Lecture 8 - Visualizing, Training & Designing ConvNets

DD2424

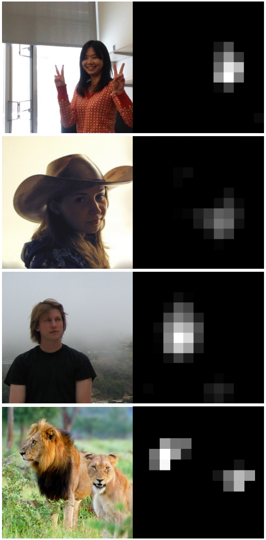
April 24, 2017

Overview of today's lecture

- **Part 1**: Visualizing what a deep ConvNet learns.
- **Part 2**: Practicalities of training & designing ConvNets
  - Data augmentation.
  - Transfer learning.
  - Stacking convolutional filters.

- Visualize patches that maximally activate neurons.
- Occlusion experiments.
- Visualize the weights.
- Deconv approaches (single backward pass).
- Optimization over image approaches (optimization).

[**Understanding Neural Networks Through Deep Visualization** by Yosinski et al, 2015]



- 13×13 activations from a channel in a conv response volume.
- 151st channel of the conv5 layer of a deep ConvNet.
- The ConvNet trained on ImageNet.
- Know this channel responds to human and animal faces.

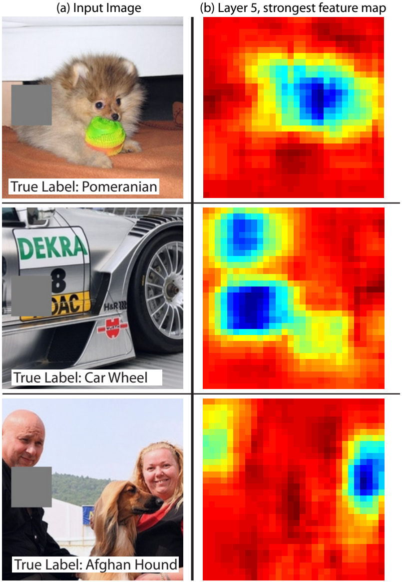
Visualize the features that maximally activate neurons

[**Rich feature hierarchies for accurate object detection and semantic segmentation** by Girshick, Donahue, Darrell & Malik, 2013]

Apply AlexNet to image regions (not used in training):

- Each row displays the 16 strongest activations for a particular pool5 unit (response volume before the 1st fully connected layer).
- Receptive fields and activation values are drawn in white.

AlexNet seems to learn class-tuned features together with a distributed representation of shape, texture, color, and material properties.



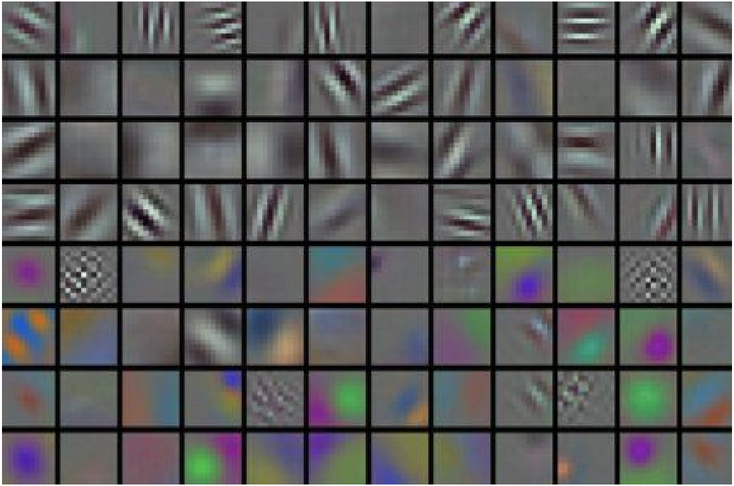
Experiment

- Occlude a small square patch of the image.
- Apply ConvNet to occluded image.
- Sum the responses from one channel in the layer 5 response volume. (channel is chosen as the one that gave the largest response for the unoccluded image.)

- Slide the occlusion patch over the whole image.

- Record the response sum for each position of the occlusion patch.

Visualize the filters/kernels (raw weights)



Only interpretable on the first layer.

Visualize the filters/kernels (raw weights)

you can still do it for higher layers, it's just not that interesting

(these are taken from ConvNetJS CIFAR-10 demo)

layer 1 weights

layer 2 weights

layer 3 weights

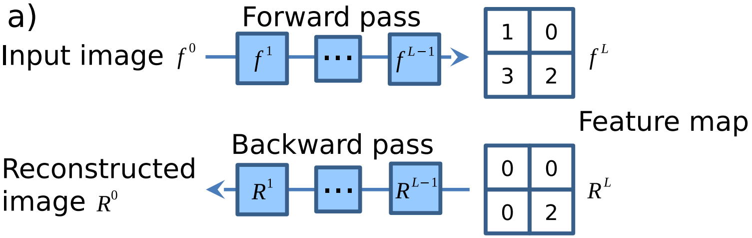
How can we visualize higher layers? DeConvNets

[**Visualizing and Understanding Convolutional Networks** by Zeiler & Fergus, 2013]

- Visualization technique that gives insight into the function of intermediate feature layers.
- DeConvNet maps a feature activity back to the input pixel space.
- Generates an input pattern that gives a certain individual activation in the feature maps.
- A DeConvNet has the same components (filtering, pooling, ReLu) as a ConvNet but applied in reverse order as it tries to invert the ConvNet operations.

Examine a particular ConvNet activation at layer l for an image:

- Apply ConvNet to image.
- Set all activations at layer l to zero except for the activation of interest.



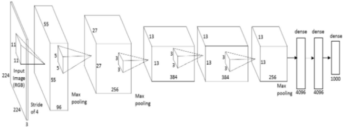
- Pass this volume as input into a **DeConvNet**.

What does a DeConvNet Do?

- Maps a feature volume pattern to a raw image (pixel values).

How?

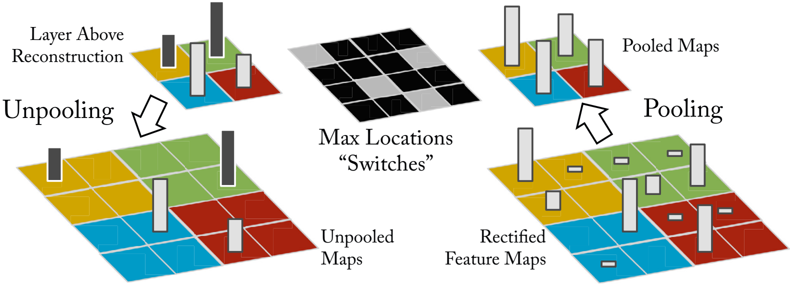
- Assume have a trained ConvNet & applied it to an image.



- DeConvNet then approximately inverts each operation (in sequence) of the original trained ConvNet
  - max-pooling,
  - ReLu,
  - convolution

  to restore the original image from the activities layer of interest.

DeConvNets: (Approx) Inverting the **Max Pool** operation



- **Switches** record the location of the local max in each pooling region during pooling in the convnet.
- The unpooling operation in the deconvnet uses these switches.

The black/white bars are negative/positive activations within the feature map.

DeConvNets: (Approx) Inverting the **Convolution** op

- Know that the convolution of image X by filter F

$$S = X * F$$

can be written as a matrix multiplication

$$\text{vec}(S) = M_F^{\text{filter}} \text{vec}(X)$$

- Let's assume M_F^{filter} is square and orthonormal (most of the columns will definitely be orthogonal as their non-zero entries will be in different rows) then

$$\left(M_F^{\text{filter}}\right)^T M_F^{\text{filter}} = I$$

$$\implies \quad \text{vec}(X) = \left(M_F^{\text{filter}}\right)^T \text{vec}(S)$$

- This matrix multiplication by $(M_F^{\text{filter}})^T$ can be re-written as

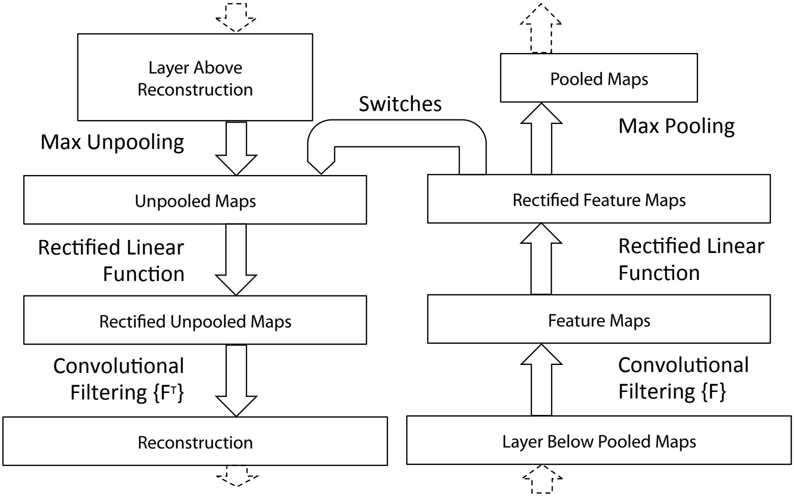
$$X = S * F^{\text{rot180}}$$

The inverting convolution applied by the DeConvNet. (**Note**: similarity to the convolution applied in the back-prop through a convolutional layer)

DeConvNets: Inverting(ish) the **ReLu** operation

- Want to obtain valid feature reconstructions at each layer $\Longrightarrow$ all entries should be non-negative
- Thus DeConvNet passes the reconstructed signal through a ReLu non-linearity.

DeConvNets: (Approx) Inverting the **Max Pool** operation



Deconvnet reconstructs an approximate version of the convnet features from the layer beneath.

Basically DeConv performs back-prop to the input image

DeConvNet procedure is similar to

- Backpropping a single strong activation to the input image.
- Or in mathematical terms computing

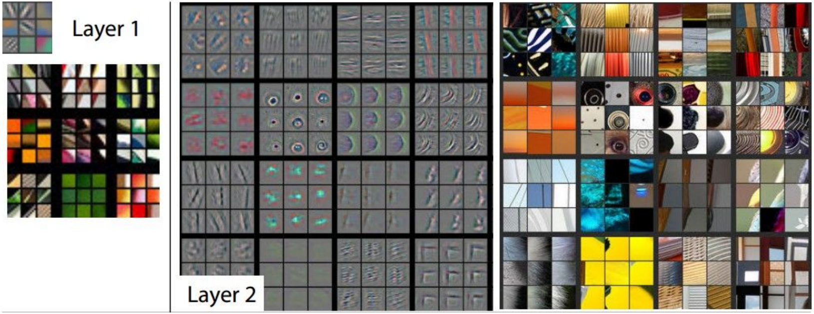
$$\frac{\partial h}{\partial X}$$

  where h is the element of the feature map with strong activation and X is the input image.

There are some technical differences between the two methods in how the ReLu operation is dealt with.

DeConvNet Visualization of arbitrary neurons

[**Visualizing and Understanding Convolutional Networks** by Zeiler & Fergus, 2013]

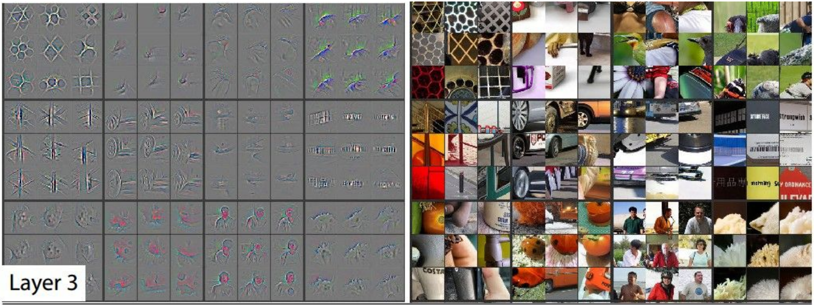


For a random subset of feature maps, show the top 9 activations from the validation set

- projected back to pixel space using the DeConvNet method and
- the corresponding image patches.

DeConvNet Visualization of arbitrary neurons

[**Visualizing and Understanding Convolutional Networks** by Zeiler & Fergus, 2013]

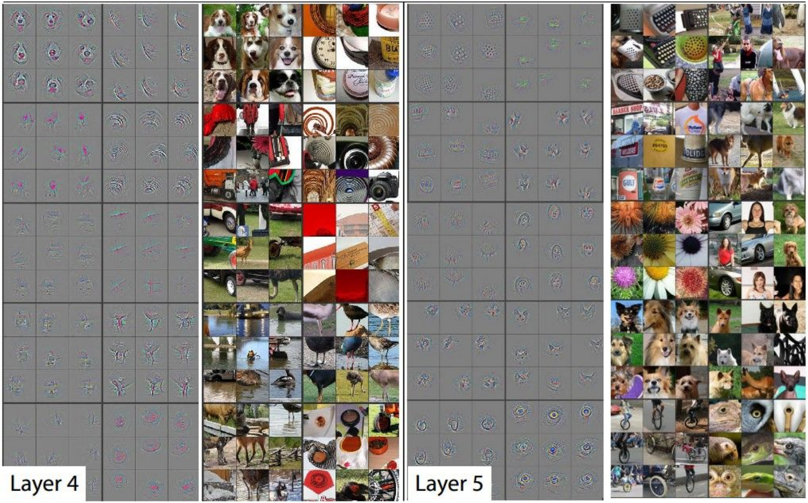


For a random subset of feature maps, show the top 9 activations from the validation set

- projected back to pixel space using the DeConvNet method and
- the corresponding image patches.

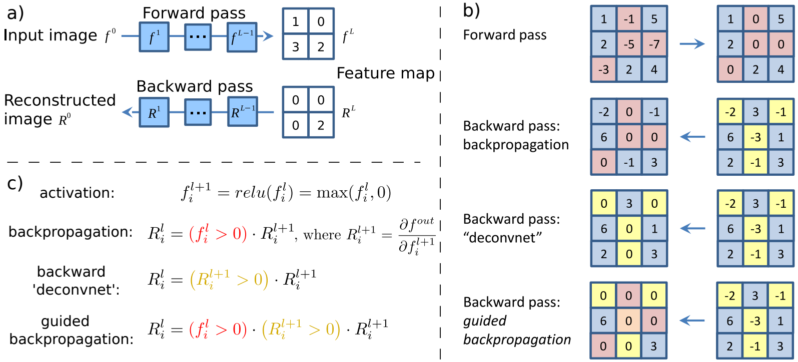
DeConvNet Visualization of arbitrary neurons

[**Visualizing and Understanding Convolutional Networks** by Zeiler & Fergus, 2013]



Guided Backprop: Alternate approach to inverting ReLu

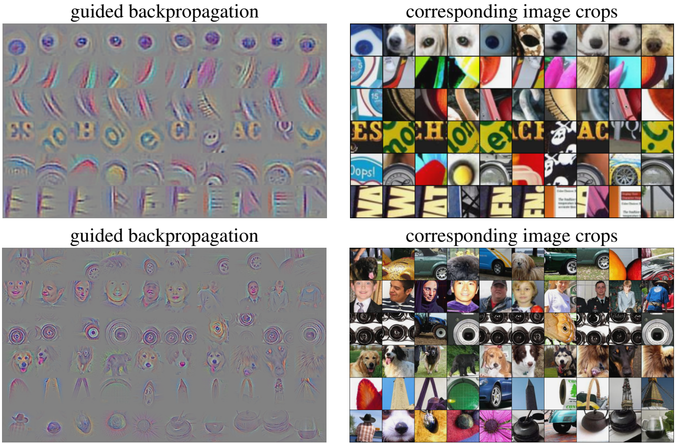
[**Striving for Simplicity: The all convolutional net** by Springenberg, Dosovitskiy, et al., 2015]



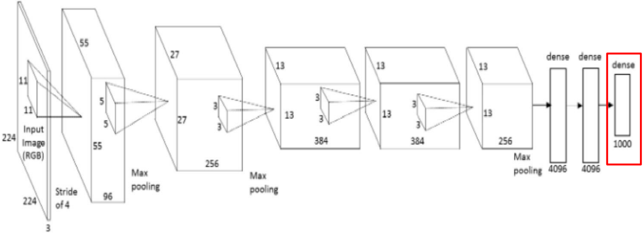
- Different methods of propagating back through a ReLU nonlinearity.
- Prevents backward flow of negative gradients, corresponding to the neurons which decrease the activation of the higher layer unit we aim to visualize.

Guided Backprop visualization of arbitrary neurons

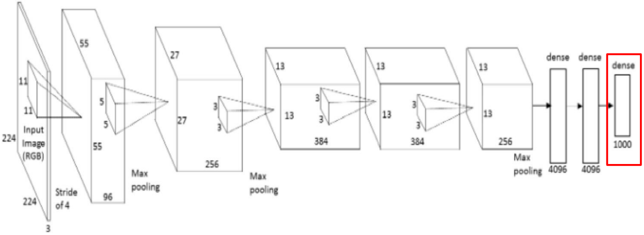
[**Striving for Simplicity: The all convolutional net** by Springenberg, Dosovitskiy, et al., 2015]



- Visualization, using guided backpropagation, of patterns learned by layers conv6 and conv9 features.
- Each row corresponds to one pattern/neuron/activity.
- Based on the top 10 (ImageNet) image patches activating this pattern.



Can we find an image that maximizes some class score?



Can we find an image that maximizes some class score?

- Let s_X represent the unnormalized scores assigned by our network to image X.
- Let y be the class of interest.
- Then problem is to solve

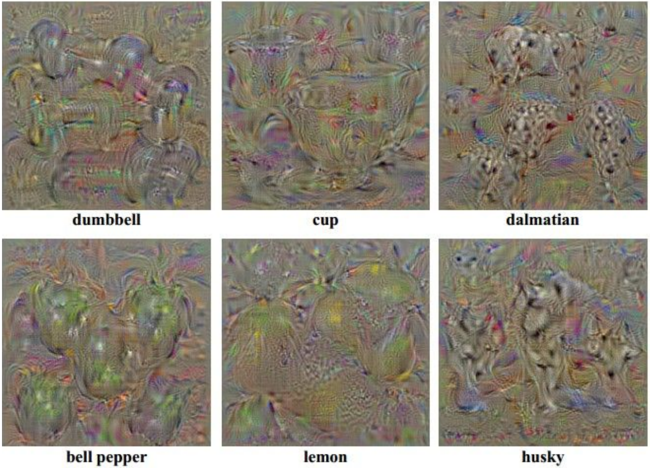
$$\arg\max_X \left(s_{X,y} - \lambda \|X\|_2^2\right)$$

Procedure to find local optimum image

1. Initialize X to be all zeros.
2. Apply ConvNet to X (forward pass)
3. Set the gradient of cost w.r.t. $\mathbf{s}$ equal to one-hot representation of y.
4. Backprop to the gradient to the image (X) node.
5. Do a small "image update".
6. Go back to step 2.

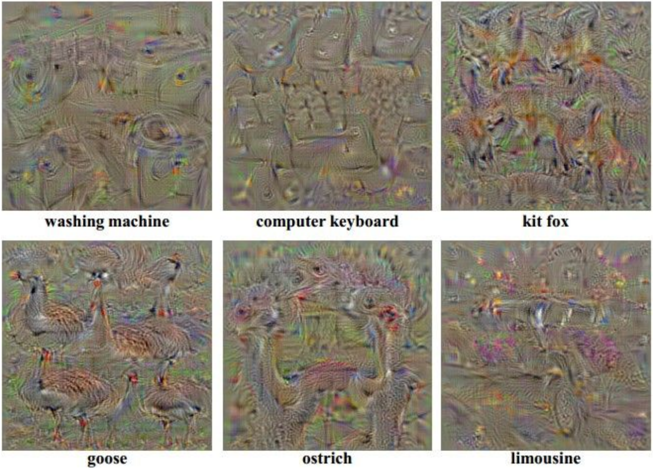
Example results

[**Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps** by Simonyan, Vedaldi & Zisserman, 2014]

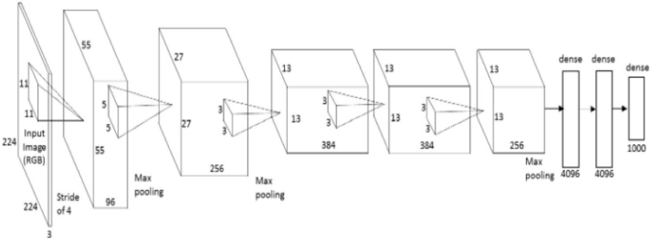


Example results

[**Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps** by Simonyan, Vedaldi & Zisserman, 2014]



Can do this for any ConvNet response

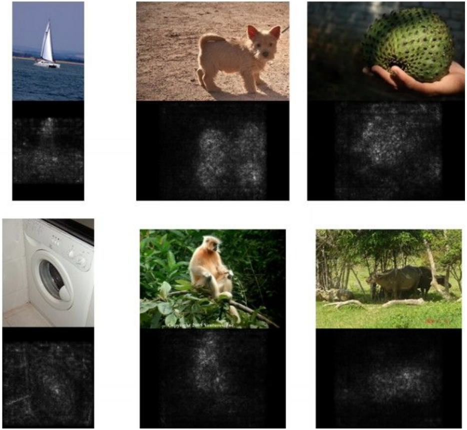


Repeat:

- Forward image estimate
- Set activations in layer of interest to all zero, except for a 1.0 for neuron of interest.
- Backprop to image.
- Update image estimate.

Visualize the data gradient

[**Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps** by Simonyan, Vedaldi & Zisserman, 2014]



$$G_{ij} = \max_k \left| \frac{\partial s_y}{dX_{ijk}} \right|$$

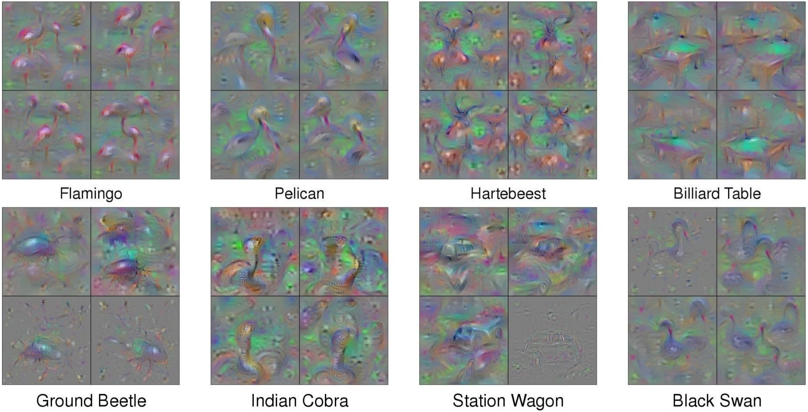
[**Understanding Neural Networks Through Deep Visualization** by Yosinski et al, 2015]

- **Problem**: Find an image that maximizes a class score + regularization term

$$\arg\max_X \ (s_{X,y} - \lambda R(X))$$

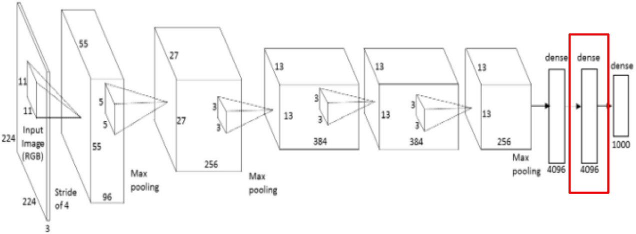
- Solution:
  Repeat
  1. Update the image X with gradient from some unit of interest.
  2. Blur X a bit.
  3. Take any pixel with small norm to zero (to encourage sparsity).

[**Understanding Neural Networks Through Deep Visualization** by Yosinski et al, 2015]



Reconstruct an image from its ConvNet encoding

Have ConvNet code: Possible to reconstruct the original image?



Reconstruct an image from its ConvNet encoding

Find an image s.t.:

- Its code is similar to a given code **and**
- It looks like a real image.

Mathematical statement:

$$X^* = \underset{X \in \mathbb{R}^{W \times H \times 3}}{\arg\max} \left(\|\Phi(X) - \Phi_0\|^2 + \lambda R(X) \right)$$

Reconstruct an image from its ConvNet encoding

[**Understanding Deep Image Representations by Inverting Them** by Mahendran and Vedaldi, 2014]

original image

Reconstructions from the 1000 class score layer.

Reconstruct an image from its ConvNet encoding

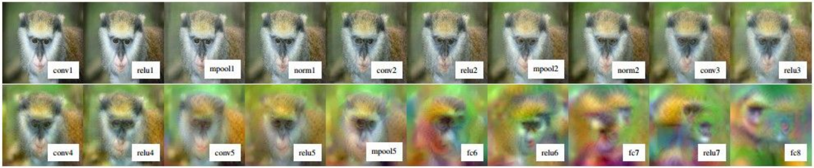
[**Understanding Deep Image Representations by Inverting Them** by Mahendran and Vedaldi, 2014]

Reconstructions from the representation after last pooling layer (immediately before the first Fully Connected layer).

Reconstruct an image from its ConvNet encoding

[**Understanding Deep Image Representations by Inverting Them** by Mahendran and Vedaldi, 2014]

original image

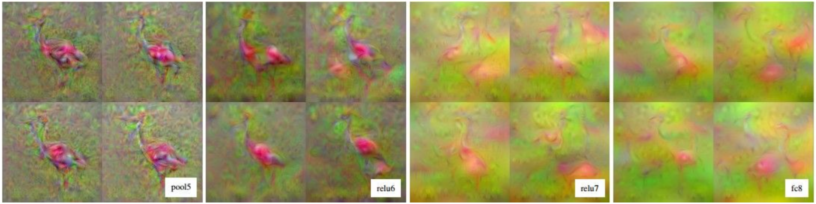


Reconstructions from intermediate layers.

Reconstruct an image from its ConvNet encoding

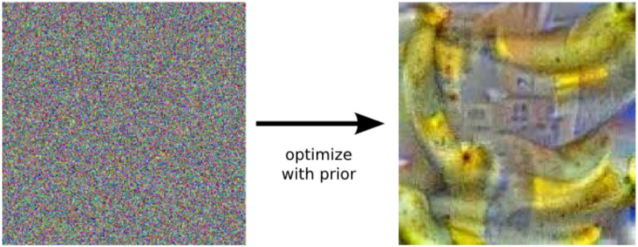
[**Understanding Deep Image Representations by Inverting Them** by Mahendran and Vedaldi, 2014]

original image



Multiple reconstructions.
Images in quadrants produce the same ConvNet encoding.

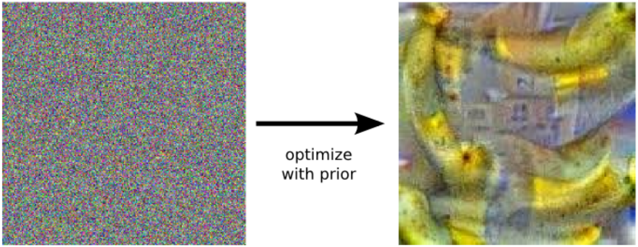
- Start with random noise image X and give it label y.
- Iterate
  - Apply ConvNet to X to get probabilities $\mathbf{p}$ for each class label.
  - Update X so p_y increases in tandem with a prior that neighbouring pixel values should be correlated.



- Start with random noise image X and give it label y.
- Iterate
  - Apply jitter translation to X to get X_{jitter}
  - Apply ConvNet to X_{jitter} (forward pass)
  - Compute gradient $\frac{\partial l}{\partial X}\Big|_{X_{\text{jitter}}}$ (backward pass)
  - Apply update step:

$$X_{\text{jitter}} = X_{\text{jitter}} + \eta \frac{\partial p_y}{\partial X}\Big|_{X_{\text{jitter}}}$$

  - Undo jitter translation $X_{\text{jitter}} \to X$

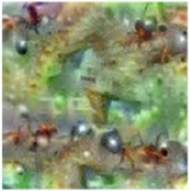


Google's DeepDream

More examples from a random initialization:

Hartebeest

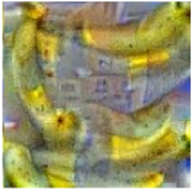
Measuring Cup



Ant

Starfish

Anemone Fish



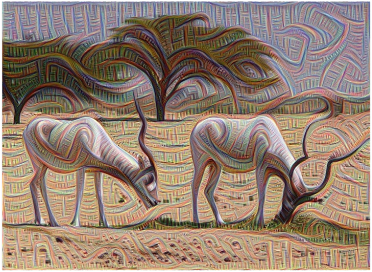
Banana

Parachute

Screw

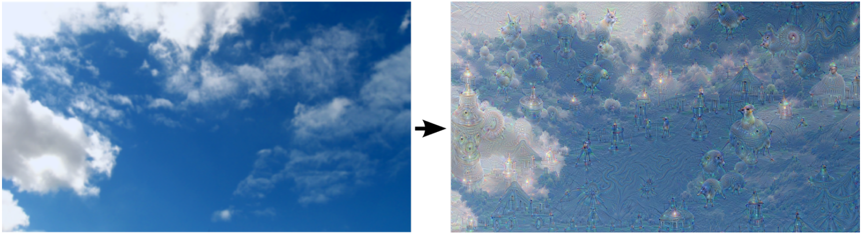
Dumb bells

- Feed the network an image.
- Pick a layer and try to increase positive responses.
- Apply a gradient ascent approach.



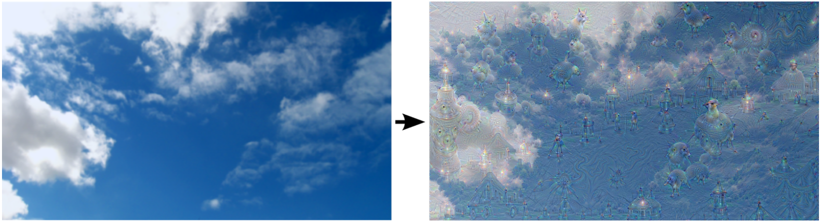
Lower layer chosen.

- Feed the network an image.
- Pick a layer and try to increase positive responses.
- Apply a gradient ascent approach.



Higher layer chosen.

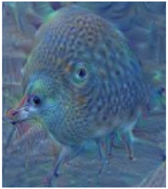
Google's DeepDream



Higher layer chosen.

"Admiral Dog!"

"The Pig-Snail"



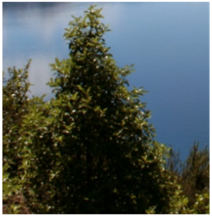
"The Camel-Bird"

"The Dog-Fish"

Close-up on some structures created

Some more examples

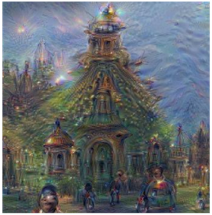
Horizon



Trees

Leaves

Towers & Pagodas



Buildings

Birds & Insects

Apply Google's DeepDream iteratively

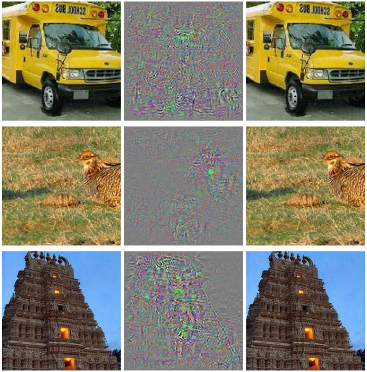
All images generated from a random noise image.

We can design an optimization problem w.r.t. the input image to maximize any class score.

Question: Can we use this to "fool" ConvNets?

Fooling a Neural Network

[**Intriguing properties of neural networks** by Szegedy et al., 2013]



$\mathbf{x}$ $\mathbf{r}$ $\mathbf{x} + \mathbf{r}$

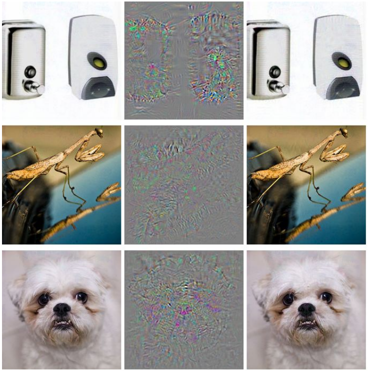
$y' =$ostrich

- Train a ConvNet.
- $\mathbf{x}$ a test image correctly classified by the ConvNet to have label y.
- Let $\mathbf{x} + \mathbf{r}$ be the closest image to $\mathbf{x}$ s.t.

  $\mathbf{x} + \mathbf{r}$ is classified by the ConvNet to have label $y' \neq y$.

Fooling a Neural Network

[**Intriguing properties of neural networks** by Szegedy et al., 2013]



$\mathbf{x}$ $\mathbf{r}$ $\mathbf{x}+\mathbf{r}$

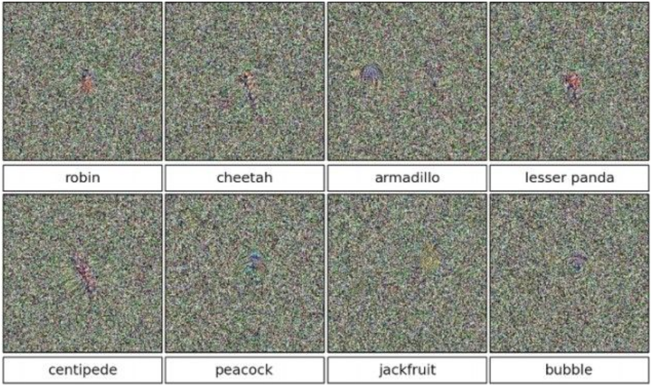
y' =ostrich

- Train a ConvNet.
- $\mathbf{x}$ a test image correctly classified by the ConvNet to have label y.
- Let $\mathbf{x}+\mathbf{r}$ be the closest image to $\mathbf{x}$ s.t.

  $\mathbf{x}+\mathbf{r}$ is classified by the ConvNet to have label $y' \neq y$.

Fooling a Neural Network

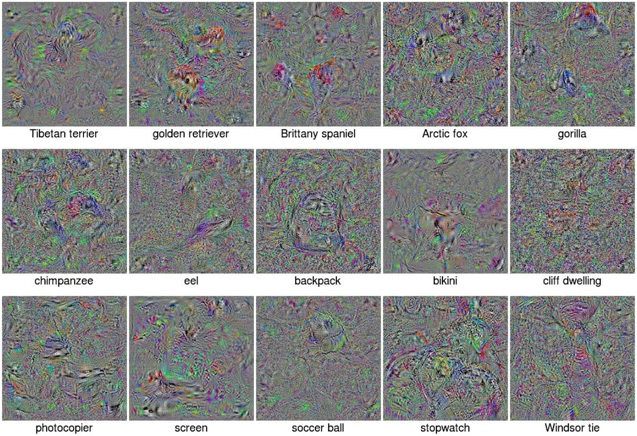
[**Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images** by Nguyen, Yosinski, Clune, 2014]



- Train a high-performance ConvNet for image classification.
- Randomly initialize an image $\mathbf{x}$.
- Iteratively update $\mathbf{x}$ to get high-confidence ConvNet score $(> 99.5\%)$ for label y.
- This paper uses a *genetic algorithm* to produce updates for $\mathbf{x}$.

Fooling a Neural Network

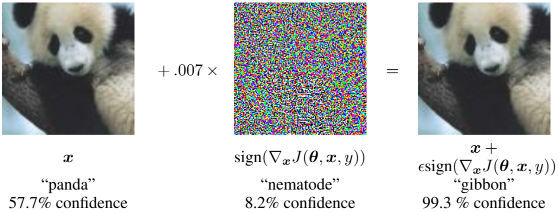
[**Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images** by Nguyen, Yosinski, Clune, 2014]



- Initialize image $\mathbf{x}$ with ImageNet mean + noise.
- Iteratively update $\mathbf{x}$ to get high-confidence ConvNet score $(> 99.99\%)$ for label y.
- This example used gradient of the loss w.r.t. $\mathbf{x}$ to produce updates.

Why can we generate these adversarial examples?

[EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES by Goodfellow, Shlens & Szegedy, 2014]



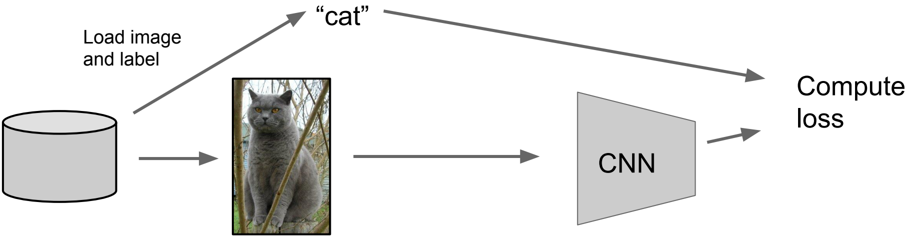
- Adversarial examples a property of high-dimensional dot products.
- They are a result of models being too linear, rather than too nonlinear.
- Direction of perturbation matters most.
- Perturbation direction results in adversarial example when highly aligned with the weight vectors of the network.
- Space is not full of pockets of adversarial examples.

Not a problem specific to Deep Learning or ConvNets.

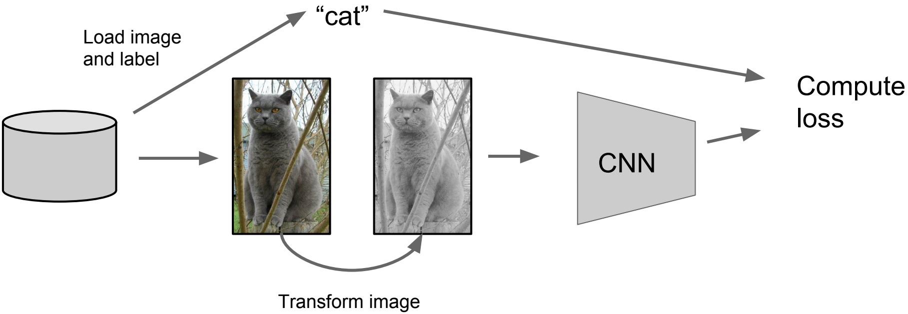
Same issue exists for shallow Neural Nets.

Data Augmentation

Data Augmentation

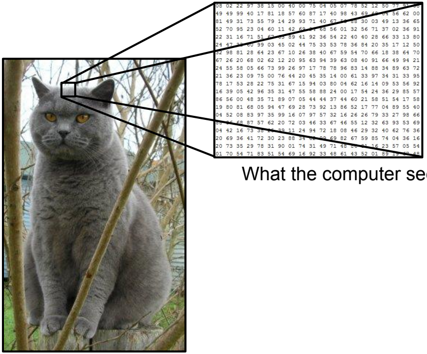


Data Augmentation



Data Augmentation

- Change the pixels without changing the label
- Train on transformed data
- VERY widely used



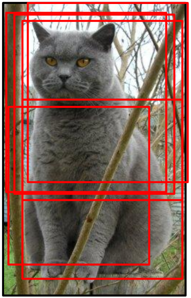
What the computer sees

Data Augmentation

1. Horizontal flips

Data Augmentation
2. Random crops/scales

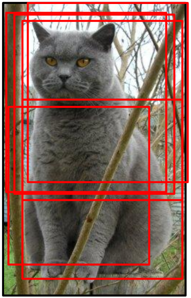
Training: sample random crops / scales



Data Augmentation
2. Random crops/scales

Training: sample random crops / scales

ResNet:
1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch



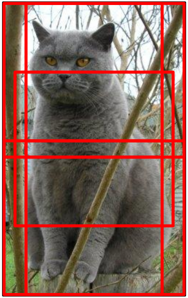
Data Augmentation
2. Random crops/scales

Training: sample random crops / scales

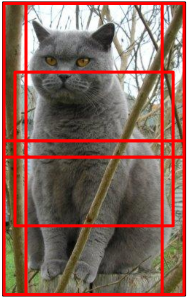
ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

Testing: average a fixed set of crops



Data Augmentation
2. Random crops/scales



Training: sample random crops / scales

ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

Testing: average a fixed set of crops

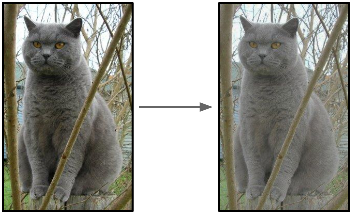
ResNet:

1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224 x 224 crops: 4 corners + center, + flips

Data Augmentation

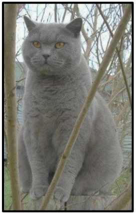
3. Color jitter

Simple:
Randomly jitter contrast



Data Augmentation
3. Color jitter

Simple:
Randomly jitter contrast



Complex:

1. Apply PCA to all [R, G, B] pixels in training set
2. Sample a “color offset” along principal component directions
3. Add offset to all pixels of a training image

(As seen in *[Krizhevsky et al. 2012],* ResNet, etc)

Data Augmentation

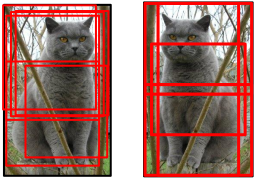
4. Get creative!

Random mix/combinations of :

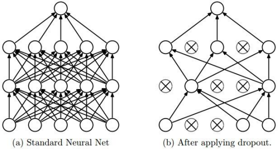
- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

A general theme:

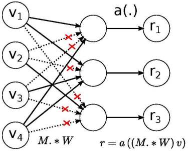
1. **Training**: Add random noise
2. **Testing**: Marginalize over the noise



Data Augmentation



Dropout



DropConnect

Batch normalization, Model ensembles

Data Augmentation: Takeaway

- Simple to implement, use it
- Especially useful for small datasets
- Fits into framework of noise / marginalization

Transfer Learning

"You need a lot of a data if you want to train/use CNNs"

Transfer Learning

“You need a lot of a data if you want to train/use CNNs”

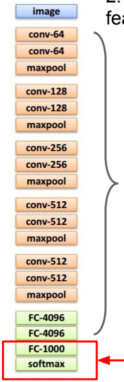
BUSTED

Transfer Learning with CNNs

1. Train on Imagenet

Transfer Learning with CNNs

1. Train on Imagenet

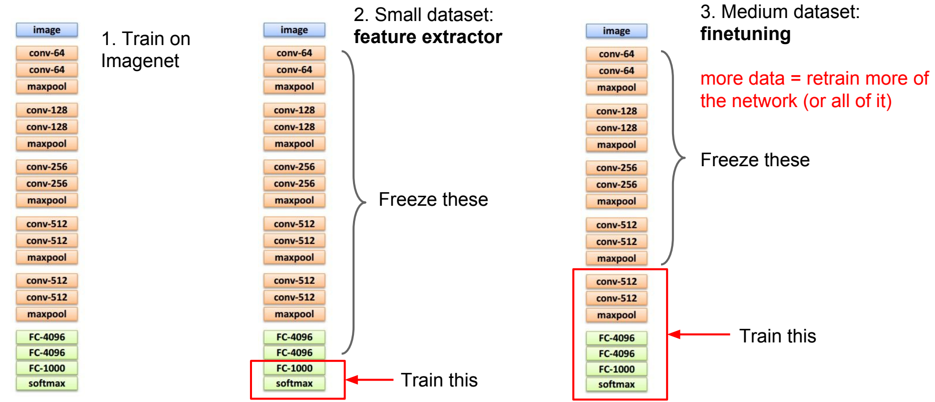


2. Small dataset: feature extractor

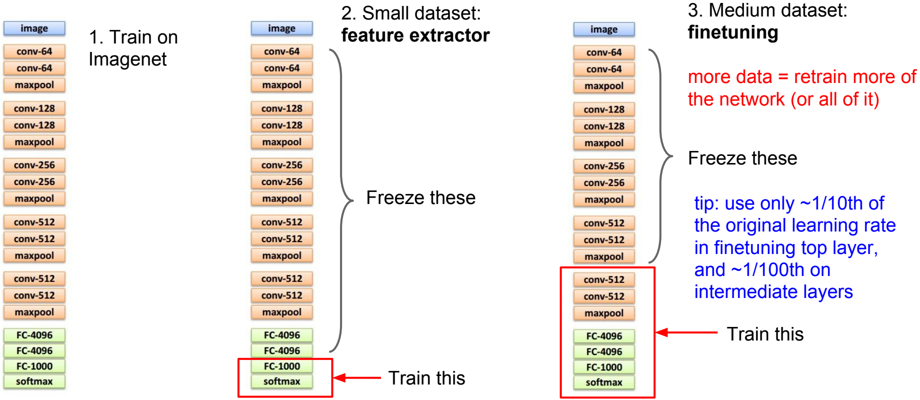
Freeze these

Train this

Transfer Learning with CNNs



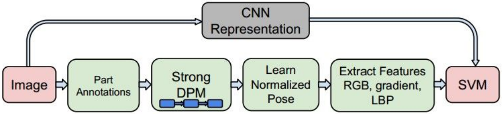
Transfer Learning with CNNs

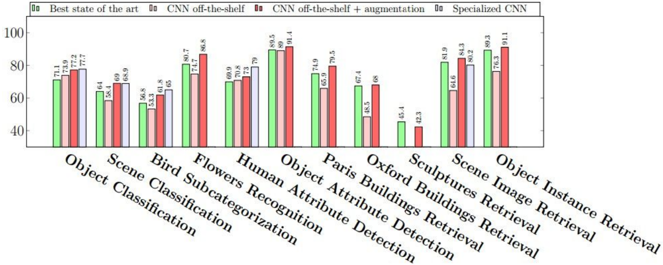


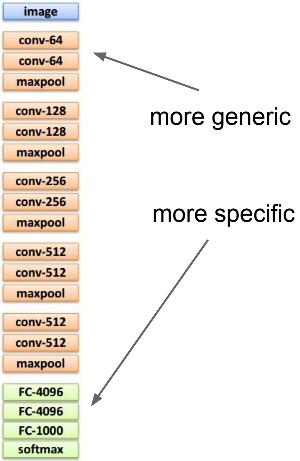
CNN Features off-the-shelf: an Astounding Baseline for Recognition [Razavian et al, 2014]

DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition [Donahue, Jia*, et al., 2013]*

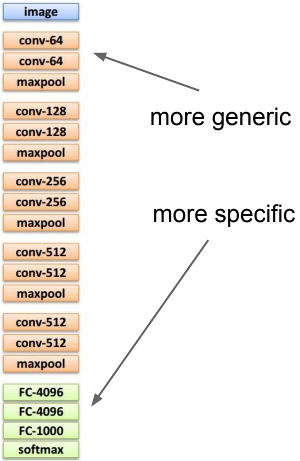
	$DeCAF_6$	$DeCAF_7$
LogReg	**40.94 ± 0.3**	40.84 ± 0.3
SVM	39.36 ± 0.3	40.66 ± 0.3
Xiao et al. (2010)	38.0	



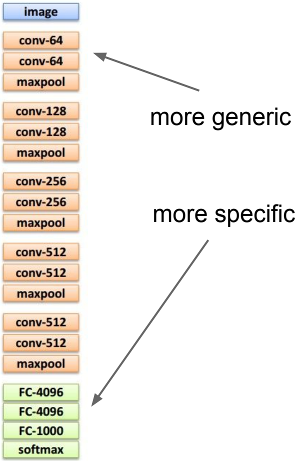




	very similar dataset	very different dataset
very little data	?	?
quite a lot of data	?	?

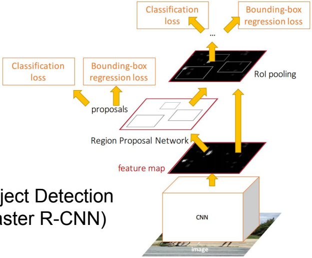


	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	?
quite a lot of data	Finetune a few layers	?



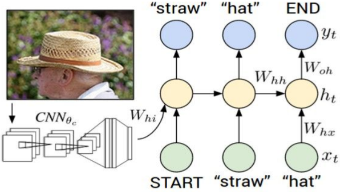
	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You’re in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Transfer learning with CNNs is pervasive…
(it's the norm, not an exception)

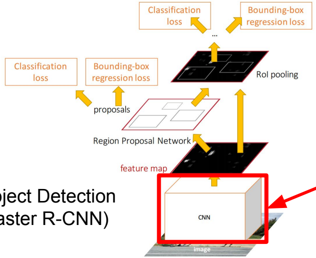


Object Detection
(Faster R-CNN)

Image Captioning: CNN + RNN



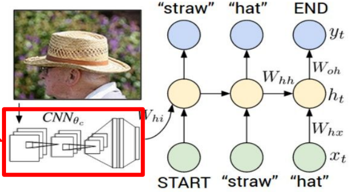
Transfer learning with CNNs is pervasive…
(it's the norm, not an exception)



Object Detection
(Faster R-CNN)

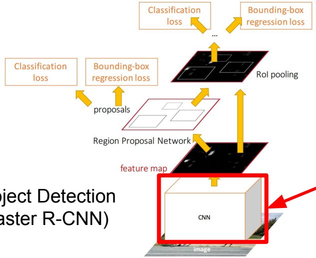
CNN pretrained
on ImageNet

Image Captioning: CNN + RNN



Transfer learning with CNNs is pervasive…

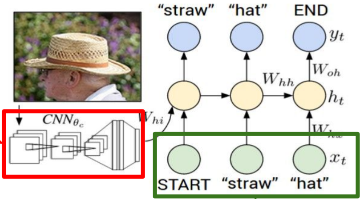
(it's the norm, not an exception)



Object Detection
(Faster R-CNN)

CNN pretrained
on ImageNet

Image Captioning: CNN + RNN



Word vectors pretrained
from word2vec

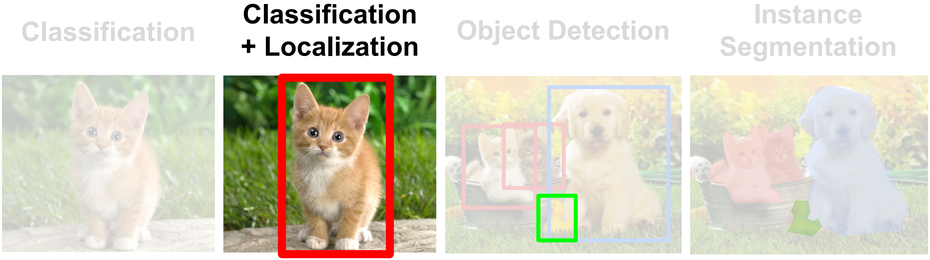
Takeaway for your projects/beyond:

Have some dataset of interest but it has < ~1M images?

1. Find a very large dataset that has similar data, train a big ConvNet there.
2. Transfer learn to your dataset

Caffe ConvNet library has a **"Model Zoo"** of pretrained models:
https://github.com/BVLC/caffe/wiki/Model-Zoo

Computer Vision Tasks



Classification + Localization: Task

Classification: C classes
Input: Image
Output: Class label
Evaluation metric: Accuracy

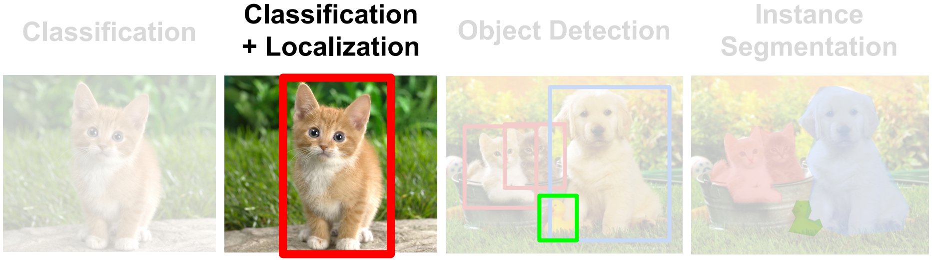
→ CAT

Localization:
Input: Image
Output: Box in the image (x, y, w, h)
Evaluation metric: Intersection over Union

→ (x, y, w, h)

Classification + Localization: Do both

Computer Vision Tasks



Classification + Localization: Task

Classification: C classes
Input: Image
Output: Class label
Evaluation metric: Accuracy

CAT

Localization:
Input: Image
Output: Box in the image (x, y, w, h)
Evaluation metric: Intersection over Union

(x, y, w, h)

Classification + Localization: Do both

Idea #1: Localization as Regression

Input: image

Neural Net

Output:
Box coordinates
(4 numbers)

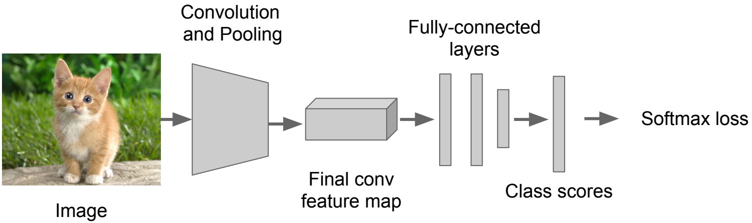
Correct output:
box coordinates
(4 numbers)

Loss:
L2 distance

Only one object,
simpler than detection

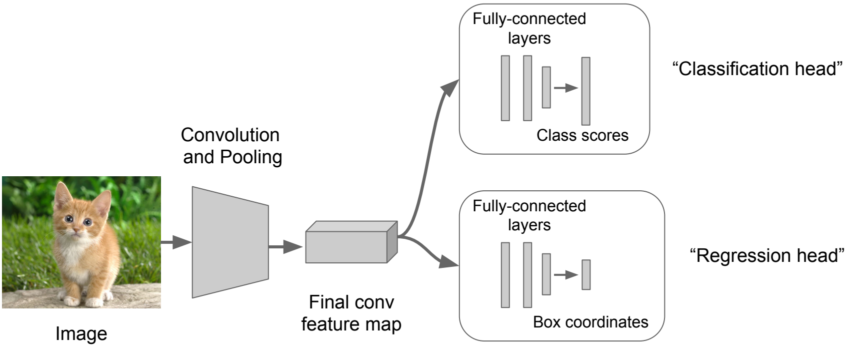
Simple Recipe for Classification + Localization

Step 1: Train (or download) a classification model (AlexNet, VGG, GoogLeNet)



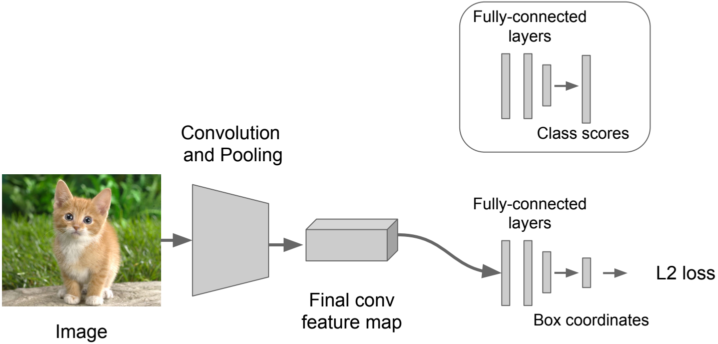
Simple Recipe for Classification + Localization

Step 2: Attach new fully-connected "regression head" to the network



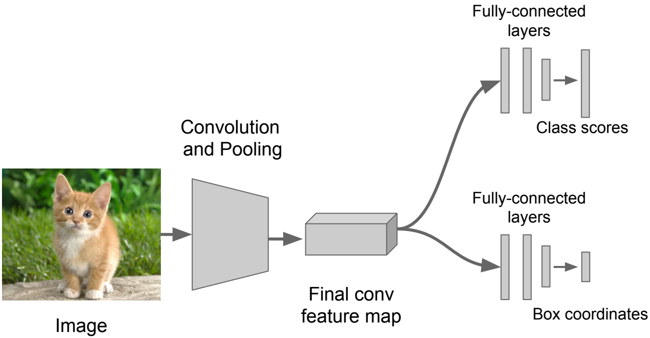
Simple Recipe for Classification + Localization

Step 3: Train the regression head only with SGD and L2 loss



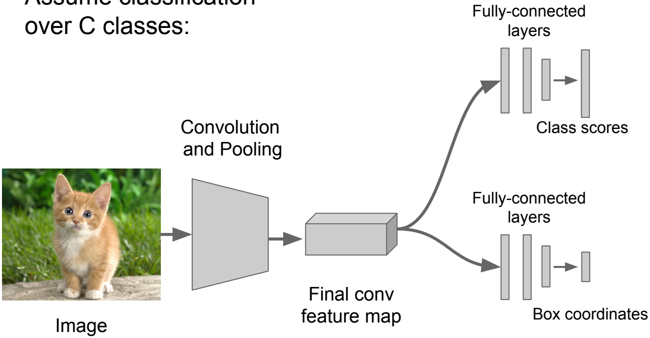
Simple Recipe for Classification + Localization

Step 4: At test time use both heads



Per-class vs class agnostic regression

Assume classification over C classes:

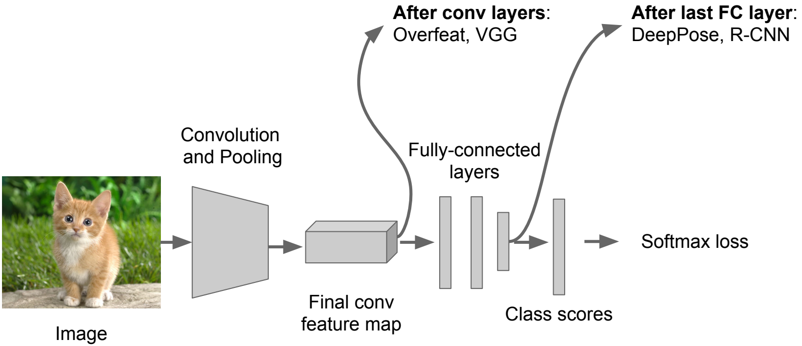


Classification head:
C numbers
(one per class)

Class agnostic:
4 numbers
(one box)

Class specific:
C x 4 numbers
(one box per class)

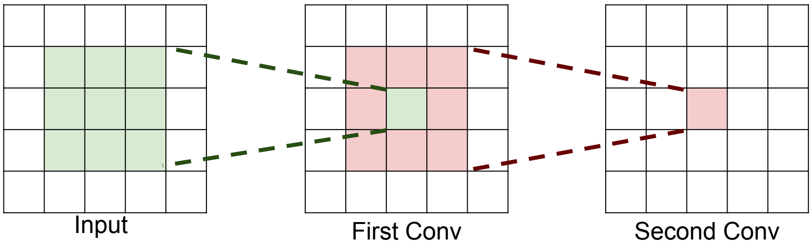
Where to attach the regression head?



How to stack convolutional layers efficiently?

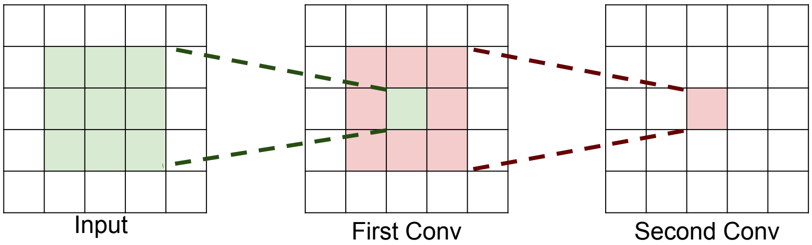
The power of small filters

Suppose we stack two 3x3 conv layers (stride 1)
Each neuron sees 3x3 region of previous activation map



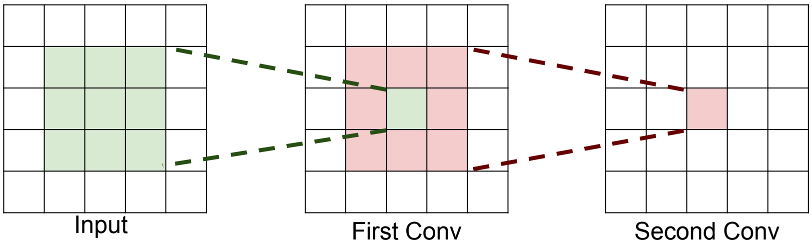
The power of small filters

Question: How big of a region in the input does a neuron on the second conv layer see?



The power of small filters

Question: How big of a region in the input does a neuron on the second conv layer see?
Answer: 5 x 5



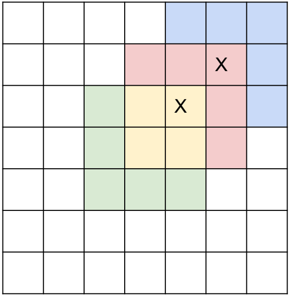
The power of small filters

Question: If we stack **three** 3x3 conv layers, how big of an input region does a neuron in the third layer see?

The power of small filters

Question: If we stack **three** 3x3 conv layers, how big of an input region does a neuron in the third layer see?

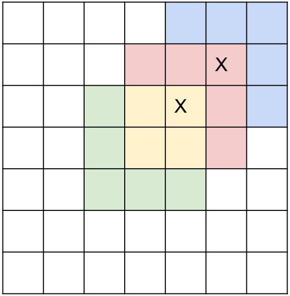
Answer: 7 x 7



The power of small filters

Question: If we stack **three** 3x3 conv layers, how big of an input region does a neuron in the third layer see?

Answer: 7 x 7



Three 3 x 3 conv gives similar representational power as a single 7 x 7 convolution

The power of small filters

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

The power of small filters

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

one CONV with 7 x 7 filters

Number of weights:

three CONV with 3 x 3 filters

Number of weights:

The power of small filters

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

one CONV with 7 x 7 filters

Number of weights:
= C x (7 x 7 x C) = **49 C^2**

three CONV with 3 x 3 filters

Number of weights:
= 3 x C x (3 x 3 x C) = **27 C^2**

The power of small filters

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

one CONV with 7 x 7 filters

Number of weights:
$= C \times (7 \times 7 \times C) =$ **$49\ C^2$**

three CONV with 3 x 3 filters

Number of weights:
$= 3 \times C \times (3 \times 3 \times C) =$ **$27\ C^2$**

Fewer parameters, more nonlinearity = GOOD

The power of small filters

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

one CONV with 7 x 7 filters

Number of weights:
$= C \times (7 \times 7 \times C) = 49\ C^2$

Number of multiply-adds:

three CONV with 3 x 3 filters

Number of weights:
$= 3 \times C \times (3 \times 3 \times C) = 27\ C^2$

Number of multiply-adds:

The power of small filters

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

one CONV with 7 x 7 filters

Number of weights:
= C x (7 x 7 x C) = 49 C^2

Number of multiply-adds:
= (H x W x C) x (7 x 7 x C)
= **49 HWC^2**

three CONV with 3 x 3 filters

Number of weights:
= 3 x C x (3 x 3 x C) = 27 C^2

Number of multiply-adds:
= 3 x (H x W x C) x (3 x 3 x C)
= **27 HWC^2**

The power of small filters

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

one CONV with 7 x 7 filters

Number of weights:
$= C \times (7 \times 7 \times C) = 49\ C^2$

Number of multiply-adds:
$= \mathbf{49\ HWC^2}$

three CONV with 3 x 3 filters

Number of weights:
$= 3 \times C \times (3 \times 3 \times C) = 27\ C^2$

Number of multiply-adds:
$= \mathbf{27\ HWC^2}$

Less compute, more nonlinearity = GOOD

The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?

The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?

$$H \times W \times C \xrightarrow{\text{Conv 1x1, C/2 filters}} H \times W \times (C / 2)$$

1. "bottleneck" 1 x 1 conv to reduce dimension

The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?

H x W x C

Conv 1x1, C/2 filters ↓

H x W x (C / 2)

Conv 3x3, C/2 filters ↓

H x W x (C / 2)

1. "bottleneck" 1 x 1 conv to reduce dimension
2. 3 x 3 conv at reduced dimension

The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?

H x W x C

↓ Conv 1x1, C/2 filters

H x W x (C / 2)

↓ Conv 3x3, C/2 filters

H x W x (C / 2)

↓ Conv 1x1, C filters

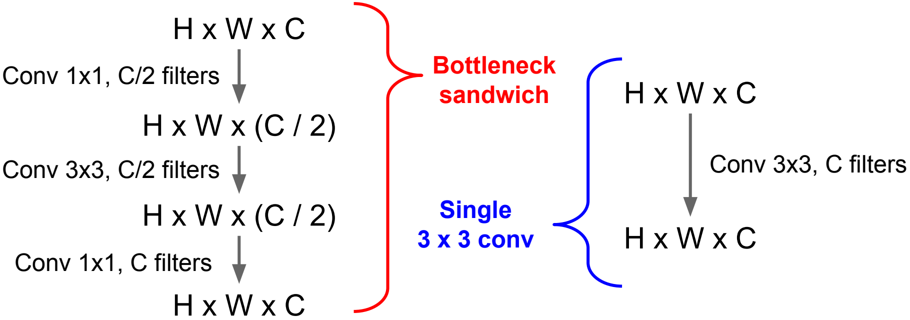
H x W x C

1. "bottleneck" 1 x 1 conv to reduce dimension
2. 3 x 3 conv at reduced dimension
3. Restore dimension with another 1 x 1 conv

[Seen in Lin et al, "Network in Network", GoogLeNet, ResNet]

The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?



The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?

More nonlinearity, fewer params, less compute!

H x W x C

Conv 1x1, C/2 filters

H x W x (C / 2)

Conv 3x3, C/2 filters

H x W x (C / 2)

Conv 1x1, C filters

H x W x C

3.25 C^2 parameters

9 C^2 parameters

H x W x C

Conv 3x3, C filters

H x W x C

The power of small filters

Still using 3 x 3 filters … can we break it up?

The power of small filters

Still using 3 x 3 filters … can we break it up?

H x W x C

Conv 1x3, C filters ↓

H x W x C

Conv 3x1, C filters ↓

H x W x C

The power of small filters

Still using 3 x 3 filters … can we break it up?

More nonlinearity, fewer params, less compute!

$$H \times W \times C \xrightarrow{\text{Conv 1x3, C filters}} H \times W \times C \xrightarrow{\text{Conv 3x1, C filters}} H \times W \times C$$

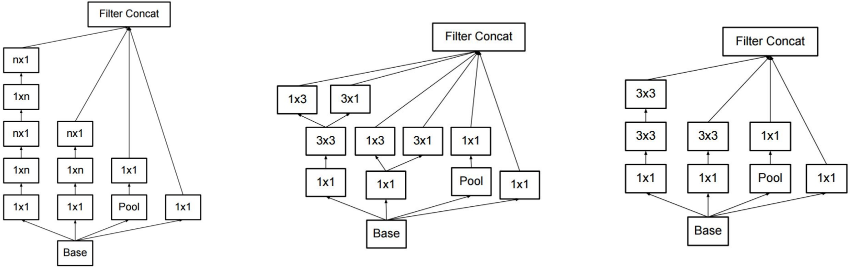
6 C² → **$6\,C^2$ parameters**

$$H \times W \times C \xrightarrow{\text{Conv 3x3, C filters}} H \times W \times C$$

$9\,C^2$ parameters

The power of small filters

Latest version of GoogLeNet incorporates all these ideas



Szegedy et al, "Rethinking the Inception Architecture for Computer Vision"

How to stack convolutions: Recap

- Replace large convolutions (5 x 5, 7 x 7) with stacks of 3 x 3 convolutions
- 1 x 1 “bottleneck” convolutions are very efficient
- Can factor N x N convolutions into 1 x N and N x 1
- All of the above give fewer parameters, less compute, more nonlinearity