cs231n lecture 12 신지연

1. First Conv Layer

- ImageNet 데이터셋으로 학습된 AlexNet, ResNet-18, ResNet-101, DenseNet-121 의 첫 번째 conv layer 의 filter 들을 시각화한 이미지

- filter 를 시각화한다는 것은 정확히 말해 filter의 weight를 시각화한다는 의미

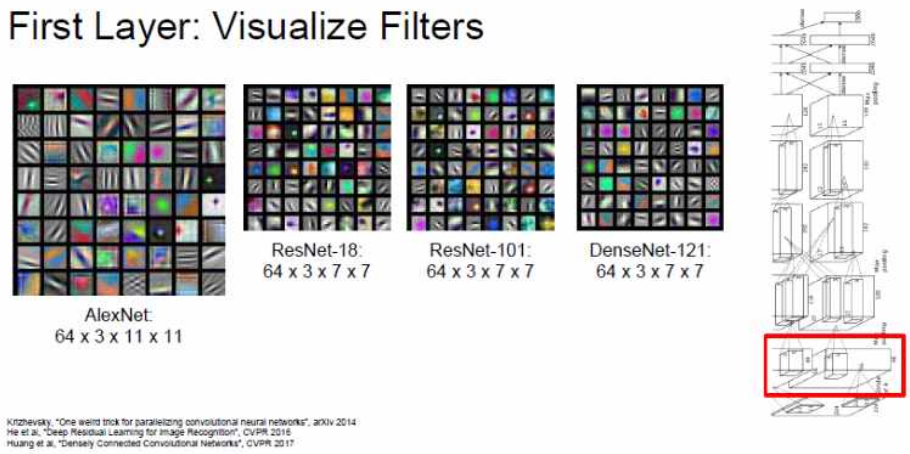
- 가장 왼쪽의 AlexNet 의 경우, 7 X 7 X 3 (H, W, 3) filter 를 64개 사용한 경우이고 각각의 filter 들을 RGB 이미지로 시각화하여 나타낸 것 (나머지 모델의 경우도 동일한 방법으로 시각화)

- 어떠한 CNN 모델을 어떠한 데이터로 학습을 시키던 첫 번째 conv layer filter 을 시각화한 모습은 비슷한 모습을 보인다는 점

- 첫 번째 conv layer 에서는 공통적으로 input 이미지의 모서리(edges), 보색(opposite color) 과 같은 특징을 찾아내는 역할

- filter의 모양을 통해 input 이미지에서 어떠한 특성을 찾는지를 알 수 있음

- input image과 filter가 conv연산(elementwise mul) 이 될 때 filter의 모양과 비슷한 부분에서 큰 activation 값이 나옴

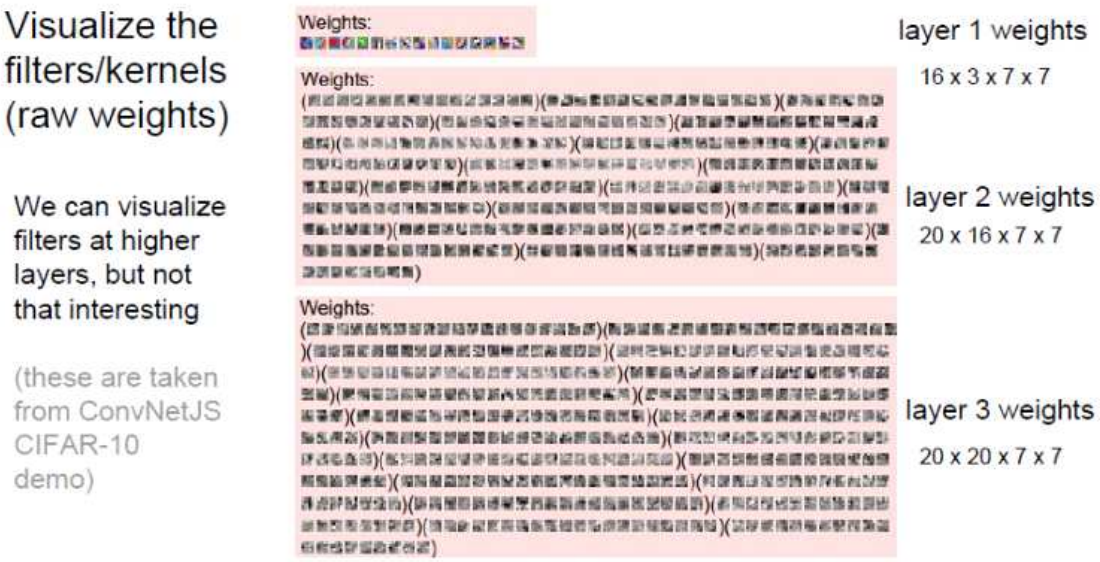


2. Intermediate Layer

- 두 번째 레이어부터 filter의 channel 수는 3 이 아니라서 filter 를 RGB 이미지로 시각화 x

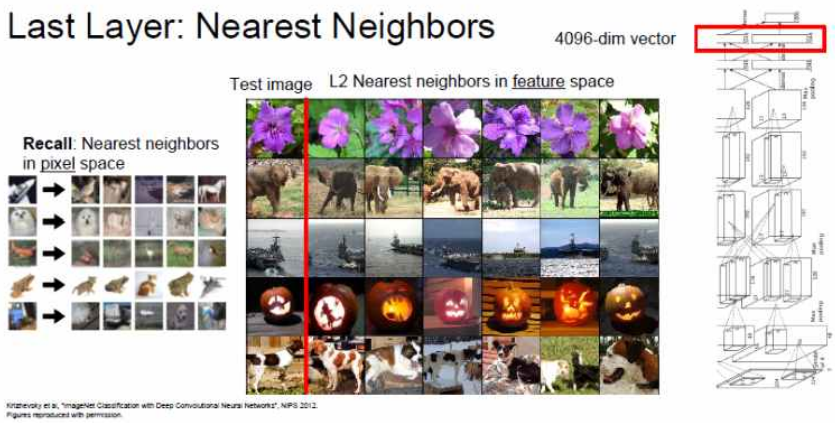
- 그래서 filter의 각 채널들을 각각 흑백 이미지로 시각화하여 나타냄

- 이러한 방법으로는 중간의 레이어가 무슨 역할을 하는지 이해하기 어려움

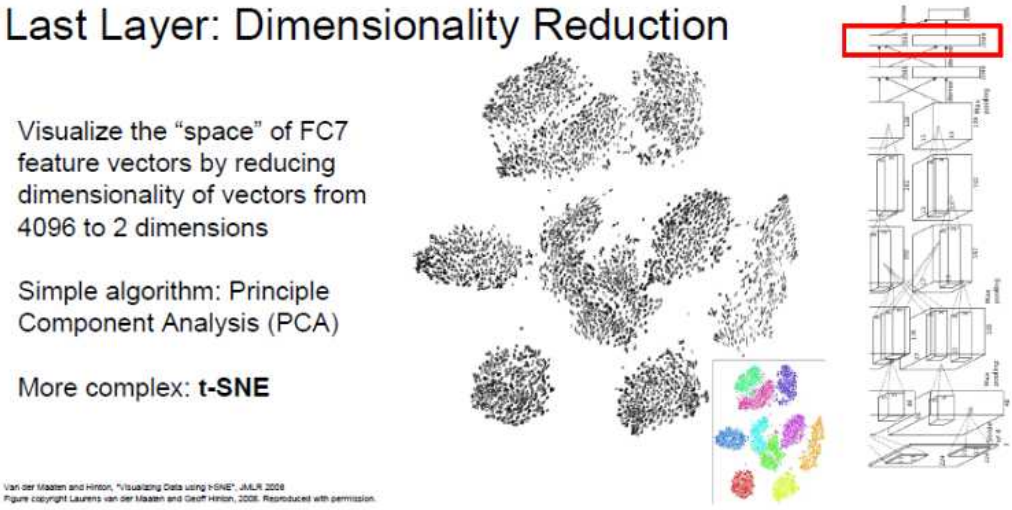


3. Last Layer

- 한 이미지에 4096차원의 특징 벡터들이 존재하는데 각 이미지 마다 특징 벡터들을 Nearest Neighbor Algorithm을 돌리게 되면 유사한 이미지들이 검출

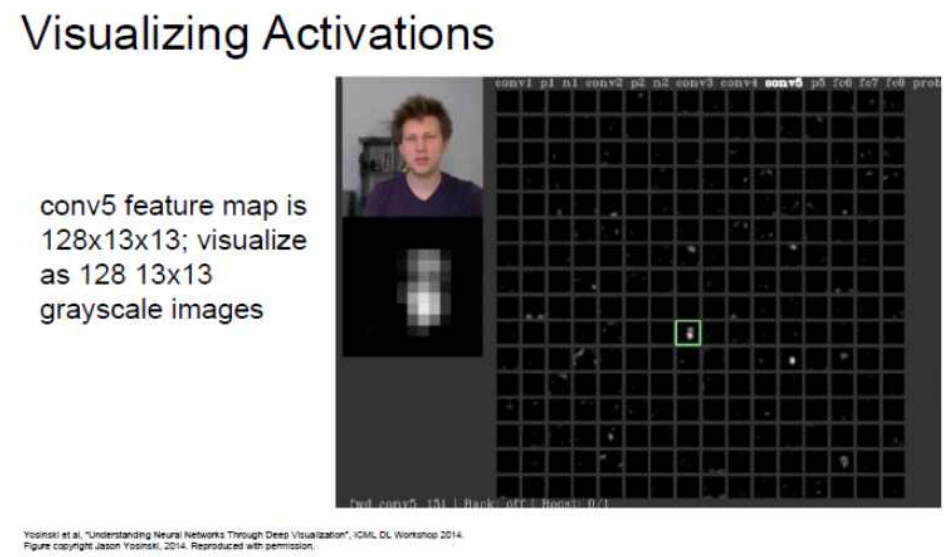


- 4096차원의 특징 벡터를 PCA Algorithm이나 t-SNE Algorithm을 이용해 2차원으로 차원 축소하면 군집화함



4. Visualizing Activations

- Activation Map을 통과시킨 Layer의 모습을 시각화하게 되면 사람의 얼굴이 있는 부분이 활성화된 Layer가 있다는 것을 확인 가능

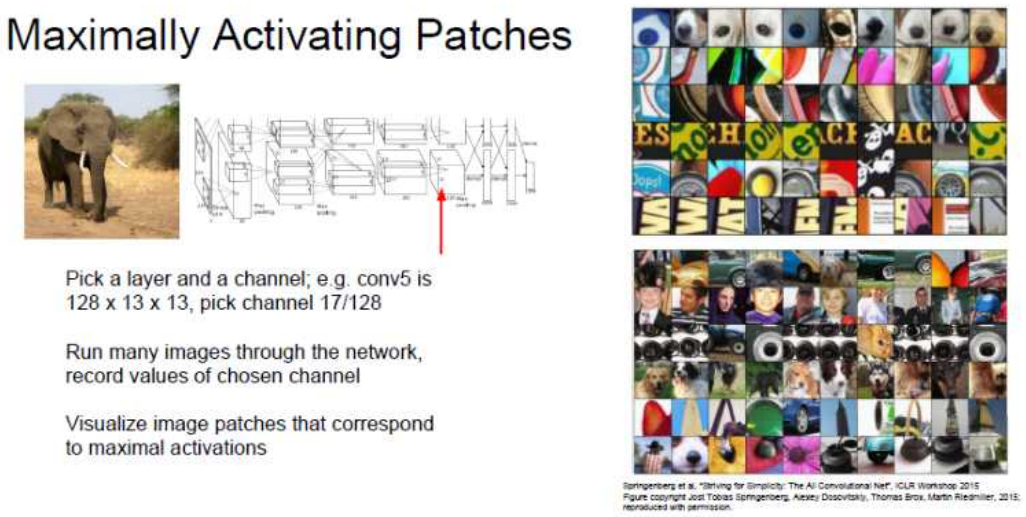


5. Maximally Activating Patches

- 여러 이미지들을 모델에 넣어 특정 activation map에서 가장 큰 값을 출력하는 (activate) 이미지들의 patch 들을 순서대로 시각화

- 오른쪽 위의 사진 : 이 Channel은 푸르스름하고 눈같이 생긴 이미지에 크게 활성화된다는 것을 추측 가능

- 오른쪽 아래의 사진 : 좀 더 깊은 Layer에서 추출한 이미지, 더 깊은 Layer에서 추출을 한다면 더 큰 구조를 추출한다는 것을 확인



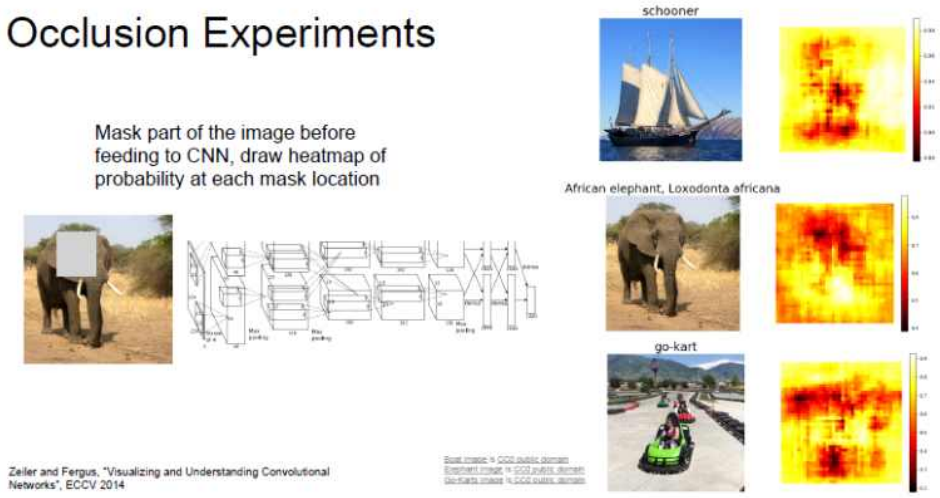
6. Occlusion Experiments

1. input 이미지의 일부를 가리고 가린 부분을 데이터셋의 평균값으로 채움

2. 일부분을 가린 이미지를 모델에 통과시키고 해당 클래스에 대한 score 값 (soft max에서 확률값) 을 얻음

3. sliding window 방식으로 이미지의 각 부분마다 위 과정을 반복하여 네트워크에 통과시키고 얻은 score 값들(확률값) 의 변화를 비교 및 시각화

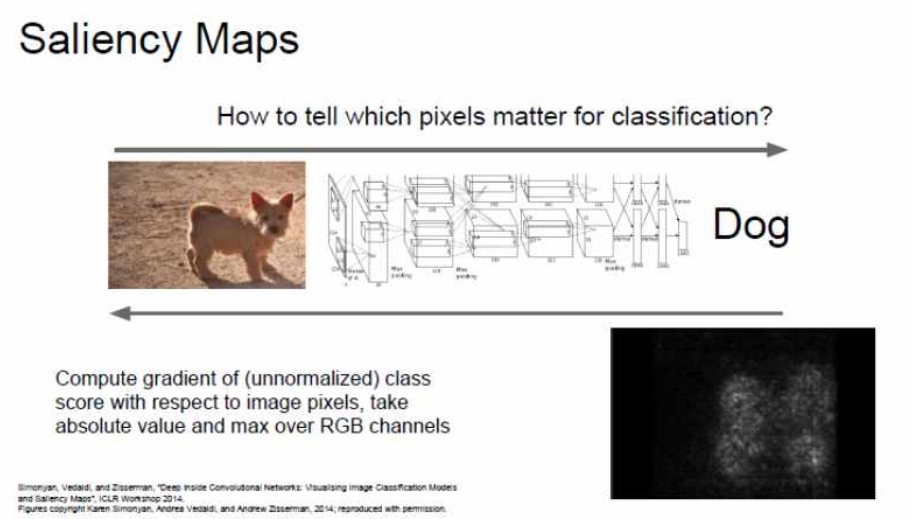
위 과정을 통해 input 이미지의 어떠한 부분을 네트워크에서 중요시 보는가에 대해 알 수 있음



7. Saliency Maps

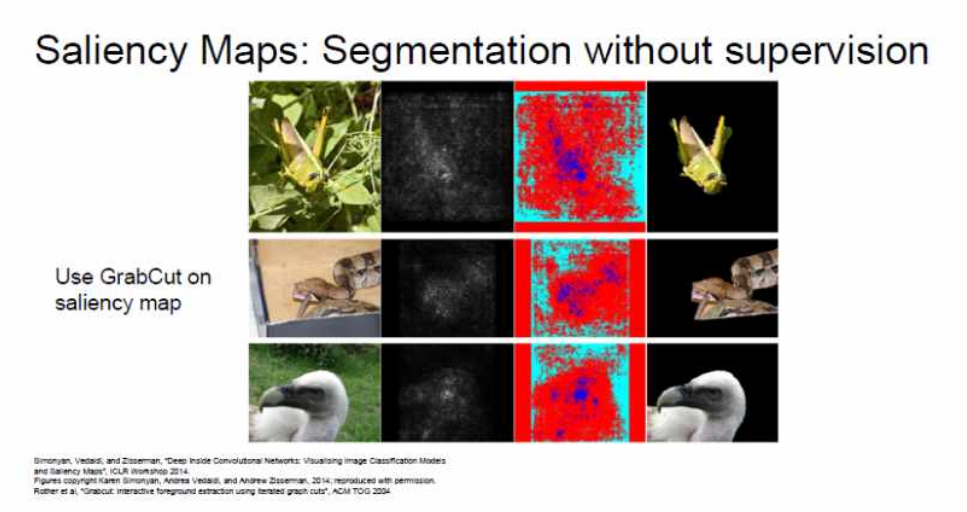
- 가중치 변화량에 대한 loss 의 변화량이 아닌 가중치(W)는 고정시켜두고 (ex. ImageNet 으로 pretrain 시킨 W) input image의 각 픽셀의 변화량에 따른 score의 변화량을 나타낸 것

- input 이미지의 각 픽셀에 대해서 예측한 class score 의 gradint 를 나타낸 시각화 방법



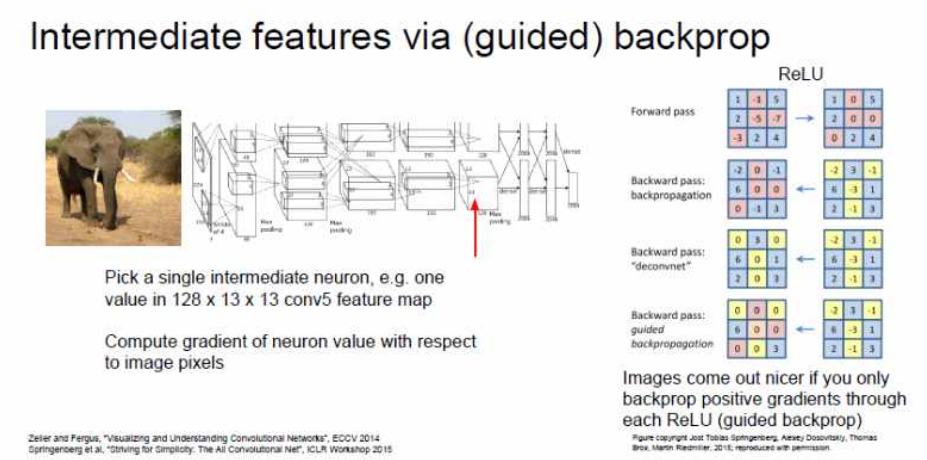
- saliency map 을 보면 이미지에 있는 객체의 윤곽이 어느정도 나타남

- 실제로 각 픽셀별로 class 가 라벨링되어있는 이미지 없이도 Saliency map 을 이용하여 semantic segmentation 이 가능하게 하는 방법도 있음 (label 이 있는 데이터로 학습하는 방법인 supervision 방식보다는 훨씬 성능이 안 좋음)



7. Guided Backpropagation

- input 이미지의 어떤 부분이 중간 레이어의 특정 뉴런(activation map 의 한 픽셀) 에 영향을 주는지 알 수 있음

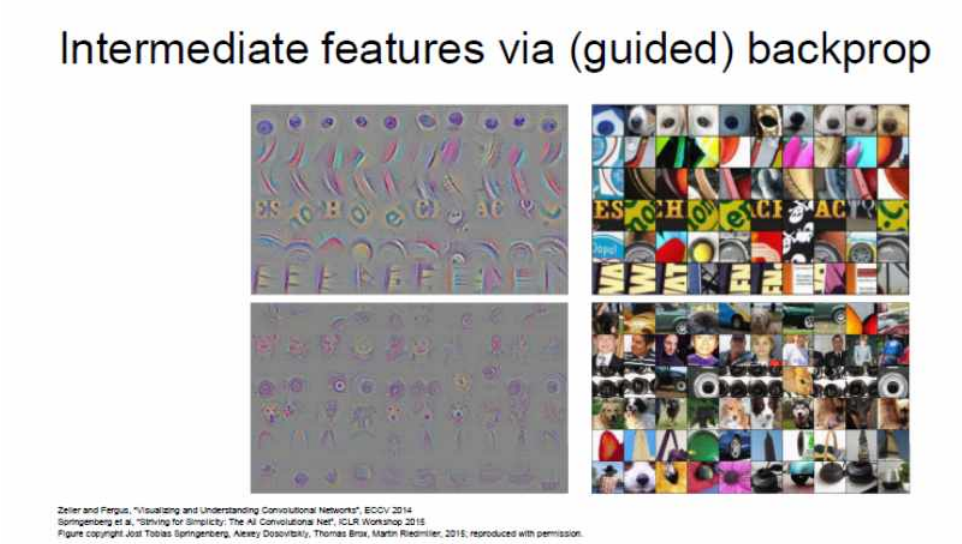


- backpropagation 과정에서 좀 더 깨끗한 결과를 시각화하기 위해 guided backpropagation을 사용

- Guided backpropagation 은 backprop 과정에서 ReLU 의 gradient 부호가 양수이면 그래도 통과시키고, 음수이면 backpropagation 을 하지 않는 방법

- 전체 네트워크는 전체 gradient 를 사용하는 것이 아니라 양의 부호인 gradient 만을 고려

- 중간 레이어의 특정 뉴런에 대한 input 이미지의 gradient를 구하여 시각화



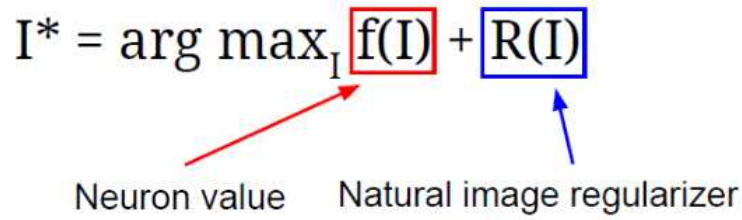
8. Gradient Ascent

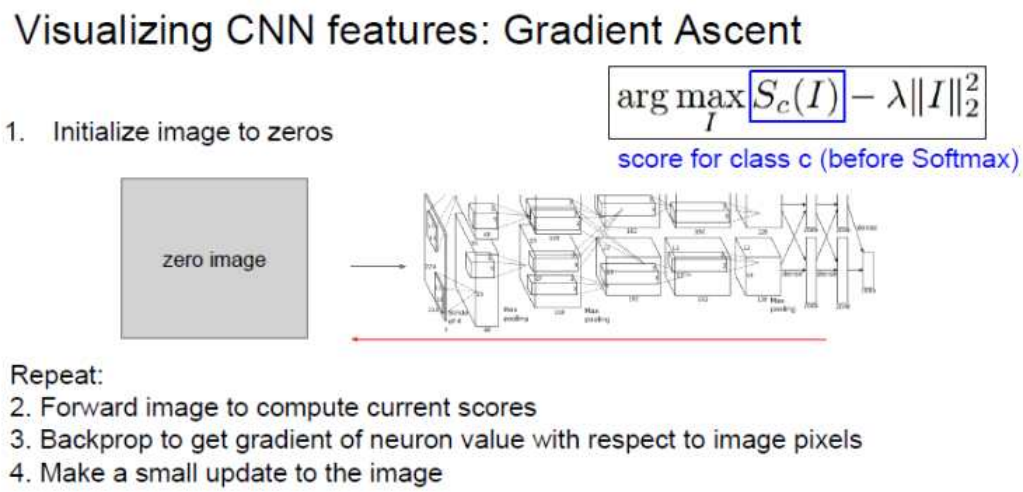
- input 이미지에 상관 없이 일반적으로 모델을 이해하는 방법

- 가중치(W)값들을 pre-trained 된 값으로 고정시키고 중간 레이어 뉴런의 activation 혹은 최종 class score 에 대한 랜덤 값으로 초기화된 input 이미지의 gradient 를 구하고 해당 gradient 방향으로 input 이미지의 픽셀 값들을 업데이트

- 최종적으로 중간 레이어 뉴런의 activation 혹은 최종 class score 을 최대화시키는 input 이미지의 픽셀값을 찾아낼수 있게됨

- gradient ascent를 통해 최종적으로 찾은 input 이미지의 픽셀값들이 네트워크의 특성(W)에 완전히 과적합되는 것을 막고 일반적인 형태의 input 이미지를 찾기 위해 Regularization term을 사용





1. input image를 0과 가까운 랜덤한 값으로 초기화

2. 초기화한 input image 를 pre-trained 된 네트워크에 통과시켜 score를 얻어냄

3. Backpropagation 을 통해 최종 score 변화량에 따른 input image 의 변화량 (gradient) 를 계산. regularization term (ex. L2 regularization) 을 추가하여 더욱 자연스러운 image를 얻어낼 수 있음

4. 얻어낸 gradient 방향으로 input image의 각 픽셀 값을 업데이트

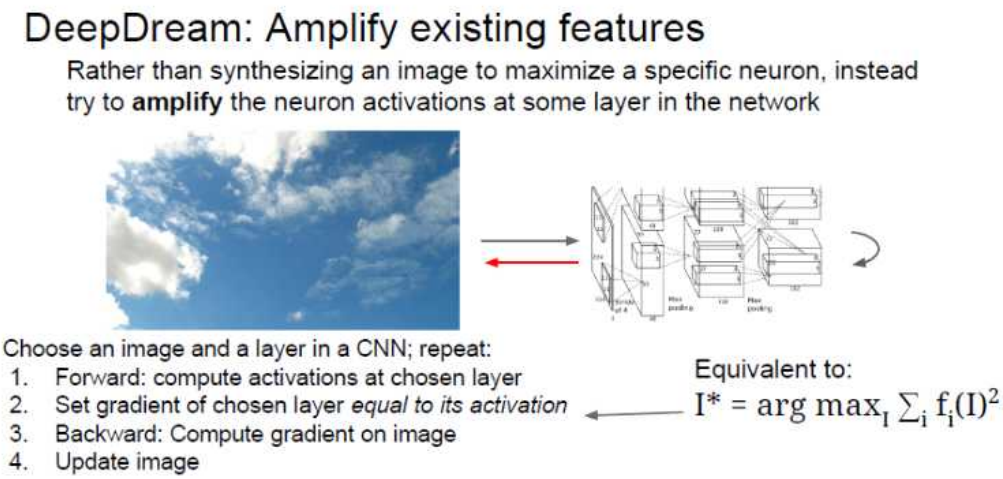
9. DeepDream

1. 입력 이미지를 CNN의 중간 Layer까지 통과 시킴

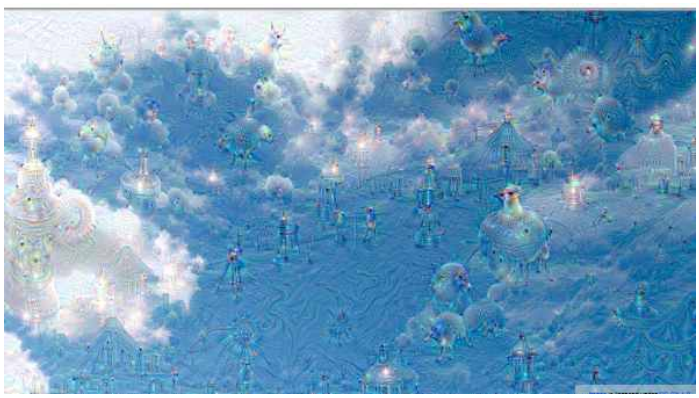
2. 해당 gradient를 activation value 값으로 저장

3. Backpropagation을 통해서 이미지를 update

4. 위의 내용을 반복



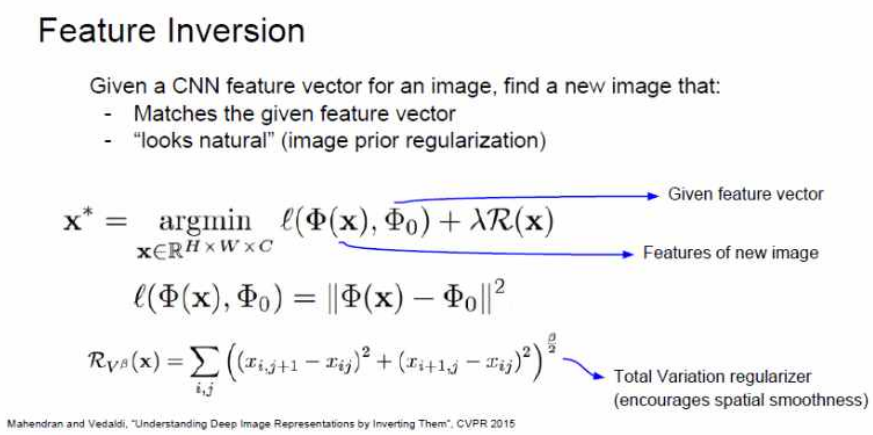
- 깊은 레이어에 대해 시각화



- 얕은 레이어에 대해 시각화

10. Feature Inversion

- 네트워크에서 추출된 feature vector (activation map) 을 이용해 input 이미지를 재구성하는 방법



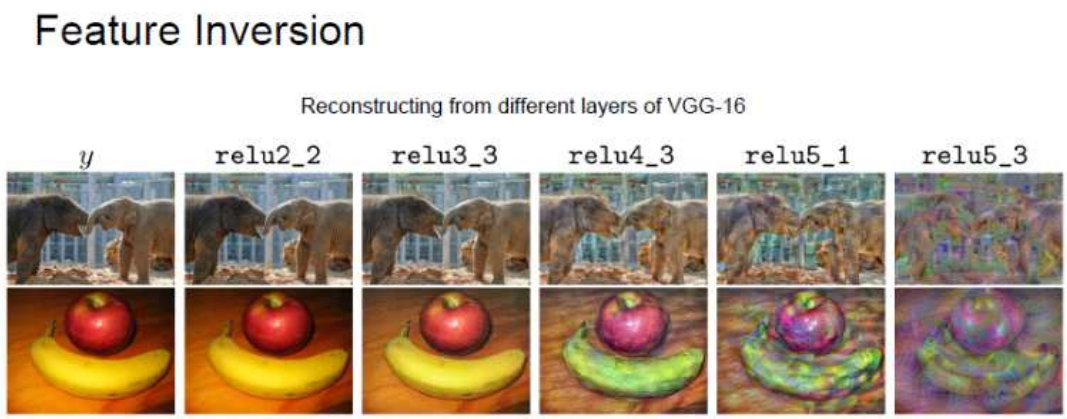
1. 특정 이미지(y) 를 네트워크에 통과시켜 feature vector(activtion map) V_y를 뽑아냄

2. (랜덤값으로 초기화된) input image 를 네트워크에 통과시켜 feature vector(activation map) V_x를 뽑아냄

3. 뽑아낸 두 feature vector V_y와 V_x들간의 distance를 구함

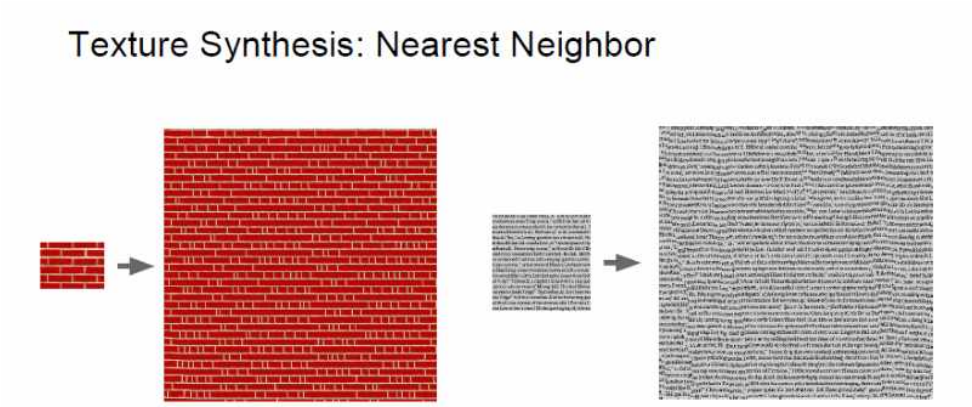
4. 해당 distance 에 대한 input image 의 변화량(gradient) 를 구하고 distance 를 최소화하는 방향으로 input 이미지의 픽셀값들을 업데이트

이 과정에서 이미지를 자연스럽게 만들기 위해 Total Variation regularizer 이라는 regularization term 을 추가



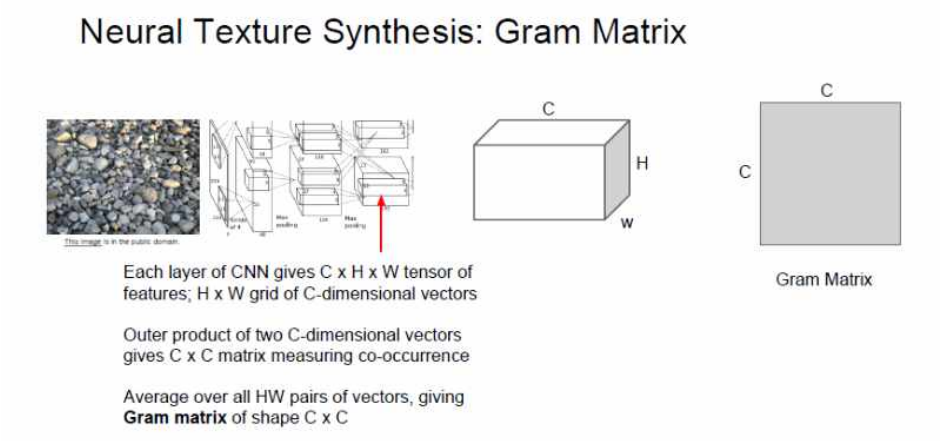
11. Texture Synthesis

- 작은 texture 이미지로부터 동일한 패턴을 가지는 큰 이미지를 출력해내는 task



- texture synthesis 를 Neural Network 를 이용해 해결하려는 Gram Matrix 를 이용하는 방법

- Gram Matrix란 서로 다른 공간 정보에 있는 Channel들을 가지고 외적을 계산하여 새로운 Matrix를 만드는 것



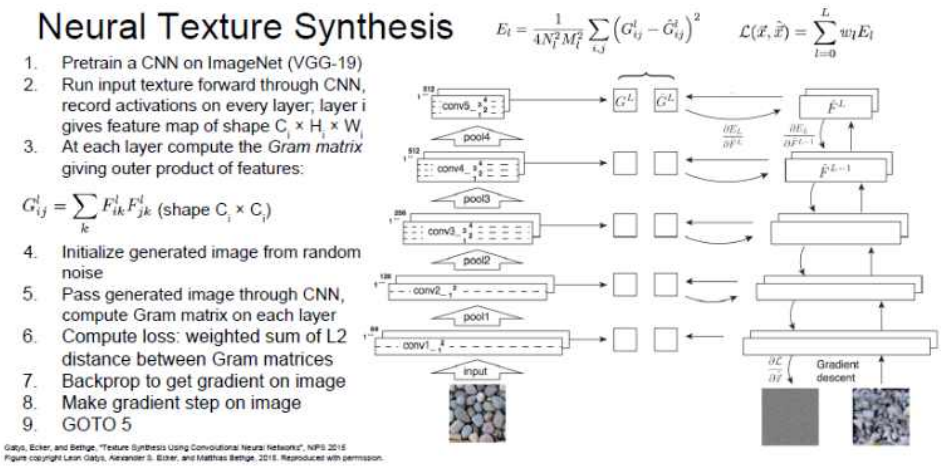
1. texture input image 를 네트워크에 통과시켜 특정 레이어에서 feature vector (C×H×W) 을 뽑아냄

2. 해당 feature vector(C×H×W)에서 서로 다른 두 개 다른 뉴런에서의 feature vector (1×1×W) 을 뽑아냄

3. 위에서 뽑아낸 두 벡터의 외적을 계산해서 C×C matrix 를 생성

4. 이 과정을 H×W에 대해 전부 수행하여, 결과에 대해 평균을 한 최종 C×C matrix 를 구함

이렇게 생성한 C×C matrix 가 Gram matrix



1. Input Image를 넣고 Pretrained 된 VGG Network에서 다양한 Gram Matrix를 생성

2. Random Noise로 초기화 된 Image를 VGG Network를 통과 시켜 Gram Matrix를 생성

3. Input Image와 만들어진 Image의 Gram Matrix를 비교하여 L2 distance가 최소가 되도록 Loss값을 계산

4. 계산된 Loss 값을 이용하여 Backpropagation을 진행하여 이미지 픽셀의 Gradient를 계산

5. Gradient Ascent 방법을 통해 이미지의 픽셀 값들을 update

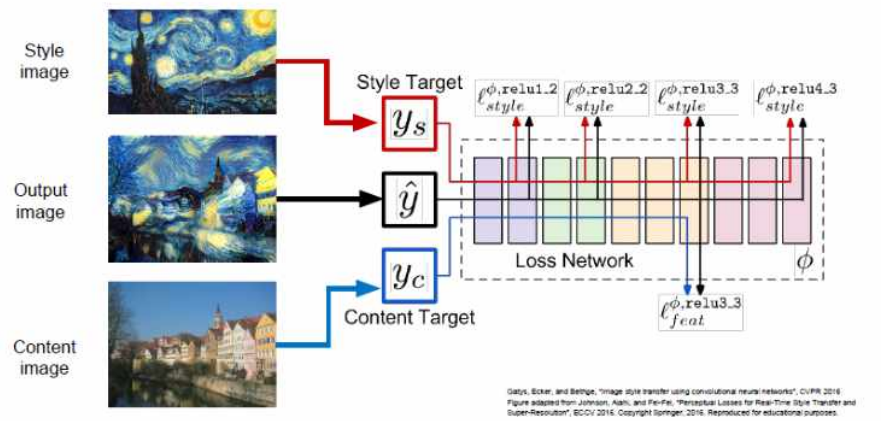
위 단계를 반복하여 입력 이미지와 유사한 이미지가 만들어지도록 함

12. Style Transfer

- Style transfer에서 input 이미지는 2가지.

- Content image: 우리가 최종적으로 생성할 이미지가 어떻게 생겼는지에 대한 input image

- Style image : 최종 이미지의 texture 가 어떤지에 대한 input image



1. content image 를 네트워크에 통과시켜 gram matrix G_c를 추출

2. style image 를 네트워크에 통과시켜 feature vector V_s를 추출

3. (초기 랜덤 값으로 초기화된) input image 를 네트워크에 통과시켜 gram matrix G_x와 feature vector V_x를 추출

4. G_c와 G_x와의 distance, V_s와 V_x와의 distance를 구하고 해당 distance에 대한 input image 의 변화량(gradient)를 구함

5. 위에서 구한 distance 들이 최소화 될 수 있게 gradient 방향으로 input image 픽셀값들을 업데이트합니다.

loss: 두 distance 의 합, 두 distance 가 합칠때 어떤 distance 를 최소화하는 것이 더 중요한지에 따라 가중치를 설정
위 과정을 반복해 최종적으로 우리가 원하는 이미지를 재구성(생성)

- style image에 대해 foward/backward pass 를 진행해야하므로 연산량이 굉장히 많고 오래걸린다는 단점

13. Fast Style Transfer

- Fast style transfer 방법은 style image를 고정시켜두고 오직 content image에 대한 연산만을 진행하여 style transfer 하는 방법으로 기존 style transfer의 속도를 개선