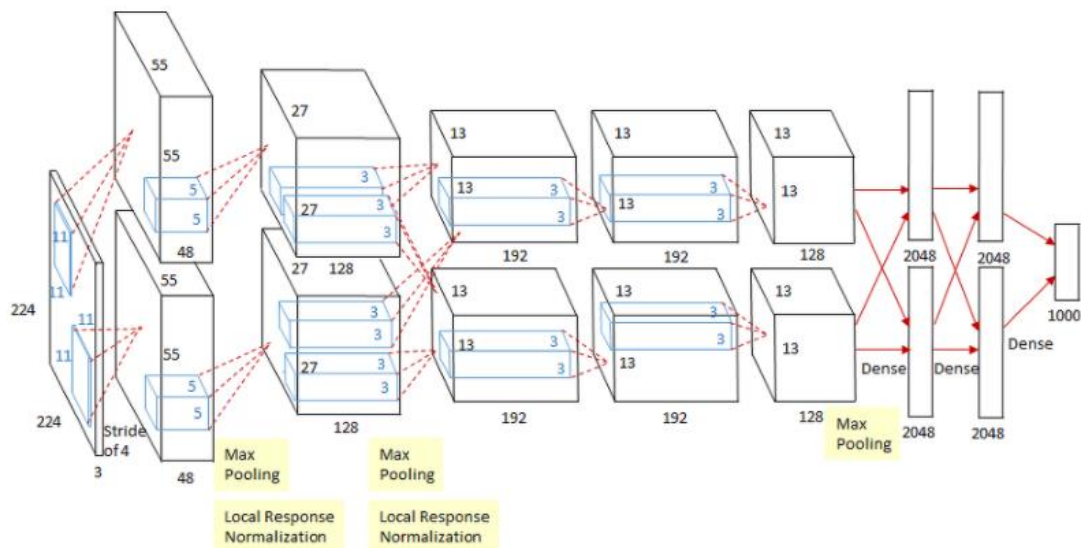


모델성능비교

AlexNet

AlexNet은 2012년에 개최된 ILSVRC 대회에서 우승을 차지한 CNN의 일종입니다. 이전에는 객체 인식 대회에서 여러 알고리즘과 방법, 기술들이 사용되었고, 이전만 해도 딥러닝은 이론만 상황하고 성능은 좋게 여겨지지 않는 기술이었는데 당시 대세였고 성능이 좋던 모델들을 제치고, 딥러닝 기술이 컴퓨터 비전에서 최고의 성능을 낼 수 있다고 증명을 했습니다. 224x224크기의 RGB 3 channel image를 input으로 사용한 구조를 사용했습니다. 아래의 구조를 보면 2갈래로 나뉘어져서 연산을 수행하며, 중간중간 결과를 공유하는 것을 알 수 있습니다. 이와 같은 이유는 그 당시에는 하나의 GPU로 실험을 하기에는 메모리가 부족하여 아래와 같은 구조를 가지게 되었습니다.

구조



활성화 함수로 ReLU를 사용합니다. 기존까지는 sigmoid를 필두로 tanh가 사용되는 추세였는데, AlexNet가 ReLU를 사용한 결과 학습, 예측 속도가 훨씬 빠르고, 정확도도 좋아졌다고 합니다. 이전함수에 비해 학습속도가 최고 6배까지 증가했습니다.

Local Response Normalization을 사용했습니다. 이 기법은 AlexNet에서 처음 사용하게 되었습니다. 활성화 함수를 적용시키기 전에 Normalization을 적용하여 함수의 결과값에서 더 좋은 일반화 결과를 만들었습니다. 이 기법을 사용하는 이유는 이미지의 특징을 부각시키기 위함입니다.

그리고 Pooling의 커널 사이즈를 stride보다 크게하는 Overlapping pooling를 사용하였습니다.

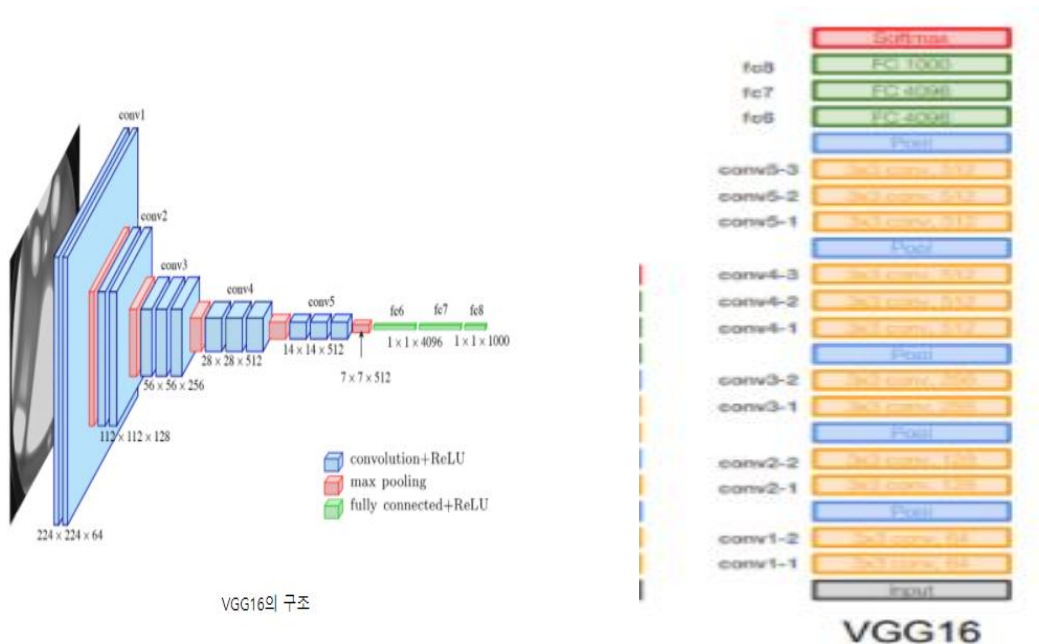
AlexNet은 총 8개의 층으로 구성되어 있습니다. 첫 5개 층은 Convolution, 그 뒤 3개 층은 Fully-Connected 층입니다. 각각의 층들은 하나의 이미지에 대해 독립적으로 특징을 추출하여 가중치를 조정함으로써 필터를 학습시킵니다.

VGG16

VGGNet은 옥스포드 대학에서 연구한 VGG팀에 의해 개발된 모델입니다. 2014년 ImageNet 이미지 인식 대회에서 준우승을 한 모델입니다. VGG16모델은 8개의 층을 가진 AlexNet과 유사한 모델입니다. 하지만 AlexNet처럼 병렬적 구조로 이루어 지지는 않았고 VGG는layer 수를 더 많이 사용합니다.

VGGNet는 네트워크의 깊이의 영향을 최대한 확인하고자 Conv 필터사이즈를 가장 작은 3x3 으로 고정했습니다. VGG가 발표한 논문에 따르면 연산하여 발생하는 파라미터의 개수가 줄어드는 효과와 ReLU 함수가 들어갈 수 있는 곳이 많아진다는 장점이 있어서 사용했다고 합니다. 파라미터의 개수가 줄어들수록 정규화를 할 때 이점을 얻을 수 있습니다. VGG16의 모든 레이어는 활성화 함수로 ReLU를 사용합니다.

구조

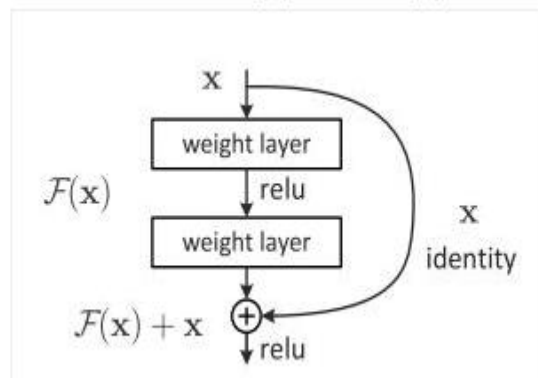
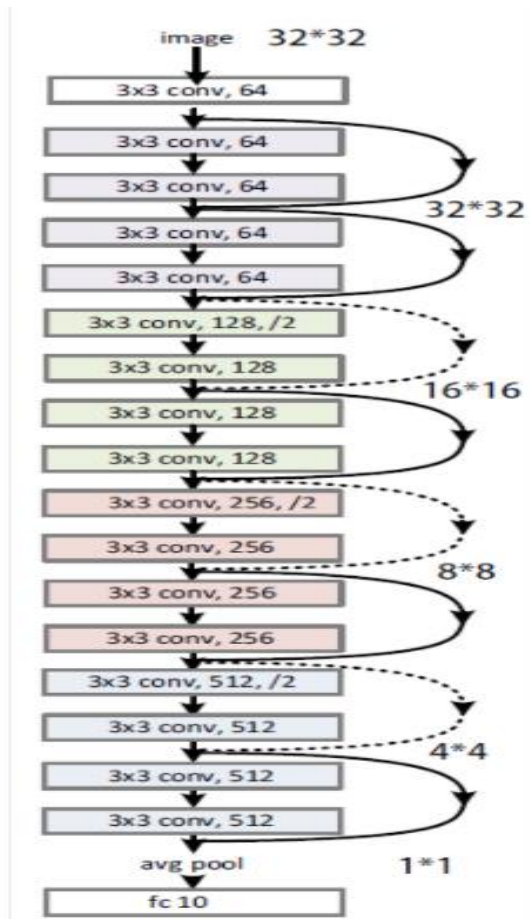


ResNet18

ResNet은 마이크로소프트사에서 개발한 알고리즘이고 2015년 ILSVRC에서 우승을 차지한 CNN 모델입니다. ResNet은 3x3 conv가 반복된다는 점에서 VGG-19의 구조를 뼈대로 하고 있다고 볼 수 있습니다.

ResNet팀은 망이 깊어질수록 어떤 결과가 나오는지에 대해 연구를 하였고 층이 개수에 대해 층

이 깊어질수록 결과가 더 좋게 나오는 것은 아니라고 확인할 수 있었습니다. 네트워크의 깊이가 깊어지면서 입력된 이미지가 갖고 있는 정보가 하위 층으로 갈수록 소실되게 되는데, 이러한 문제 때문에 깊이가 깊어지는 것이 모델에 있어 최선은 아니라고 설명했습니다.

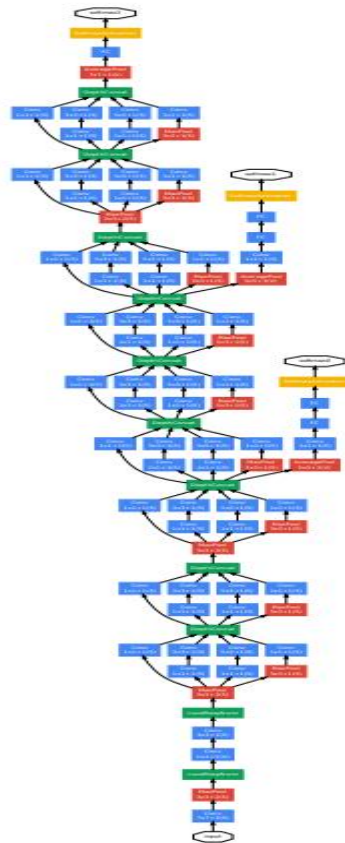


2개의 conv layer마다 옆으로 화살표가 빠져나간 뒤 합쳐지는 식으로 그려져 있습니다. 이러한 구조를 Shortcut이라고 합니다. 일반적으로 Shortcut으로는 identity shortcut, 즉 input feature map x 를 그대로 output에 더해주는 방식을 사용합니다.

Output feature map의 개수가 2배로 커질 때 마다 feature map의 가로, 세로 size는 절반으로 줄여주는 방식을 이용하고 있으며, 이 때는 pooling 대신 stride=2를 갖는 convolution 연산을 이용하는 점이 특징입니다. 이 경우, Shortcut에서도 feature map size를 줄여주어야 하며, 이 때는 identity shortcut 대신 projection shortcut을 이용합니다. 이러한 shortcut 구조를 통해 vanishing gradient에 학습을 수행할 수 있게됩니다.

GoogLeNet는 2014년 ILSVRC에서 VGGNet를 이기고 우승을 차지한 알고리즘 입니다. GoogLeNet는 22층으로 구성되어 있고 구글이 이 알고리즘을 만들었습니다. 총 22개의 layer로 구성되어 있으며 아래의 구조 모습처럼 굉장히 길고 복잡한 구조로 되어있습니다.

구조



googLeNet은 inception module라고 불리는 block 구조를 제안했습니다. 기존에는 하나의 layer안에 하나의 convolution, pooling연산으로 결합 했다면 inception module는 총 4가지 서로 다른 연산을 거친 뒤 feature map를 channel 방향으로 합쳐주고 다양한 convolution 연산을 섞어서 사용합니다. 여기에 3x3, 5x5 conv연산이 많은 연산량을 차지하기 때문에 두 연산 앞에 1x1conv연산을 추가하여 feature map 개수를 줄인 다음 다시 연산을 수행합니다. 이 덕에 inception module의 연산량이 크게 줄었습니다.

googLeNet는 inception module를 총 9번 쌓아 구성이 됩니다. 3번째와 6번째 모듈 뒤에 classifier를 추가로 붙여 총 3개의 classifier을 사용하고, 이를 Auxiliary Classifier라 부릅니다. 가장 뒷부분에 Classifier 하나가 존재하면 input과 가까운 쪽에는 gradient가 잘 전파되지 않을 수 있다고 생각하여 Network의 중간부분, 앞부분에 추가로 classifier을 붙여주었습니다.

Global Average Pooling을 이용하여 파라미터의 수를 크게 줄일 수 있었습니다. Global Average Pooling는 feature map의 모든 element의 평균을 구하여 하나의 node로 바꿔주는 연산을 뜻하고

feature map의 개수만큼 node를 출력하게 됩니다. 1024개의 node를 만든 뒤 class 개수 (ImageNet=1000)의 output을 출력하도록 하나의 Fully-Connected layer만 사용하여 classifier를 구성하였습니다. 그 덕에 AlexNet, ZFNet, VGG 등에 비해 훨씬 적은 수의 parameter를 갖게 되었습니다.

pytorch에서 제공되는 alexnet, vgg16, resnet18, googlenet 성능 비교

먼저 다운받아온 데이터셋을 가지고 옵니다.

```
[ ] import torchvision

testset = torchvision.datasets.ImageNet(root="/content/drive/MyDrive/imagenet", split='val')
```

Pytorch에서 제공하는 학습된 모델들을 가져 옵니다.

#alexnet

```
from torchvision.models import alexnet

model = alexnet(pretrained=True)

Downloading: "https://download.pytorch.org/models/alexnet-owt-4df8aa71.pth" to /root/.cache/torch/hub/checkpoints/alexnet-owt-4df8aa71.pth
100%  233M/233M [00:03<00:00, 70.0MB/s]
```

#vgg16

```
from torchvision.models import vgg16

model = vgg16(pretrained=True)

Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.cache/torch/hub/checkpoints/vgg16-397923af.pth
100%  528M/528M [00:02<00:00, 225MB/s]
```

#resnet18

```
from torchvision.models import resnet18


model = resnet18(pretrained=True)

Downloading: "https://download.pytorch.org/models/resnet18-5c106cde.pth" to /root/.cache/torch/hub/checkpoints/resnet18-5c106cde.pth
100%  44.7M/44.7M [00:00<00:00, 205MB/s]
```

#googlenet

```
] from torchvision.models import googlenet
```

```
model = googlenet(pretrained=True)
```

Downloading: "<https://download.pytorch.org/models/googlenet-1378be20.pth>" to /root/.cache/torch/hub/checkpoints/googlenet-1378be20.pth
100%  49.7M/49.7M [00:00<00:00, 82.5MB/s]

데이터 세트와 데이터를 로드하고 이를 모델에 넣어 top-1 accuracy를 측정합니다.

Batch_size는 128, num_worker은 2로 하고 실행을 했습니다

```
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=[0.5, 0.5, 0.5],
                                     std=[0.5, 0.5, 0.5])

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

    test_set = torchvision.datasets.ImageNet(root="/content/drive/MyDrive/imagenet", transform=transform, split='val')
    test_loader = data.DataLoader(test_set, batch_size=128, shuffle=True, num_workers=2)

    model = torchvision.models.alexnet(pretrained=True).to(device)
    model.eval()

    correct_top1 = 0
    correct_top5 = 0
    total = 0

    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()

            print("step : {} / {}".format(idx + 1, len(test_set)/int(labels.size(0))))
            print("top-1 percentage : {0:0.2f}%".format(correct_top1 / total * 100))

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

다음은 실행 결과값입니다

#alexnet

```
top-1 percentage : 39,51%  
step : 260 / 790,3333333333334  
top-1 percentage : 39,50%  
top-1 percentage : 39,50%
```

#vgg16

```
step : 259 / 259,320125  
top-1 percentage : 65,28%  
step : 260 / 790,3333333333334  
top-1 percentage : 65,28%  
top-1 percentage : 65,28%
```

#resnet18

```
top-1 percentage : 63,64%  
step : 260 / 790,3333333333334  
top-1 percentage : 63,64%  
top-1 percentage : 63,64%
```

#googlenet

```
step : 259 / 259,320125  
top-1 percentage : 68,22%  
step : 260 / 790,3333333333334  
top-1 percentage : 68,23%  
top-1 percentage : 68,23%
```

Top-1 accuracy

Googlenet(68.23%) > vgg16(65.28%) > resnet18(63.64%) > alexnet(39.50%)

이와 같은 순서로 정확도가 나왔습니다.