CS 4644-DL / 7643-A: LECTURE 15
DANFEI XU

Instance Segmentation (Continued)

Network Visualization

- **Assignment 2**
  - 🚨 We are into the grace period!🚨
  - No exception other than for emergencies.

- **Project Proposal Feedback is Out**
  - Talk to the TA (over OHs) who graded your proposal for more detailed feedback.

- **Assignment 3 out soon**

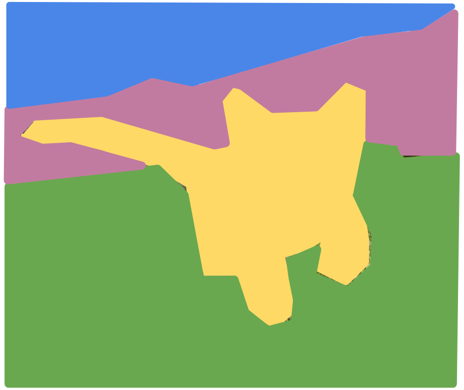
Computer Vision Tasks

Classification

CAT

No spatial extent

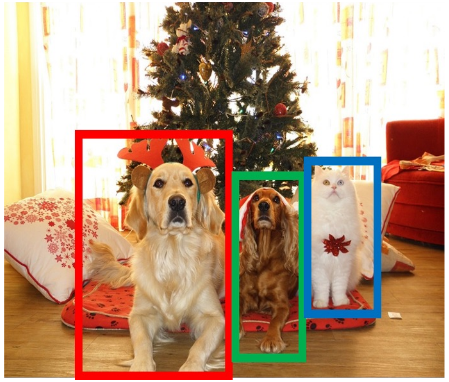
Semantic Segmentation



GRASS, CAT, TREE, SKY

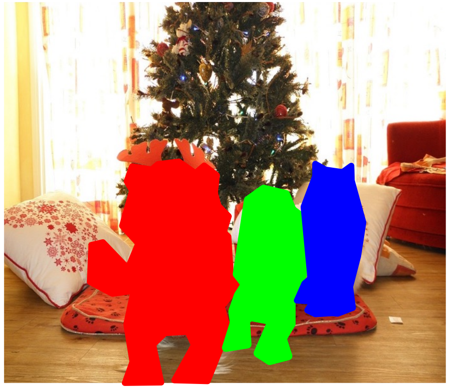
No objects, just pixels

Object Detection



DOG, DOG, CAT

Instance Segmentation

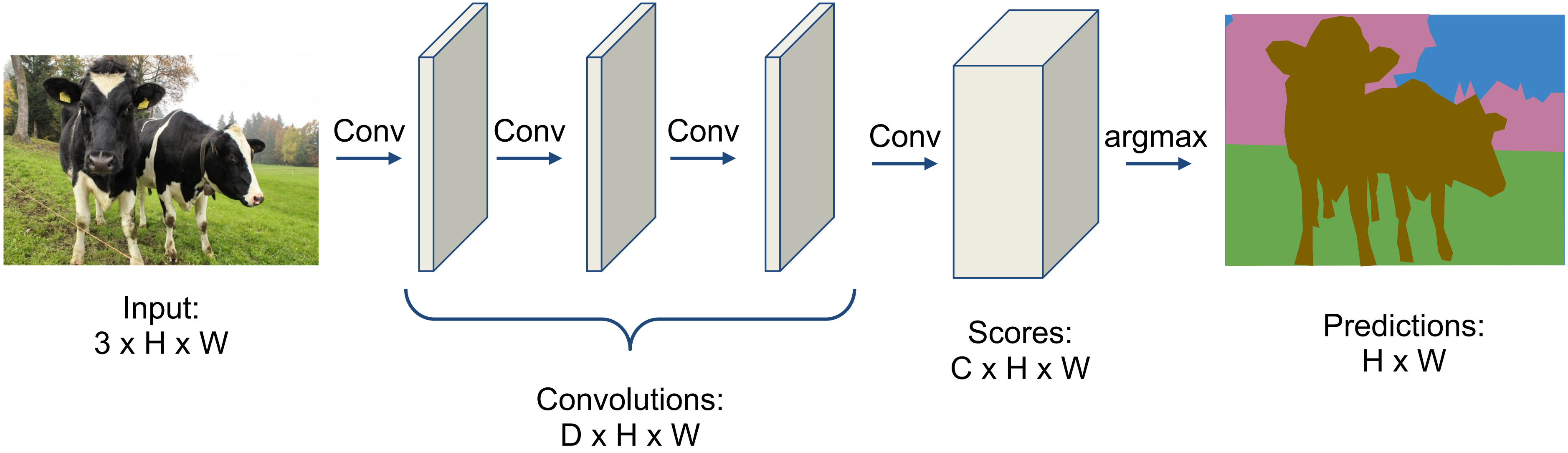


DOG, DOG, CAT

Multiple Object

Semantic Segmentation Idea: Fully Convolutional

Design a network with only convolutional layers without downsampling operators to make predictions for pixels all at once!



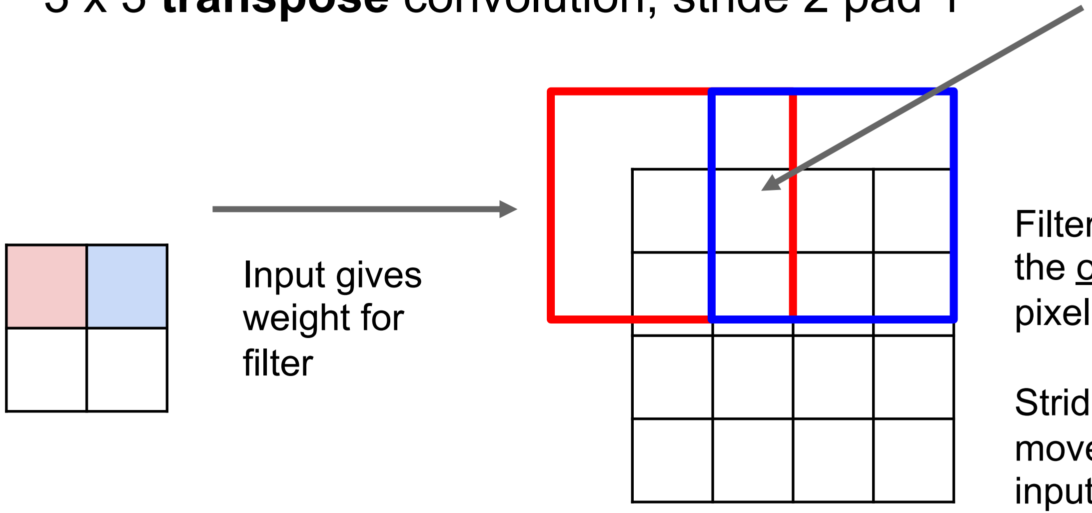
Loss: Pixel-wise cross entropy!

Learnable Upsampling: Transposed Convolution

Q: Why is it called transpose convolution?

3 x 3 **transpose** convolution, stride 2 pad 1

Sum where output overlaps



Input gives weight for filter

Filter moves 2 pixels in the <u>output</u> for every one pixel in the <u>input</u>

Stride gives ratio between movement in output and input

Input: 2 x 2

Output: 4 x 4

Semantic Segmentation Idea: Fully Convolutional

Downsampling: Pooling, strided convolution

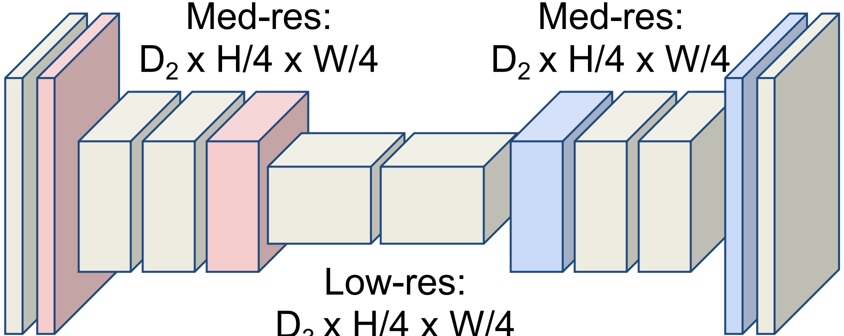
Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling: Unpooling or strided transpose convolution

Input:
3 x H x W

Med-res:
D_2 x H/4 x W/4

Med-res:
D_2 x H/4 x W/4



Low-res:
D_3 x H/4 x W/4

High-res:
D_1 x H/2 x W/2

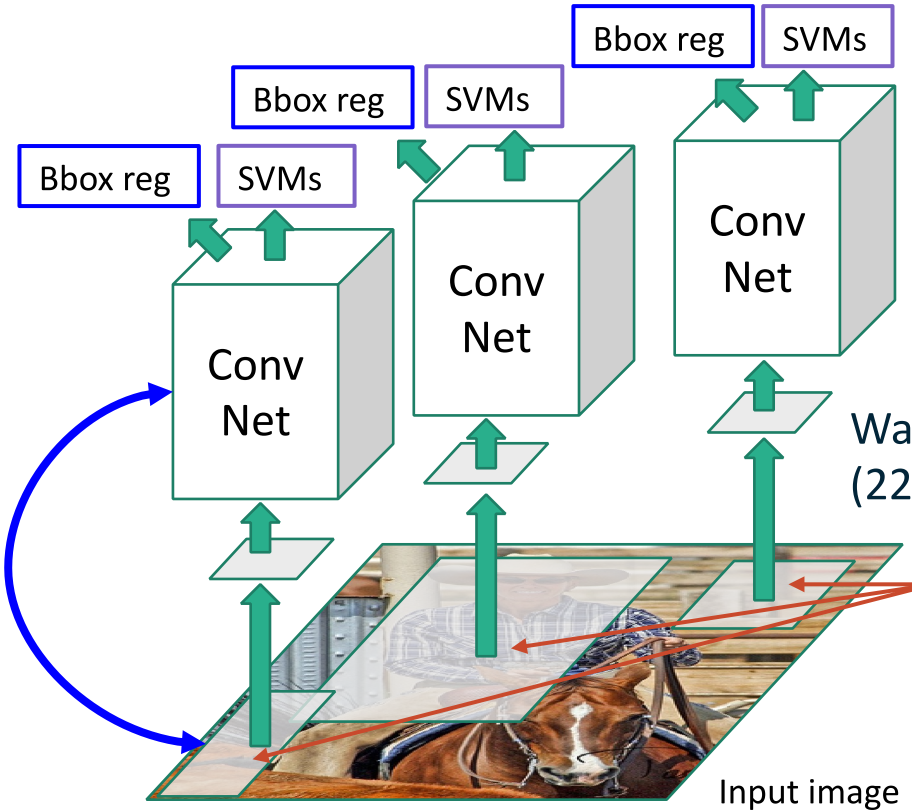
High-res:
D_1 x H/2 x W/2

Predictions:
H x W

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

"Slow" R-CNN

Predict "corrections" to the RoI: 4 numbers: (dx, dy, dw, dh)



Classify regions with SVMs

Forward each region through ConvNet

Warped image regions (224x224 pixels)

Regions of Interest (RoI) from a proposal method (~2k)

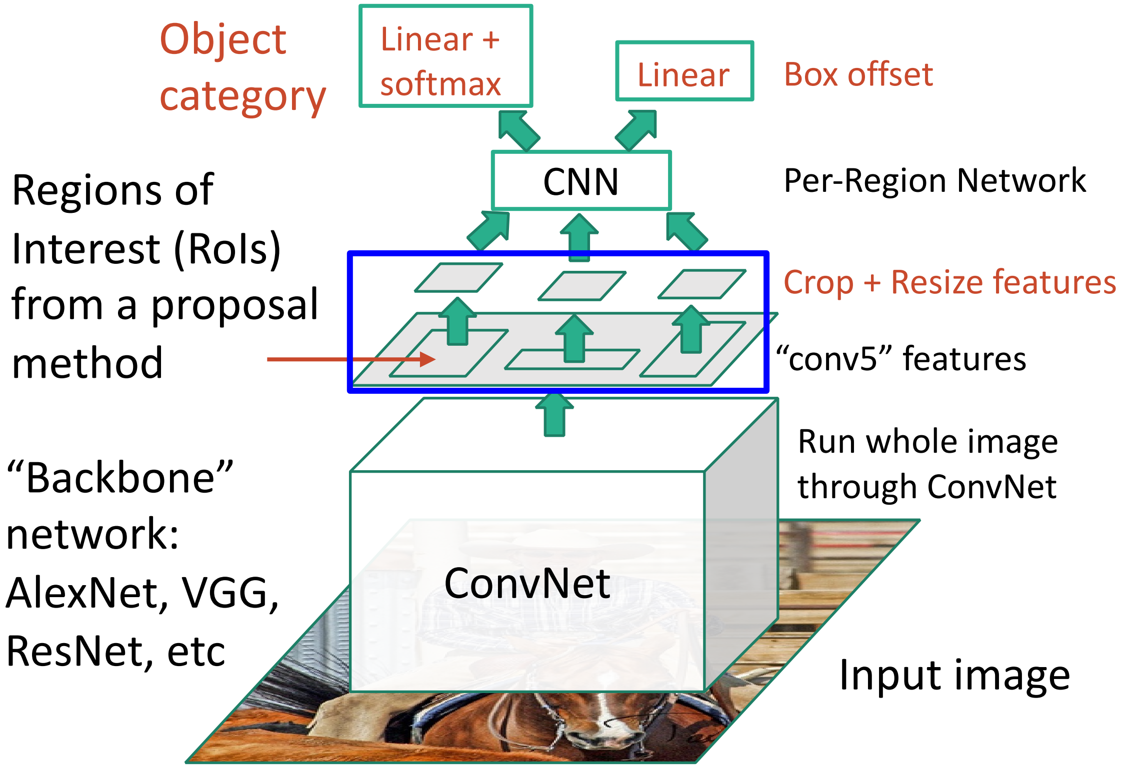
Input image

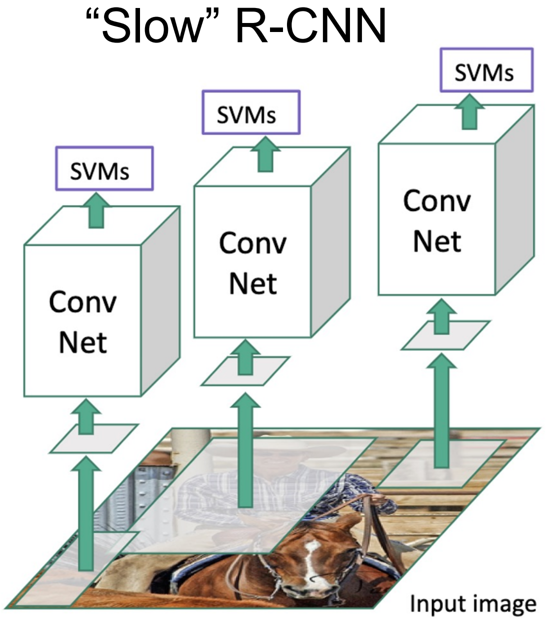
Problem: Very slow! Need to do ~2k independent forward passes for each image!

Idea: Pass the image through convnet before cropping! Crop the conv feature instead!

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

Fast R-CNN





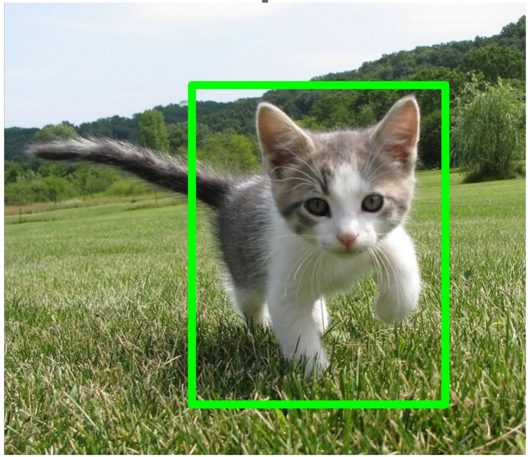
Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Cropping Features: RoI Pool

"Snap" to grid cells

Divide into 2x2 grid of (roughly) equal subregions

Project proposal onto features



Input Image
(e.g. 3 x 640 x 480)

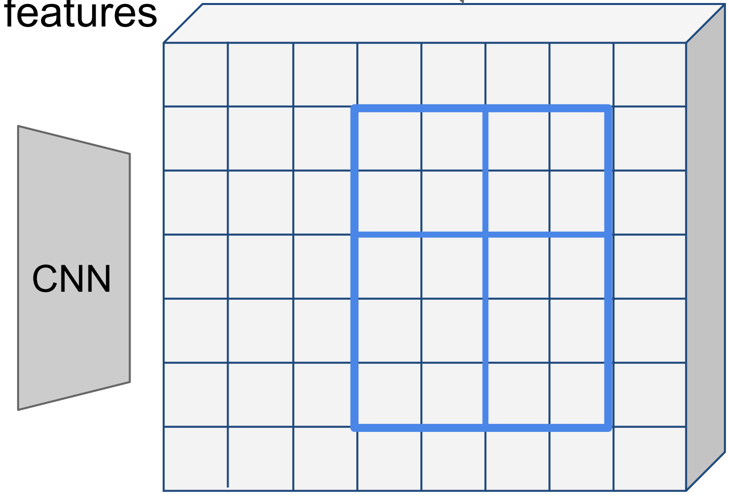
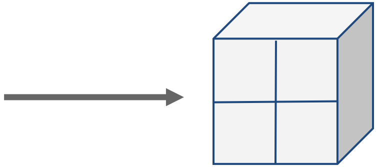


Image features: C x H x W
(e.g. 512 x 20 x 15)

Max-pool within each subregion



Region features
(here 512 x 2 x 2;
In practice e.g 512 x 7 x 7)

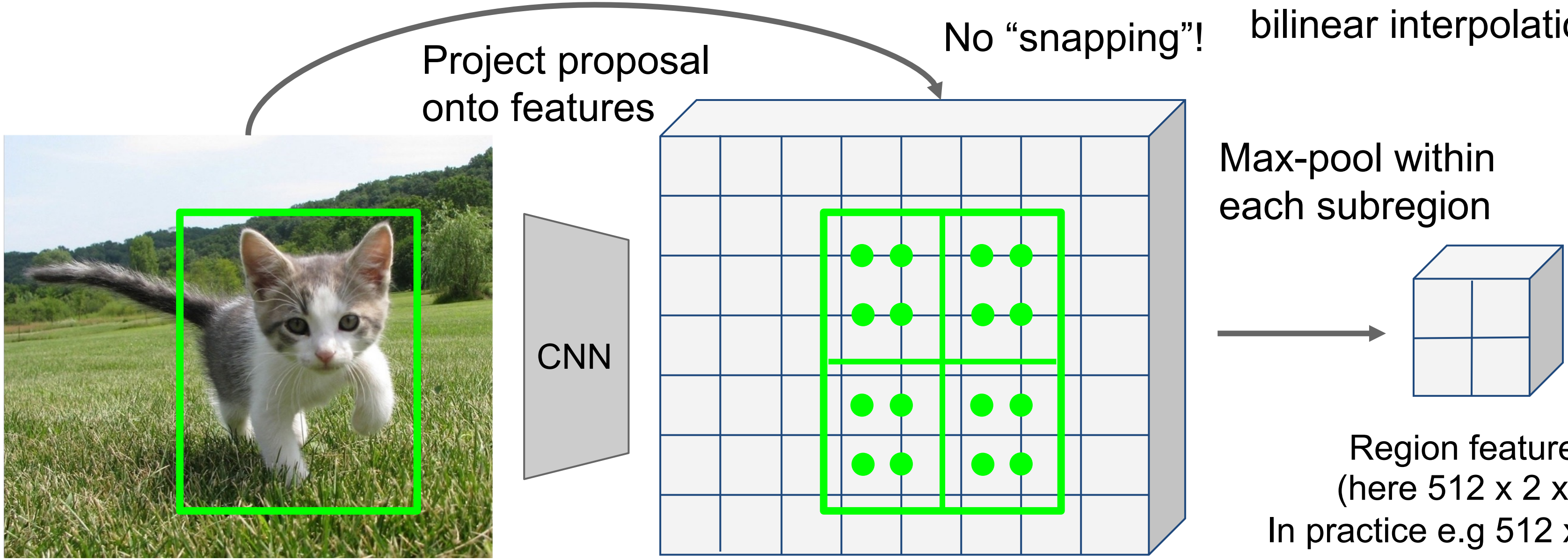
Region features always the same size even if input regions have different sizes!

Problem: Region features slightly misaligned

Girshick, "Fast R-CNN", ICCV 2015.

Cropping Features: RoI Align

Sample at regular points
in each subregion using
bilinear interpolation



Input Image
(e.g. 3 x 640 x 480)

Image features: C x H x W
(e.g. 512 x 20 x 15)

Region features
(here 512 x 2 x 2;
In practice e.g 512 x 7 x 7)

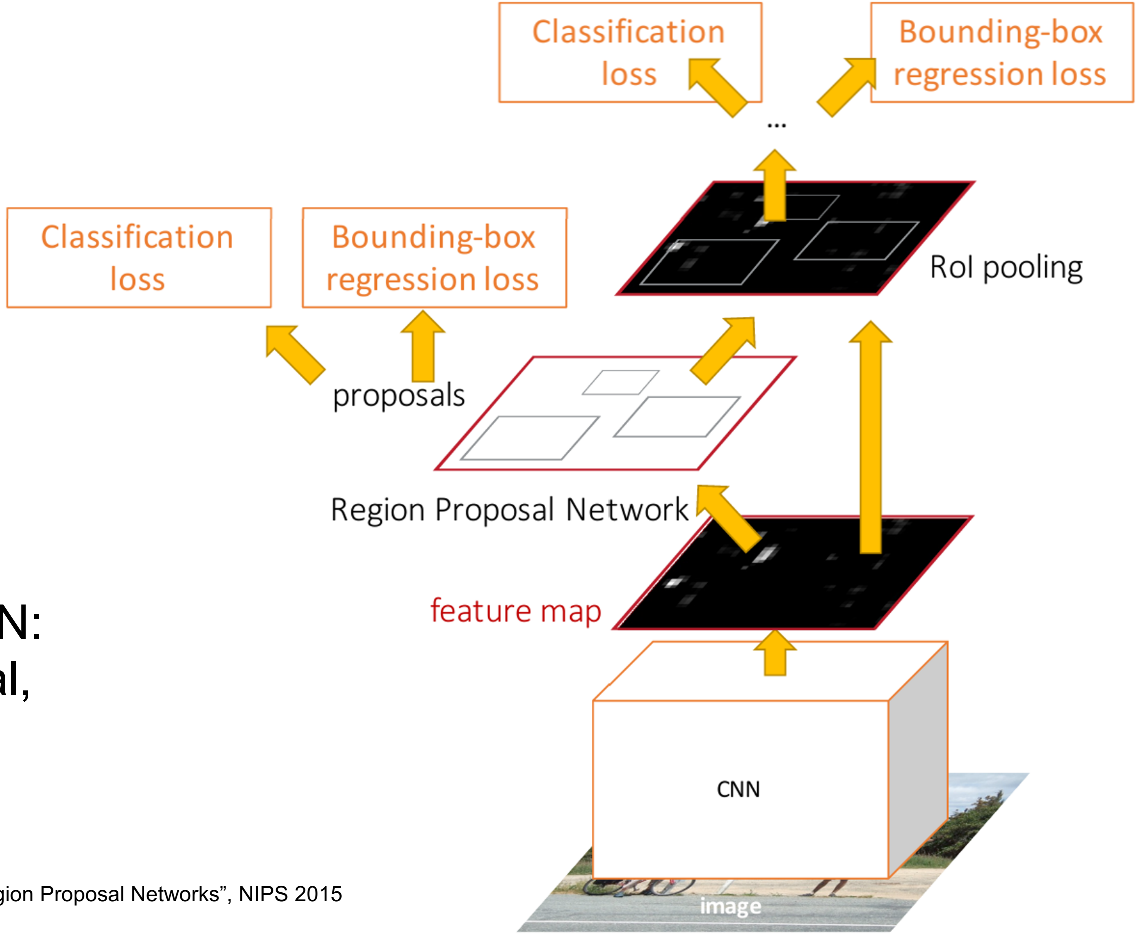
He et al, "Mask R-CNN", ICCV 2017

Fast**<u>er</u>** R-CNN:

Make CNN do proposals!

Insert **Region Proposal Network (RPN)** to predict proposals from features

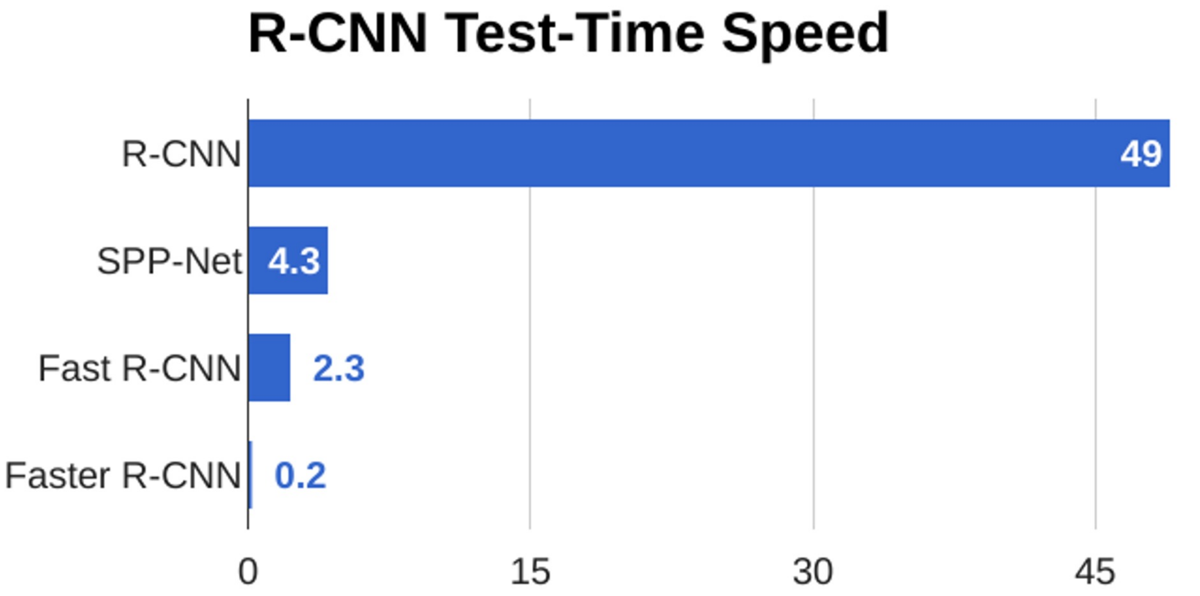
Otherwise same as Fast R-CNN: Crop features for each proposal, classify each one



Ren et al, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, NIPS 2015

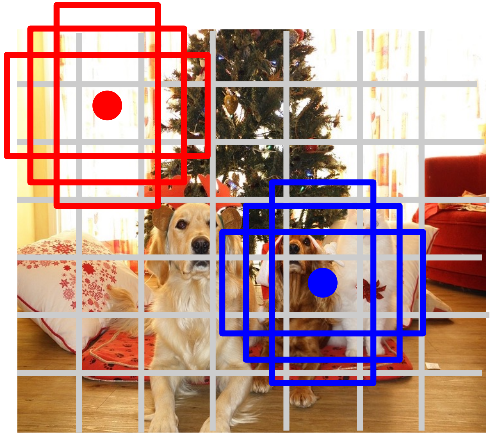
Faster R-CNN:

Make CNN do proposals!



Single-Stage Object Detectors: YOLO / SSD / RetinaNet

Input image
3 x H x W



Divide image into grid
7 x 7

Image a set of **base boxes** centered at each grid cell
Here B = 3

Within each grid cell:
- Regress from each of the B base boxes to a final box with 5 numbers: (dx, dy, dh, dw, confidence)
- Predict scores for each of C classes (including background as a class)
- Looks a lot like RPN, but category-specific!

Output:
7 x 7 x (5 * B + C)

Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016
Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

Instance Segmentation

Classification

CAT

No spatial extent

Semantic Segmentation

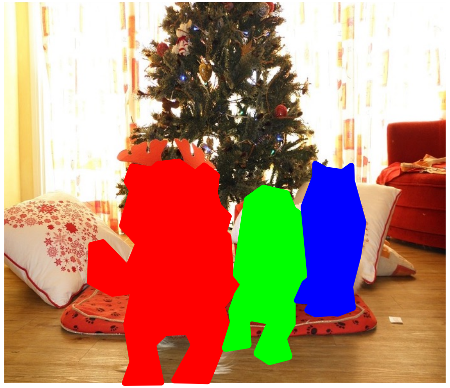
GRASS, CAT, TREE, SKY

No objects, just pixels

Object Detection

DOG, DOG, CAT

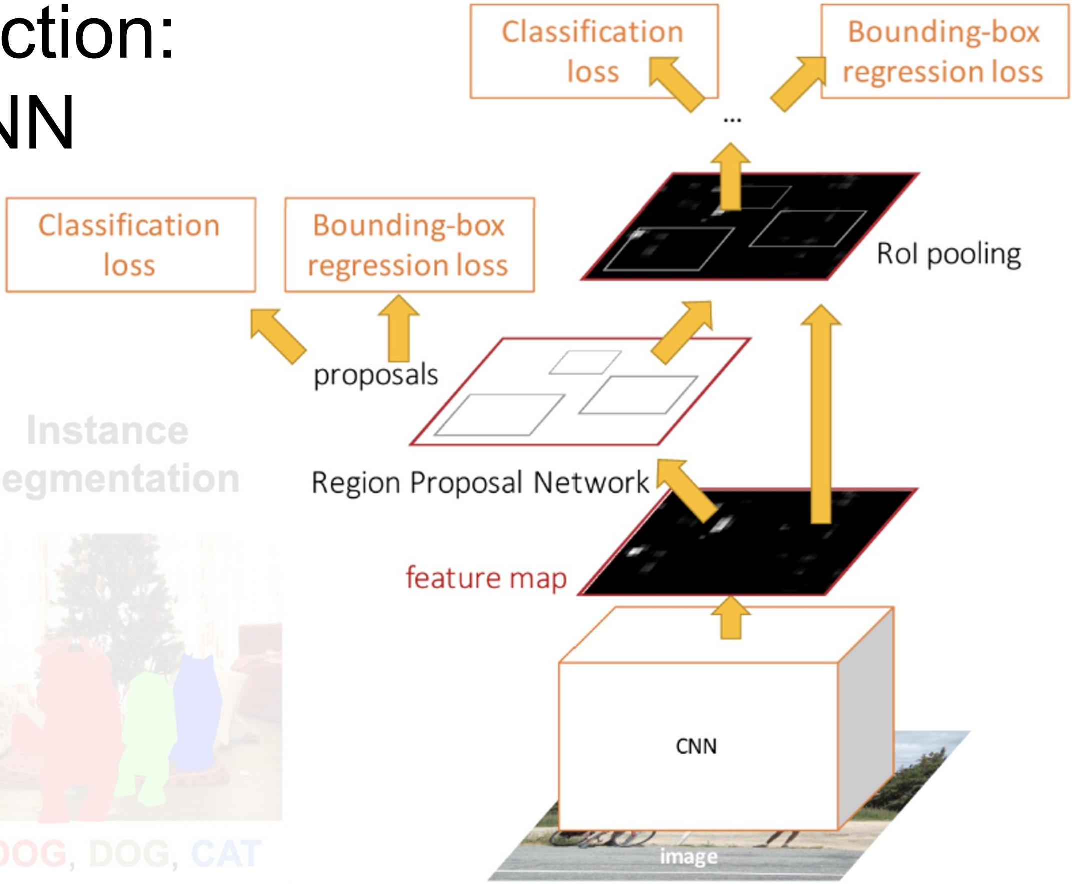
Instance Segmentation



DOG, DOG, CAT

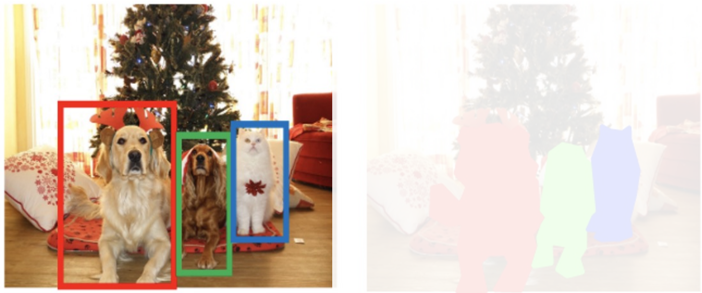
Multiple Object

Object Detection: Faster R-CNN



Object Detection

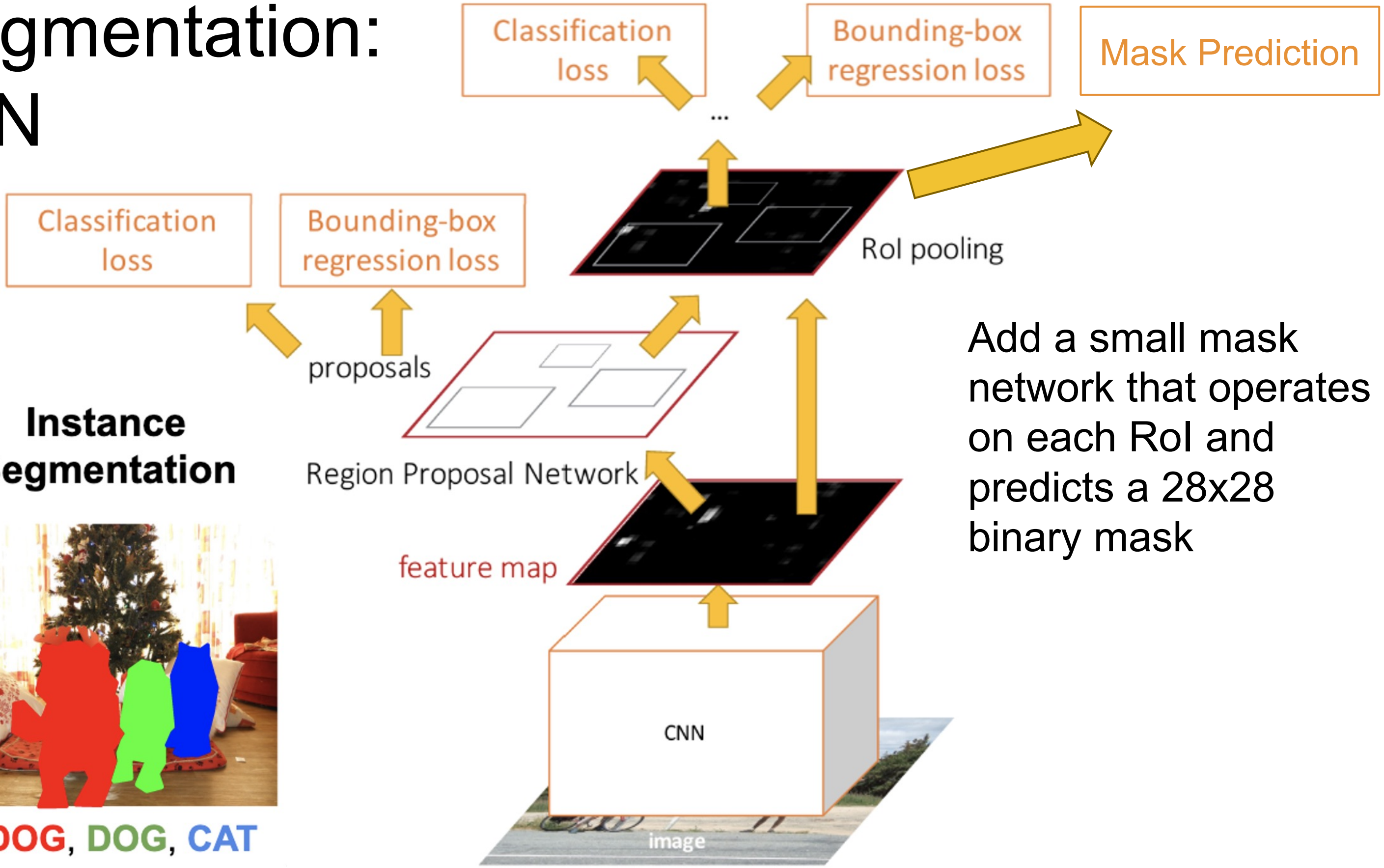
Instance Segmentation



DOG, DOG, CAT

DOG, DOG, CAT

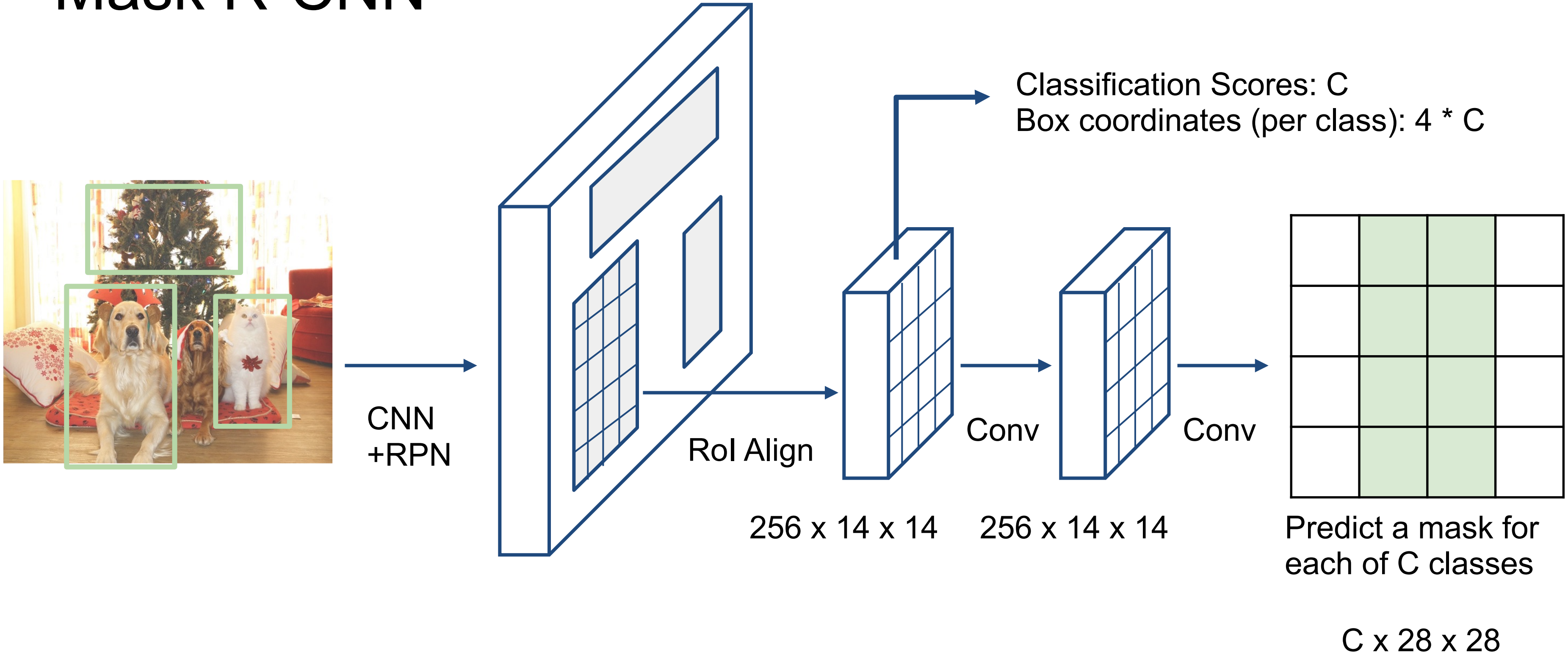
Instance Segmentation: Mask R-CNN



Add a small mask network that operates on each RoI and predicts a 28x28 binary mask

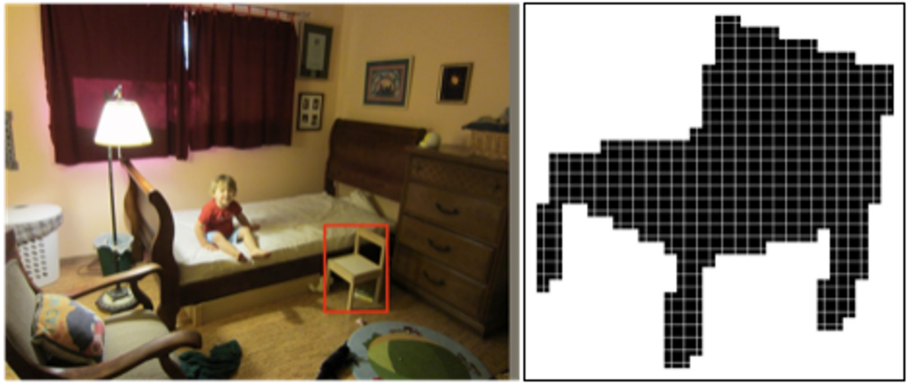
He et al, “Mask R-CNN”, ICCV 2017

Mask R-CNN

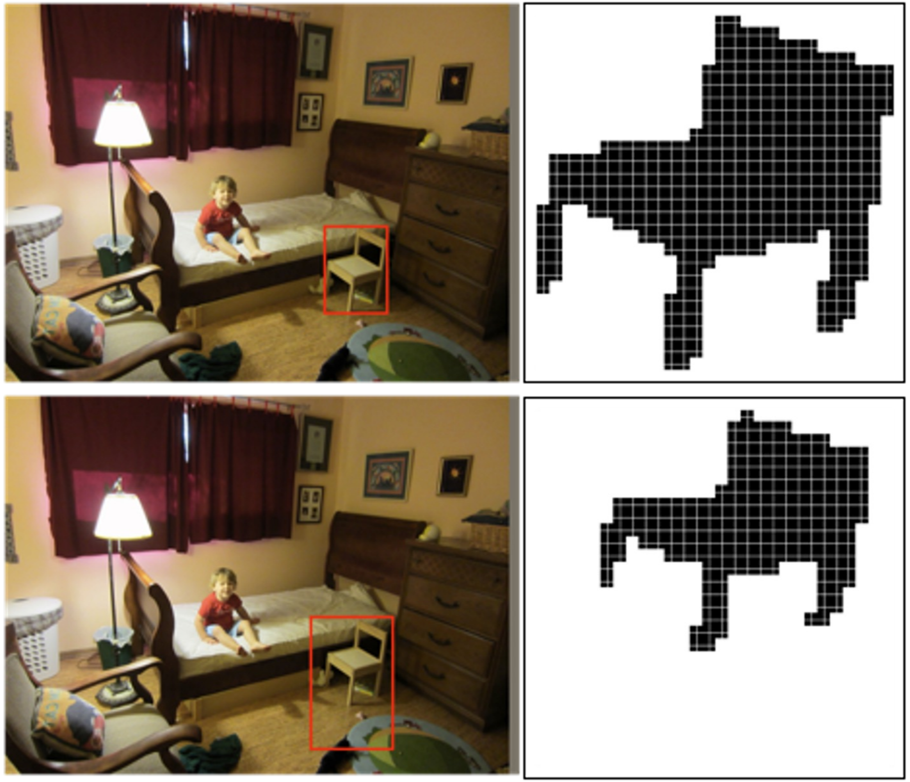


He et al, "Mask R-CNN", arXiv 2017

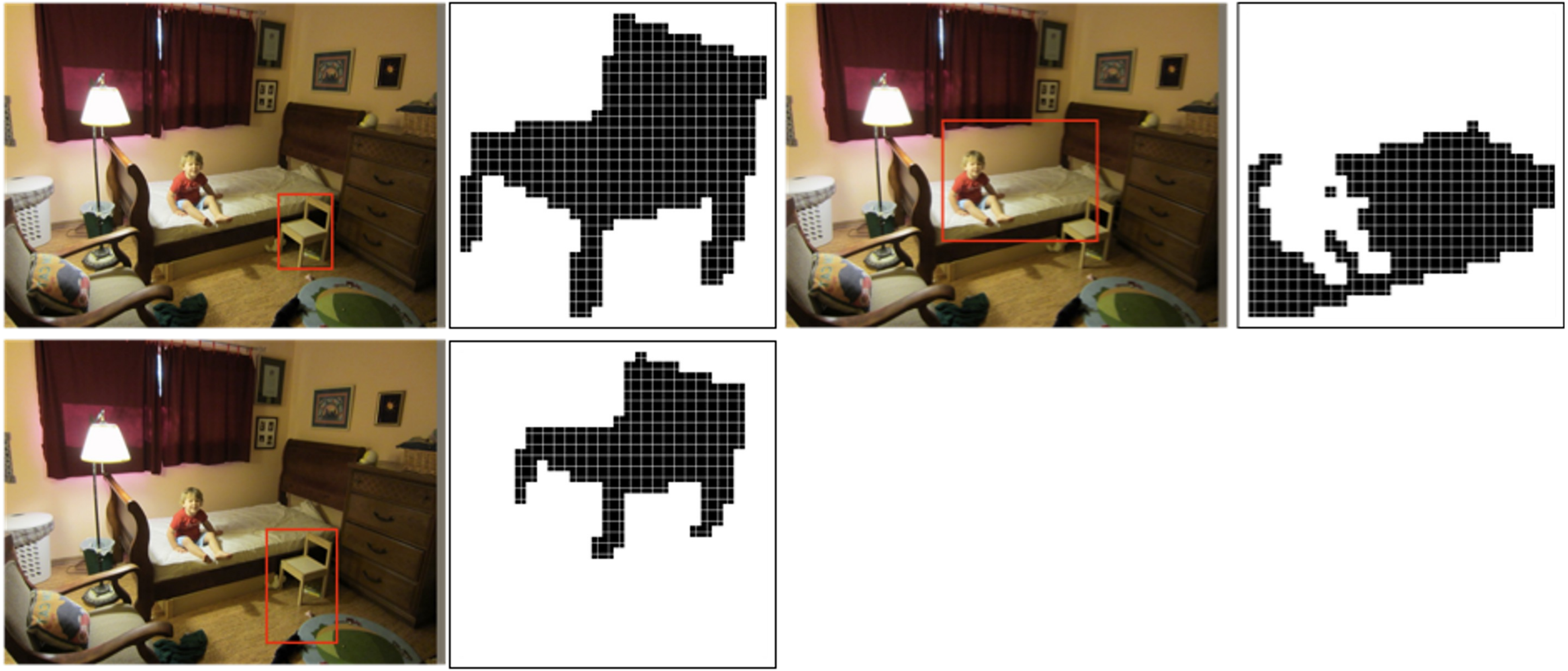
Mask R-CNN: Example Mask Training Targets



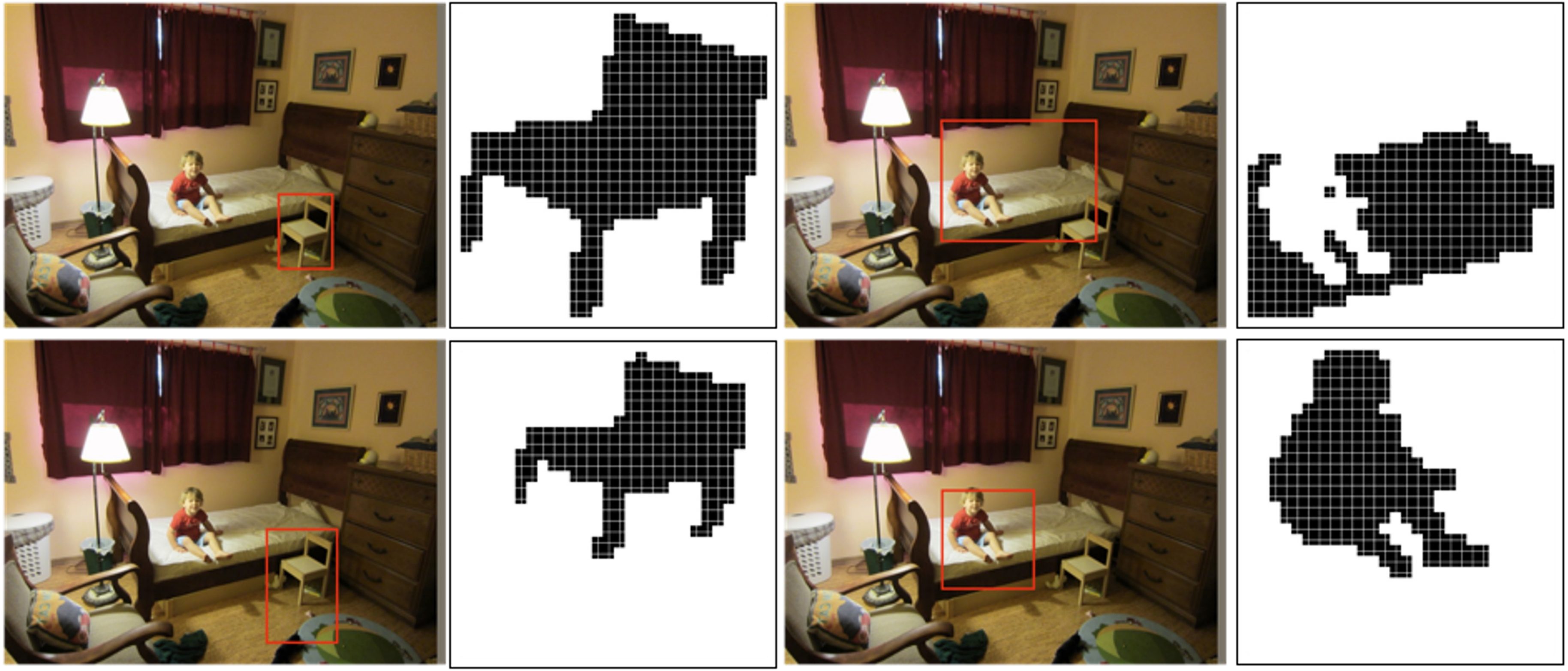
Mask R-CNN: Example Mask Training Targets



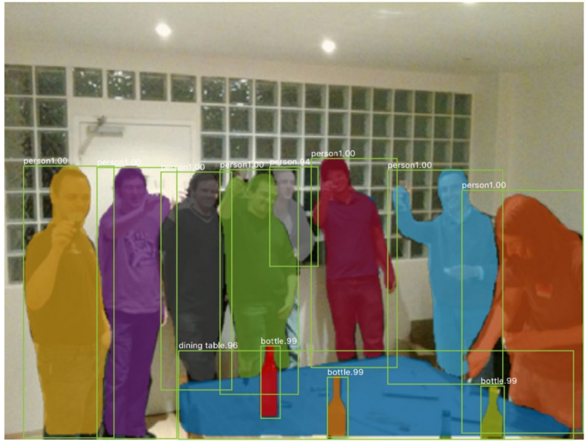
Mask R-CNN: Example Mask Training Targets



Mask R-CNN: Example Mask Training Targets



Mask R-CNN: Very Good Results!



He et al, “Mask R-CNN”, ICCV 2017

Mask R-CNN
Also does pose

He et al, “Mask R-CNN”, ICCV 2017

Open Source Frameworks

Lots of good implementations on GitHub!

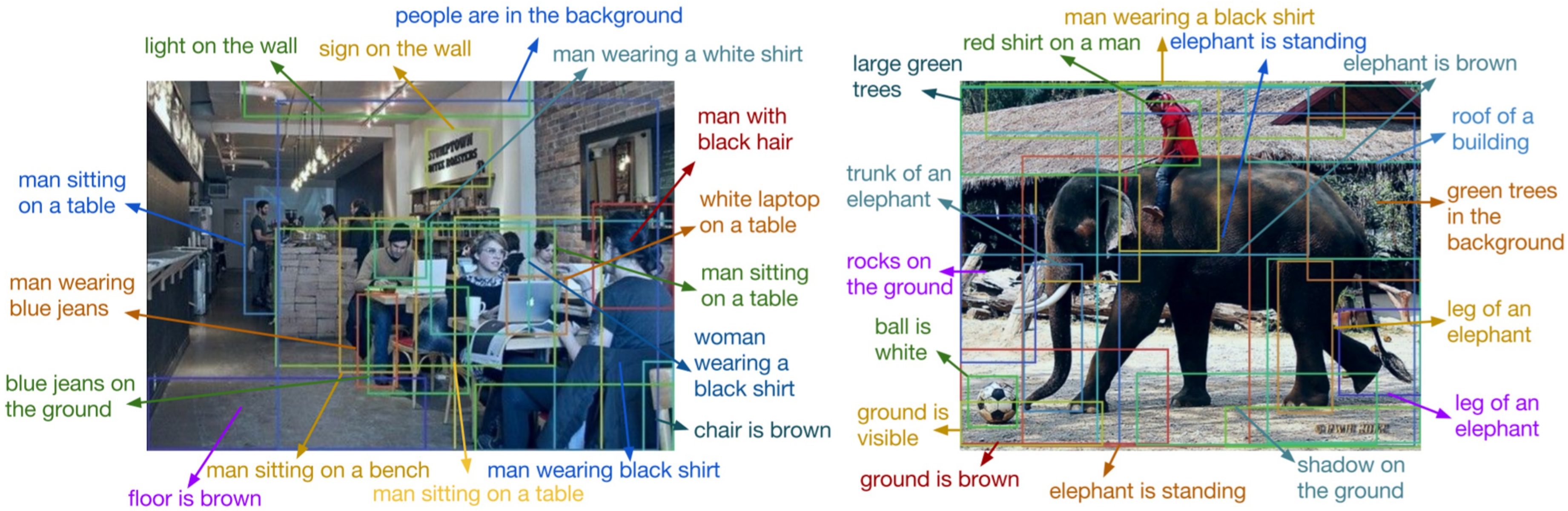
TensorFlow Detection API:
https://github.com/tensorflow/models/tree/master/research/object_detection
Faster RCNN, SSD, RFCN, Mask R-CNN, ...

Detectron2 (PyTorch)
https://github.com/facebookresearch/detectron2
Mask R-CNN, RetinaNet, Faster R-CNN, RPN, Fast R-CNN, R-FCN, ...

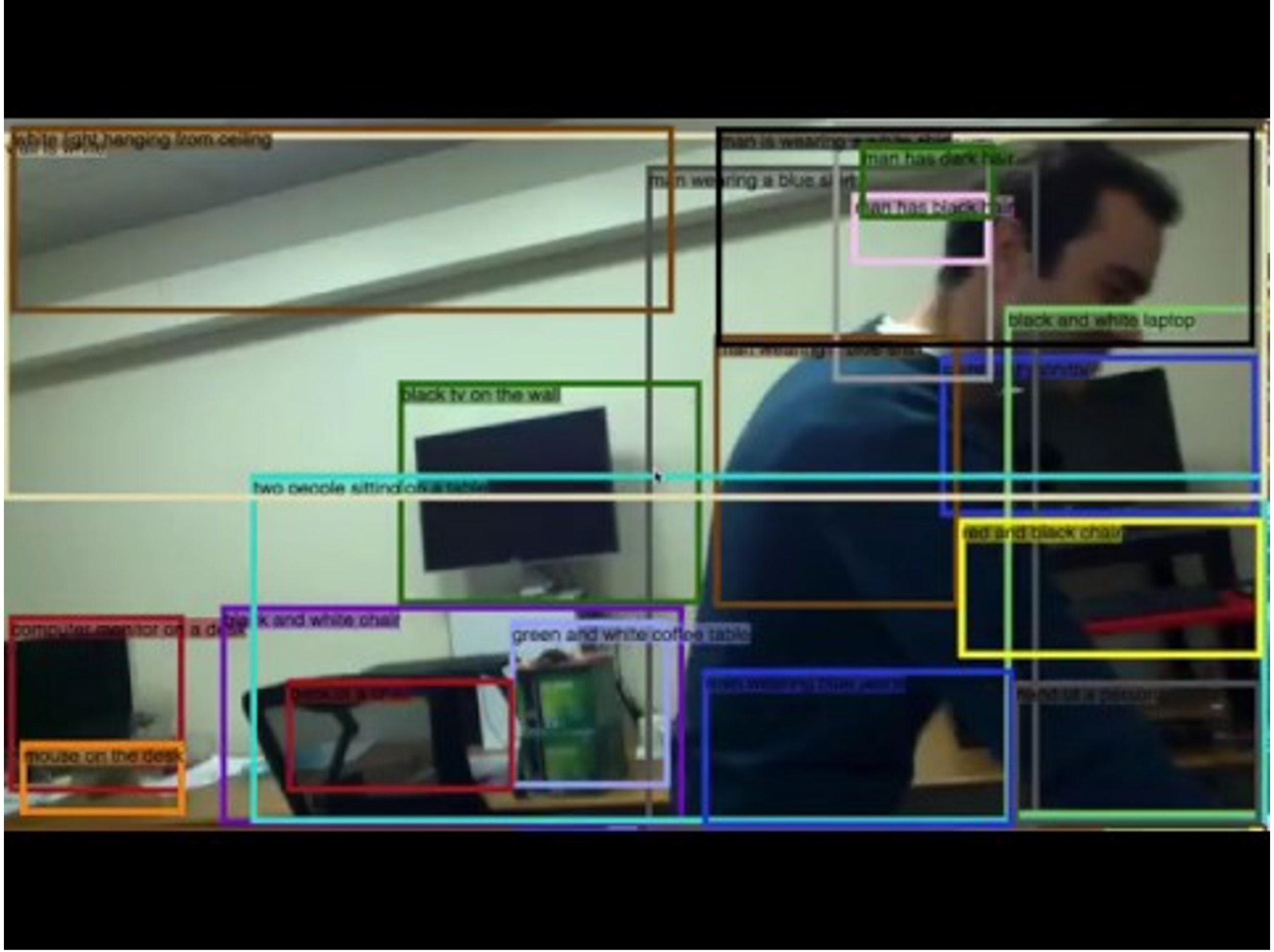
Finetune on your own dataset with pre-trained models

Beyond 2D Object Detection...

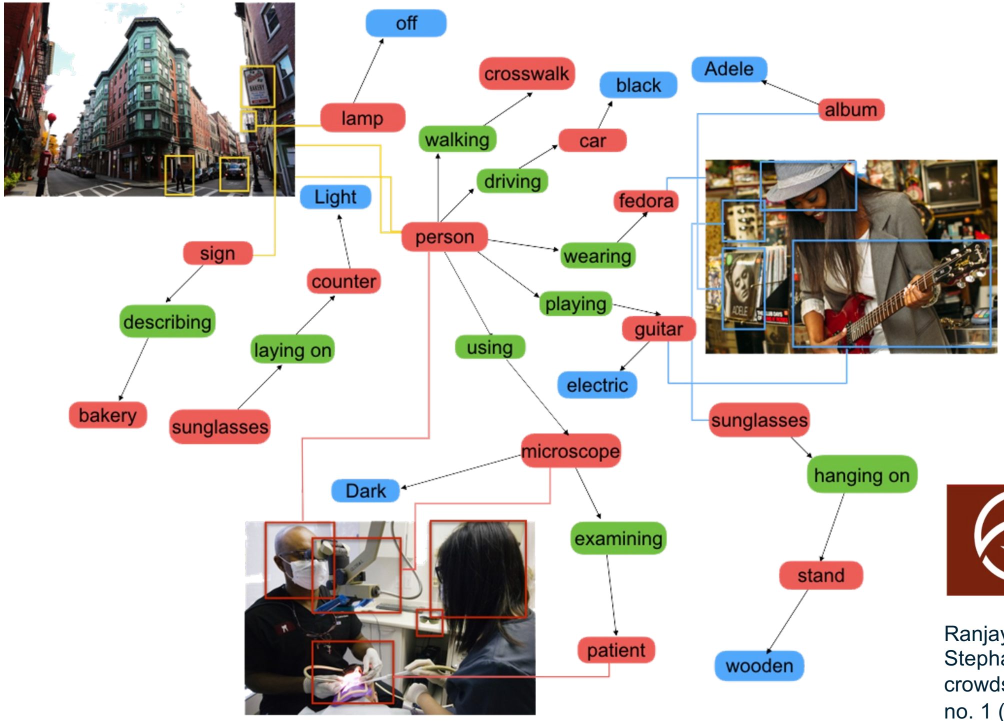
Object Detection + Captioning = Dense Captioning



Johnson, Karpathy, and Fei-Fei, “DenseCap: Fully Convolutional Localization Networks for Dense Captioning”, CVPR 2016


white light hanging from ceiling
man has dark hair
black and white laptop
black tv on the wall
red and black chair
green and white coffee table
mouse on the desk

Objects + Relationships = Scene Graphs

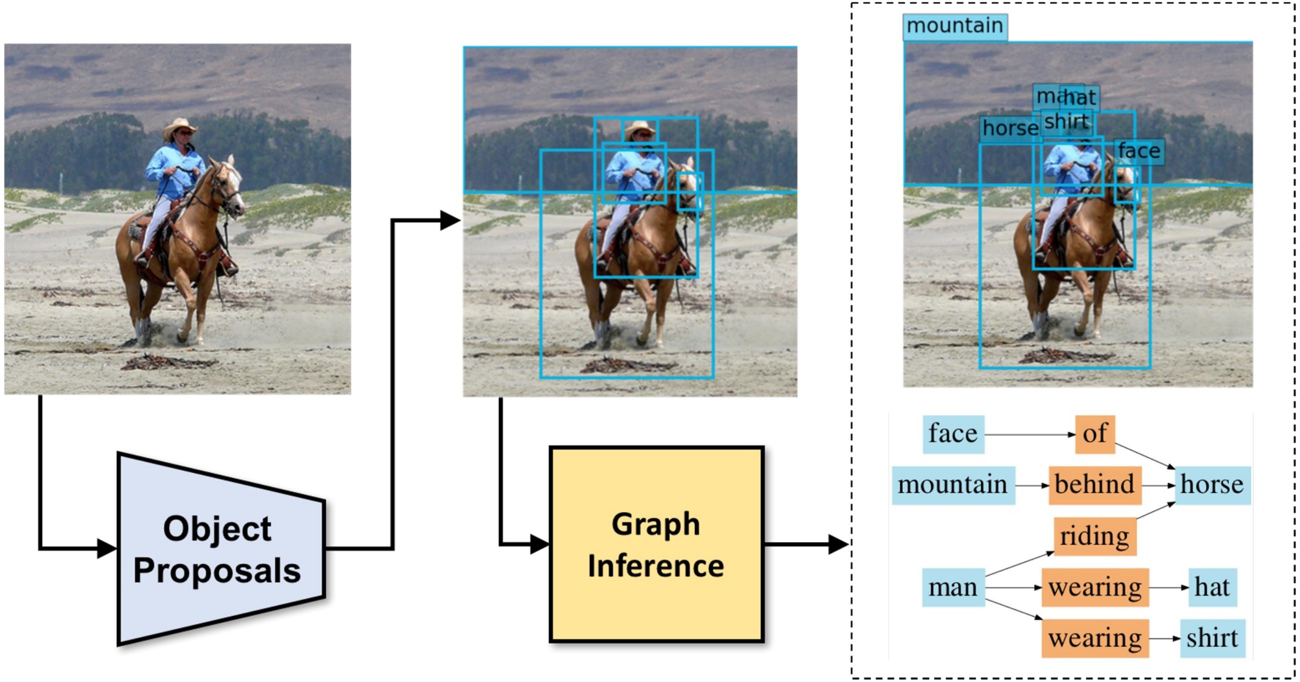


108,077 Images
5.4 Million Region Descriptions
1.7 Million Visual Question Answers
3.8 Million Object Instances
2.8 Million Attributes
2.3 Million Relationships
Everything Mapped to Wordnet Synsets

VISUALGENOME

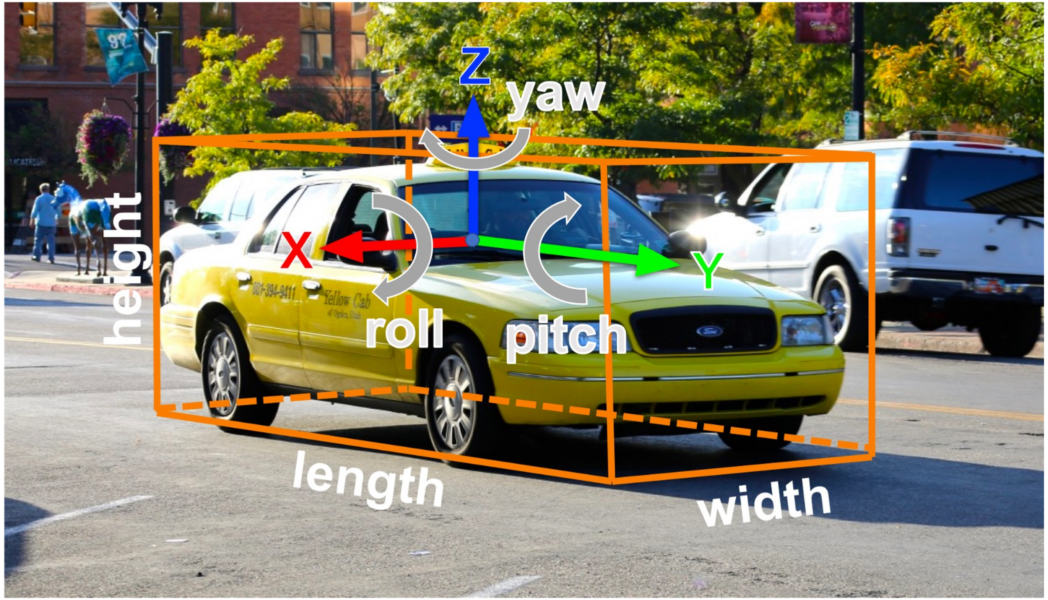
Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen et al. "Visual genome: Connecting language and vision using crowdsourced dense image annotations." International Journal of Computer Vision 123, no. 1 (2017): 32-73.

Scene Graph Prediction



Xu, Zhu, Choy, and Fei-Fei, “Scene Graph Generation by Iterative Message Passing”, CVPR 2017

3D Object Detection



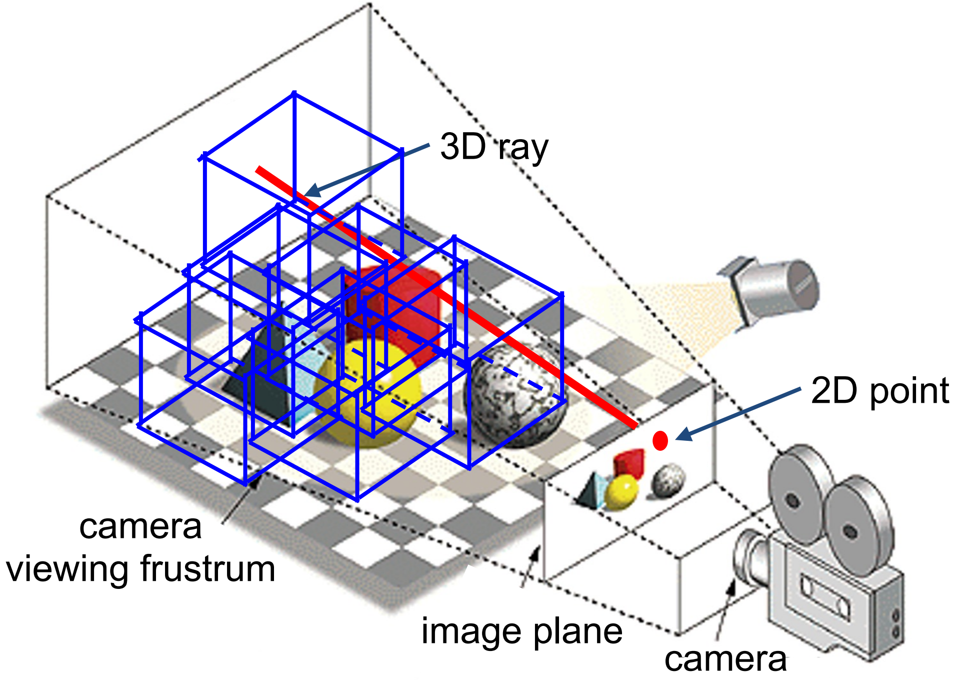
2D Object Detection:
2D bounding box
(x, y, w, h)

3D Object Detection:
3D oriented bounding box
(x, y, z, w, h, l, r, p, y)

Simplified bbox: no roll & pitch

Much harder problem than 2D object detection!

3D Object Detection: Simple Camera Model



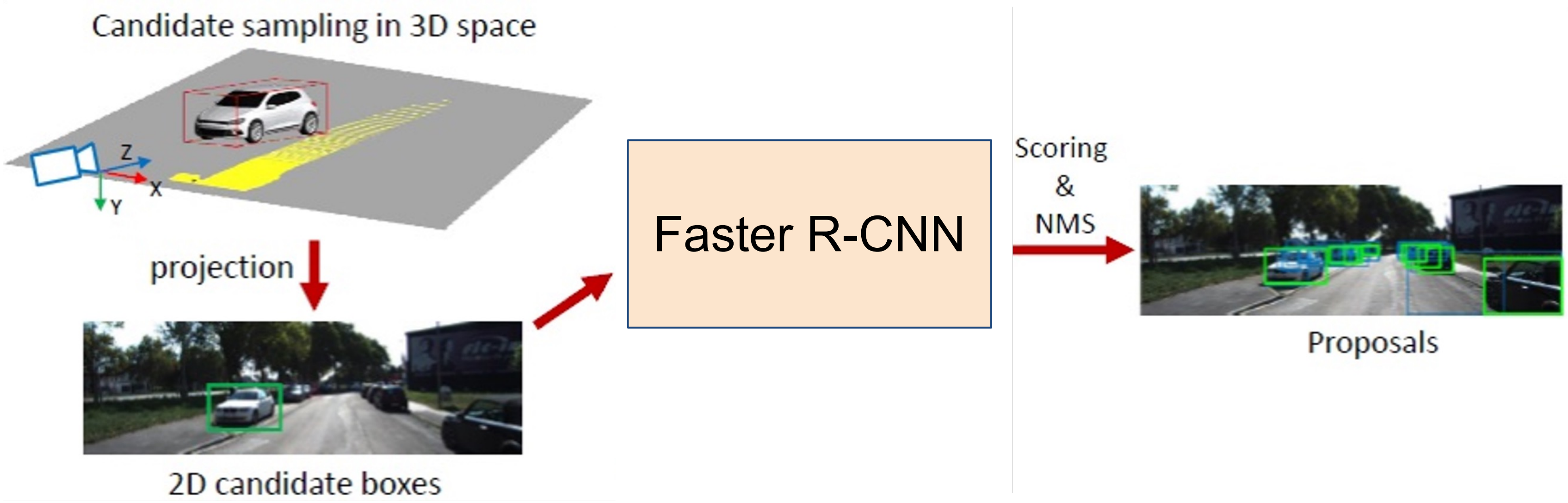
A point on the image plane corresponds to a **ray** in the 3D space

A 2D bounding box on an image is a **frustrum** in the 3D space

Localize an object in 3D:
The object can be anywhere in the **camera viewing frustrum**!

Image source: https://www.pcmag.com/encyclopedia_images/_FRUSTUM.GIF

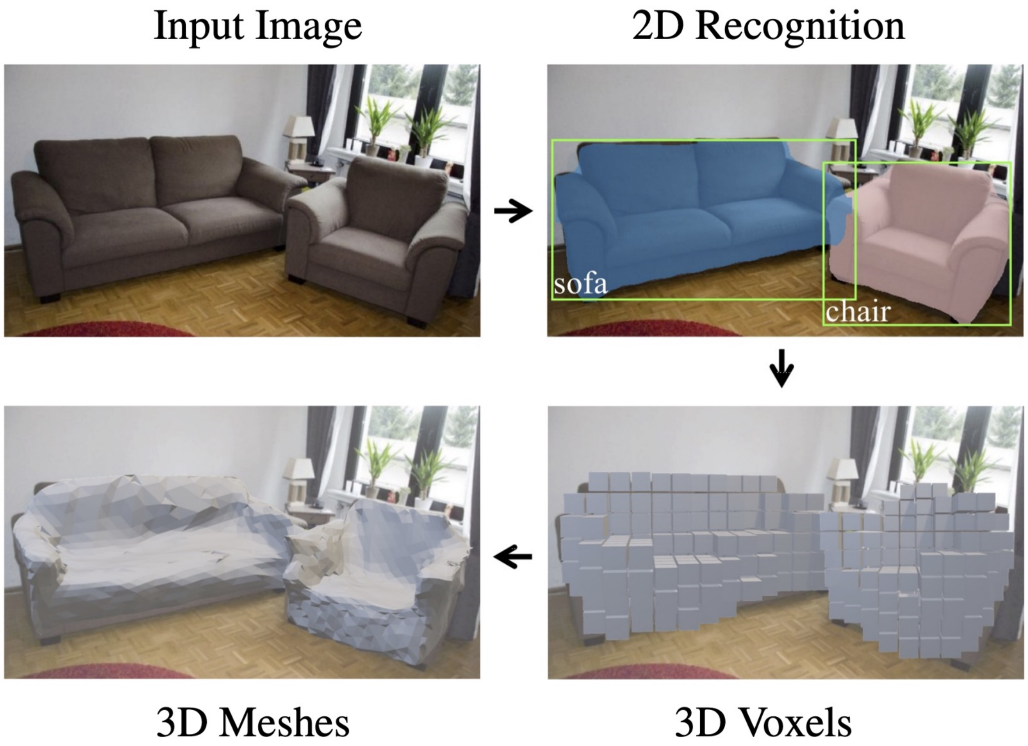
3D Object Detection: Monocular Camera



- Same idea as Faster RCNN, but proposals are in 3D
- 3D bounding box proposal, regress 3D box parameters + class score

Chen, Xiaozhi, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. "Monocular 3d object detection for autonomous driving." CVPR 2016.

3D Shape Prediction: Mesh R-CNN



Gkioxari et al., Mesh RCNN, ICCV 2019

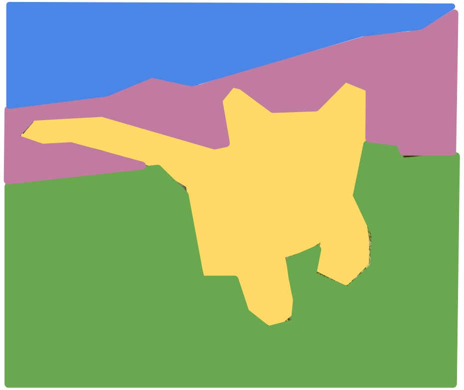
Recap: Lots of computer vision tasks!

Classification

CAT

No spatial extent

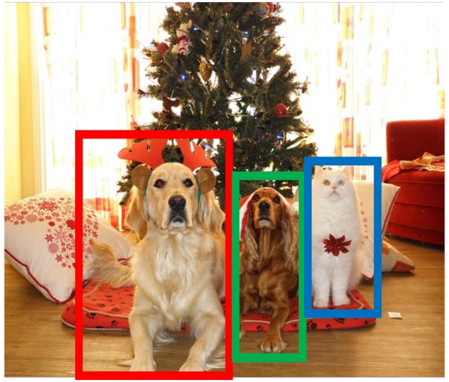
Semantic Segmentation



GRASS, CAT, TREE, SKY

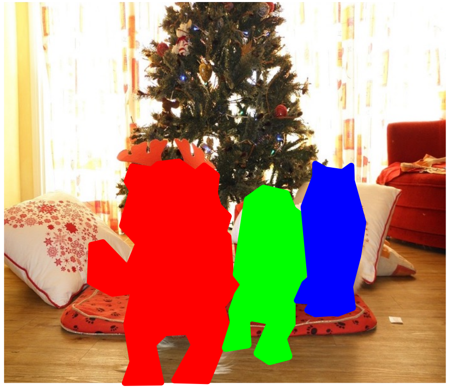
No objects, just pixels

Object Detection



DOG, DOG, CAT

Instance Segmentation

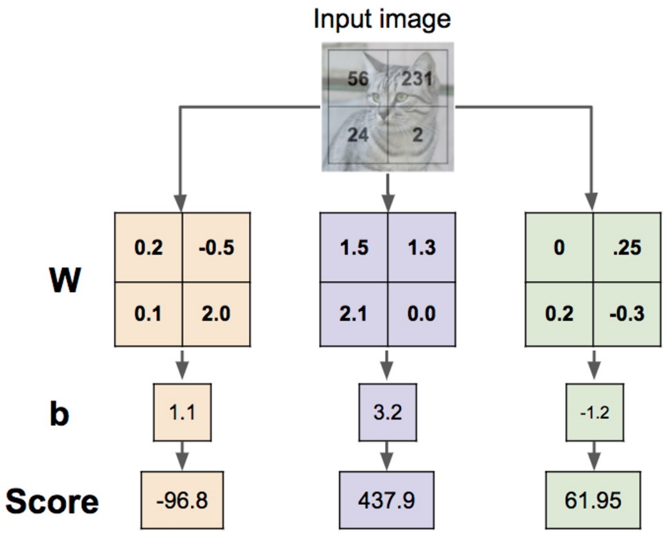


DOG, DOG, CAT

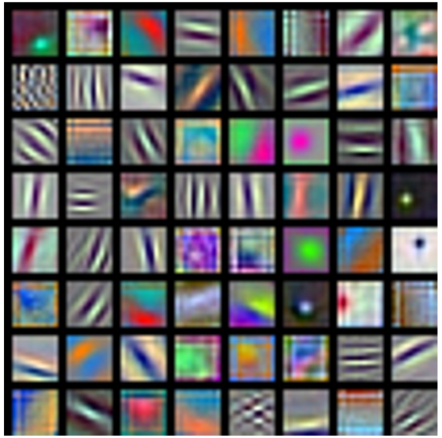
Multiple Object

Visualizing Neural Networks

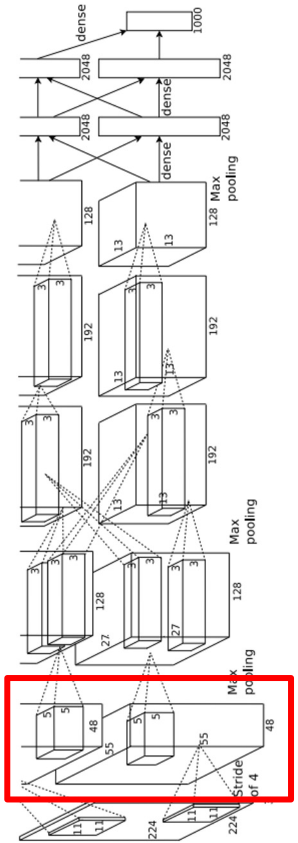
Interpreting a Linear Classifier: Visual Viewpoint



First Layer: Visualize Filters

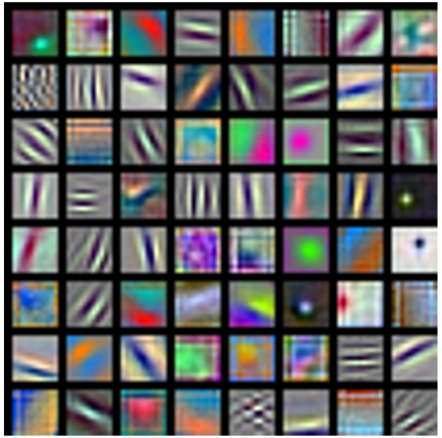


AlexNet:
64 x 3 x 11 x 11

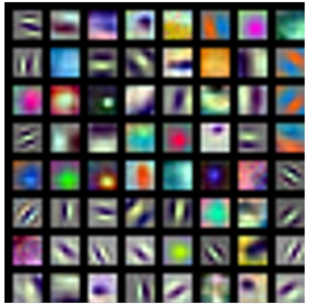


Krizhevsky, “One weird trick for parallelizing convolutional neural networks”, arXiv 2014
He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016
Huang et al, “Densely Connected Convolutional Networks”, CVPR 2017

First Layer: Visualize Filters

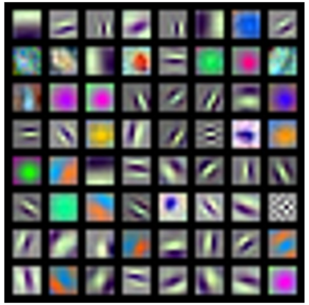


AlexNet:
64 x 3 x 11 x 11

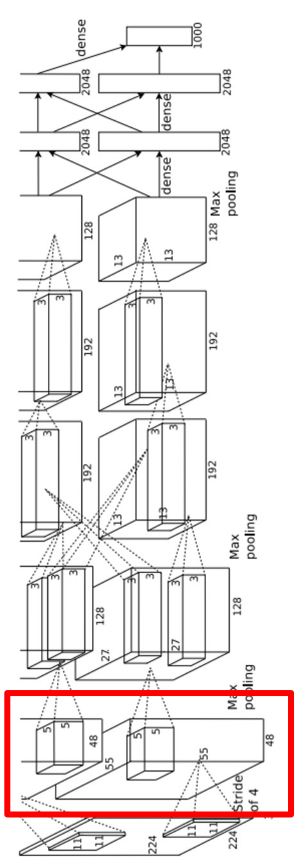


ResNet-18:
64 x 3 x 7 x 7

ResNet-101:
64 x 3 x 7 x 7



DenseNet-121:
64 x 3 x 7 x 7



Krizhevsky, “One weird trick for parallelizing convolutional neural networks”, arXiv 2014
He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016
Huang et al, “Densely Connected Convolutional Networks”, CVPR 2017

Visualize the filters/kernels (raw weights)

We can visualize filters at higher layers, but not that interesting

(these are taken from ConvNetJS CIFAR-10 demo)

layer 1 weights

16 x 3 x 7 x 7

layer 2 weights

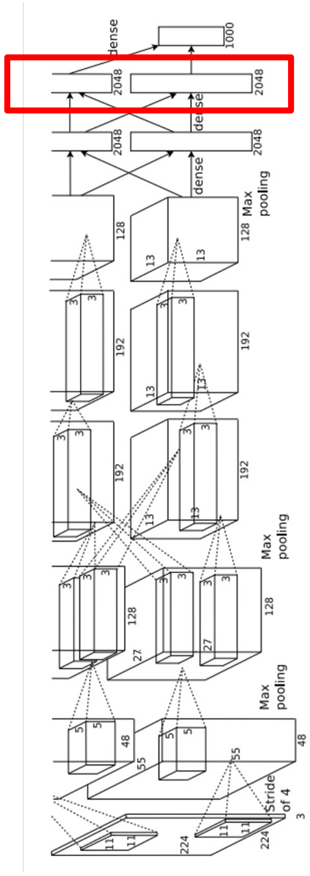
20 x 16 x 7 x 7

layer 3 weights

20 x 20 x 7 x 7

Last Layer

FC7 layer



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

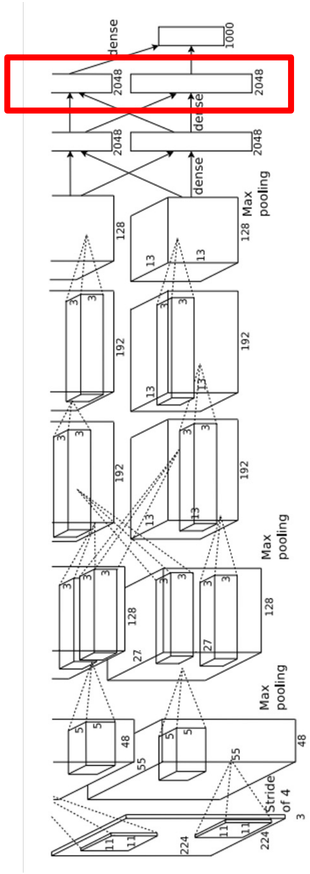
Run the network on many images, collect the
feature vectors

Last Layer: Nearest Neighbors

4096-dim vector

Test image L2 Nearest neighbors in feature space

Recall: Nearest neighbors in pixel space



Krizhevsky et al, “ImageNet Classification with Deep Convolutional Neural Networks”, NIPS 2012.
Figures reproduced with permission.

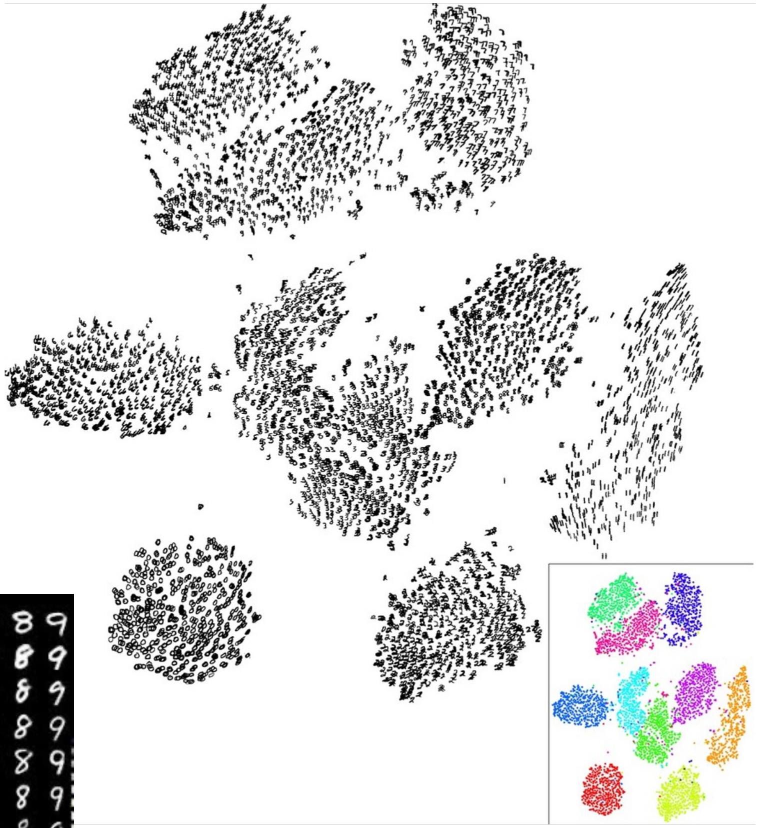
Last Layer: Dimensionality Reduction

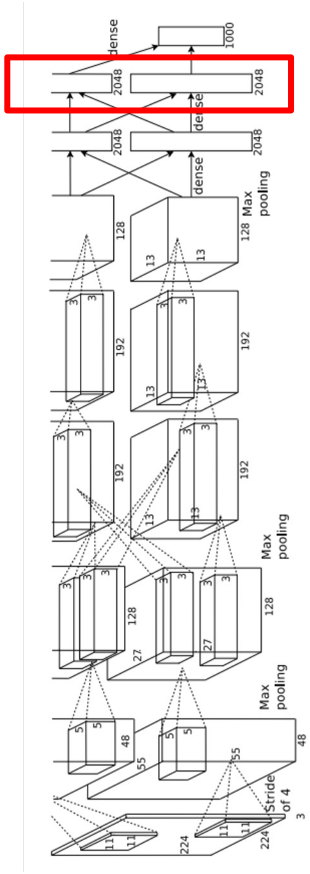
Visualize the “space” of FC7 feature vectors by reducing dimensionality of vectors from 4096 to 2 dimensions

Simple algorithm: Principal Component Analysis (PCA)

More complex: **t-SNE**

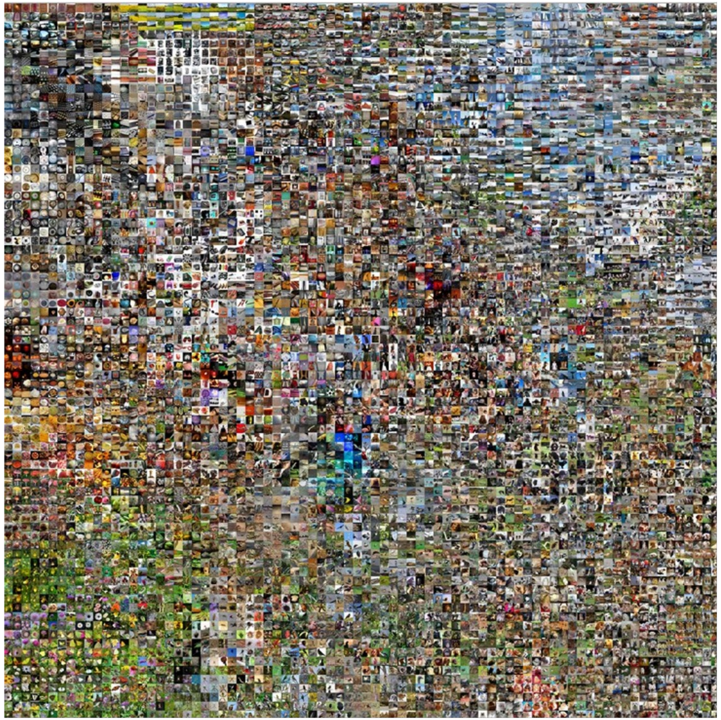
Visualize MNIST:

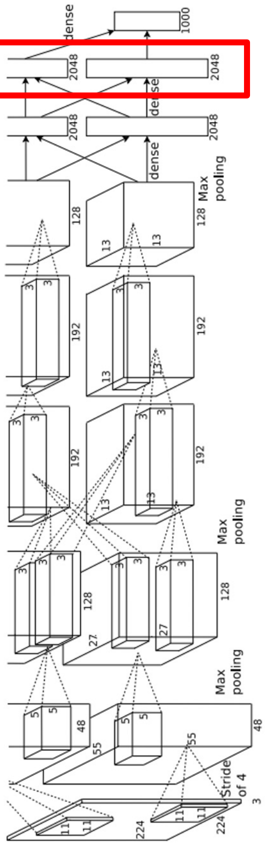




Van der Maaten and Hinton, “Visualizing Data using t-SNE”, JMLR 2008

Last Layer: Dimensionality Reduction



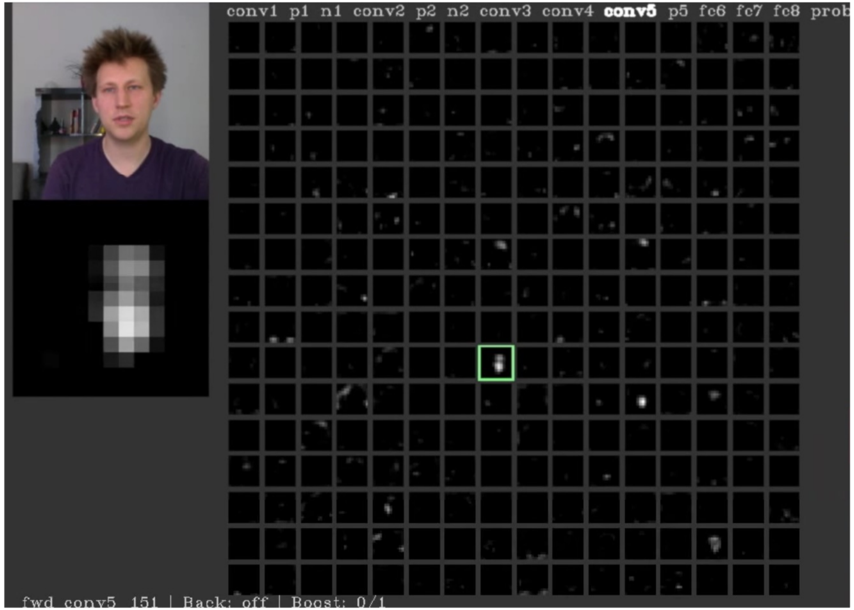


Van der Maaten and Hinton, "Visualizing Data using t-SNE", JMLR 2008
Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012.
Figure reproduced with permission.

See high-resolution versions at
http://cs.stanford.edu/people/karpathy/cnnembed/

Visualizing Activations

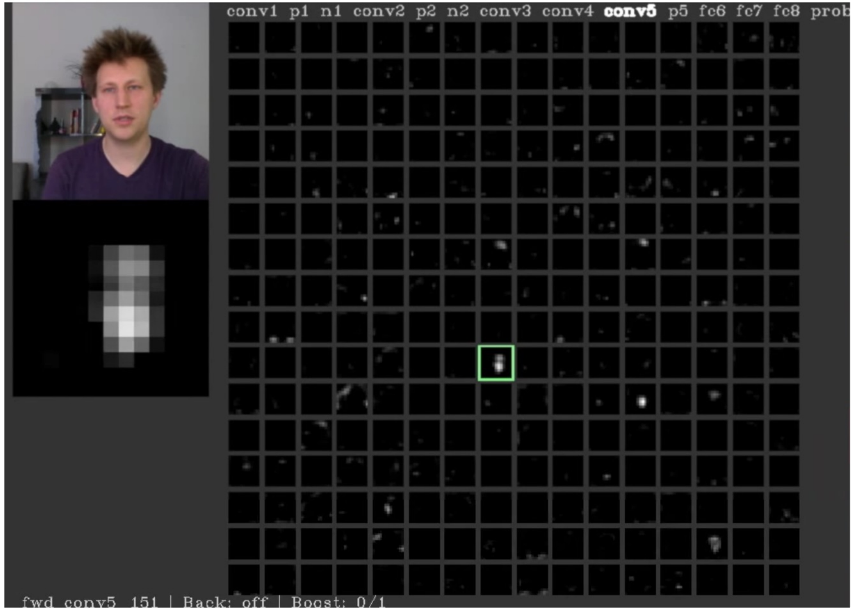
conv5 feature map is 128x13x13; visualize as 128 13x13 grayscale images



Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.

Visualizing Activations

conv5 feature map is 128x13x13; visualize as 128 13x13 grayscale images

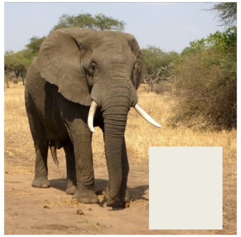


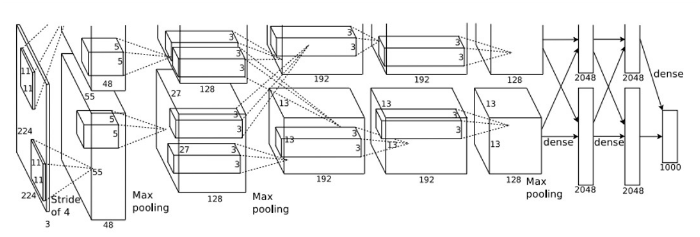
Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.

Neural nets learn distributed representations over many layers. Difficult to visualize everything!

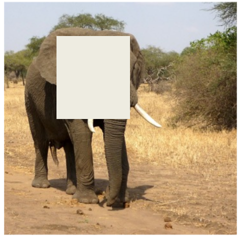
Which pixels matter: Saliency via Occlusion

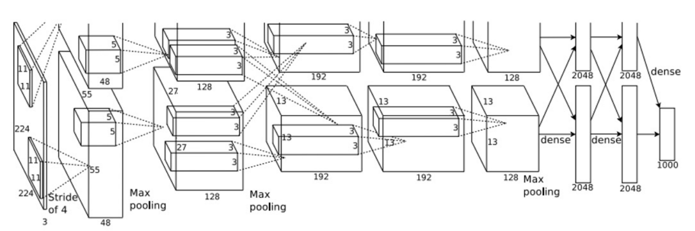
Mask part of the image before feeding to CNN, check how much predicted probabilities change





P(elephant) = 0.95



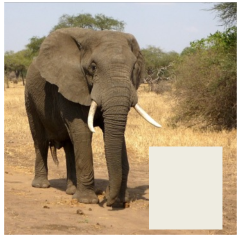


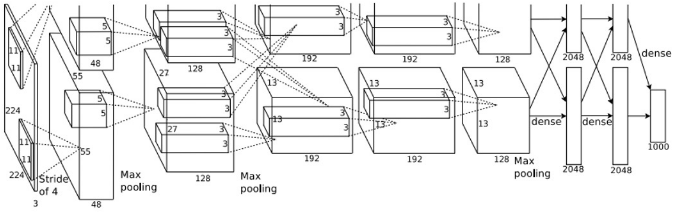
P(elephant) = 0.75

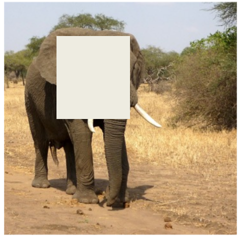
Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014

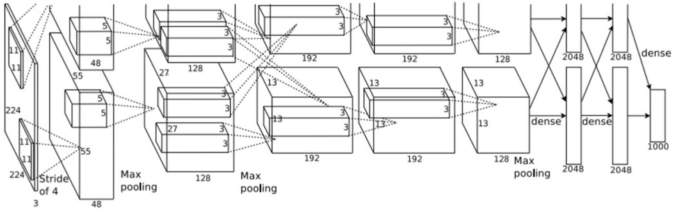
Which pixels matter: Saliency via Occlusion

Mask part of the image before feeding to CNN, check how much predicted probabilities change

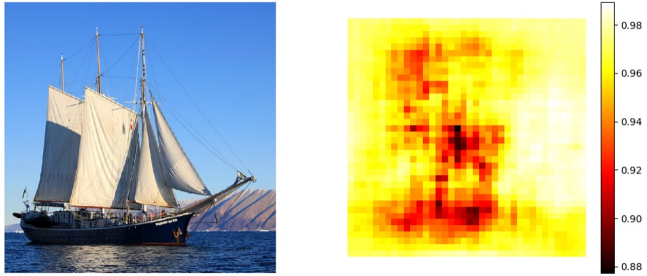








schooner



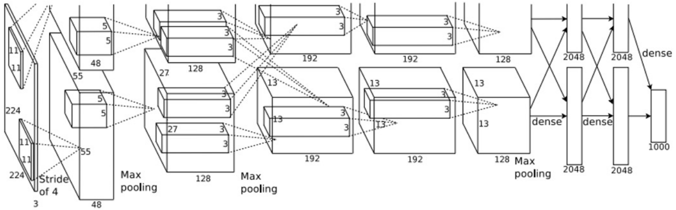
African elephant, Loxodonta africana

go-kart

Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

Which pixels matter: Saliency via Backprop

Forward pass: Compute probabilities

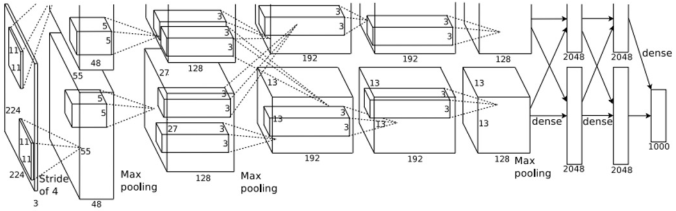


Dog

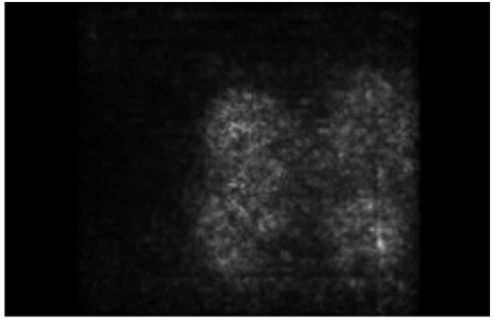
Simonyan, Vedaldi, and Zisserman, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”, ICLR Workshop 2014.

Which pixels matter: Saliency via Backprop

Forward pass: Compute probabilities



Dog



Compute gradient of (unnormalized) class score with respect to image pixels, take absolute value and max/sum over RGB channels

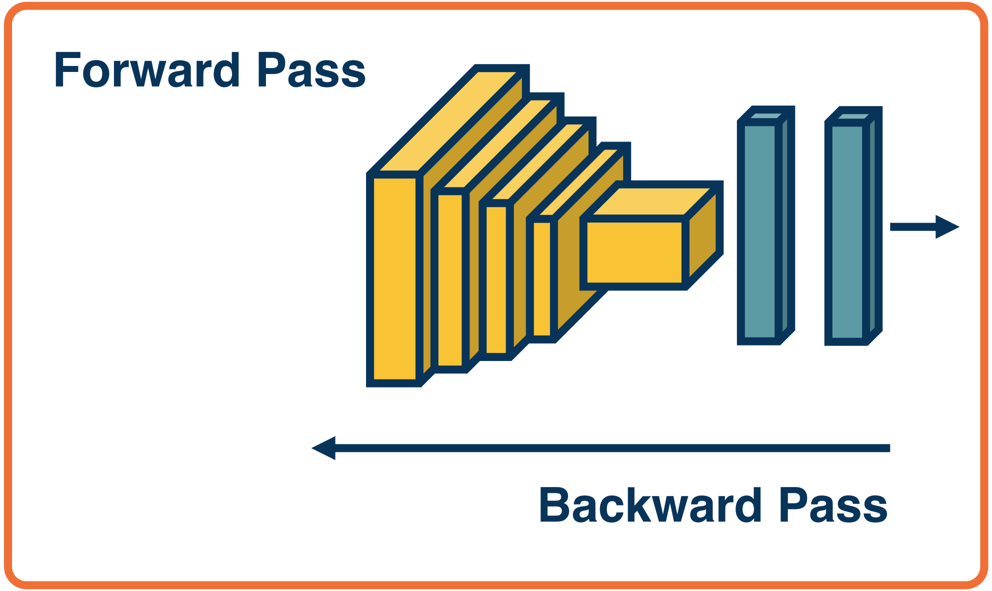
Simonyan, Vedaldi, and Zisserman, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”, ICLR Workshop 2014.

Saliency Maps

Simonyan, Vedaldi, and Zisserman, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”, ICLR Workshop 2014.

Gradient-based Saliency Visualization

Given a **trained** model, we can perform forward pass given an input to get scores, softmax probabilities, loss and then backwards pass to get gradients

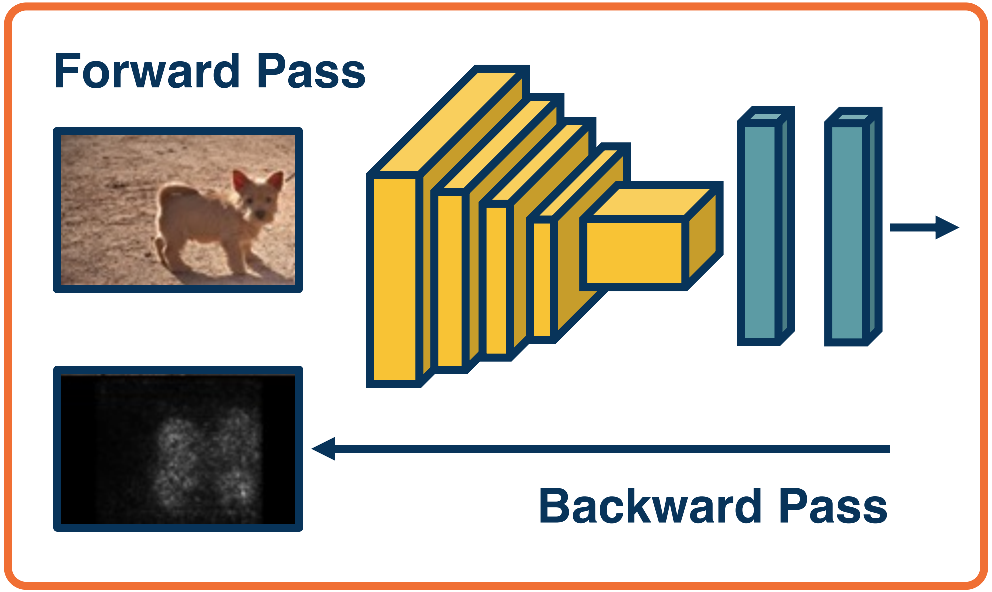


- Note: We are keeping parameters/weights **frozen**
  - Do not use gradients w.r.t. weights to perform updates

Gradient-based Saliency Visualization

Idea: We can backprop to the image

- Sensitivity of loss to individual pixel changes
- Large sensitivity implies important pixels
- Called **Saliency Maps**



In practice:

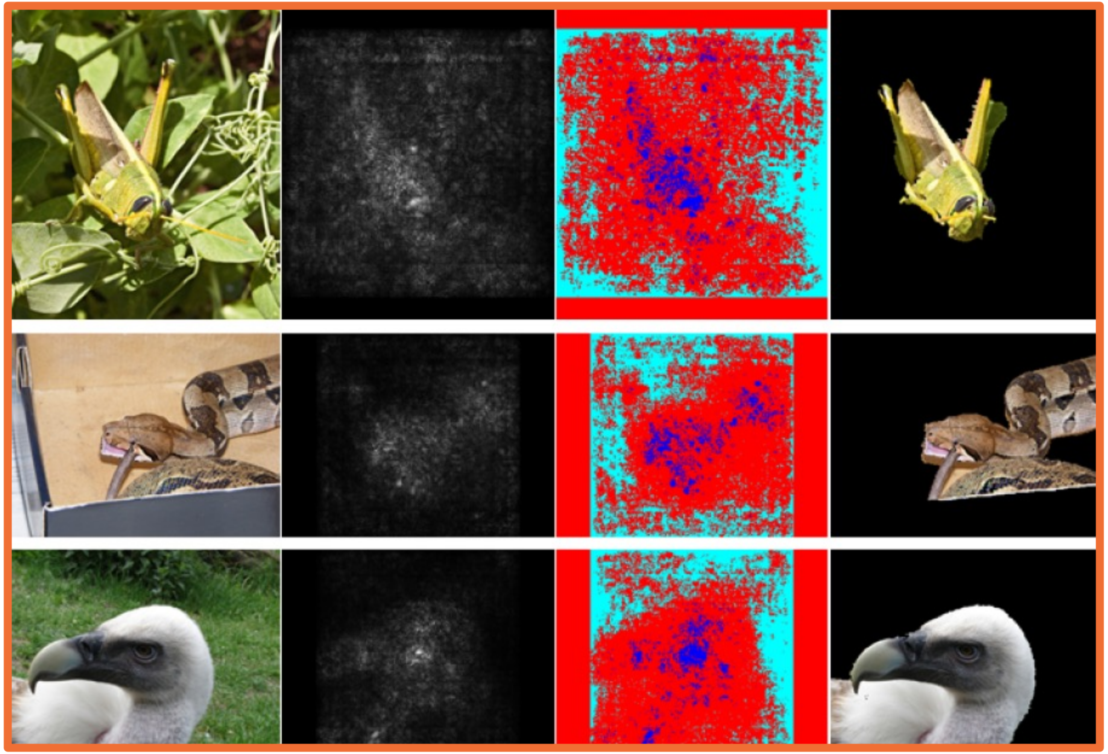
- Instead of loss, find gradient of classifier **scores** (pre-softmax)
- Take absolute value of gradient
- Sum across all channels

From: Simonyan et al., "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", 2013

Gradient-based Saliency Visualization

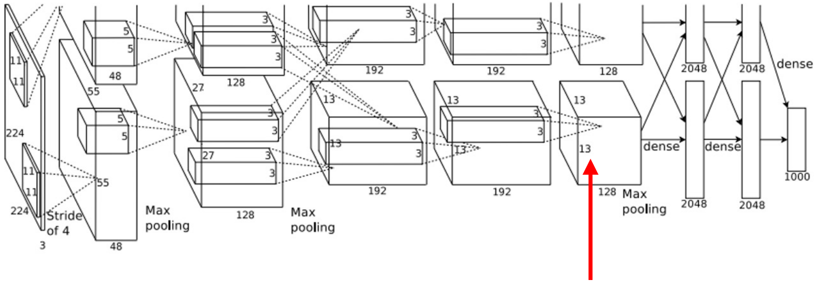
Applying traditional (non-learned) computer vision segmentation algorithms on gradients gets us **object segmentation for free**!

Surprising because **not part of supervision**



From: Simonyan et al., "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", 2013

Intermediate Features via (guided) backprop

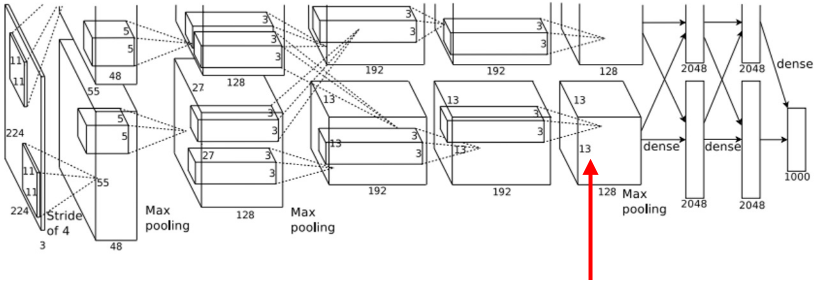


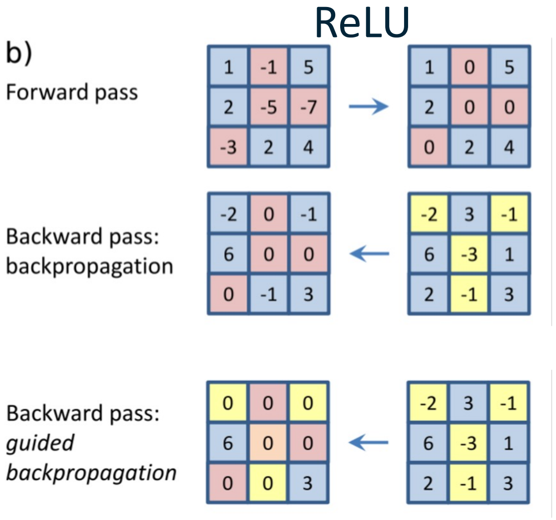
Pick a single intermediate neuron, e.g. one value in 128 x 13 x 13 conv5 feature map

Compute gradient of activation value with respect to image pixels

Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014
Springenberg et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015

Intermediate Features via (guided) backprop





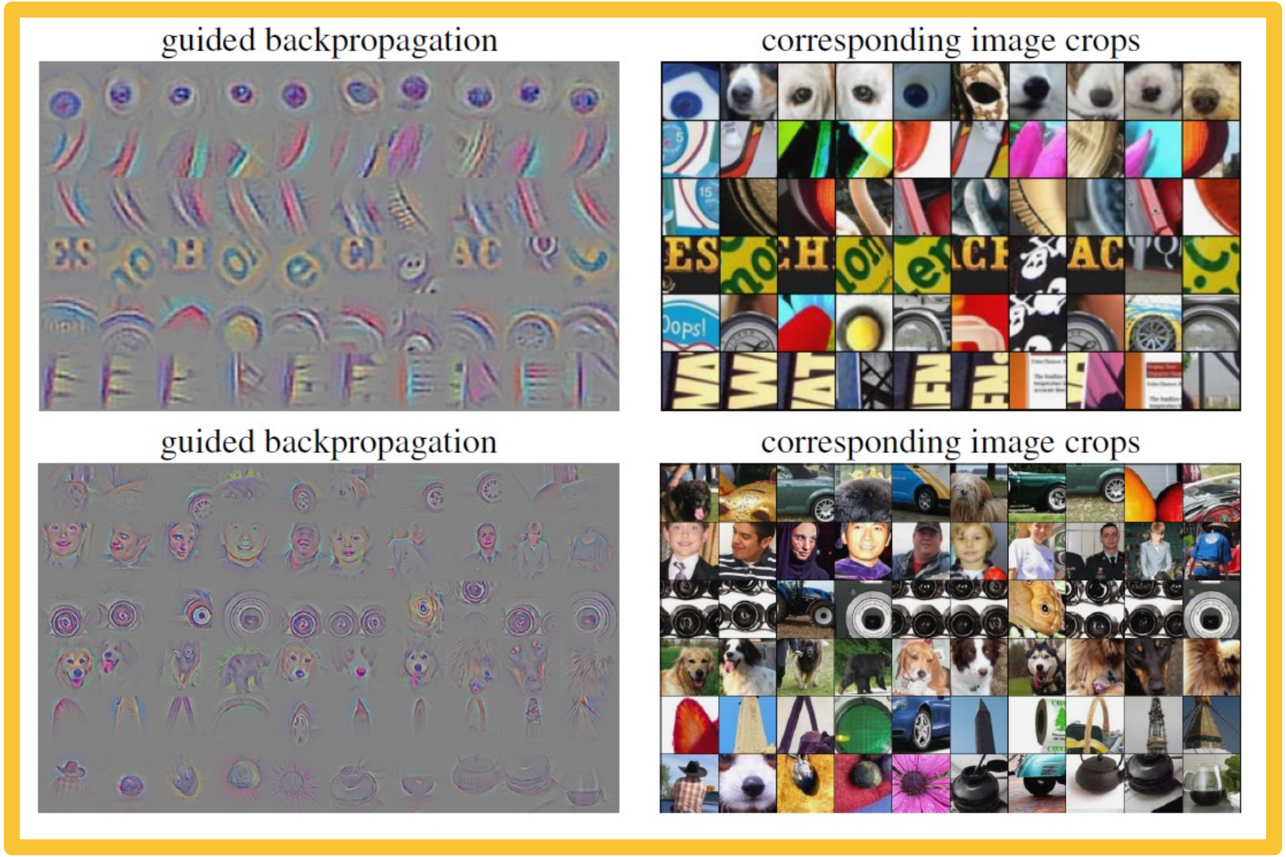
Pick a single intermediate neuron, e.g. one value in 128 x 13 x 13 conv5 feature map

Compute gradient of activation value with respect to image pixels

Guided backprop: suppress pathways that have negative gradients --- only backprop positive gradients through each ReLU

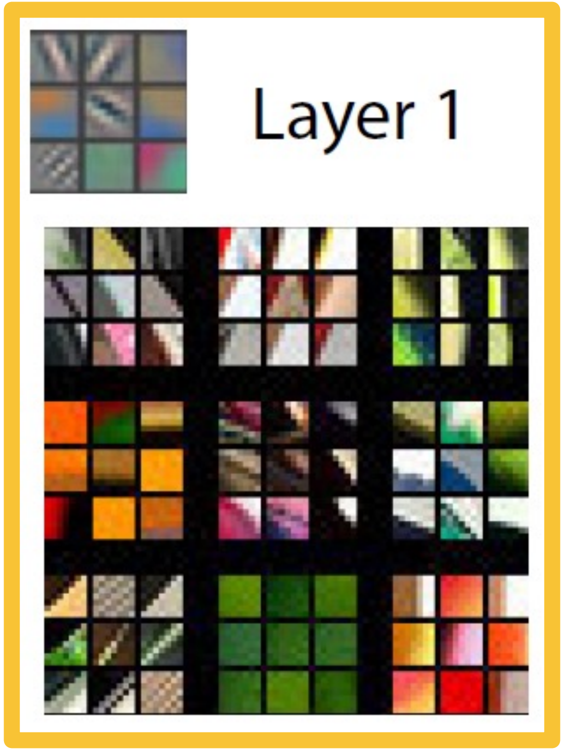
Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014
Springenberg et al, “Striving for Simplicity: The All Convolutional Net”, ICLR Workshop 2015

Guided Backprop Results



From: Springenberg et al., "Striving For Simplicity: The All Convolutional Net"

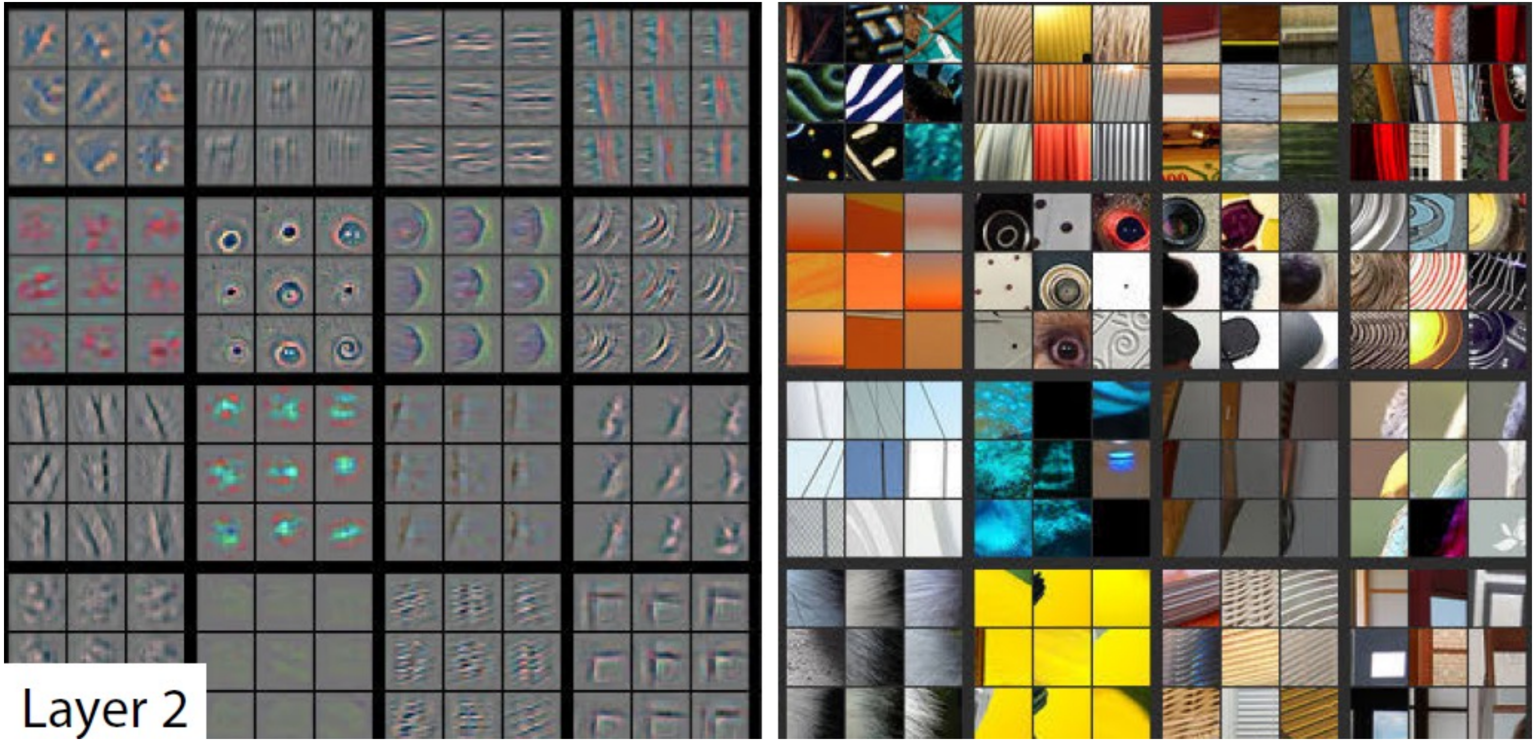
VGG Layer-by-Layer Visualization



Note: These images were created by a slightly different method called **deconvolution,** which ends up being similar to guided backprop

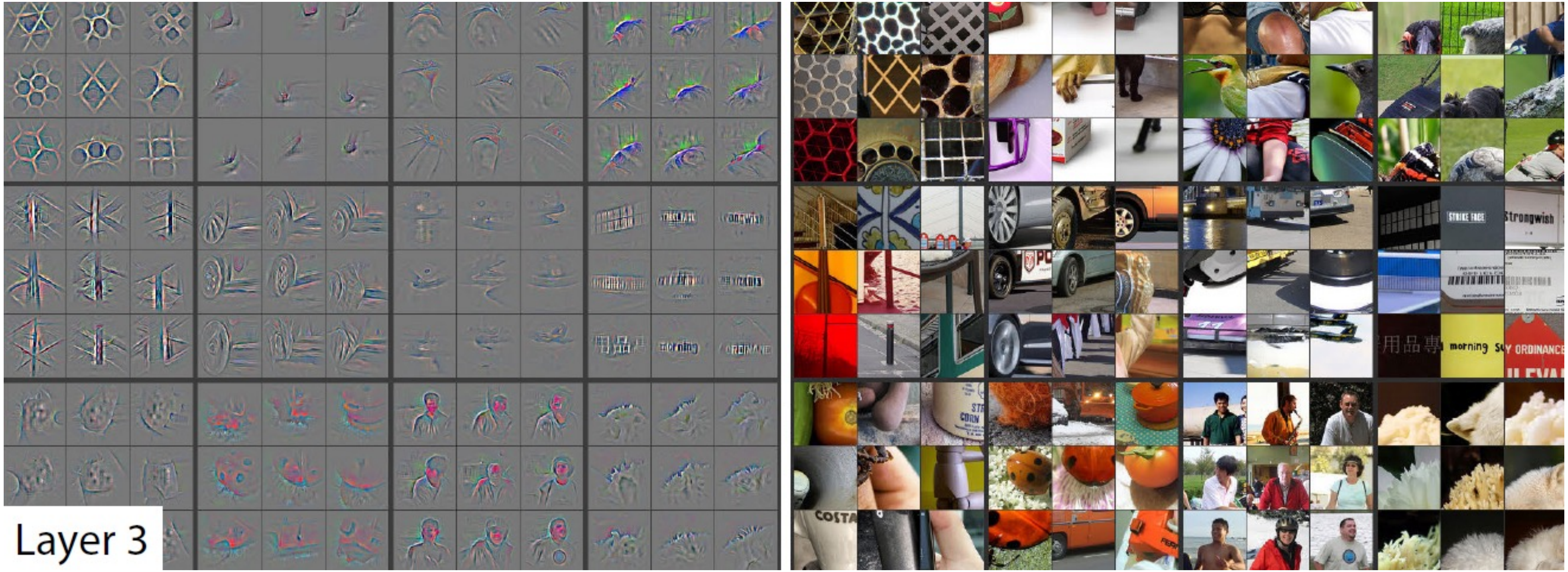
From: "Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014.

VGG Layer-by-Layer Visualization



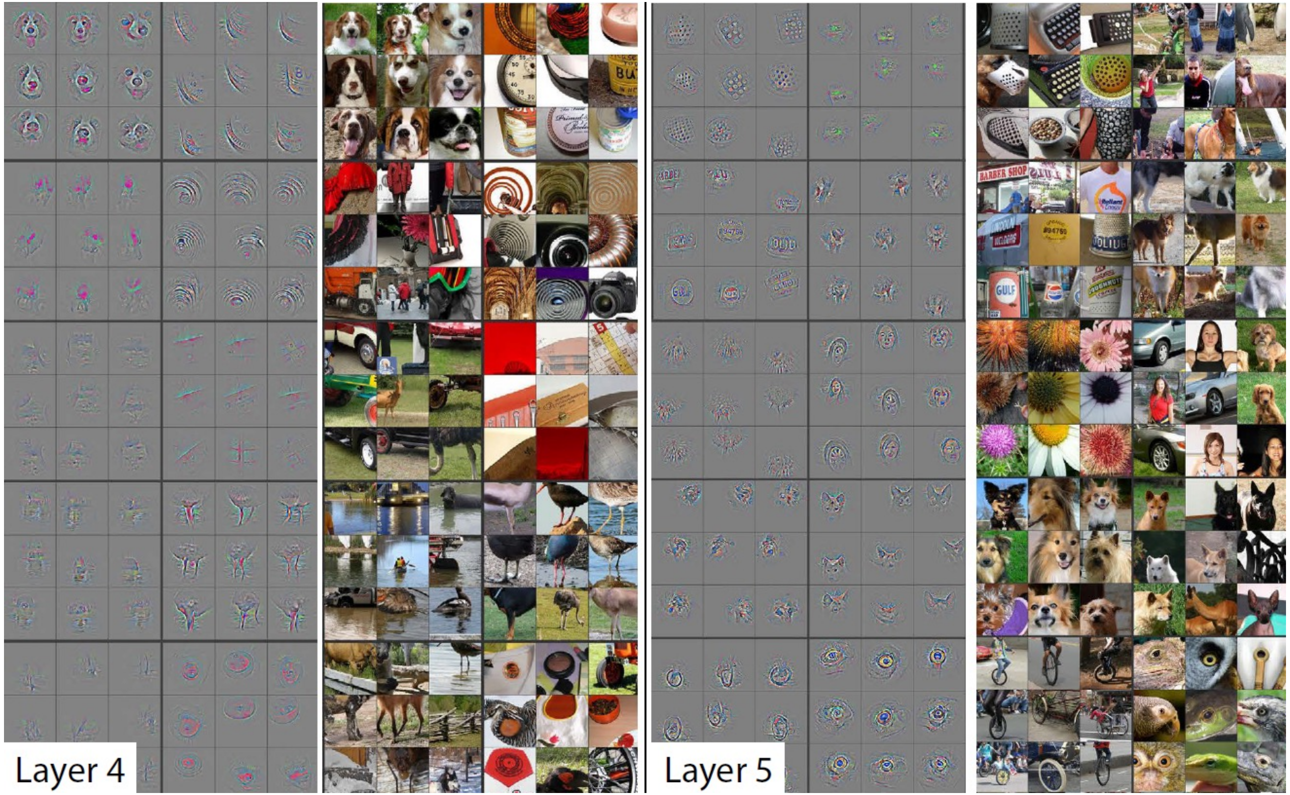
From: "Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014.

VGG Layer-by-Layer Visualization



From: "Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014.

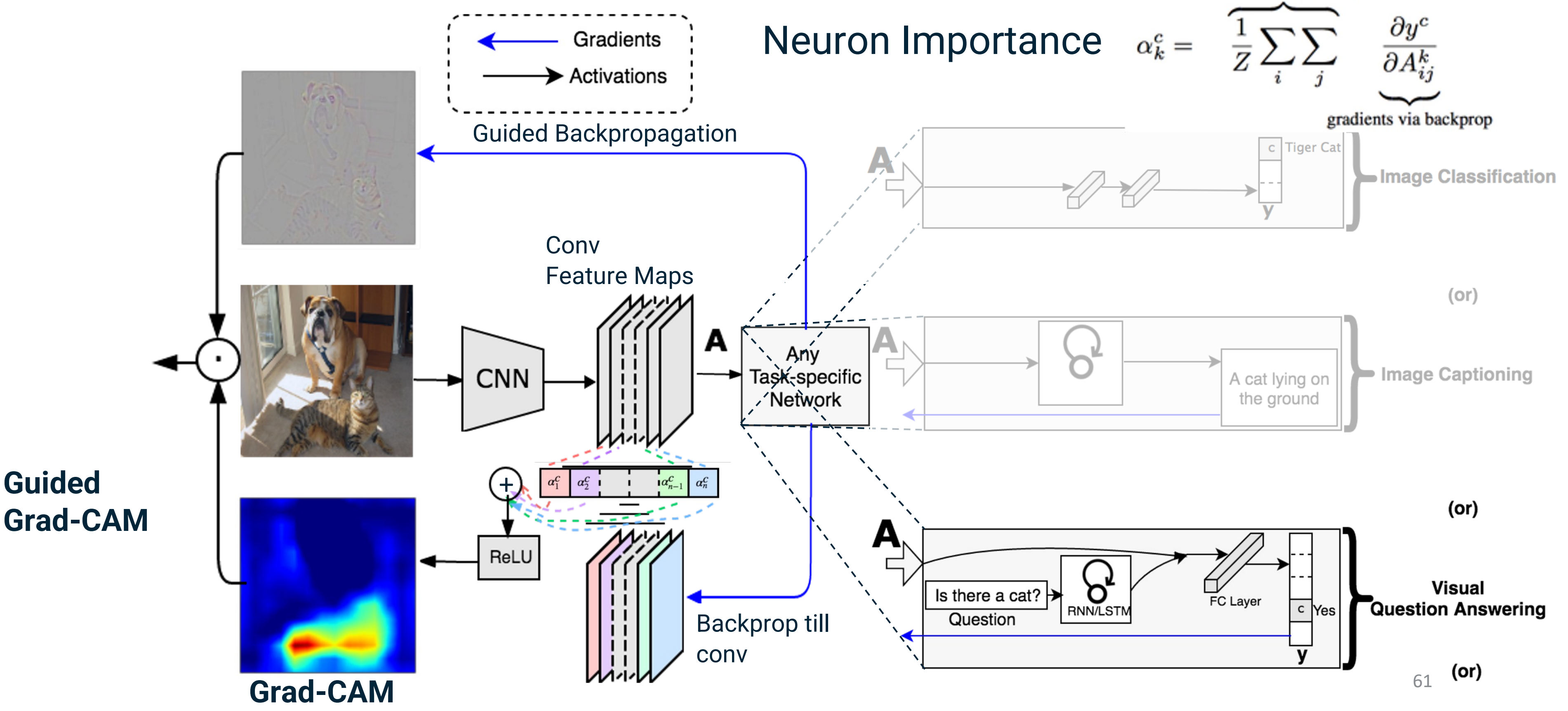
VGG Layer-by-Layer Visualization



From: “Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014.

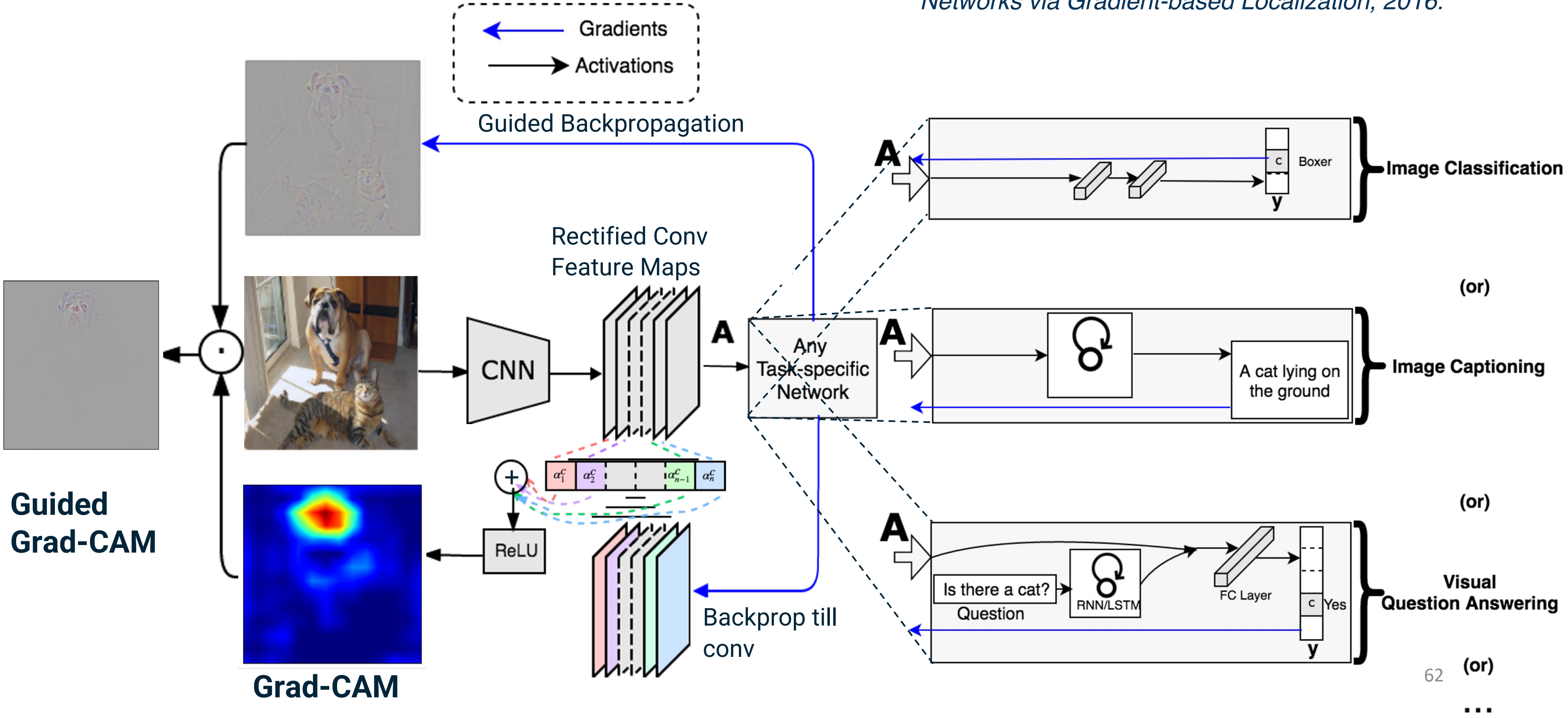
Neuron Importance

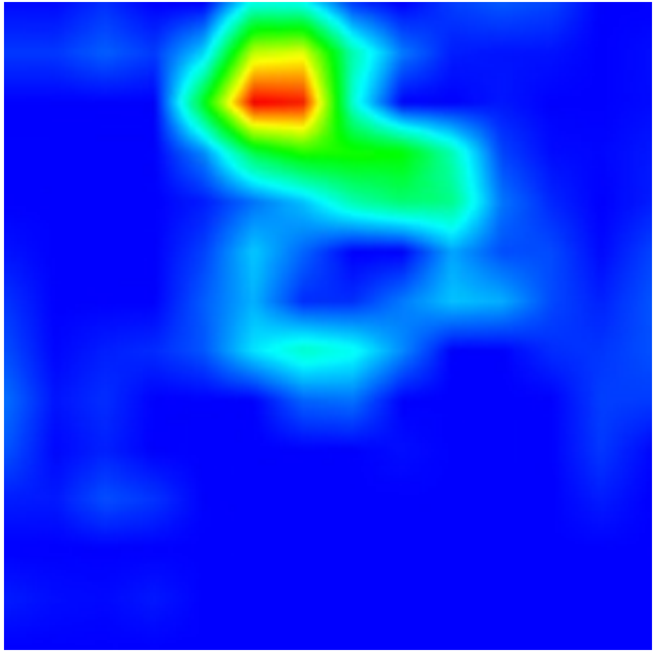
$$\alpha_k^c = \overbrace{\frac{1}{Z}\sum_i \sum_j}^{\text{global average pooling}} \underbrace{\frac{\partial y^c}{\partial A_{ij}^k}}_{\text{gradients via backprop}}$$



Selfvaraju et al., Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization, 2016.

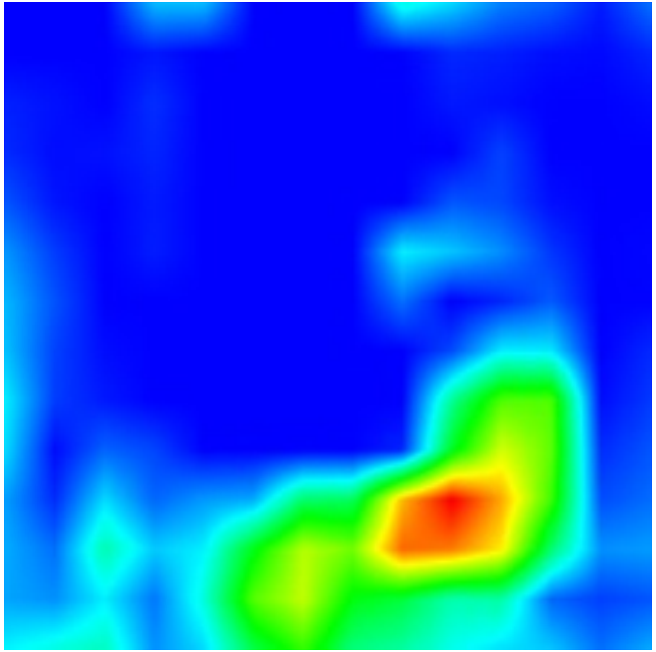
Selfvaraju et al., Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization, 2016.





What animal is in this picture? Dog

Selfvaraju et al., Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization, 2016.



What animal is in this picture? Cat

Selfvaraju et al., Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization, 2016.

Summary

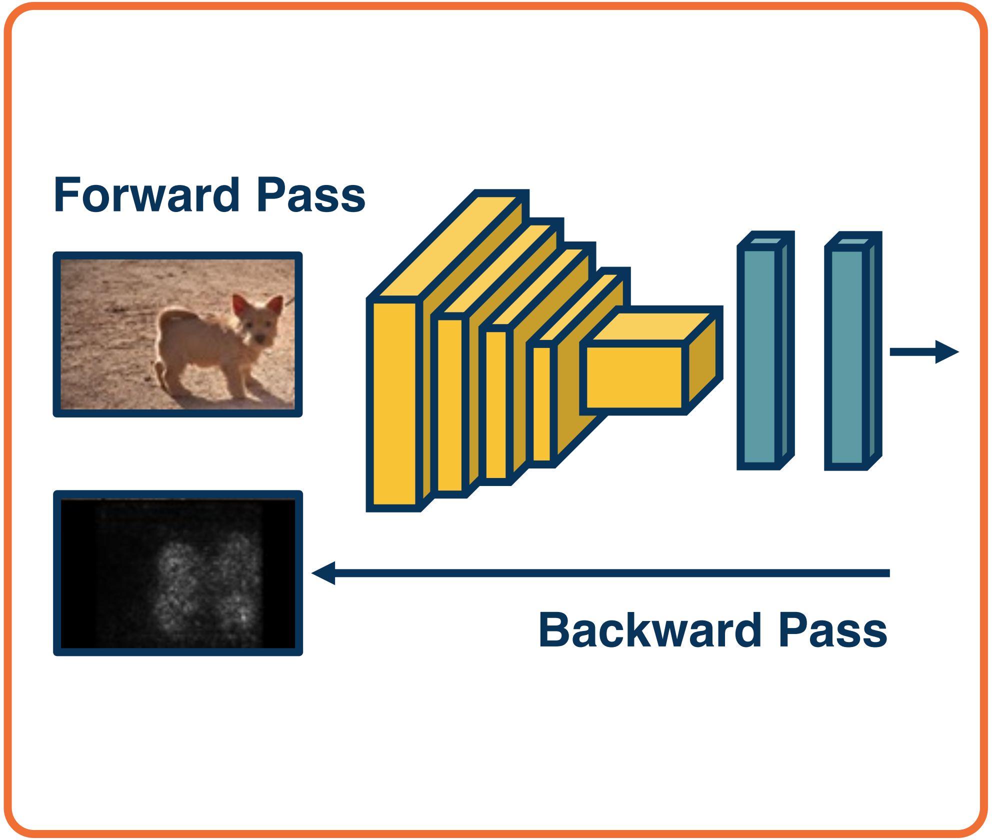
- Gradients are important **not just for optimization**, but also for **analyzing** what neural networks **have learned**
- Standard backprop **not always the most informative** for visualization purposes
- Several ways to **modify the gradient flow** to improve visualization results

Optimizing the Input Images

Idea: Since we have the gradient of scores w.r.t. inputs, can we *optimize* the image itself to maximize the score?

Why?

- Generate images from scratch!
- Adversarial examples



From: Simonyan et al., "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", 2013

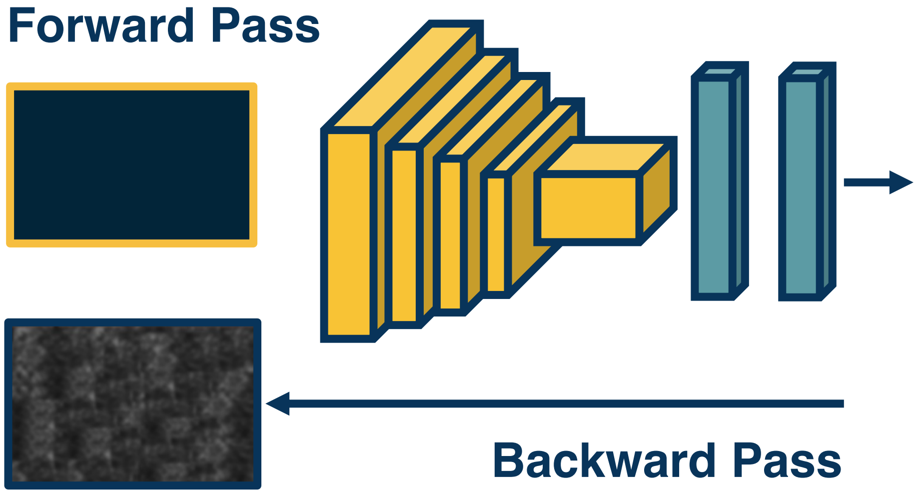
We can perform **gradient ascent** on image for image generation

- Start from random/zero image
- Use scores to avoid minimizing other class scores instead

Often need **regularization term** to induce statistics of natural imagery

- E.g. small pixel values, spatial smoothness

$$\arg\max S_c(I) - \lambda \, ||I||_2^2$$



$$I = I + \alpha \frac{\partial S_c}{\partial I}$$

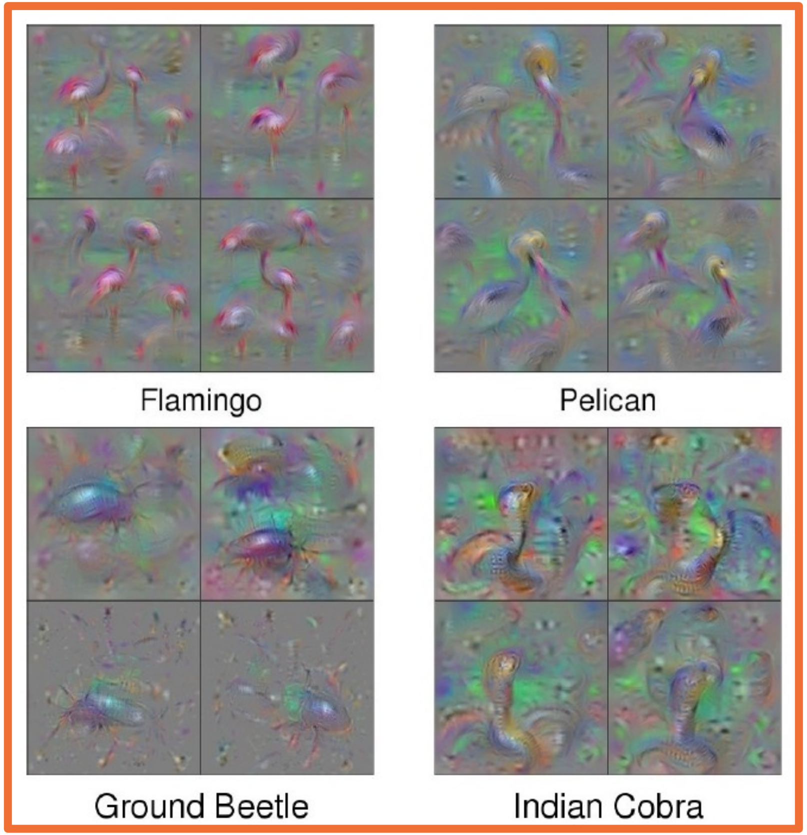
From: Simonyan et al., "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", 2013

Note: You might have to squint!

From: Simonyan et al., "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", 2013

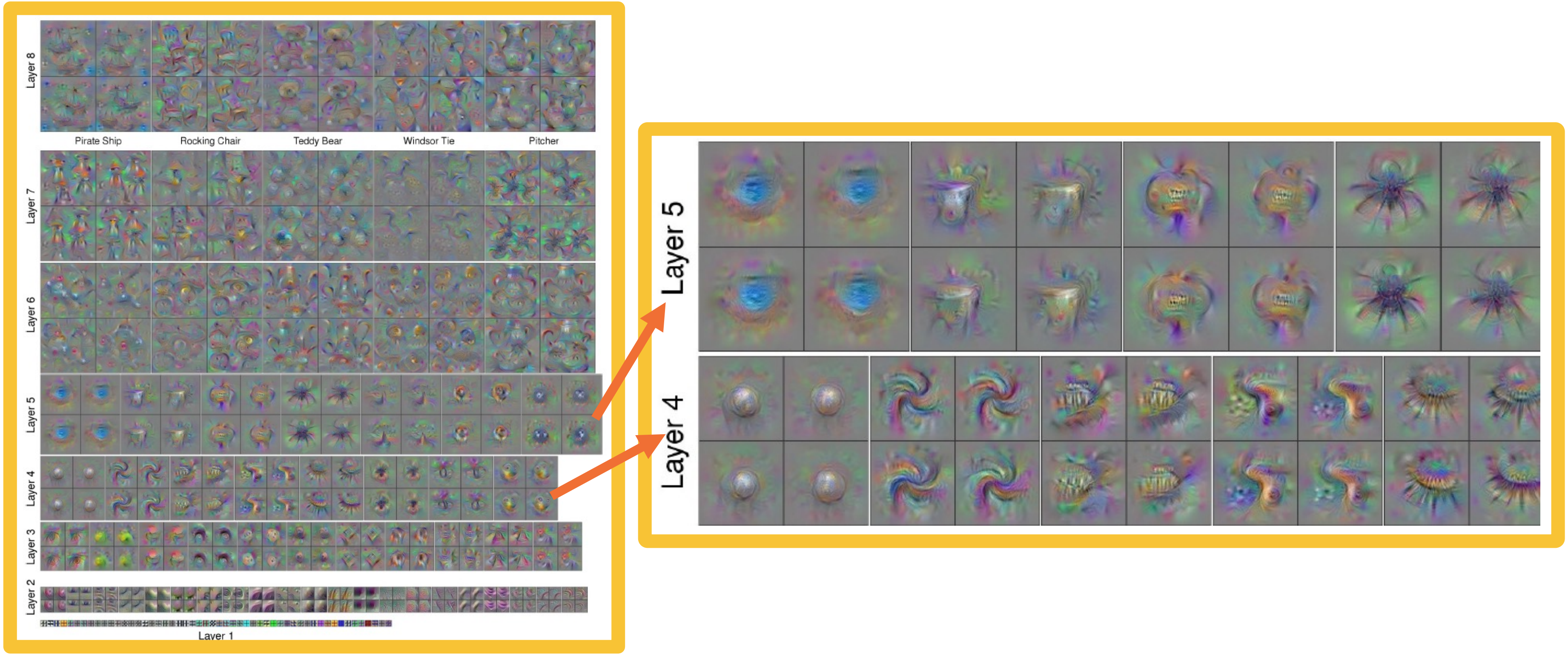
Can improve results with **various tricks**:

- Clipping or normalization of small values & gradients
- Gaussian blurring



From: Yosinski et al., "Understanding Neural Networks Through Deep Visualization", 2015

Improved Results



From: Yosinski et al., "Understanding Neural Networks Through Deep Visualization", 2015

Summary

We can optimize the input image to **generate** examples to increase class scores or activations

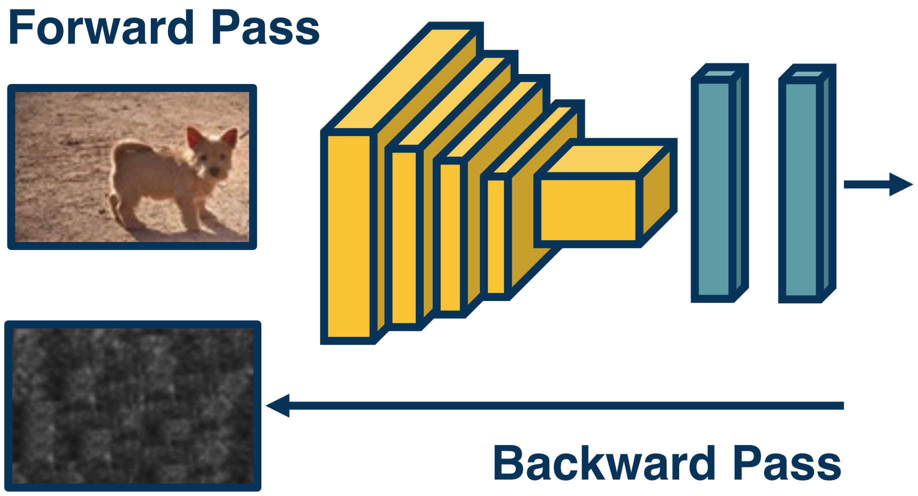
This can show us a great deal about what examples (not in the training set) **activate the network**

- We can perform **gradient ascent** on image
- Rather than start from zero image, why not real image?
- And why not optimize the score of an **arbitrary** (incorrect!) class

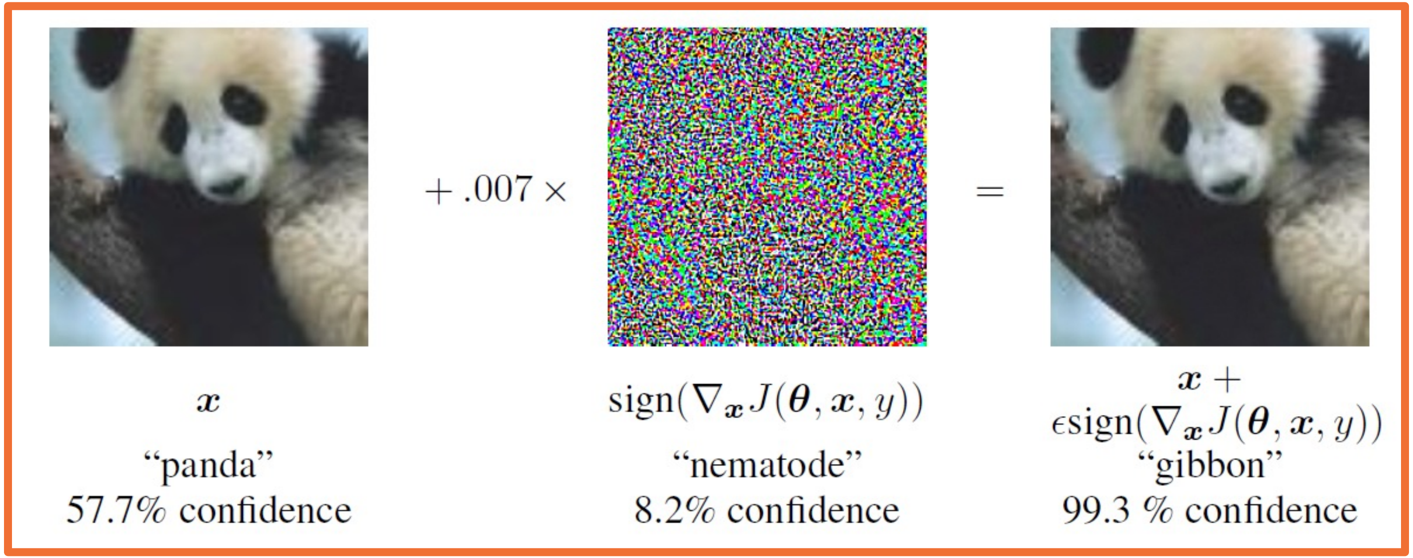
Surprising result: You need very small amount of pixel changes to make the network confidently wrong!

$$\boldsymbol{argmax}\, S_c(I) - \lambda \left|\left|I\right|\right|_2^2$$

where $c = cat$



From: Simonyan et al., "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", 2013



Note this problem is not specific to deep learning!

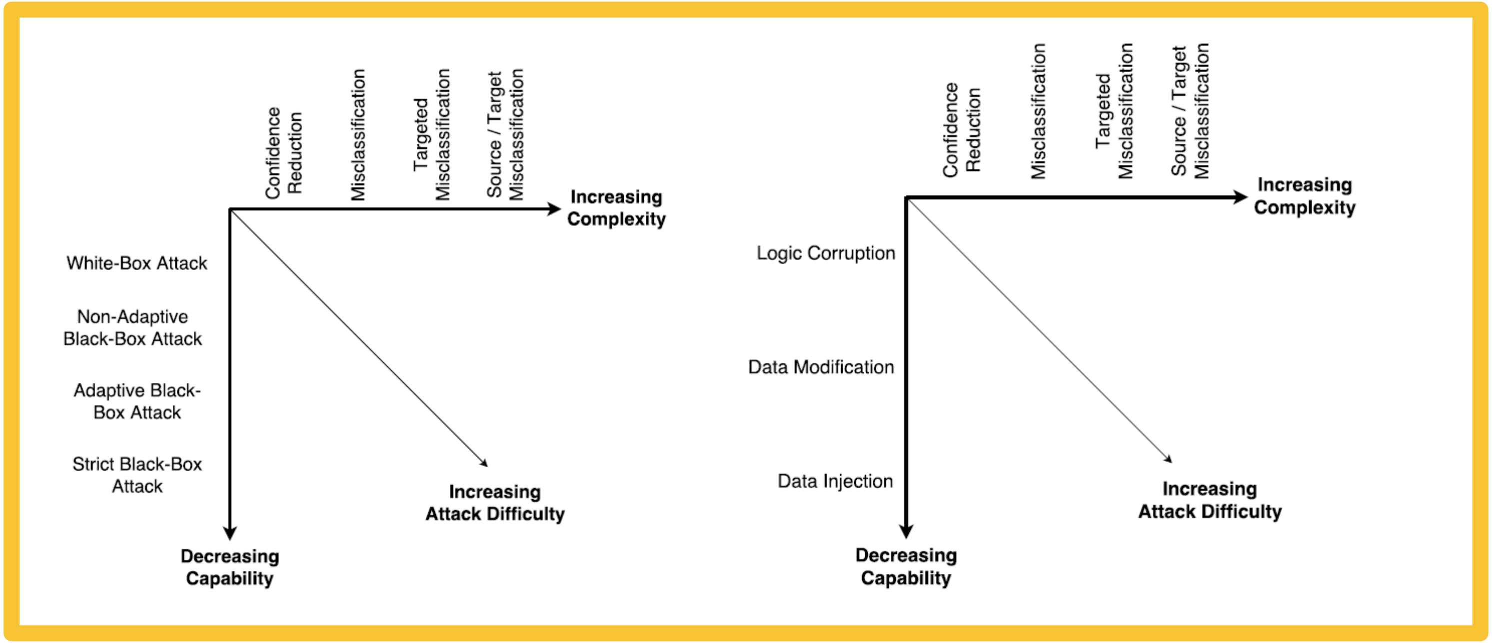
- Other methods also suffer from it
- Can show how **linearity** (even at the end) can bring this about
  - Can add many small values that add up in right direction

From: Goodfellow et al., "Explaining and Harnessing Adversarial Examples", 2015

Variations of Attacks

Single-Pixel Attacks!

Su et al., "One Pixel Attack for Fooling Deep Neural Networks", 2019.



White vs. Black-Box Attacks of Increasing Complexity

Chakraborty et al., Adversarial Attacks and Defences: A Survey, 2018

Summary of adversarial Attacks/Defenses

Similar to other security-related areas, it's an active **cat-and-mouse game**

Several defenses such as:

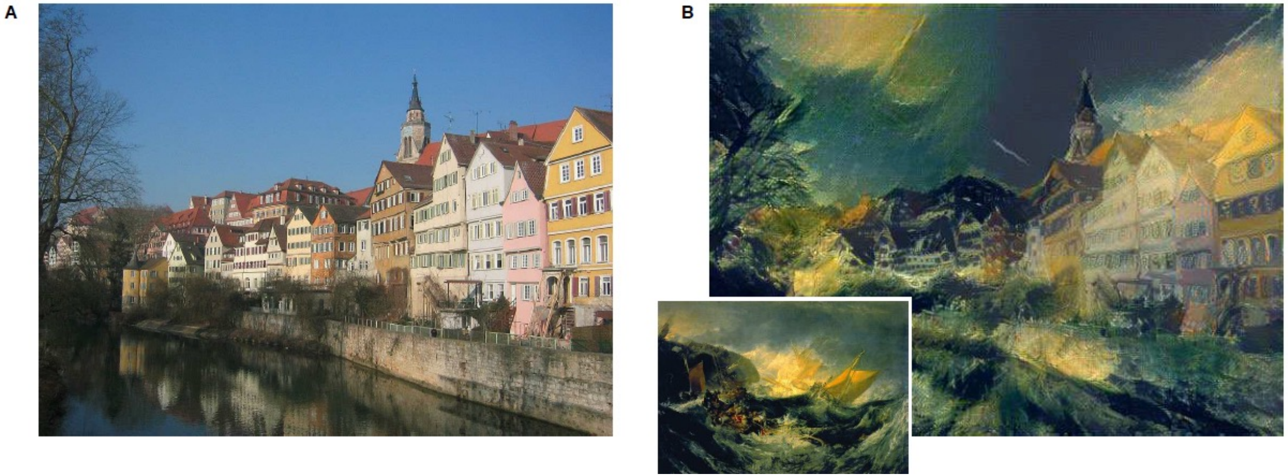
- Training with adversarial examples
- Perturbations, noise, or re-encoding of inputs

There are **not universal methods** that are robust to all types of attacks

Style Transfer: Separating Style from Content

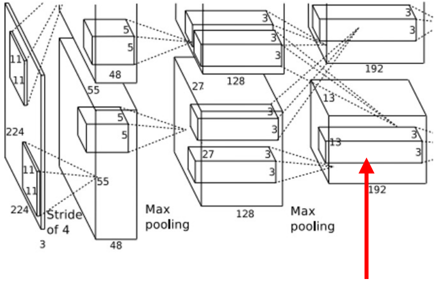
So far, we've seen how to generate images for certain classes / activations through backpropagation / gradient-based optimization.

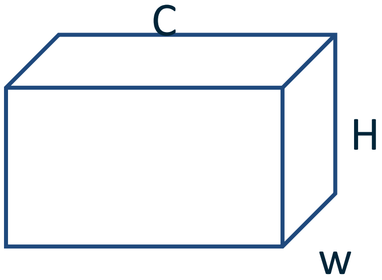
Can we use similar ideas to generate images by combining the **style** and the **content** from different images?



Neural Texture Synthesis: Gram Matrix

This image is in the public domain.

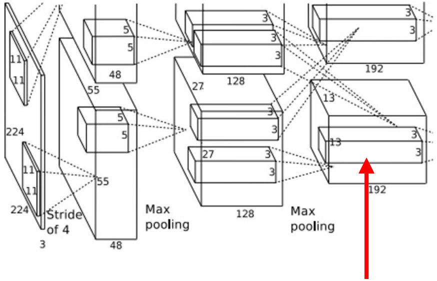


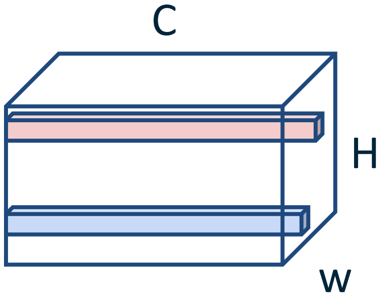


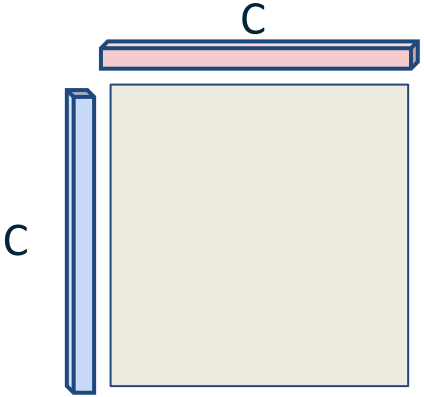
Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

Neural Texture Synthesis: Gram Matrix

This image is in the public domain.





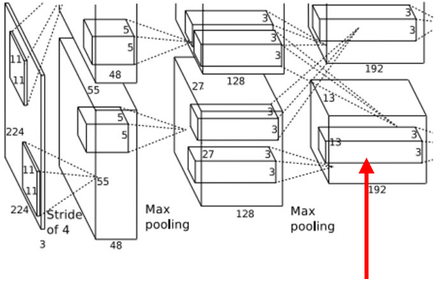


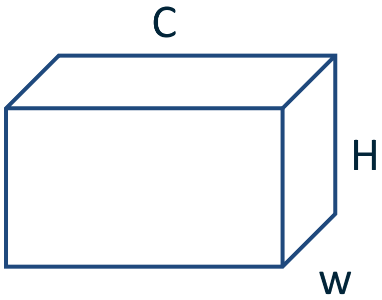
Each layer of CNN gives C x H x W tensor of features;
H x W grid of C-dimensional vectors

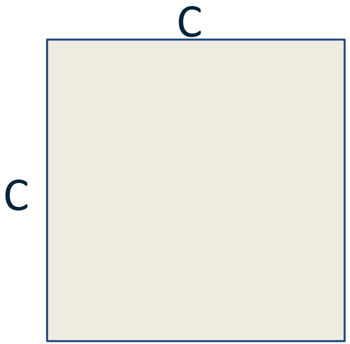
Outer product of two C-dimensional vectors gives C x C matrix measuring co-occurrence

Neural Texture Synthesis: Gram Matrix

This image is in the public domain.







Each layer of CNN gives C x H x W tensor of features;
H x W grid of C-dimensional vectors

Outer product of two C-dimensional vectors gives C x C matrix measuring co-occurrence

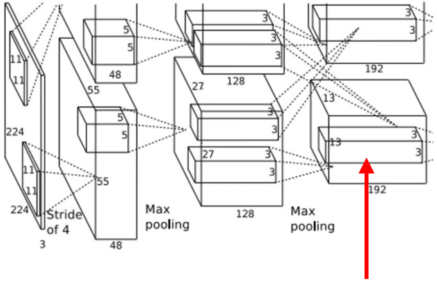
Average over all HW pairs of vectors, giving **Gram matrix G** of shape C x C

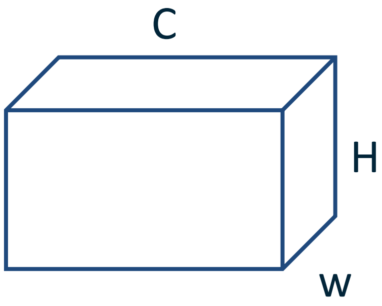
Gram
Matrix

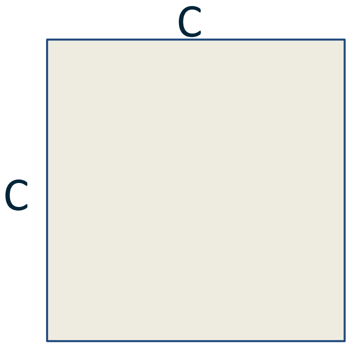
Gram matrix captures the statistics of the **texture** rather than the content of the image

Neural Texture Synthesis: Gram Matrix

This image is in the public domain.







Each layer of CNN gives C x H x W tensor of features;
H x W grid of C-dimensional vectors

Outer product of two C-dimensional vectors gives C x C matrix measuring co-occurrence

Average over all HW pairs of vectors, giving **Gram matrix G** of shape C x C

Efficient to compute; reshape features from

C x H x W to =C x HW

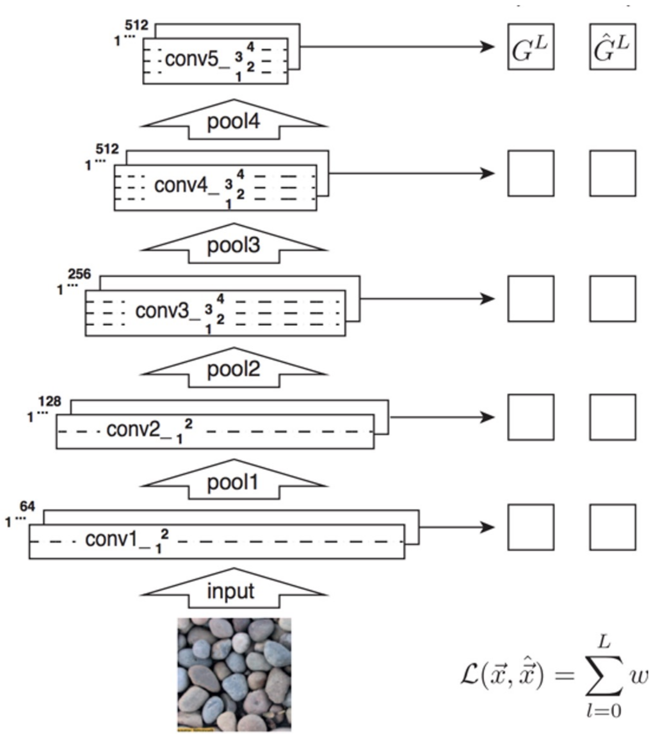
then compute G from all pairs of feature vectors

Gram matrix captures the statistics of the **texture** rather than the content of the image

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:

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



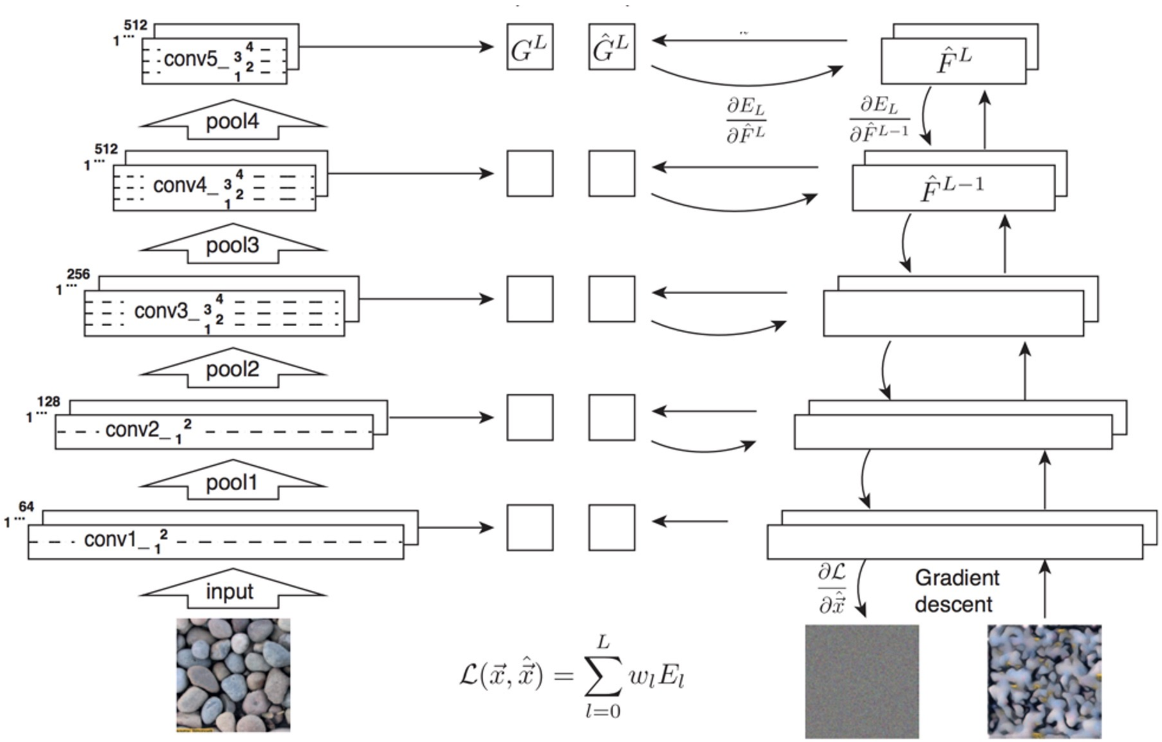
Gatys, Ecker, and Bethge, “Texture Synthesis Using Convolutional Neural Networks”, NIPS 2015

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:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$ hape $C_i \times C_i$)

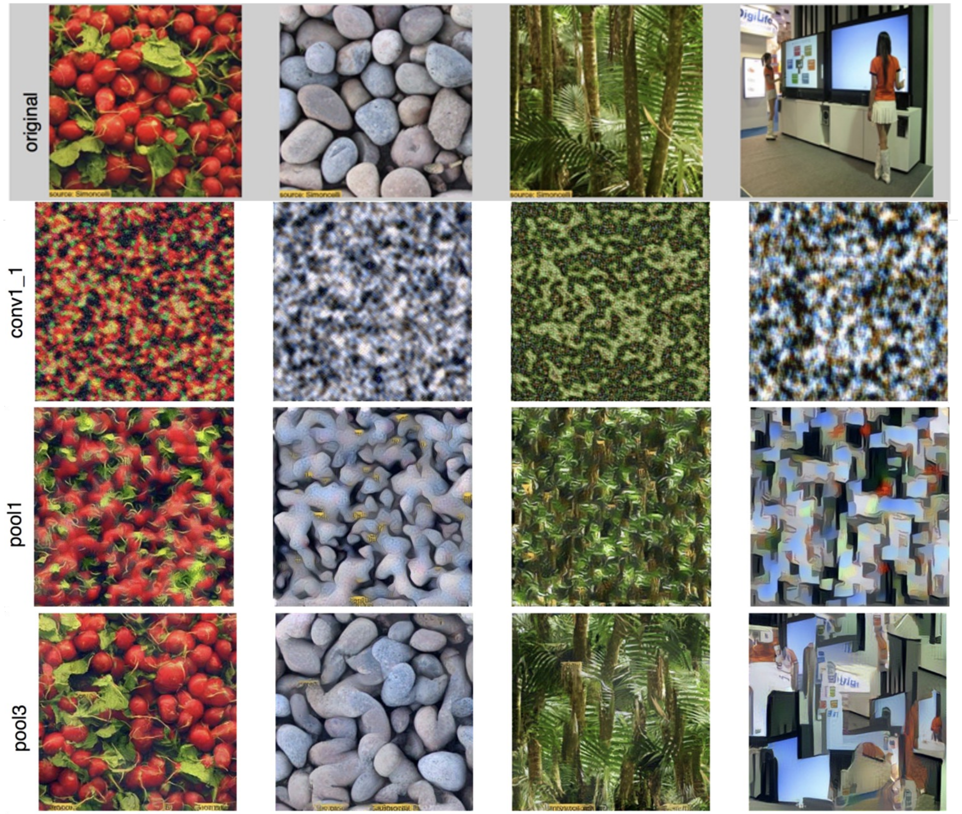
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5



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

Neural Texture Synthesis

Reconstructing texture from higher layers recovers larger features from the input texture



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

Neural Style Transfer

Content Image

+

Style Image

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

Neural Style Transfer

Content Image

+

Style Image

=

Style Transfer!

Gatys, Ecker, and Bethge, “Image style transfer using convolutional neural networks”, CVPR 2016

Feature Inversion

Given a CNN feature vector for an image, find a new image that:

- Matches the given feature vector
- “looks natural” (image prior regularization)

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

Given feature vector

Features of new image

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

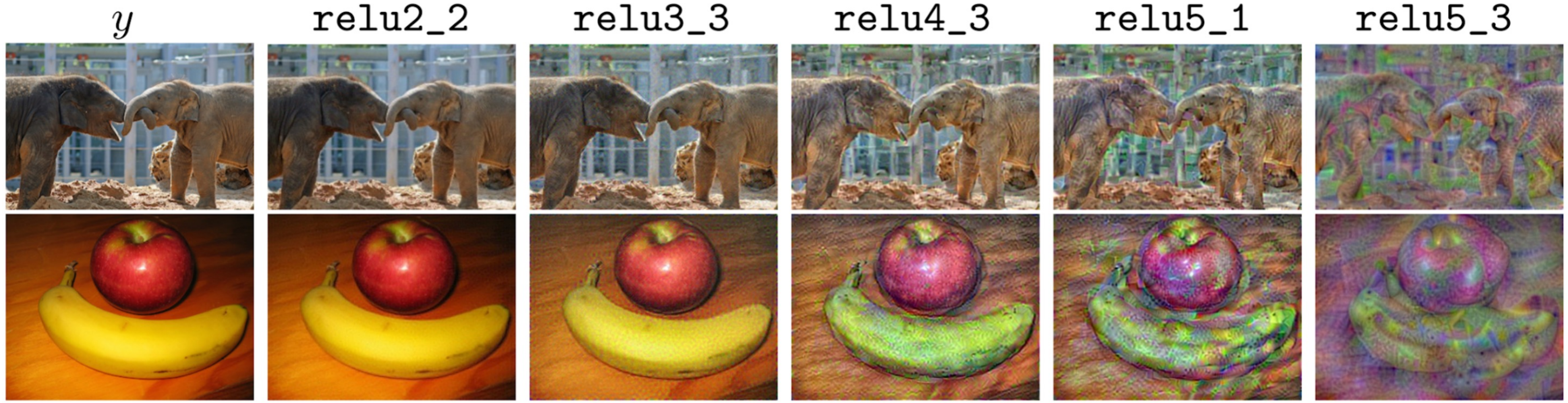
$$\mathcal{R}_{V^\beta}(\mathbf{x}) = \sum_{i,j} \left((x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\frac{\beta}{2}}$$

Total Variation regularizer (encourages spatial smoothness)

Mahendran and Vedaldi, “Understanding Deep Image Representations by Inverting Them”, CVPR 2015

Feature Inversion

Reconstructing from different layers of VGG-16

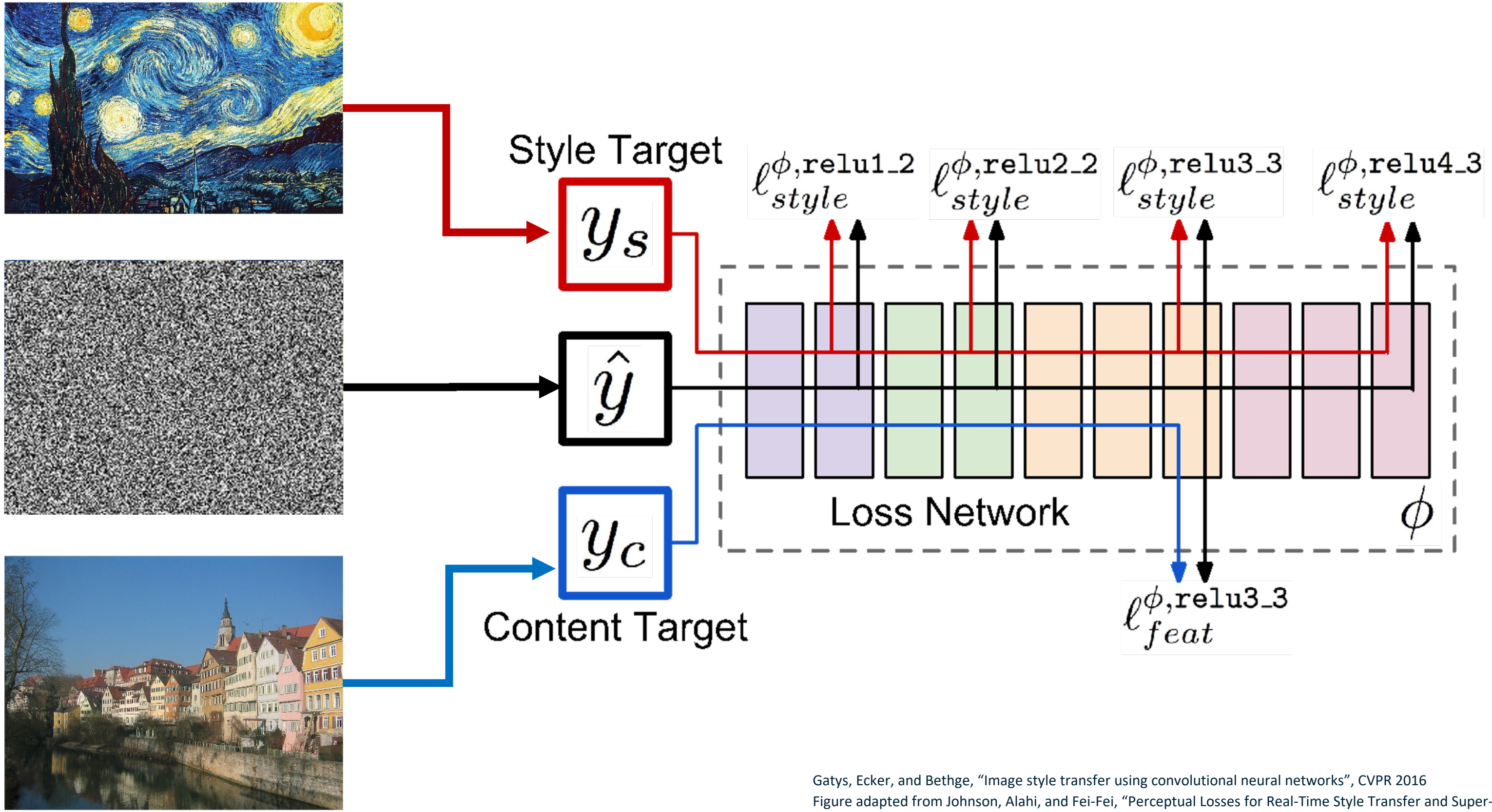


Mahendran and Vedaldi, “Understanding Deep Image Representations by Inverting Them”, CVPR 2015
Figure from Johnson, Alahi, and Fei-Fei, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016.

Style image

Output image (Start with noise)

Content image

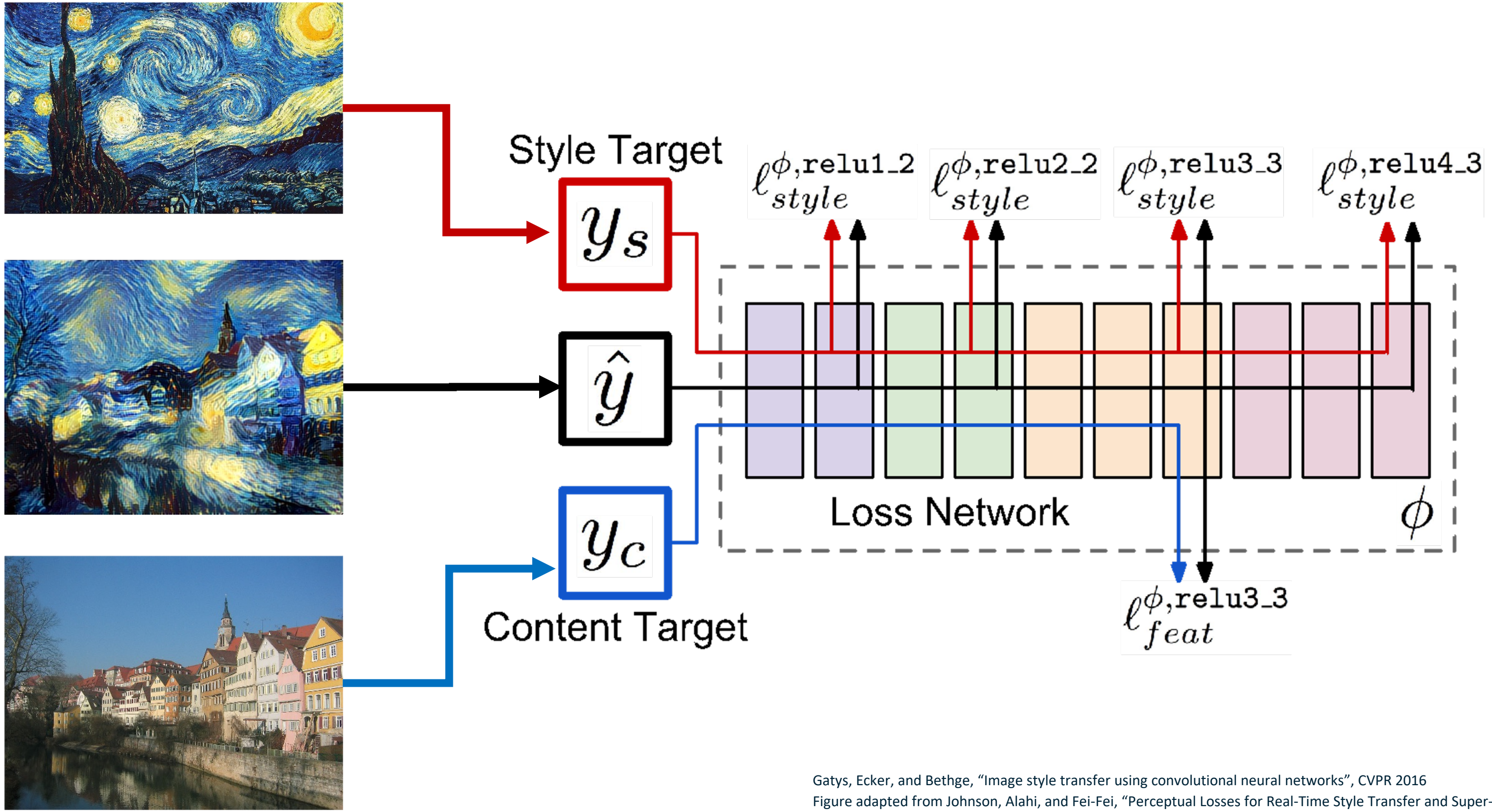


Gatys, Ecker, and Bethge, “Image style transfer using convolutional neural networks”, CVPR 2016
Figure adapted from Johnson, Alahi, and Fei-Fei, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016.

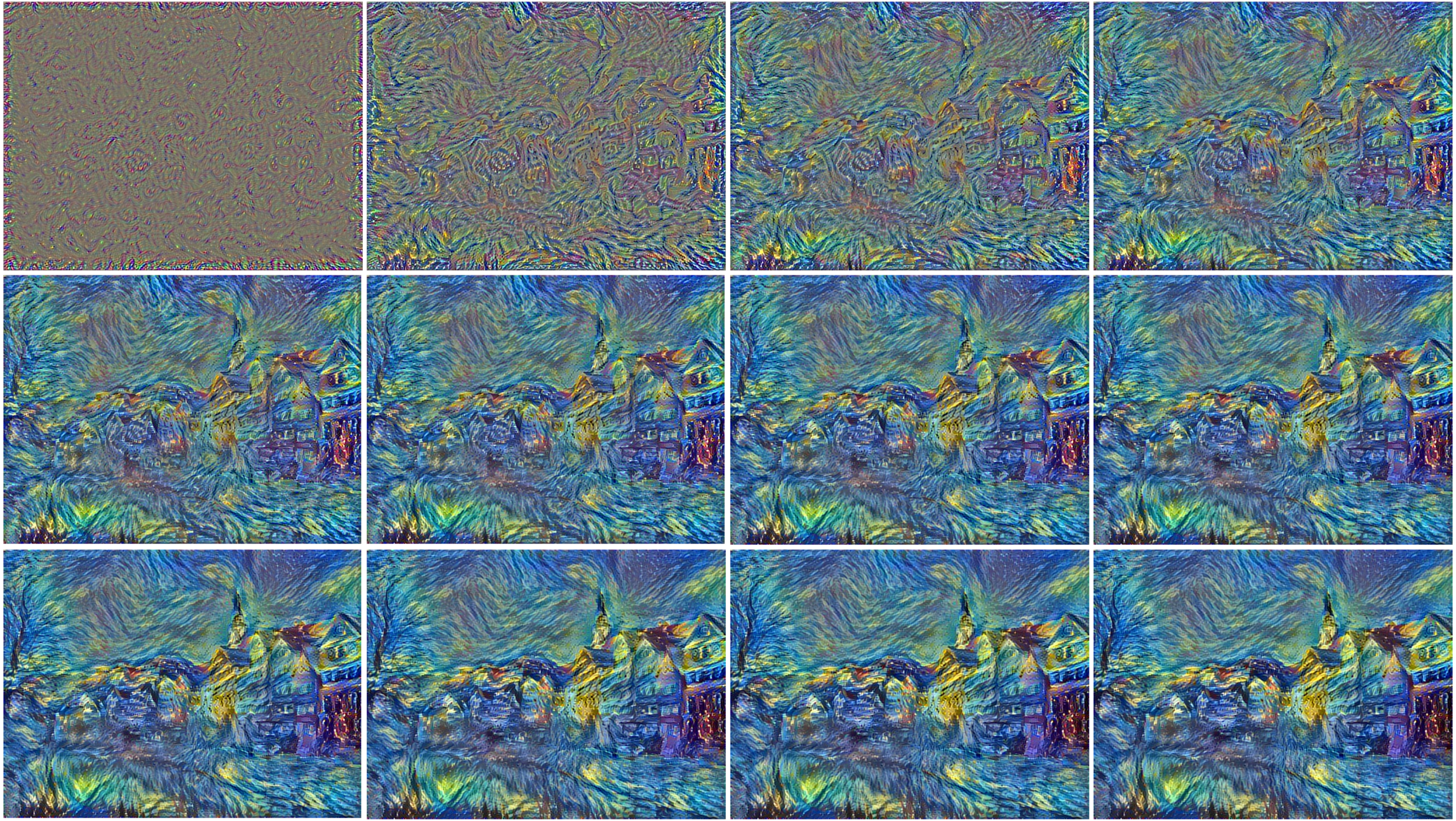
Style image

Output image

Content image



Gatys, Ecker, and Bethge, “Image style transfer using convolutional neural networks”, CVPR 2016
Figure adapted from Johnson, Alahi, and Fei-Fei, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016.



Neural Style Transfer

Gatys, Ecker, and Bethge, “Image style transfer using convolutional neural networks”, CVPR 2016
Figure copyright Justin Johnson, 2015.

Summary

- Generating images through optimization is a powerful concept!

- Besides fun and art, methods such as stylization also useful for understanding what the network has learned

- Also useful for other things such as data augmentation