

CNN이 비난 아닌 비난(?)을 받는 가장 큰 이유는 바로 **Black Box**

**Problem** 때문입니다.

Black-Box Problem이란 말 그대로 CNN 안에서 일어나는 과정이 마치 Black Box 안에 들어있는거 같다는 비유로 붙여진 이름입니다.

모델을 작성한 당사자도 CNN 안에서 어떤 과정을 거쳐서 학습이 되는지 정확히 알 수가 없다는 문제가 있죠.

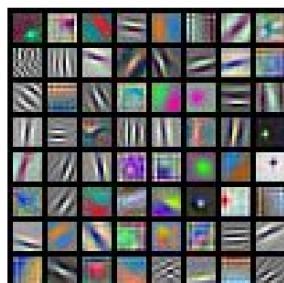
12강에서는 이런 문제점들에 맞서 CNN의 중관 과정들을 좀 더 직관적으로 나타내는 방법들을 배웁니다.

그 중에서도 input output과 직접적인 연관이 있는 first & last layer들을 먼저 살펴봅시다.

## First Layer

이미지의 raw pixel들을 받아서 처리하는 첫번째 레이어를 자세히 들여다 보면 아래 그림과 같은 결과를 얻을 수 있습니다.

## First Layer: Visualize Filters



AlexNet:  
64 x 3 x 11 x 11



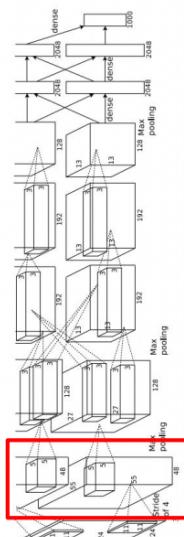
ResNet-18:  
64 x 3 x 7 x 7



ResNet-101:  
64 x 3 x 7 x 7



DenseNet-121:  
64 x 3 x 7 x 7



First Layer 는 Image raw pixel 에 W 를 내적해서 feature map 을 생성하고, 이때 W 는  $64 \times 3 \times 11 \times 11$  의 사이즈를 가집니다.

W vector 를 64 개의  $11 \times 11$  RGB 이미지로 변환하면 위와 같은 결과가 나옵니다.  
(trained 된 W)

언뜻 보면 격자무늬나 직선의 이미지들이 잔뜩 있는것을 볼 수있는데 이는 first layer 의 Weight 가 Edge 와 Corner 를 찾는다고 이해할 수 있습니다.



잠깐 Filter 가 왜그렇게 해석이 되는지에 대해 간단히 설명해보자면~

Filter 는 곧 이미지에서 무엇을 찾는지로 연결이 됩니다.

아래와 같은 filter 가 있고 동일 사이즈의 img 가 4 개 있다고 했을때, convolution 한 값이 가장 큰 건 몇번일까요?

2	0	0
0	2	0
0	0	2

Filter

1	1	1
1	1	1
1	1	1

0	0	0
0	0	0
3	3	3

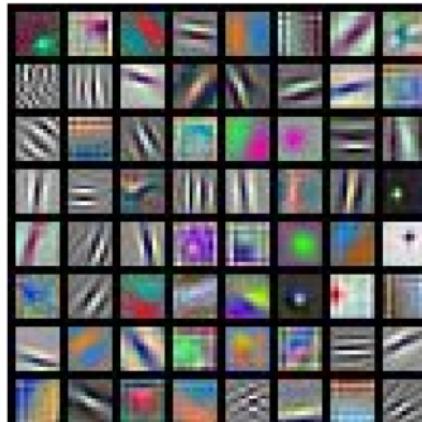
3	0	0
0	3	0
0	0	3

0	0	0
1	2	1
3	1	1

weight\_sum 은 9로 통일했을 때

왼쪽부터 차례대로 6, 6, 18, 6이 나오고 가장 큰 건 3번째가 됩니다.

결국, Filter 와 비슷한, 겹치는 형태를 떨수록 inner product 내적의 결과가 큰 값이 나올 확률이 높아집니다.



다시 돌아와서, first layer 를 다시 보면, 엣지 성분이 많이 검출되는 것을 확인할 수 있고

첫 번째 layer 에 한해서 유의미한 결과를 해석해낼 수 있다고 볼 수 있습니다.

하지만 Intermediate layer 부턴 뭔가 이상해집니다. input 은 채널수가 3(RGB) or 1(BW)이기 때문에 filter 도 동일한 채널사이즈를 갖지만, AlexNet 만해도 2 번째 layer 부터는 채널사이즈가 16 으로 대폭 상승합니다.

Weights:  


layer 1 weights

Weights:  
()

$16 \times 3 \times 7 \times 7$

layer 2 weights

$20 \times 16 \times 7 \times 7$

Weights:  
()

layer 3 weights

$20 \times 20 \times 7 \times 7$

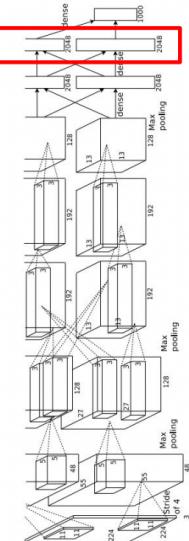
$20 \times 16 \times 7 \times 7$  을 어찌저찌 잘 나눠서  $7 \times 7 \times 1$ (BW)filter 16 개의 묶음 x 20 개 로 시각화해도 크게 의미있는 경향은 찾아볼 수 없습니다.

첫번째 레이어는 input 이미지에 직접적으로 연결되어 있지만 두번째 레이어 부터는 이전 레이어의 activation map 과 연결되어 있기 때문에 interpretable 하지 않습니다.

## Last Layer - Nearest Neighbor

### Last Layer

FC7 layer



4096-dimensional feature vector for an image  
(layer immediately before the classifier)

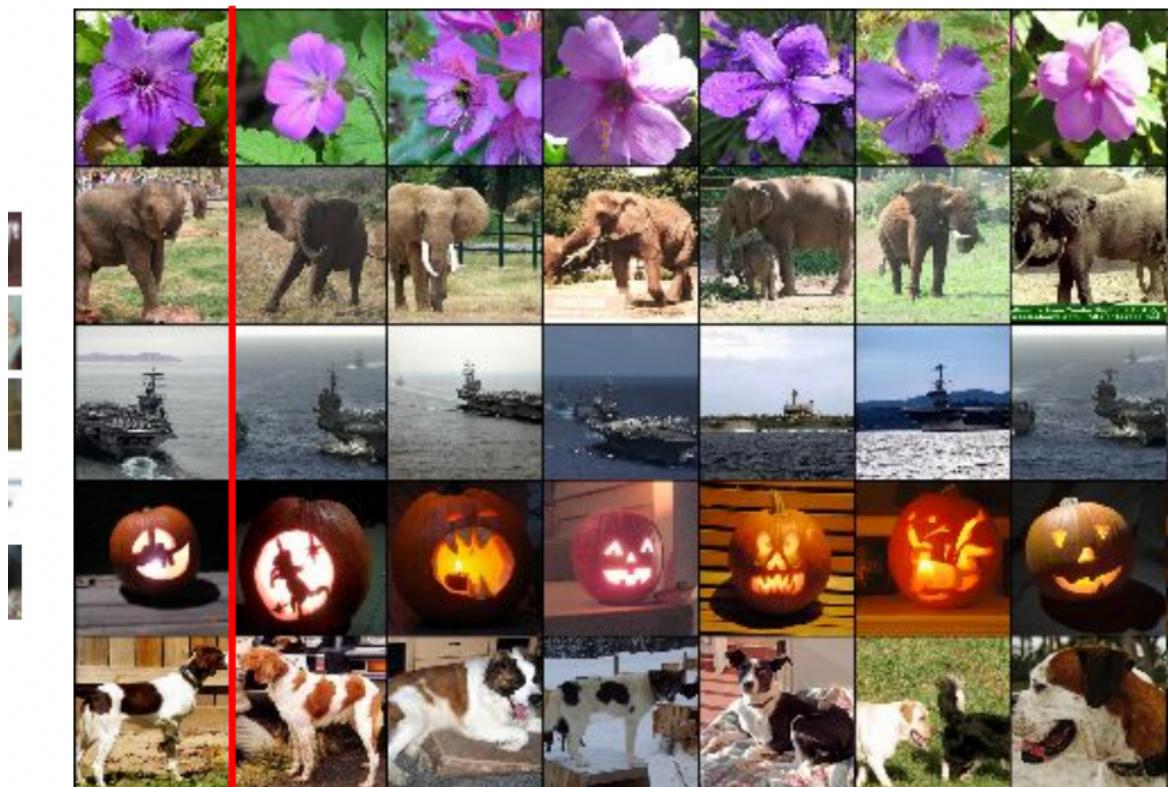
Run the network on many images, collect the  
feature vectors

마지막 Layer에서도 유의미한 결과를 확인할 수 있습니다.

보통 Image Classification에서 마지막 layer는 dimension을 class의 개수로 축소해주는 fully-connected layer로, 위의 그림(Alexnet)에서는 마지막 fc-layer에 입력이 되는 vector가 4096 - D입니다.

이렇게 input 이미지부터 여러 layer를 거친 마지막 4096-D vector(=feature)들로 L2 Nearest Neighbor를 수행하면 비슷한 vector들끼리 묶이는데, 각 vector의 입력이미지를 모았더니 아래와 같은 결과가 나왔습니다.

## Test image L2 Nearest neighbors in feature space

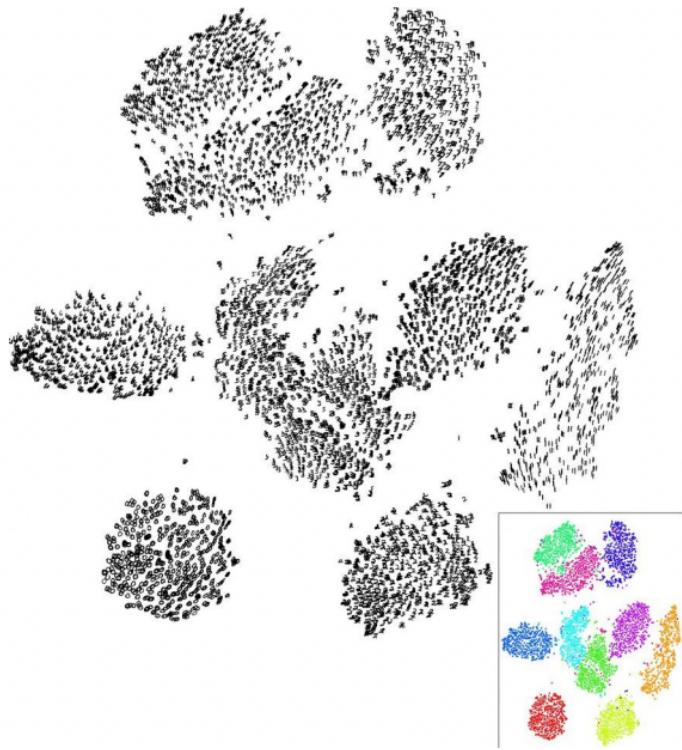


feature 가 유사한 이미지들은 실제로 비슷한 이미지였다는걸 알 수 있습니다.

놀라운 점은 두번째 줄의 경우, test image 의 코끼리는 오른쪽을 향하고 있고, 다른 코끼리들은 정면을보거나 왼쪽을 보고 있다는 것입니다. 마지막줄의 강아지도 마찬가지입니다.

단순히 pixel 끼리 분석해서 얻을 수 없는, 물체가 무엇인지 feature 들이 이해하고 있다는것을 알 수 있게 해준 실험입니다.

Last Layer - Dimensionality reduction



이번에는 마지막 layer의 4096 vector를 PCA로 군집화 시켰습니다.

PCA는 차원축소의 일종으로 여기서는 4096 차원의 vector를 x,y 2 차원으로 축소해 좌표평면에 나타냈습니다.

그 결과, 같은 class들의 이미지들끼리 가깝게 분포함을 확인할 수 있습니다.

## Last Layer - tSNE

강의에서 가볍게 언급하고 넘어간 tSNE에 대해서 간단하게 설명하면,

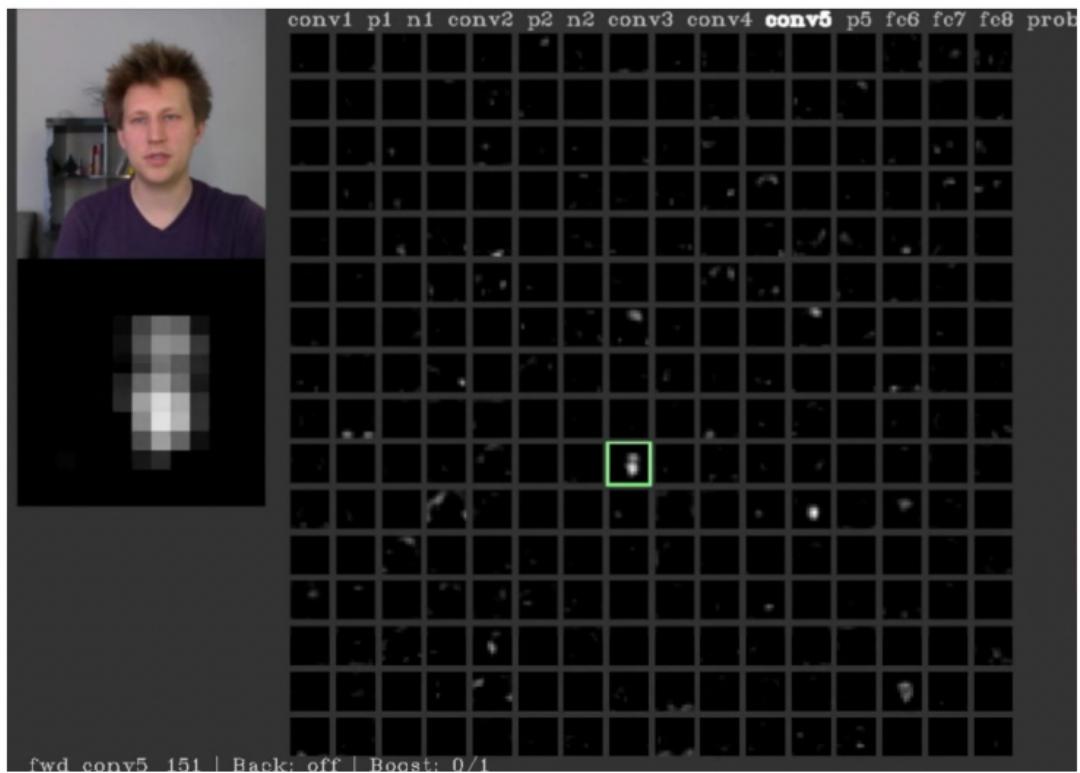
SNE는 [Stochastic Neighbor Embedding](#)의 약자로 고차원의 원공간에 존재하는 데이터  $x$ 의 이웃간 거리를 최대한 보존하는 저차원의  $y$ 를 학습하는 방법입니다. SNE는 가우시안 확률분포를 전제하고,  $t$  분포를 사용하면 t-SNE가 됩니다.

자세한건 [t-SNE](#) 잘 정리된 블로그를 참고해주세요.



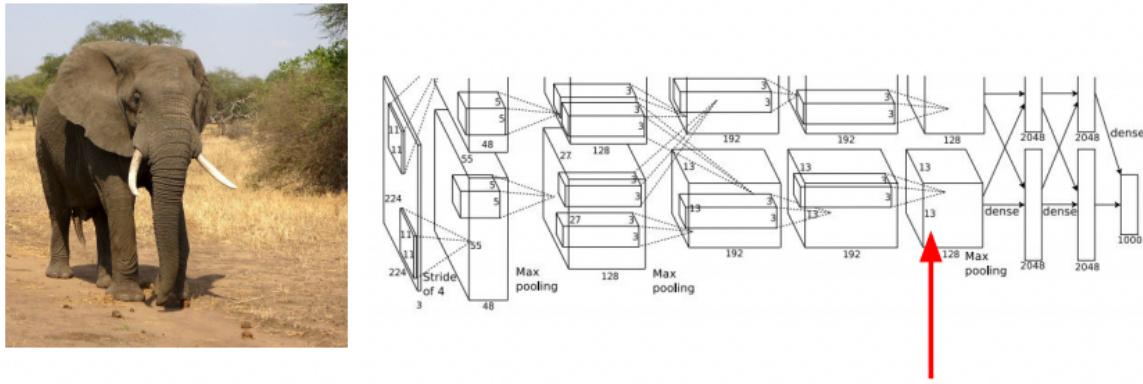
## Visualizing Activations

앞서 말했듯이 intermediate layer들은 input에 직접적으로 연결되어 있는게 아니기 때문에 인간이 해석하기가 쉽지 않다고 했습니다. 하지만 얻어걸린건지 아닌지는 모르겠지만 AlexNet의 conv5 activation map에서, 사람사진을 입력했을 때 얼굴의 모양과 위치가 비스무리한 image를 확인할 수 있었다고 합니다.



## Maximally Activating Patches

Maximally Activation Patches는 input 이미지의 어떤 patch(부분, crop)이 neuron을 가장 활성화 시키는지를 확인하는 방법입니다.



### AlexNet conv5

강의에서는 AlexNet의 conv5 layer을 예시로 설명했는데, conv5 layer는 128x13x13의 feature map을 output 합니다.

이중에서 임의로 하나의 채널을 고릅니다. 128 개의 채널중에 17 번째 채널을 골랐다면, 13x13의 feature map이 나옵니다.

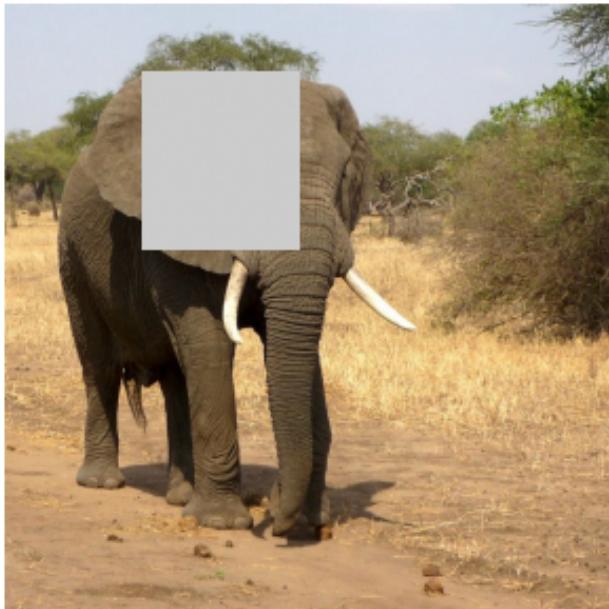
이제 input에서부터 network를 쭉 통과시키며 conv5의 17 번째 layer에서 가장 높은 값을 나타낸 위치를 찾고, 그 지점에서부터 receptive field들을 쭉 거슬러 올라 input에서 나타냈더니 아래와 같은 결과가 나왔습니다.



여기서 알 수 있는 것은 특정 layer의 특정 neuron에서 어떤 feature들을 찾고 있는지에 대한 대략적인 아이디어입니다.

## Occlusion Experiments

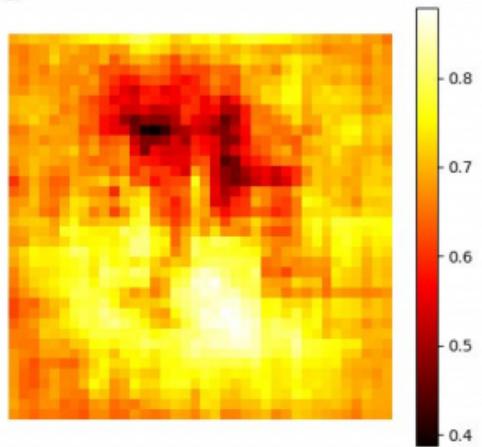
Occlusion Experiment 는 이미지의 어떤 부분을 가렸을때 예측 성능이 얼마나 줄어드는지를 heatmap 으로 나타낸것입니다.



왼쪽같이 코끼리 이미지에서 이마, 귀 부분을 가리면 예측 성능에 어떤 변화가 있을까요?

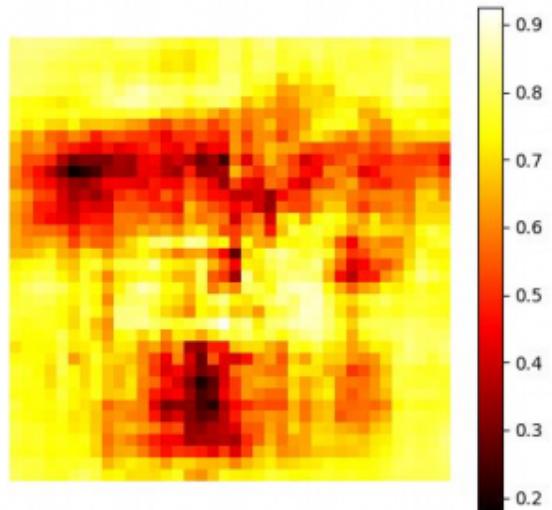
가리는 부분을 옮겨가면서 어떤 변화가 있는지 측정을 해서 heatmap 으로 나타내 봅시다.

African elephant, *Loxodonta africana*



heatmap에서 색깔이 진할 수록 예측확률이 떨어지는것을 의미합니다. 따라서 진한 부분일 수록 예측에 critical 한 역할을 합니다.

go-kart



go-kart 예시를 보면 확 와닿는게, 맨 앞에 있는 카트를 가리면 예측성능이 확 줄어듭니다.

카트라고 이미지를 분류하는 과정에서 go-kart가 실제로 예측에 사용되는것을 알 수 있습니다.

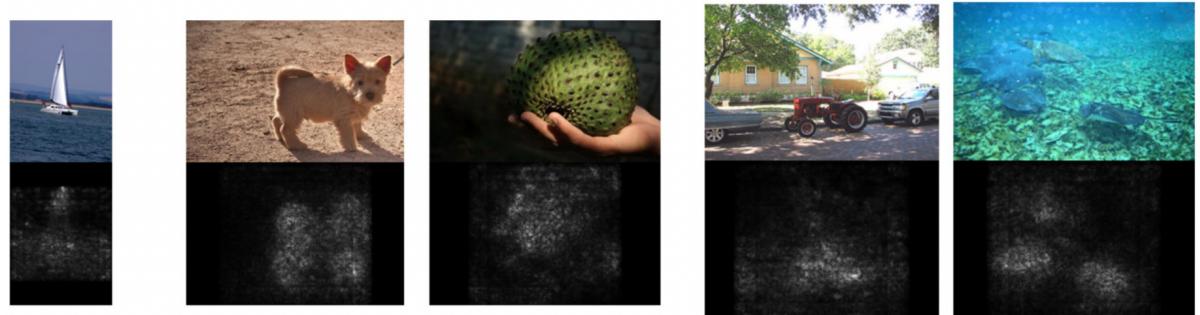


## Saliency Maps

saliency는 한국어로 '가장 중요한, 핵심적인; 가장 두드러진, 현저한' 등의 뜻을 가지는 단어입니다.

Saliency Map은 이미지의 어떤 pixel이 classification에 영향을 줬는지 체크합니다.

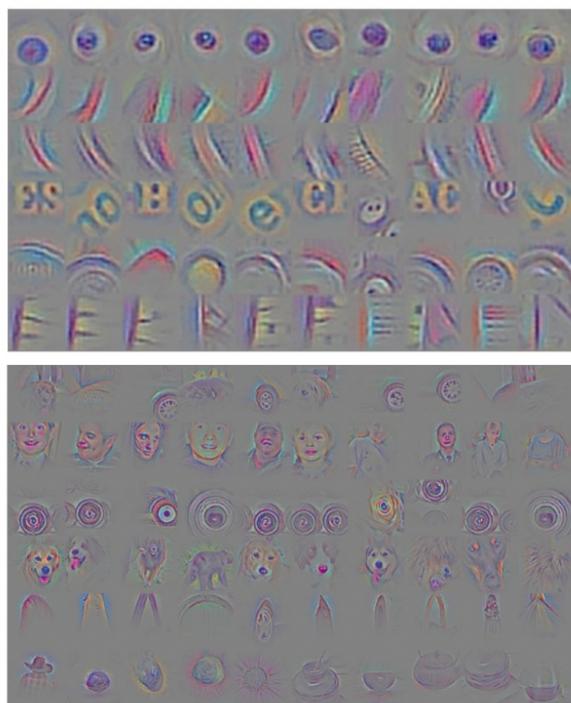
각 픽셀별로 결과에 얼마나 영향을 끼치는지 Gradient Descent 방식으로 접근해 영향력이 큰 pixel들을 찾아냅니다.



## Intermediate Features via guided BackProp

Guided Backpropagation 도 위에서 본것과 마찬가지로 중간의 뉴런을 골라서 이미지의 어떤 patch 가 영향을 크게 줬는지를 확인합니다.

이때 비슷한 실험을 굳이 Guided Backpropagation 을 사용하면서 실험하는 데서 오는 장점은, patch 내부에서도 '어떤 픽셀'이 크게 영향을 줬는지 알 수 있기 때문입니다.

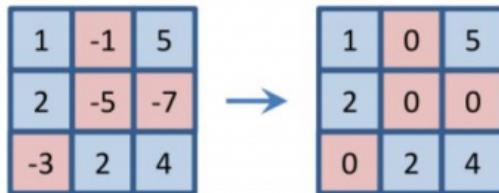


좌: guided backprop, 우: Activating Patches

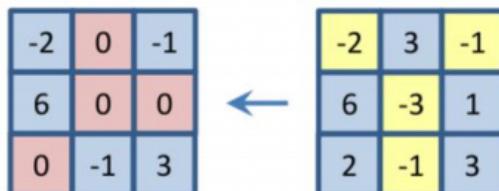


## ReLU

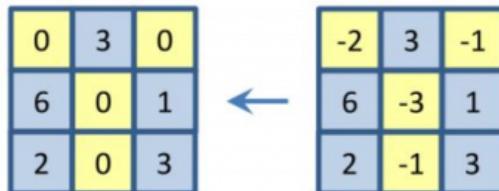
Forward pass



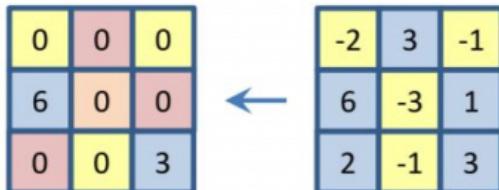
Backward pass:  
backpropagation



Backward pass:  
“deconvnet”



Backward pass:  
*guided  
backpropagation*



**Images come out nicer if you only  
backprop positive gradients through  
each ReLU (guided backprop)**

### 1. Forward Pass ( ReLU )

: Relu 를 활성화 함수로 사용해 0 이하 값들은 전부 0 이됩니다.

### 2. Backward Pass: Back Propagation

: 구해진 Gradient 들 중 ReLU 를 활성화했던 것들에만 값을 그대로 전달.

(ReLU Gradient 전달 과정 참고)

### 3. Backward Pass: "deconvnet"

: gradient 가 음수면 backward pass 하지 않고 0 전달, 양수면 그대로 전달

### 4. Backward Pass: *Guided* Back Propagation

: 기존의 Back Propagation + deconvnet

이 경우에는 ReLU 활성화 안된거 0, gradient 값 음수인거 0으로 전달



## Visualizing CNN features: Gradient Ascent

위에서 다룬 거의 모든 실험적 방법들은 전부 어떤 이미지 I의 어떤 부분, **요소들이** neuron 을 활성화하는지, input 이미지에 따라 다 다른 결과가 나왔습니다. (이미지에 상당히 rely 하는 방법이라고 할 수 있습니다.)

그렇다면 반대로 어떤 neuron(weight)이 주어졌을 때 그 neuron 을 활성화 시키는 generalized 한 image 는 어떤게 있을까요?

이를 알아보기 위해 **Gradinet Ascent** 를 진행합니다.

※ Gradient Ascnet 는 말 그대로 Loss 가 최대가 되는 Parameter 를 찾는 방법입니다.

※ 고정된 W에 대해 input image 의 pixel value 를 gradient ascent 로 바꿔가면서 neuron 이나 class score 를 극대화 시키는것.

$$I^* = \text{argmax}_I f(I) + R(I)$$

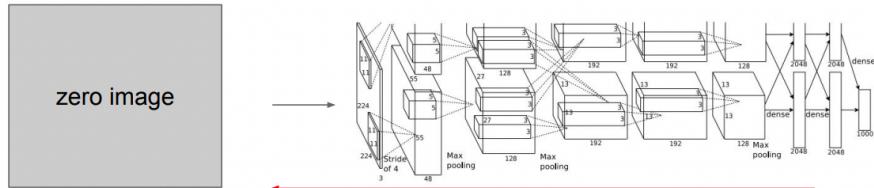
※ Regularization  $R(I)$ 가 필요. Generated Pixel 들이 network 에 overfitting 되지 않도록 하기 위해서 꼭 필요!

# Visualizing CNN features: Gradient Ascent

1. Initialize image to zeros

$$\arg \max_I [S_c(I) - \lambda \|I\|_2^2]$$

score for class c (before Softmax)

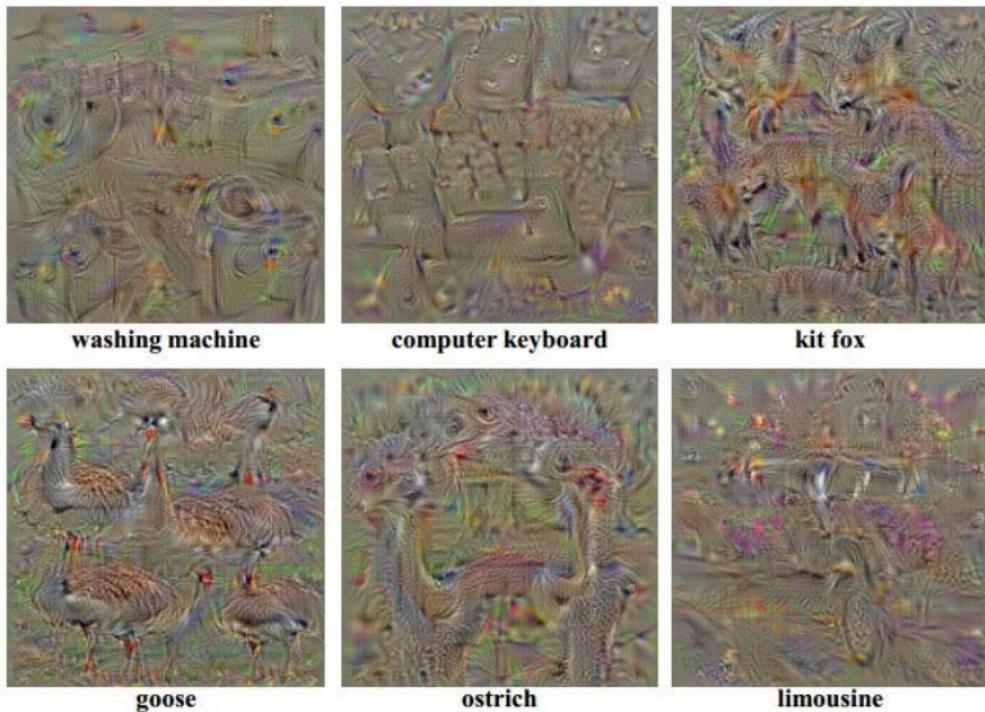


Repeat:

2. Forward image to compute current scores
3. Backprop to get gradient of neuron value with respect to image pixels
4. Make a small update to the image

실험 결과는 아래와 같습니다. 아래 이미지들은 Gradient Ascent를 통해 뉴런을 최대로 활성화 시키는 합성 이미지들입니다.

이제 중요한 Feature들을 부각시켜 볼 수 있습니다



여기서 추가로 아래 Regularization들을 더하면 결과가 더 좋아진다고 합니다.

1. Gaussian Blur Image

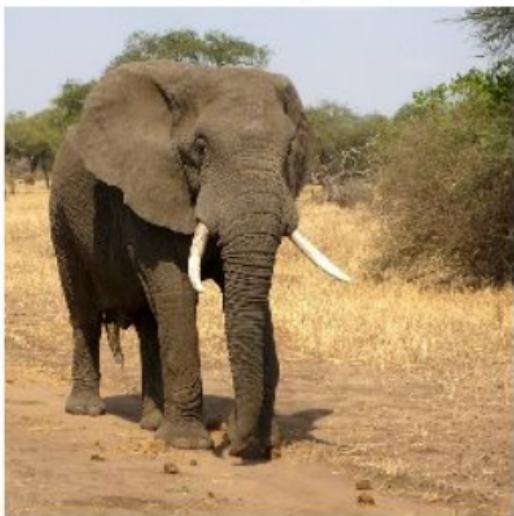
2. Clip pixels with small values to 0
3. Clip pixels with small gradient to 0



## Fooling Images / Adversarial Examples

이번에는 하나의 class 가 다른 class 로 classify 되도록 image 를 계속 update 하면 어떤 결과가 나오는지 살펴봅시다.

African elephant



koala

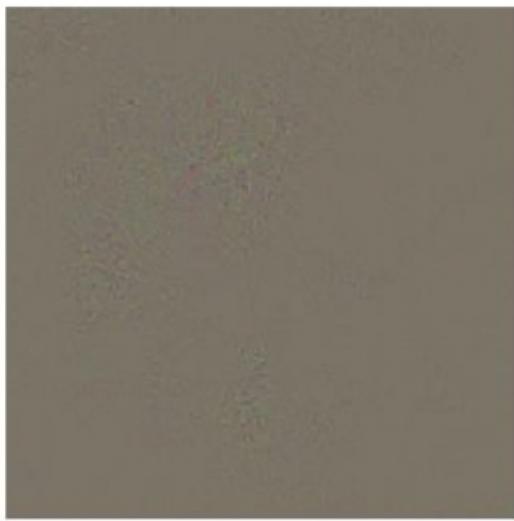


코끼리가 두마리 있습니다. 근데 왼쪽은 코끼리로 정확한 classification 이 됐지만 오른쪽은 koala 로 classify 됐네요.

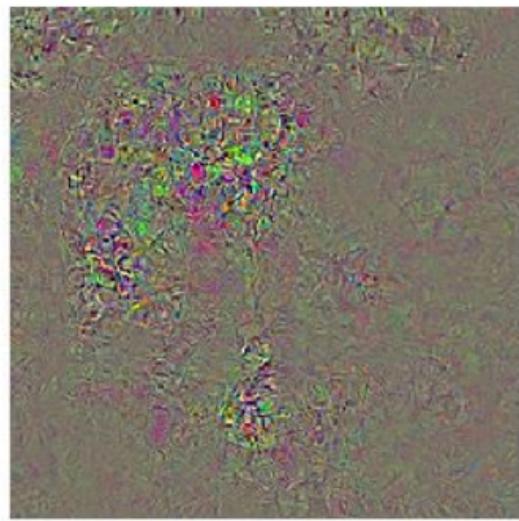
어떻게 된 일일까요?

이 두 이미지의 pixelwise 차이(Difference)를 보면 큰 차이가 있지는 않습니다. 그러나 차이를 10 배로 magnify 하면 차이가 나타나지만, koala 같은 모양이나 elephant 같은 모양같이 의미있는 형태라기보단 노이즈가 보일 뿐입니다.

Difference



10x Difference



### DeepDream: Amplify existing features

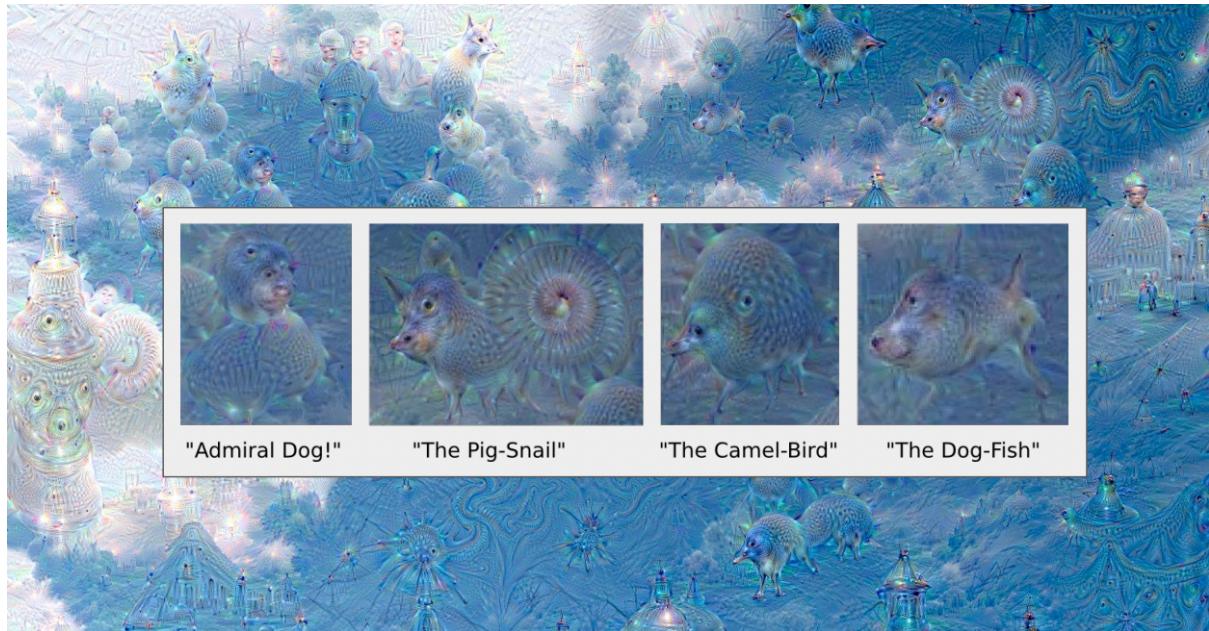
DeepDream은 구글에서 만든것으로, 거의 entertain | 몰↗? 눈호강 + feature들이 무엇을 찾는지를 대략적으로 파악할 수 있게끔 만든 것입니다.

과정 :

1. image 와 CNN 의 한 layer 를 선택한다.
2. layer 까지 forward pass 하고 activation 을 계산한다.
3. layer 의 gradient 를 activation 과 같게 설정한다. (**amplify neuron activations**)
4. Backward pass, update image.

위에서는 특정 **neuron 을 maximaize** 하는 방향으로 시각화를 했다면,

Deep Dream 은 **neuron activations 를 증가시키는** 방향의 차이가 있습니다.



## Feature Inversion

Feature Inversion 은 CNN에서 구한 feature 들만으로 통해 역으로 input 이미지를 생성하는 방법입니다.

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathbb{R}^{H \times W \times C}}{\operatorname{argmin}} \ell(\Phi(\mathbf{x}), \Phi_0) + \lambda \mathcal{R}(\mathbf{x})$$

Given feature vector

Features of new image

$$\ell(\Phi(\mathbf{x}), \Phi_0) = \|\Phi(\mathbf{x}) - \Phi_0\|^2$$

$$\mathcal{R}_{V^\beta}(\mathbf{x}) = \sum_{i,j} \left( (x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\frac{\beta}{2}}$$

Total Variation regularizer  
(encourages spatial smoothness)

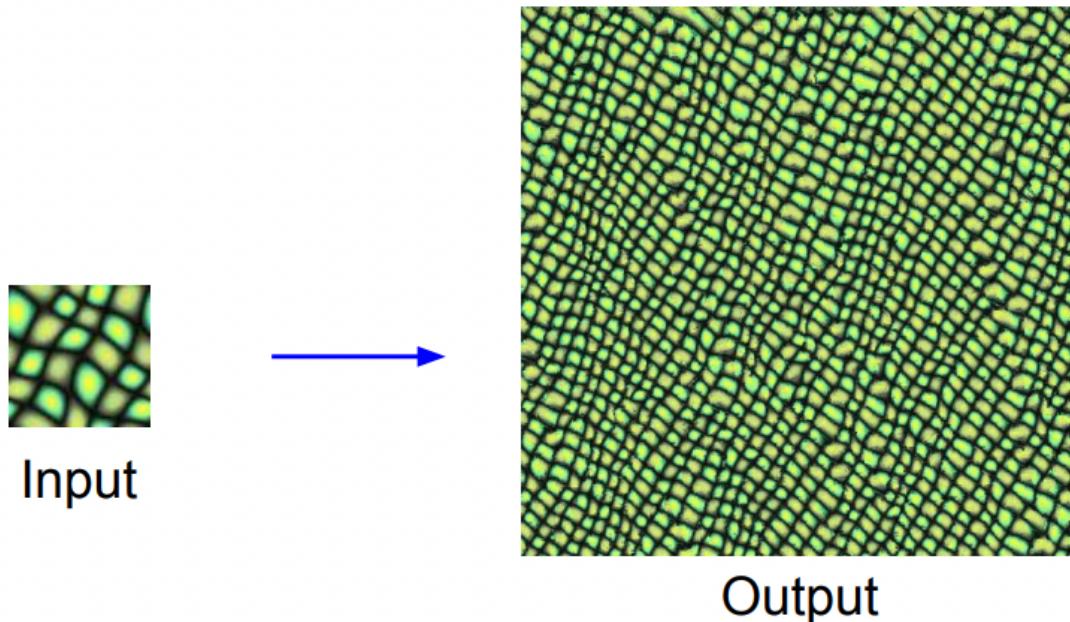
여기서 input  $\mathbf{x}^*$ 는 주어진 feature 와 새롭게 생성할 이미지의 feqture 간의 간극을 최소화 시키는 방향으로 gradient ascent 됩니다.

## Texture Synthesis

Given a sample patch of some texture, can we generate a bigger image  
of the same texture?

Texture Synthesis 는 꼭 Neural Network 를 통해서만 할 수 있지 않고 다른  
방법들로도 할 수 있습니다. Nearest Neighbor 등등,,

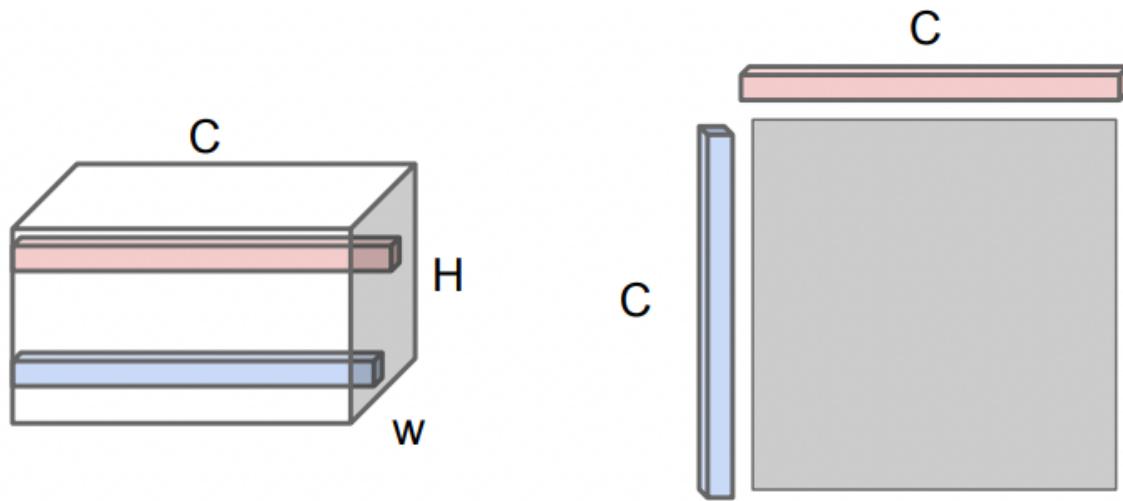
강의는 Computer Vision 강의이기 때문에 NN 방법론에 대해 자세히 다루진  
않습니다.



Gram Matrix (Somewhat Gradient Acsent-ish?)

Texture Synthesis 를 NN 을 적용해 하기 위해서는 Gram Matrix 라는것을 사용합니다.

1. input texture 를 가지고 CNN 에 넣습니다.
2. 어떤 layer 에서 convolution 된 feature  $C \times H \times W$  를 가져옵니다.



$H \times W$  는 spatial grid 로 생각할 수 있고,  $H \times W$  의 한 점마다  $C$  dimension 의 vector 를 지닙니다.

이  $C$  vector 는 이미지의 해당 지점에서 어떤 형상을 띠는지에 대한 정보를 rough 하게 담고 있습니다.

3. 그래서 이 convolution feature 에서  $C$  vector 두개를 선택하고, 외적을 진행해서  $C \times C$  matrix 를 만듭니다.

이  $C \times C$  vector 는 특정 포인트 두점에서 공통적으로 발현되는 성질에 대한 정보를 담고 있습니다.

4. 이  $C \times C$  matrix 를 모든 point 에 대하여 만들고 average 하면 gram matrix 가 생성됩니다.

Gram matrix 는 input image 의 texture 의 정보를 담는 일종의 descriptor 역할을 합니다.

\* Gram Matrix 는 average 과정에서 spatial information 을 다 버렸습니다.

\* 모든 경우의 수를 다 구해서 computation complexity 가 되게 별로일거 같지만,  $Cx(HW)$ 와  $Cx(HW)$ 의 Transpose 를 곱하면 되므로 계산이 오히려 굉장히 수월합니다.

이제 구한 descriptor 를 가지고 아래의 neural process 를 거치면 Neural Texture Synthesis 가 완성됩니다.!

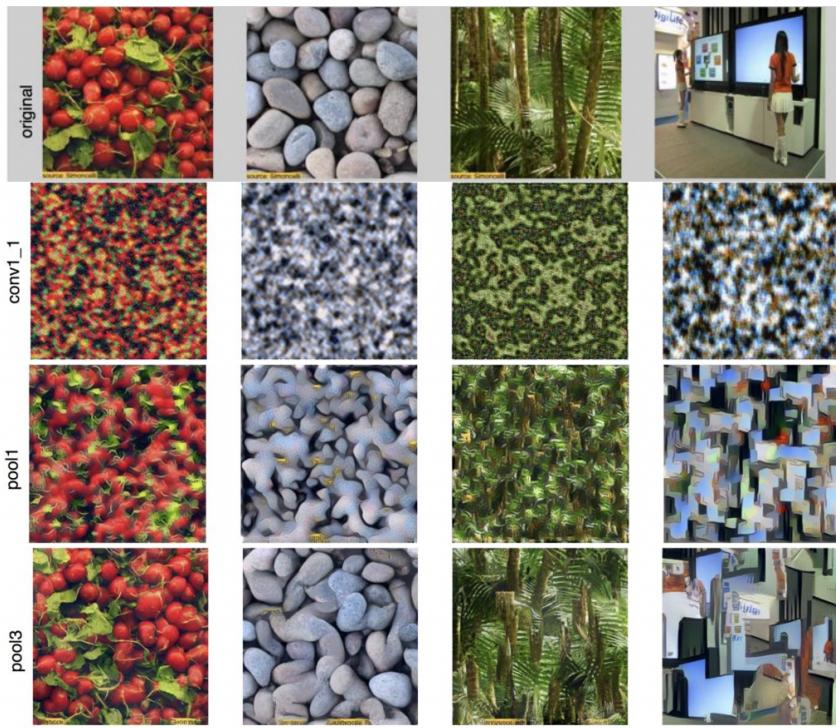
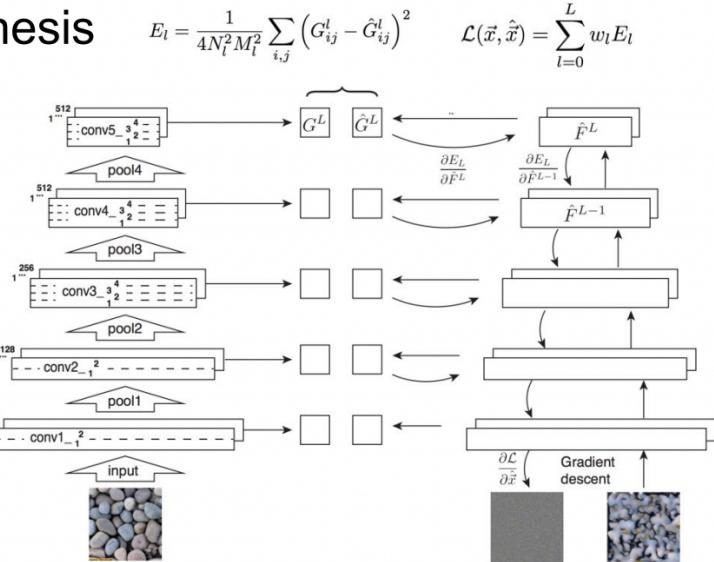
# Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer  $i$  gives feature map of shape  $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l \text{ (shape } C_i \times C_i\text{)}$$

4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5

Gatys, Ecker, and Bethge, "Texture Synthesis Using Convolutional Neural Networks", NIPS 2015



결과