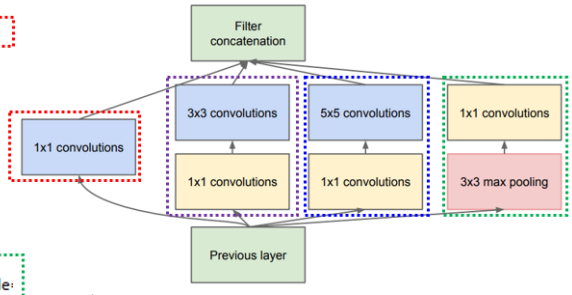
(1) 일반 Convolution 계층

먼저 inception 모듈 구현의 편의성을 위해 Convolution - Batch Normalization - ReLU의 일반적인 conv-olution 계층을 하나의 함수로 구현합니다

```
class BasicConv2d(nn.Module):

    def __init__(self, in_channels, out_channels, **kwargs):
        super(BasicConv2d, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, bias=False, **kwargs)
        self.bn   = nn.BatchNorm2d(out_channels, eps=0.001)

    def forward(self, x):
        x = self.conv(x)
        x = self.bn(x)
        return F.relu(x, inplace=True)
```

(2) Inception 모듈

```
class Inception(nn.Module):

    def __init__(self, in_channels, ch1x1, ch3x3red, ch3x3, ch5x5red, ch5x5, pool_proj):

        super(Inception, self).__init__()
        self.branch1 = BasicConv2d(in_channels, ch1x1, kernel_size=1)
        self.branch2 = nn.Sequential(
            BasicConv2d(in_channels, ch3x3red, kernel_size=1),
            BasicConv2d(ch3x3red, ch3x3, kernel_size=3, padding=1)
        )
        self.branch3 = nn.Sequential(
            BasicConv2d(in_channels, ch5x5red, kernel_size=1),
            BasicConv2d(ch5x5red, ch5x5, kernel_size=3, padding=1)
        )
        self.branch4 = nn.Sequential(
            nn.MaxPool2d(kernel_size=3, stride=1, padding=1, ceil_mode
            BasicConv2d(in_channels, pool_proj, kernel_size=1)
        )
```



이해를 돕기 위해 우측의 inception 모듈의 구조에 해당하는 좌측의 코드에 표시를 하였습니다. 코드의 branch는 inception 모듈 내의 각 병렬 처리 convolution 과정을 의미합니다. __init__부분의 variable 값인 ch3x3red, ch5x5red는 각 convolution 수행 전, 1x1 convolution을 통해 축소된 채널의 출력 개수를 입력합니다.

(3) 순전파 과정

순전파 과정은 입력 이미지에 대하여 각 branch마다 연산을 진행하고 맨 아랫줄의 torch.cat 함수를 이용하여 각 branch의 결과를 취합합니다.

```
class Inception(nn.Module):

    def __init__(self, in_channels, ch1x1, ch3x3red, ch3x3, ch5x5red, ch5x5, pool_proj):

        super(Inception, self).__init__()

        self.branch1 = BasicConv2d(in_channels, ch1x1, kernel_size=1)

        self.branch2 = nn.Sequential(
            BasicConv2d(in_channels, ch3x3red, kernel_size=1),
            BasicConv2d(ch3x3red, ch3x3, kernel_size=3, padding=1)
        )

        self.branch3 = nn.Sequential(
            BasicConv2d(in_channels, ch5x5red, kernel_size=1),
            BasicConv2d(ch5x5red, ch5x5, kernel_size=5, padding=2)
        )

        self.branch4 = nn.Sequential(
            nn.MaxPool2d(kernel_size=3, stride=1, padding=1, ceil_mode=True),
            BasicConv2d(in_channels, pool_proj, kernel_size=1)
        )
    def _forward(self, x):
        branch1 = self.branch1(x)
        print(branch1.size())
        branch2 = self.branch2(x)
        print(branch2.size())
        branch3 = self.branch3(x)
        print(branch3.size())
        branch4 = self.branch4(x)
        print(branch4.size())

        outputs = [branch1, branch2, branch3, branch4]
        return outputs

    def forward(self, x):
        outputs = self._forward(x)
        return torch.cat(outputs, 1)
```

(4) 결과 확인하기

구현한 inception 모듈의 결과를 확인하기 위해 이미지를 순전파 시키겠습니다.

```
from PIL import Image
from torchvision import transforms
input_image = Image.open(filename)
input_image.resize((224, 224))
```

```
In [23]: input_image.resize((224, 224))
Out[23]:
```

```
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
input_tensor = preprocess(input_image)
input_batch = input_tensor.unsqueeze(0)
```

각 branch들의 결과를 확인하기 위해 print 함수를 추가하고 병합이 일어나는 과정을 추적합니다.

아래 그림의 Inception함수의 파라미터 의미는 다음과 같습니다

```
Inception(3, 64, 96, 128, 16, 32, 32)(input_tensor.unsqueeze(0))
```

이미지는 3 x 224 x 224 크기의 이미지이기 때문에 입력 채널은 3
branch1(1x1)의 출력 채널은 64
branch2(3x3)의 축소된 채널은 96, 출력 채널은 128
branch3(5x5)의 축소된 채널은 16, 출력 채널은 32
branch4(max-pooling)의 축소된 채널은 32

```
In [44]: Inception(3, 64, 96, 128, 16, 32, 32)(input_tensor.unsqueeze(0))
branch1 :  torch.Size([1, 64, 224, 224])
branch2 :  torch.Size([1, 128, 224, 224])
branch3 :  torch.Size([1, 32, 224, 224])
branch4 :  torch.Size([1, 32, 224, 224])
```

```
In [48]: output = Inception(3, 64, 96, 128, 16, 32, 32)(input_tensor.unsqueeze(0))
    ...: output.size()
    ...:
Out[48]: torch.Size([1, 256, 224, 224])
```

In [44]에서 각 branch에서 출력된 채널의 크기가 함수에 입력한 파라미터와 일치하며, 최종 병합된 이미지의 채널은 $64+128+32+32=256$이 되는 것을 확인할 수 있습니다. 각 branch의 출력 이미지 크기는 3x3 필터는 padding 1, 5x5 필터는 padding 2를 적용하여 출력 이미지의 크기가 통일되도록 조정합니다.

(5) 일반 Convolution과 Inception 모듈의 비교

일반적으로 inception 모듈의 차원 축소 효과는 입력 이미지의 채널 수가 많을 때, 그 효과를 발휘합니다. 이를 증명하기 위해 입력 이미지의 크기, 출력 이미지의 크기, 출력 채널 수, 필터 사이즈를 동일하게 한 상태에서, 입력 이미지의 차원을 다르게 하여 inception 모듈과 일반 convolution의 파라미터 수 비교를 통해 차원 축소의 효과를 정량적으로 평가하였습니다.

실험의 변화 조건은 일반 convolution의 필터 사이즈(3x3, 5x5), 입력 이미지 채널 수(저차원 3, 고차원 256)입니다. 출력 채널은 256개로 고정하며, 입력 이미지 크기는 224 x 224로 고정하였습니다. inception 모듈과 일반 convolution의 파라미터 수 산출에는 pytorch의 부가기능인 torchsummary 함수를 사용하였습니다.

(5.1) 필터 사이즈 3x3

필터 사이즈가 3x3인 상태에서 입력 이미지의 채널 수가 3인 저차원 이미지의 inception 모듈과

일반 convolution의 파라미터 수 비교 결과, inception 모듈의 파라미터는 124,752개, 일반 convolution 파라미터는 7,424개로 차원 축소의 효과가 없었습니다.

In_channels = 3

Inception	BasicConv2d
Total params: 124,752	Total params: 7,424
Trainable params: 124,752	Trainable params: 7,424
Non-trainable params: 0	Non-trainable params: 0

고차원의 이미지에서는 Inception 모듈의 차원 축소 효과가 큰 것을 볼 수 있습니다. inception 모듈의 파라미터 수는 일반 convolution의 3분의 1 수준인 것을 볼 수 있습니다. 채널의 증가에 따른 파라미터의 증가량이 Inception 모듈이 일반 convolution에 비해 굉장히 낮은 것을 볼 수 있습니다. 이는 inception 모듈은 입력 이미지의 채널 수의 변화가 1x1 convolution에만 영향을 미치기 때문입니다.

In_channels = 256

Inception	BasicConv2d
Total params: 177,376	Total params: 590,336
Trainable params: 177,376	Trainable params: 590,336
Non-trainable params: 0	Non-trainable params: 0

(5.2) 필터 사이즈 5x5

In_channels = 3

Inception	BasicConv2d
Total params: 124,752	Total params: 19,712
Trainable params: 124,752	Trainable params: 19,712
Non-trainable params: 0	Non-trainable params: 0

In_channels = 256

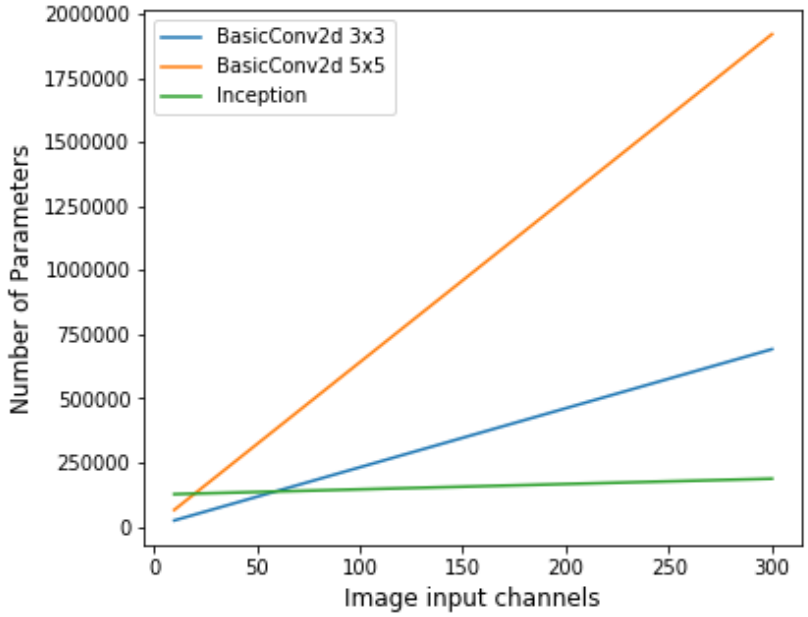
Inception	BasicConv2d
Total params: 177,376	Total params: 1,638,912
Trainable params: 177,376	Trainable params: 1,638,912
Non-trainable params: 0	Non-trainable params: 0

필터 사이즈가 5x5일 경우에도 저차원 이미지에서는 차원 축소 효과가 없지만, 고차원 이미지에서 차원 축소 효과가 큰 것을 볼 수 있습니다. 일반 convolution은 필터 사이즈가 커질수록 파라미터 수가 증가하기 때문에 같은 고차원의 이미지라도 일반 convolution의 필터 사이즈가 큰 경우, inception 모듈의 효과를 극대화할 수 있습니다.

(5.3) 입력 이미지 채널 변화에 따른 파라미터 수 변화

필터 사이즈에 따라 입력 이미지 채널 변화에 따른 파라미터 수의 변화를 측정하면, inception 모듈이 일반 convolution보다 효과를 볼 수 있는 지점을 찾아낼 수 있습니다. 이미지의 출력 채널이 256일 때, inception 모듈이 효과를 볼 수 있는 입력 이미지의 채널 개수의 지점이 어디인지 분석하였습니다.

Changes in number of parameters according to input image channel change



Inception 모듈의 입력 이미지 채널 증가에 따른 파라미터 수 증가량은 일반 Convolution에 비해 굉장히 낮은 것을 알 수 있습니다. 3x3 convolution과 비교 시 입력 이미지의 채널이 약 60개인 시점부터 inception 모듈이 효과를 보기 시작합니다. 5x5 convolution은 입력 이미지의 채널이 약 10개부터인 지점부터 효과를 볼 수 있습니다.

입력 채널의 개수에 따른 파라미터 수의 변화 그래프는 inception 모듈을 활용하여 커스텀 CNN 모델을 구현할 때, 좋은 인사이트를 제공할 수 있습니다. 이미지의 입력 및 출력 채널을 설계할 때, inception 모듈이 효과적일 것인지, 일반적인 convolution이 효과적일 것인지에 대한 판단을 제공할 수 있을 것입니다. 또한 inception 모듈 무조건 쓰는 것이 아니라, 맨 하단부의 가장 많은 feature map을 추출하고, 가장 연산량이 많은 convolution 계층들에 대하여 일부만 대체하는 방식으로 사용한다면, 연산량이 적고 성능이 좋은 CNN 모델을 만들 수 있을 것입니다.

5. Conclusion

지금까지 GoogLeNet을 차원 축소의 관점에서 리뷰하고, inception 모듈을 구현하여 차원 축소에 대한 효과에 대하여 직접 구현하고 분석하였습니다. 또한 inception 모듈을 효율적으로 활용하기 위해서 입력 및 출력 이미지의 채널 변화에 따른 파라미터 수를 측정하여, 일반 convolution을 대체할 수 있는 지점을 제공하였습니다. 현재 진행 중인 제 연구에서 CNN 기반의 모델을 사용하고 있는데, 연산량을 줄이고, 성능을 유지할 수 있는 방안을 고민하면서, inception 모듈이라는 주제를 선정하게 되었습니다. 본 리뷰 논문을 공부하면서 inception 모듈의 구조를 반드시 사용하지 않더라도, 1x1 convolution을 사용한 motivation을 응용한다면 제 연구에 충분히 활용될 수 있을 것이라는 생각을 하였습니다.