VGG

Very Deep Convolutional Networks for Large-Scale Image Recognition

서울과학기술대학교 국방인공지능응용학과
이찬호

Part 1

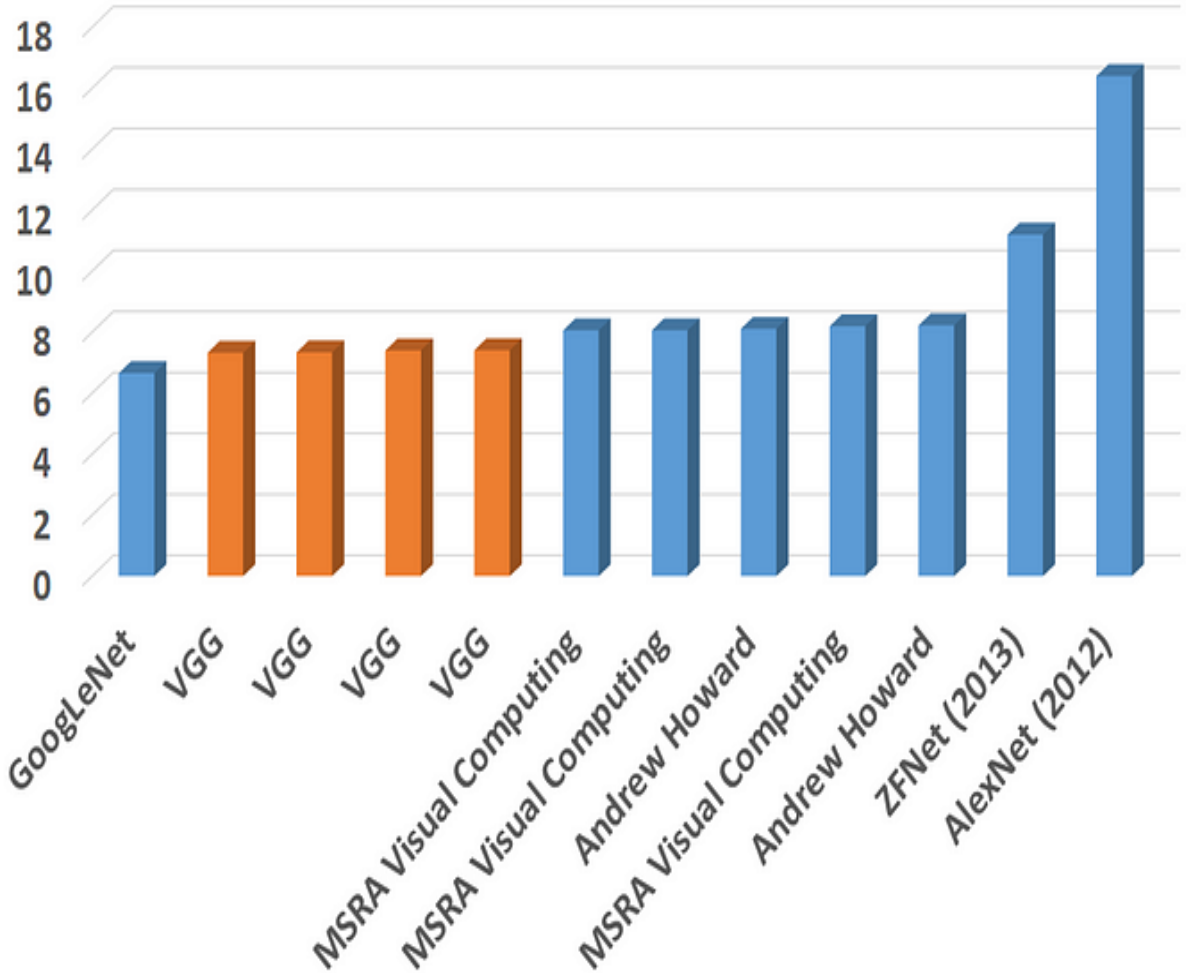
VGG

Part 1

VGG

Introduction

- VGGNet 등장 이전에 AlexNet이 등장하면서 이미지 분류에서 CNN 모델의 인기가 상승
- 그 당시 Deep layer라고는 8 layers였지만 ILSVRC 2014에 VGG(2위)와 GoogleNet(1위)이 등장하면서 더욱 깊은 layer(19, 22 layers)의 효과를 발표
- Very Deep Convolutional Networks for Large-Scale Image Recognition은 VGGNet을 발표해 CNN 구조 설계에 있어 네트워크의 depth가 정확도에 미치는 영향에 대해 실험함
- Depth를 늘리면서 vanishing/exploding gradient 문제 등 다양한 문제를 어떻게 해결했는지에 대해 설명함



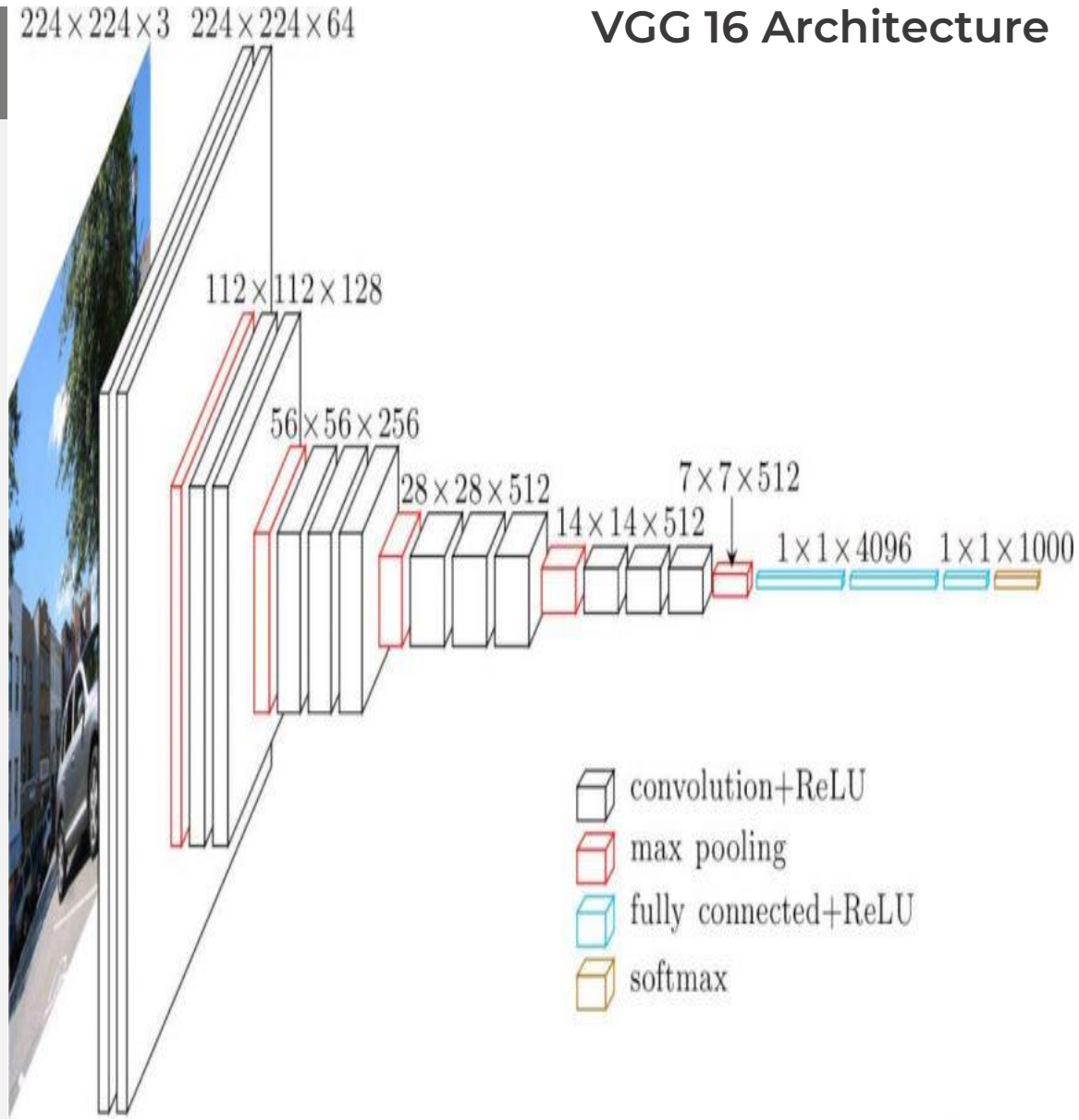
Part 1

VGG

ARCHITECTURE

- Input Image
  - 224 x 224 RGB
  - 전처리는 RGB 평균 값을 빼주는 것만 적용
- Conv layer
  - 3 x 3 Conv filter : 3 x 3 사이즈가 이미지 요소의 상하좌우 등을 파악 할 수 있는 최소한의 receptive field*이기 때문
  - 1 x 1 Conv filter : 차원을 줄이고 non-linearity를 증가를 위함
  - Stride : 1, Padding : 1
- Pooling layer
  - Conv layer 다음에 적용되며 총 5개의 max pooling layer
  - 2 x 2 사이즈, Stride : 2
- FC layer
  - 처음 두 FC layer는 4096 채널
  - 마지막 FC layer는 1000 채널
  - 모든 layer에 ReLU*를 사용, 마지막 layer에 Softmax 적용

VGG 16 Architecture



Part 1

VGG

CONFIGURATIONS

- Depth에 따라 모델 구조가 조금씩 변형
- 11 depth인 A 구조부터 19 depth인 E 구조까지 있음*
- Conv layer의 폭은 64부터 시작해서 max pooling layer를 통과할 때마다 2의 제곱만큼 커져 최대 512까지 커짐
- Depth가 늘어남에도 더 큰 Conv layer를 사용한 얕은 신경망 보다 오히려 파라미터 수가 줄어듦
- Conv3 : 3 x 3 Conv layer
- Conv1 : 1 x 1 Conv layer
- LRN : Local Response Normalization의 약자이며 AlexNet에 처음 도입된 기법으로 인접 화소들을 억제 시키며 특징을 부각시키기 위한 방법

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					

Table 2: **Number of parameters** (in millions).

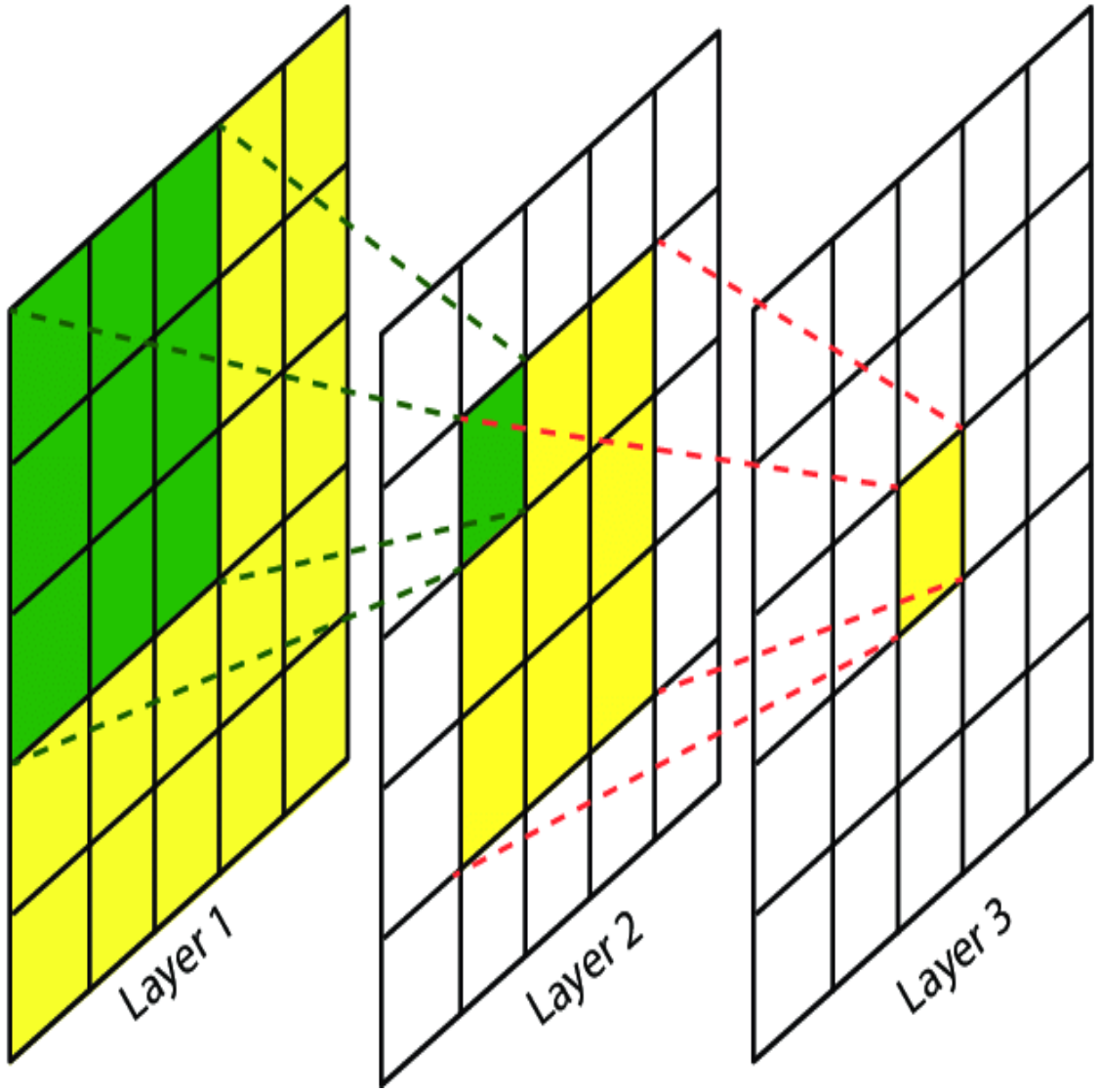
Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Part 1

VGG

DISCUSSION

- VGGNet 이전 모델들은 비교적 큰 Receptive Field를 갖는 11 x 11 filter나 7 x 7 filter를 사용
- VGGNet은 오직 3 x 3 filter만 사용해 성능 개선
- 3 x 3 filter 2개가 5 x 5 filter 1개의 성능을 내고 3 x 3 filter 3개가 7 x 7 filter 1개의 성능을 냄
- 3 x 3 filter 3개 사용 시 $3 \times 3^2 \times C^2 = 27 \times C^2$인 반면 7 x 7 filter 1개 사용 시 $1 \times 7^2 \times C^2 = 49 \times C^2$으로 7 x 7 filter 1개만 사용 했음에도 3 x 3 filter 3개 사용한 것보다 파라미터 수가 많고 수치적으로 보면 81%가 많음
- 3 x 3 filter 3개가 7 x 7 filter 1개보다 파라미터 수가 줄어 들어 마치 7 x 7에 일반화를 적용한 것과 같아 오버 피팅을 줄이는 효과도 있음



Part 1

VGG

TRAINING – Hyper Parameter

- Cost function
  - Multinomial logistic regression objective(Cross Entropy)
- Mini batch
  - 256 Size
- Optimizer
  - Momentum = 0.9
- Regularization
  - L2 = 5.10^{-4}, Dropout = 0.5
- Learning Rate
  - 10^{-2}(valid 향상 없으면 10^{-1} 감소)
- Pre-initialization
  - A 구조를 먼저 학습하고 다음 구조를 학습 할 때 A 구조에 학습된 layer를 사용해 최적의 초기값을 설정

TRAINING – Image Size(S 값)

- 이미지의 가로, 세로 중 더 작은 쪽을 224로 원 사이즈의 비율을 지키며 rescaling
- Rescale 된 이미지에서 랜덤으로 224 x 224 사이즈로 crop해서 input 사이즈를 맞춤
- 두 가지의 S 값 설정 방식이 있음
  - Single-scale training : S를 256 또는 384((256+512) / 2)로 고정, S가 384인 경우 학습 속도를 높이기 위해 S를 256으로 설정해 학습시킨 가중치 값들을 기반으로 S를 384로 설정해 다시 학습
  - Multi-scale training : S를 고정하지 않고, 256~512 값 중 랜덤하게 값을 설정, 이미지가 모두 같은 사이즈가 아니므로 랜덤하게 multi-scale로 학습하면 효과가 더 좋음 (속도 상의 이유로 S=384로 Pre-trained된 single-scale을 모델로 fine-tuning함)

VGG

TESTING

- Valid Set을 이용해 평가와 학습을 동시에 함
- 테스트 시에도 rescale함
- Q(테스트 이미지 scale 파라미터)가 S와 같을 필요는 없음
- Training 할 때의 구조와 Valid을 할 때 구조가 다름
- Valid 수행 시 FC layer를 Conv layer로 변경한 뒤 Uncropped 이미지에 적용 (첫 FC layer를 7 x 7 conv layer로, 마지막 두 FC layer를 1 x 1 Conv layer로 변경)*
- Uncropped 이미지 사용 시 input 사이즈에 따라 output 사이즈가 달라지는데 1 x 1 사이즈가 아닌 output feature map을 class score map이라 하며, class score map에 sum-pooled를 적용하며 이후, softmax를 거쳐 flipped 이미지와 original 이미지의 평균값을 통해 최종 score를 출력함

Part 2

VGG 구현

Part 2

VGG-16 & VGG-19

<VGG-16 구현>

```
def made_VGG16():
    model = Sequential()
    model.add(Conv2D(filters = 64, kernel_size = (3,3), padding = "same" ,activation="relu", input_shape = (224, 224, 3)))
    model.add(Conv2D(filters = 64, kernel_size = (3,3), padding = "same", activation = "relu"))
    model.add(MaxPooling2D(2))

    model.add(Conv2D(filters = 128, kernel_size = (3,3), activation = "relu", padding = "same"))
    model.add(Conv2D(filters = 128, kernel_size = (3,3), activation = "relu", padding = "same"))
    model.add(MaxPooling2D(2))

    model.add(Conv2D(filters = 256, kernel_size = (3,3), activation = "relu", padding = "same"))
    model.add(Conv2D(filters = 256, kernel_size = (3,3), activation = "relu", padding = "same"))
    model.add(Conv2D(filters = 256, kernel_size = (3,3), activation = "relu", padding = "same"))
    model.add(MaxPooling2D(2))

    model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = "relu", padding = "same"))
    model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = "relu", padding = "same"))
    model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = "relu", padding = "same"))
    model.add(MaxPooling2D(2))

    model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = "relu", padding = "same"))
    model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = "relu", padding = "same"))
    model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = "relu", padding = "same"))
    model.add(MaxPooling2D(2))

    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(4096, activation = "relu"))
    model.add(Dense(4096, activation = "relu"))
    model.add(Dense(1000, activation = "relu"))
    model.add(Dense(2, activation = "softmax"))
```

<VGG-19 구현>

```
def made_VGG19():
    model = Sequential()
    model.add(Conv2D(filters = 64, kernel_size = (3,3), padding = "same" ,activation="relu", input_shape = (224, 224, 3)))
    model.add(Conv2D(filters = 64, kernel_size = (3,3), padding = "same", activation = "relu", ))
    model.add(MaxPooling2D(2))

    model.add(Conv2D(filters = 128, kernel_size = (3,3), activation = "relu", padding = "same" ))
    model.add(Conv2D(filters = 128, kernel_size = (3,3), activation = "relu", padding = "same" ))
    model.add(MaxPooling2D(2))

    model.add(Conv2D(filters = 256, kernel_size = (3,3), activation = "relu", padding = "same" ))
    model.add(Conv2D(filters = 256, kernel_size = (3,3), activation = "relu", padding = "same" ))
    model.add(Conv2D(filters = 256, kernel_size = (3,3), activation = "relu", padding = "same" ))
    model.add(MaxPooling2D(2))

    model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = "relu", padding = "same" ))
    model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = "relu", padding = "same" ))
    model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = "relu", padding = "same" ))
    model.add(MaxPooling2D(2))

    model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = "relu", padding = "same" ))
    model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = "relu", padding = "same" ))
    model.add(Conv2D(filters = 512, kernel_size = (3,3), activation = "relu", padding = "same" ))
    model.add(MaxPooling2D(2))

    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(4096, activation = "relu"))
    model.add(Dense(4096, activation = "relu"))
    model.add(Dense(1000, activation = "relu"))
    model.add(Dense(2, activation = "softmax"))
```