과제 2

자동차IT융합학과 20175330 현석종

과제 내용

Pytorch(torchvision)에서 제공되는 Pre-trained Model 성능 비교

-성능 비교에 사용할 데이터셋 : ImageNet Validation Set (다운로드는 직접 하시기 바랍니다.)

-비교해야하는 Pre-trained Model

- AlexNet
- VGG16
- ResNet18
- GoogLeNet

각 모델의 간단한 특징을 이해 및 조사하고, 제시된 데이터셋을 사용하여 모델의 성능을 비교합니다.

Task는 Classification이므로, 비교 지표는 Top-1 Accuracy(정확도)로 합니다.

목차

1. Dataset

ImageNet Validation Set (ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012))

이용 방법 :ImageNet (http://image-net.org/index)에서 로그인한 뒤, 이메일을 통해 dataset 다운로드 권한 요청을 하여 dataset을 다운 받습니다.

사용한 dataset은 2012년 버전입니다.(ILSVRC2012)

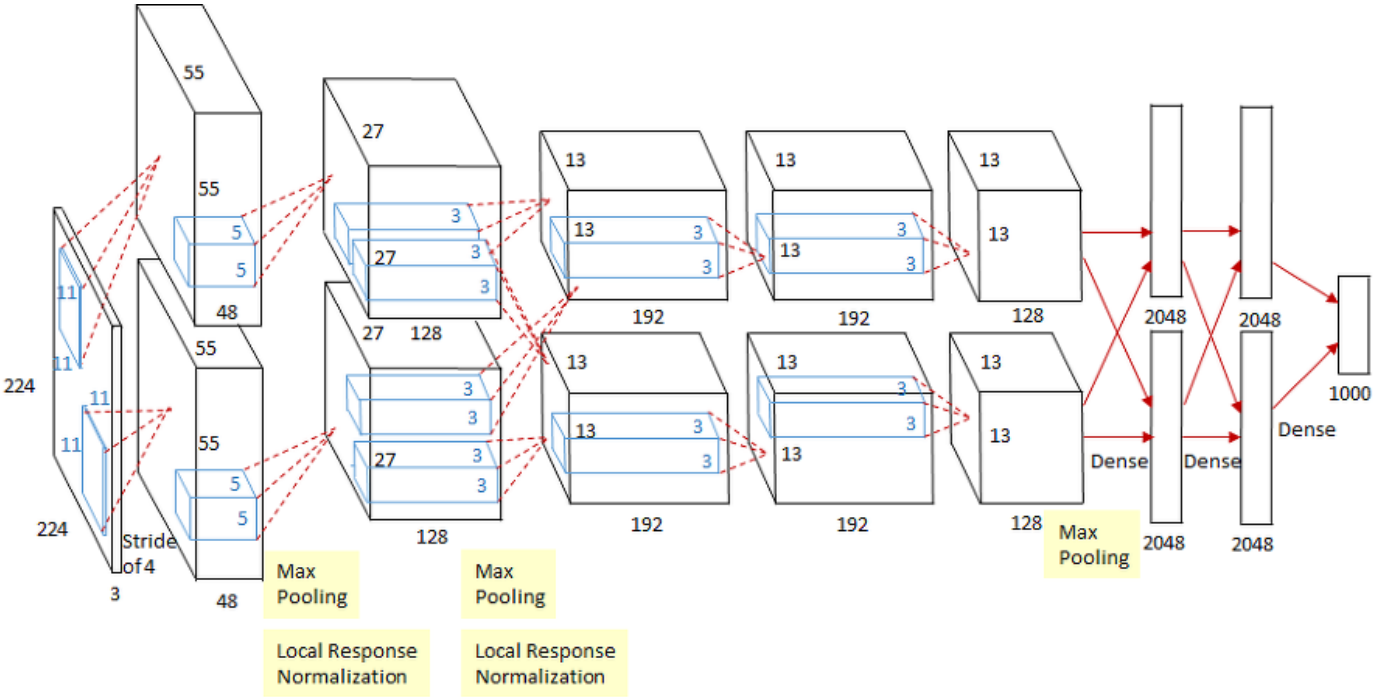
ILSVRC 2012 validation 데이터셋은 총 1000개의 카테고리로 분류되어 있으며 5만개의 이미지로 구성되어있습니다.train 데이터셋의 경우 120만개의 이미지로 구성되어 있습니다. label데이터는 이미지와 별도로 다운받아 사용할 수 있습니다.

link: http://image-net.org/challenges/LSVRC/2012/2012-downloads (http://image-net.org/challenges/LSVRC/2012/2012-downloads)

2. Pre-trained Model

1. AlexNet

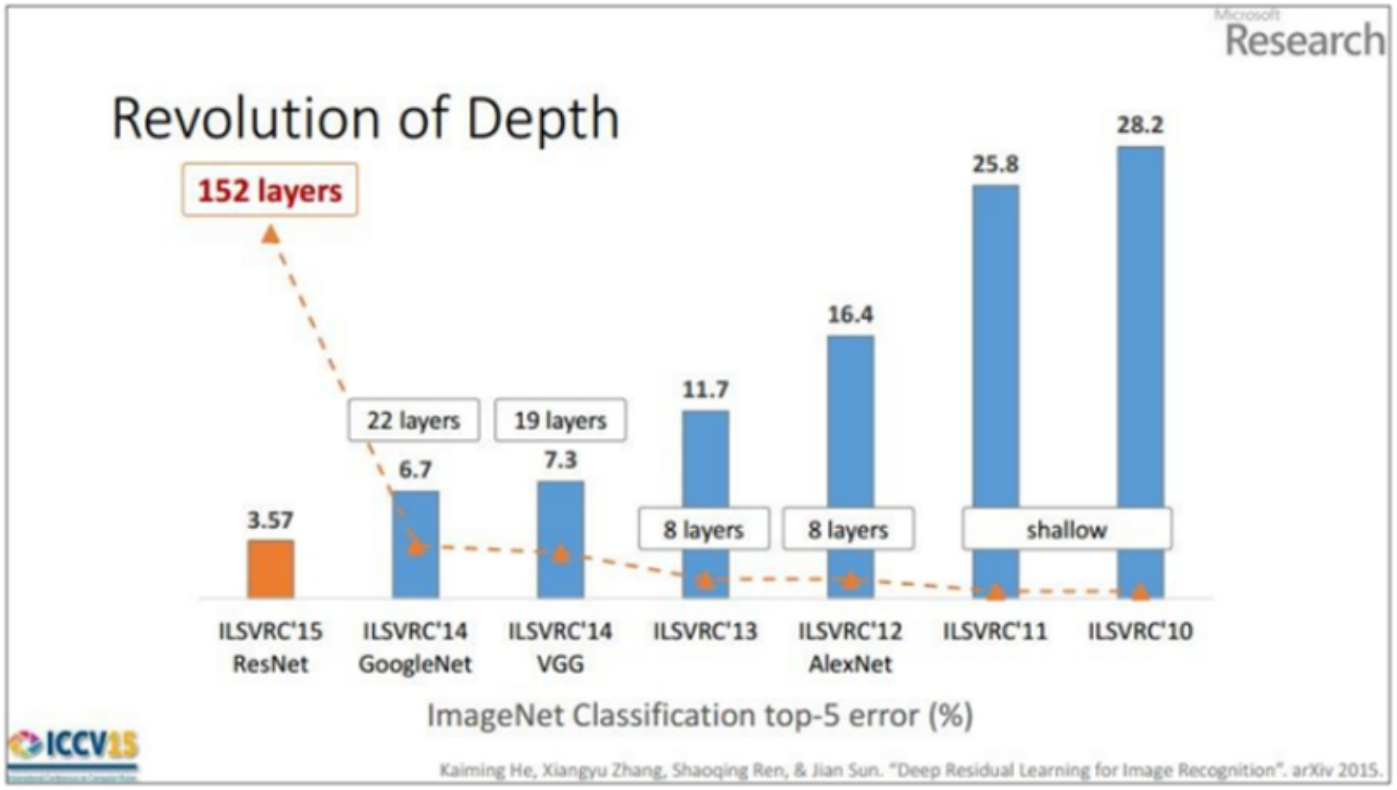
AlexNet은 2012년에 개최된 ILSVRC(ImageNet Large Scale Visual Recognition Challenge) 대회의 우승을 차지한 컨볼루션 신경망(CNN) 구조입니다.



AlexNet의 구조는 위 이미지와 같은 구조로 되어 있으며 2개의 GPU로 병렬 연산을 수향하기 위해서 병렬적인 구조로 설계되었다는 점이 이전 model에 있어 가장 큰 변화입니다.

VGG16

VGGNet은 옥스포드 대학의 연구팀 VGG에 의해 개발된 모델로써, 2014년 이미지넷 이미지 인식 대회에서 준우승을 한 모델입니다. 16이라는 숫자는 16개의 층으로 구성된 모델을 의미 합니다.



위 그림을 참고했을 때, VGGNet 모델부터 네트워크의 깊이가 깊어진 것을 확인할 수 있습니다.

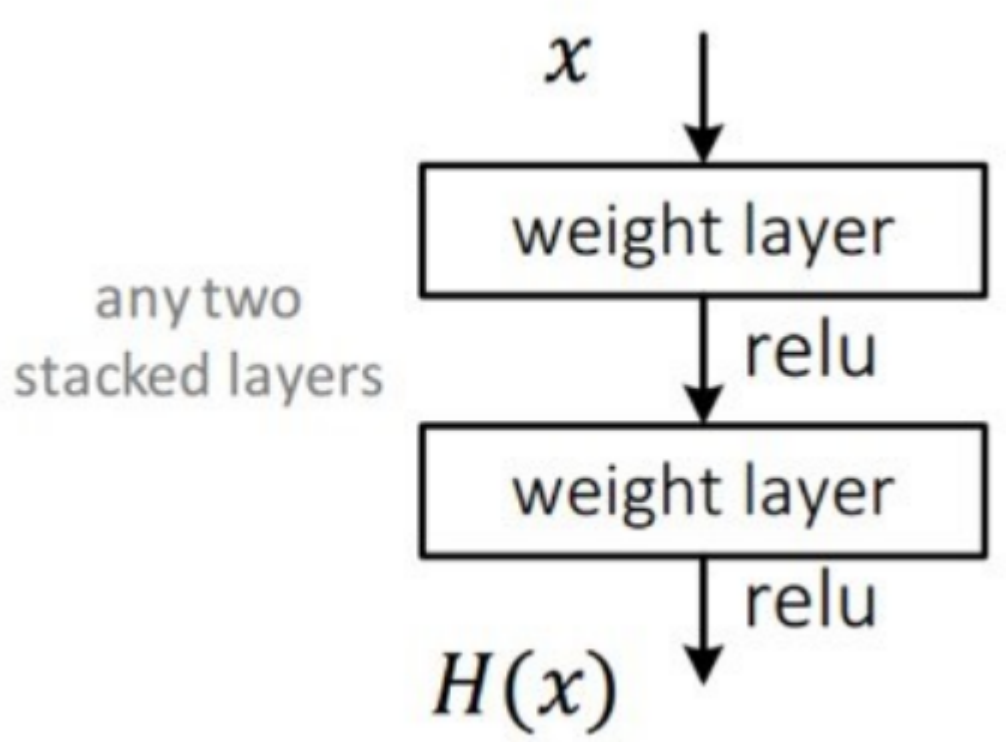
2012,2013년 우승 모델은 8개의 층으로 구성되었던 반면 2014년의 VGGNet(VGG19)는 19개의 층으로 구성되었음을 알 수 있습니다.
따라서 네트워크의 깊이가 깊어질 수록 성능이 좋아졌음을 위 그림을 통해 확인할 수 있습니다.

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**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

VGG의 구조는 위 표에 나타난 것처럼 이루어져 있으며, 이 중 D구조가 VGG16의 구조입니다. 표에서 확인할 수 있듯이, 2개 또는 3개의 CONV층과 1개의MAXPool층이 하나의 block으로 구성되어 있습니다. 이러한 단순한 구조가 VGG가 인기를 끄는 이유가 되었습니다.

ResNet18

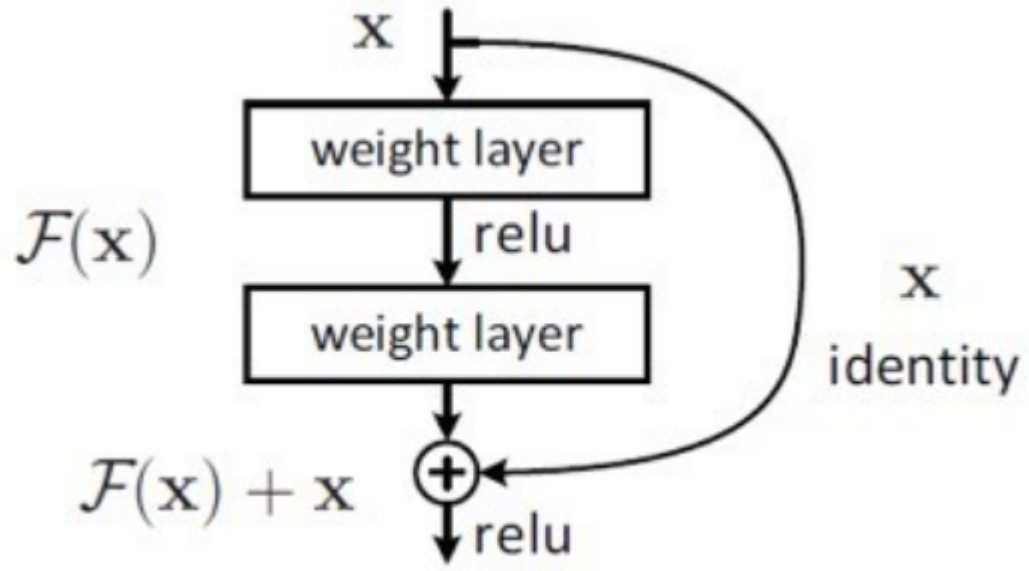
ResNet의 핵심 기술은 Residual Learning입니다. Residual Learning이란 VGGNet과 같은 기존 방식으로는 어느 정도 이상의 layer수를 넘어서게 되면 결과가 나빠지는 문제를 해결하기 위함입니다. 대표적인 문제점으로는 graduent vanishing/exoloding과 degradation을 꼽을 수 있습니다.



위 그림은 입력X를 받아 2개의 layer를 거쳐 H(x)를 도출하며 이는 다음 layer의 입력으로 적용되는 구조입니다.

위와 같은 구조의 일반적인 CNN구조로 깊이를 깊게하여 학습을 진행하게 되면 앞에 언급했던 문제들이 발생하게 됩니다.

따라서 ResNet은 layer의 입력을 layer의 출력에 바로 연결 시키는 skip connection을 적용합니다.



위 그림과 같이 출력 H(x)가 F(x)+x로 변경됩니다. 이러한 과정에서 F(x)=H(x)-x고 F(x)를 학습한다는 건 나머지인 residual을 학습한다는 것입니다.

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				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
		[3×3, 512]	[3×3, 512]	[1×1, 512]	[1×1, 512]	[1×1, 512]

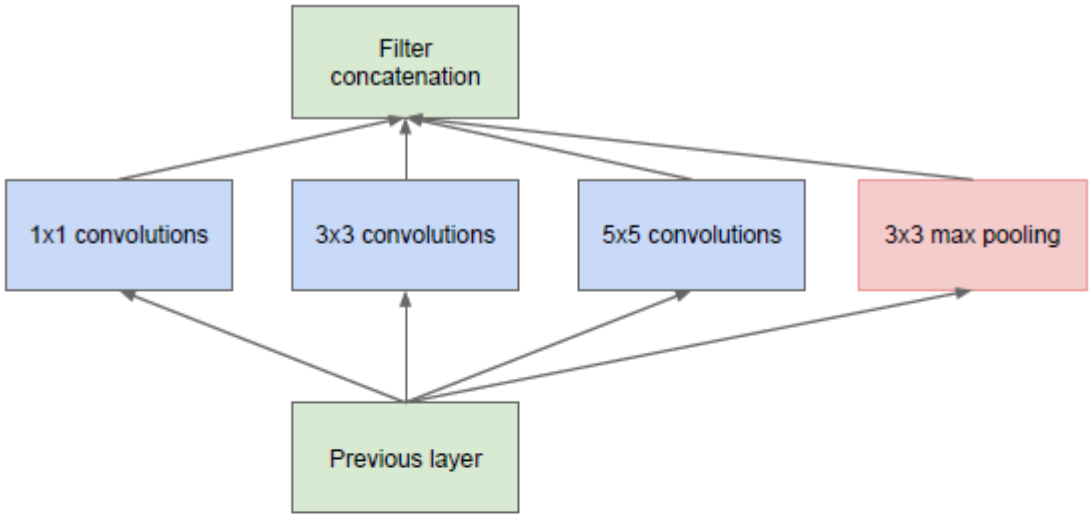
GoogLeNet

GoogLeNet은 2014년 이미지넷 이미지 인식 대회(ILSVRC)에서 VGGNet(VGG19)을 이기고 우승을 차지한 알고리즘입니다. GoogLeNet은 22개의 층으로 구성되어 있습니다. 이름에서도 알 수 있듯이 구글이 이 알고리즘 개발에 참여하였습니다.

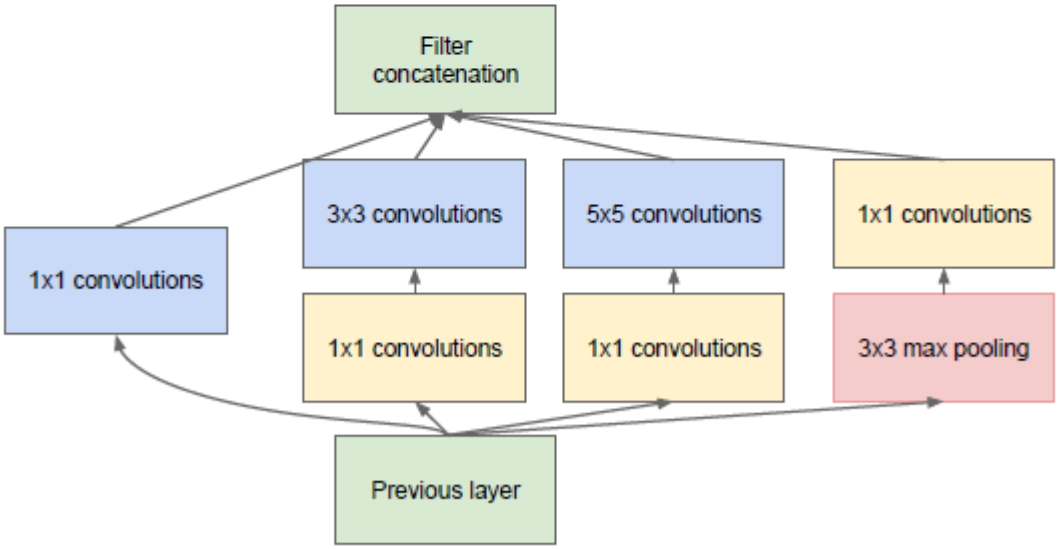
GoogLeNet의 특징은 크게 3개로 볼 수 있습니다.

1. 1 X 1 컨볼루션
GoogLeNet에서 1 x 1 컨볼루션은 특성맵의 갯수를 줄이는 목적으로 사용됩니다. 특성맵의 갯수가 줄어들면 그만큼 연산량이 줄어들기 때문입니다.

2. Inception 모듈
Inception모듈은 GoogLeNet의 가장 핵심적인 부분이라고 할 수 있습니다.



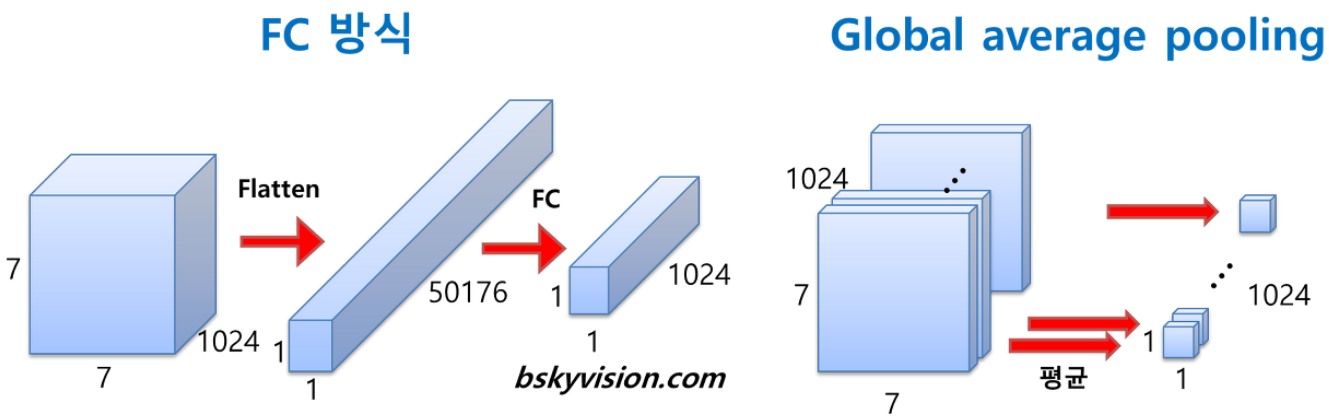
(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

위 그림을 참고했을 때, 노란색 블럭으로 표현된 1x1 컨볼루션을 제외한 naive 버전을 살펴보면, 이전 층에서 생성된 특성맵을 1x1 컨볼루션, 3x3 컨볼루션, 5x5 컨볼루션, 3x3 최대풀링해준 결과 얻은 특성맵들을 모두 함께 쌓아줍니다. AlexNet, VGGNet 등의 이전 CNN 모델들은 한 층에서 동일한 사이즈의 필터커널을 이용해서 컨볼루션을 해줬던 것과 차이가 있습니다. 따라서 좀 더 다양한 종류의 특성이 도출된다는 것을 알 수 있습니다.

3. global average pooling



AlexNet, VGGNet 등에서는 fully connected (FC) 층들이 망의 후반부에 연결되어 있습니다. 그러나 GoogLeNet은 FC 방식 대신에 global average pooling이란 방식을 사용합니다. global average pooling은 전 층에서 산출된 특성맵들을 각각 평균낸 것을 이어서 1차원 벡터를 만들어주는 것입니다. 1차원 벡터를 만들어줘야 최종적으로 이미지 분류를 위한 softmax 층을 연결해줄 수 있기 때문입니다.

3. 실습

이용한 코드는 pytorch 공식 사이트와 여러 블로그와 깃허브를 참조하여 작성하였습니다.

In [16]:
```
import torch
import torchvision
import torch.utils.data as data
import torchvision.transforms as transforms


if __name__ == "__main__":
    device = torch.device('cuda:0' if torch.cuda.is_available() else
'cpu')
    normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                     std=[0.229, 0.224, 0.225])

    transform = transforms.Compose(
        [transforms.Resize(256),
         transforms.CenterCrop(224),
         transforms.ToTensor(),
         normalize,
         ])

    test_set = torchvision.datasets.ImageNet(root="/home/ssac16/차량지
능기초/imagenet", transform=transform, split='val')
    test_loader = data.DataLoader(test_set, batch_size=100, shuffle=T
rue, num_workers=4)

    alexnet_model = torchvision.models.alexnet(pretrained=True).to(de
vice)
    vgg16_model = torchvision.models.vgg16(pretrained=True).to(device
)
    resnet18_model = torchvision.models.resnet18(pretrained=True).to(
device)
    googlenet_model = torchvision.models.googlenet(pretrained=True).t
o(device)

    model_name_list=["AlexNet","VGG16","ResNet18","GoogLeNet"]
    model_list=[]

    model_list.append(alexnet_model.eval())
    model_list.append(vgg16_model.eval())
    model_list.append(resnet18_model.eval())
    model_list.append(googlenet_model.eval())

    correct_top1 = 0
    correct_top5 = 0
    total = 0



    for i,j in enumerate(model_name_list):
        model=model_list[i]
        print("<<<",j,">>>")

        with torch.no_grad():
            for idx, (images, labels) in enumerate(test_loader):

                images = images.to(device)      # [100, 3, 224, 224]
                labels = labels.to(device)      # [100]
```

```
                outputs = model(images)

                # ------------------------------------------------
--------------------------
                # rank 1
                _, pred = torch.max(outputs, 1)
                total += labels.size(0)
                correct_top1 += (pred == labels).sum().item()

                # ------------------------------------------------
--------------------------
                # rank 5
                _, rank5 = outputs.topk(5, 1, True, True)
                rank5 = rank5.t()
                correct5 = rank5.eq(labels.view(1, -1).expand_as(rank
5))

                # ------------------------------------------------
--------------------------
                for k in range(6):
                    correct_k = correct5[:k].reshape(-1).float().sum(
0, keepdim=True)

                correct_top5 += correct_k.item()

    #             print("step : {} / {}".format(idx + 1, len(test_se
t)/int(labels.size(0))))
    #             print("top-1 percentage :  {0:0.2f}%".format(correc
t_top1 / total * 100))
    #             print("top-5 percentage :  {0:0.2f}%".format(correc
t_top5 / total * 100))

        print(" top-1 percentage :  {0:0.2f}%".format(correct_top1 /
total * 100))
        print(" top-5 percentage :  {0:0.2f}%".format(correct_top5 /
total * 100))
```

```
<<< AlexNet >>>
 top-1 percentage :  56.52%
 top-5 percentage :  79.07%
<<< VGG16 >>>
 top-1 percentage :  64.05%
 top-5 percentage :  84.73%
<<< ResNet18 >>>
 top-1 percentage :  65.96%
 top-5 percentage :  86.18%
<<< GoogLeNet >>>
 top-1 percentage :  66.91%
 top-5 percentage :  87.01%
```

4. 고찰

위 실습을 통해 결과를 도출해 보았을 때, GoogLeNet , ResNet18 , VGG16 , AlexNet 순으로 성능이 좋은 것을 확인할 수 있습니다. 각 모델들이 논문을 통해 발표된 순서는 AlexNet, VGG, GoogLeNet, ResNet 으로 알고 있는데 데이터를 학습하거나 Fine Tuning한 것이 아니기 때문에 결과가 이와 상관관계 없이 도출된 것 같습니다.