Topics:

- Questions on convolution layers
- Visualization

CS 4803-DL / 7643-A
ZSOLT KIRA

- **Assignment 2**
  - Due in **4 days!!!**

- **GPU resources**
  - Google Cloud Credits
  - Google Colab
  - Should not be necessary for assignments though

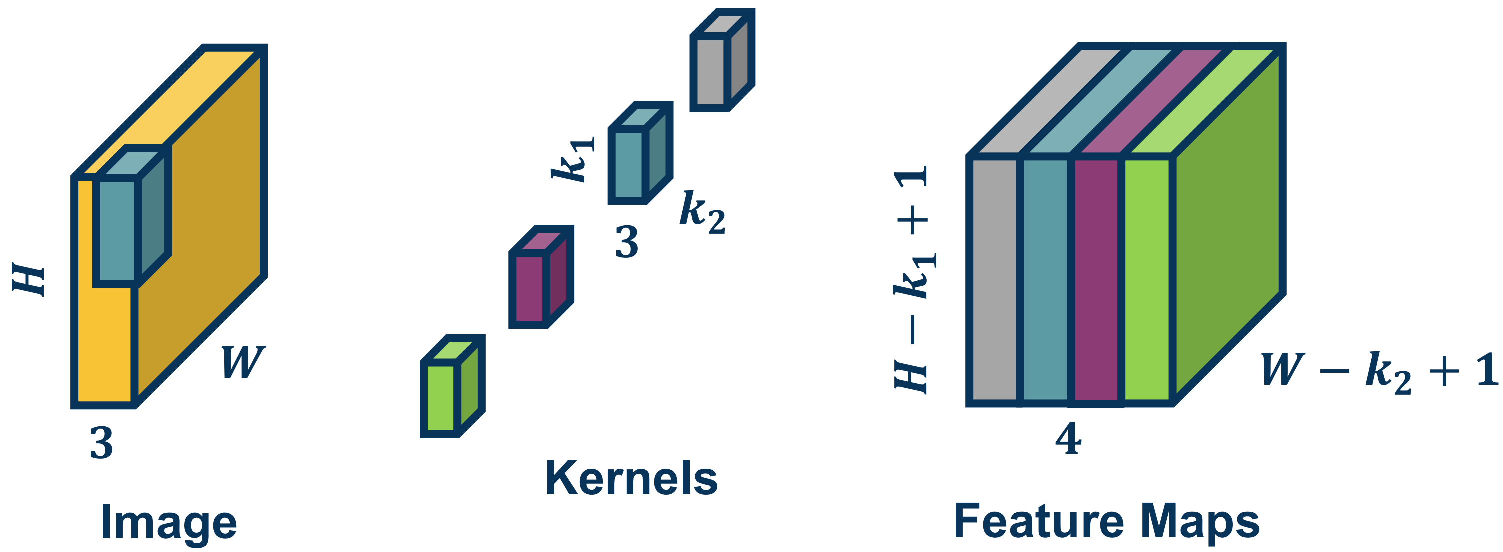
- **Projects**
  - Released catme, fill out by **02/28!** If you have a team, no need.
  - Rubric/description released, my office hours went over it
  - Some interesting topics here. FB topics coming out this month.
  - Project proposal due **mid-March** (will re

- 4803 special office hours

We can have **multiple kernels per layer**

- We stack the feature maps together at the output

Number of channels in output is equal to *number of kernels*



Multiple Kernels

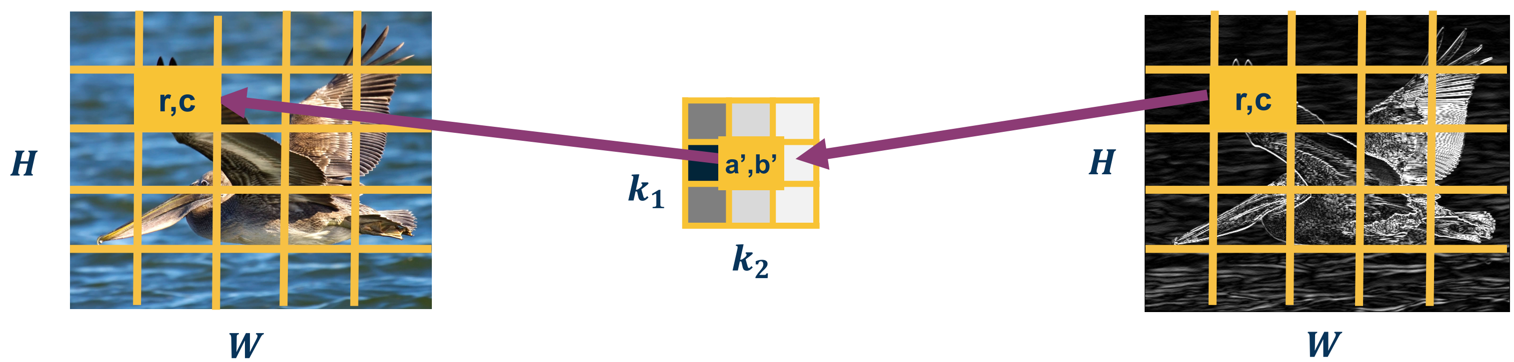
$$\frac{\partial y(r,c)}{\partial k(a',b')} = x(r+a', c+b')$$

$$\frac{\partial L}{\partial k(a',b')} = \sum_{r=0}^{H-1} \sum_{c=0}^{W-1} \frac{\partial L}{\partial y(r,c)} x(r+a', c+b')$$

Does this look familiar?

Cross-correlation between upstream gradient and input!

(until $k_1 \times k_2$ output)



Gradients and Cross-Correlation

Plugging in to earlier equation:

$$\frac{\partial L}{\partial x(r',c')} = \sum_{a=0}^{k_1-1}\sum_{b=0}^{k_2-1} \frac{\partial L}{\partial y(r'-a,c'-b)} \frac{\partial y(r'-a,c'-b)}{\partial x(r',c')}$$

Does this look familiar?

$$= \sum_{a=0}^{k_1-1}\sum_{b=0}^{k_2-1} \frac{\partial L}{\partial y(r'-a,c'-b)} k(a,b)$$

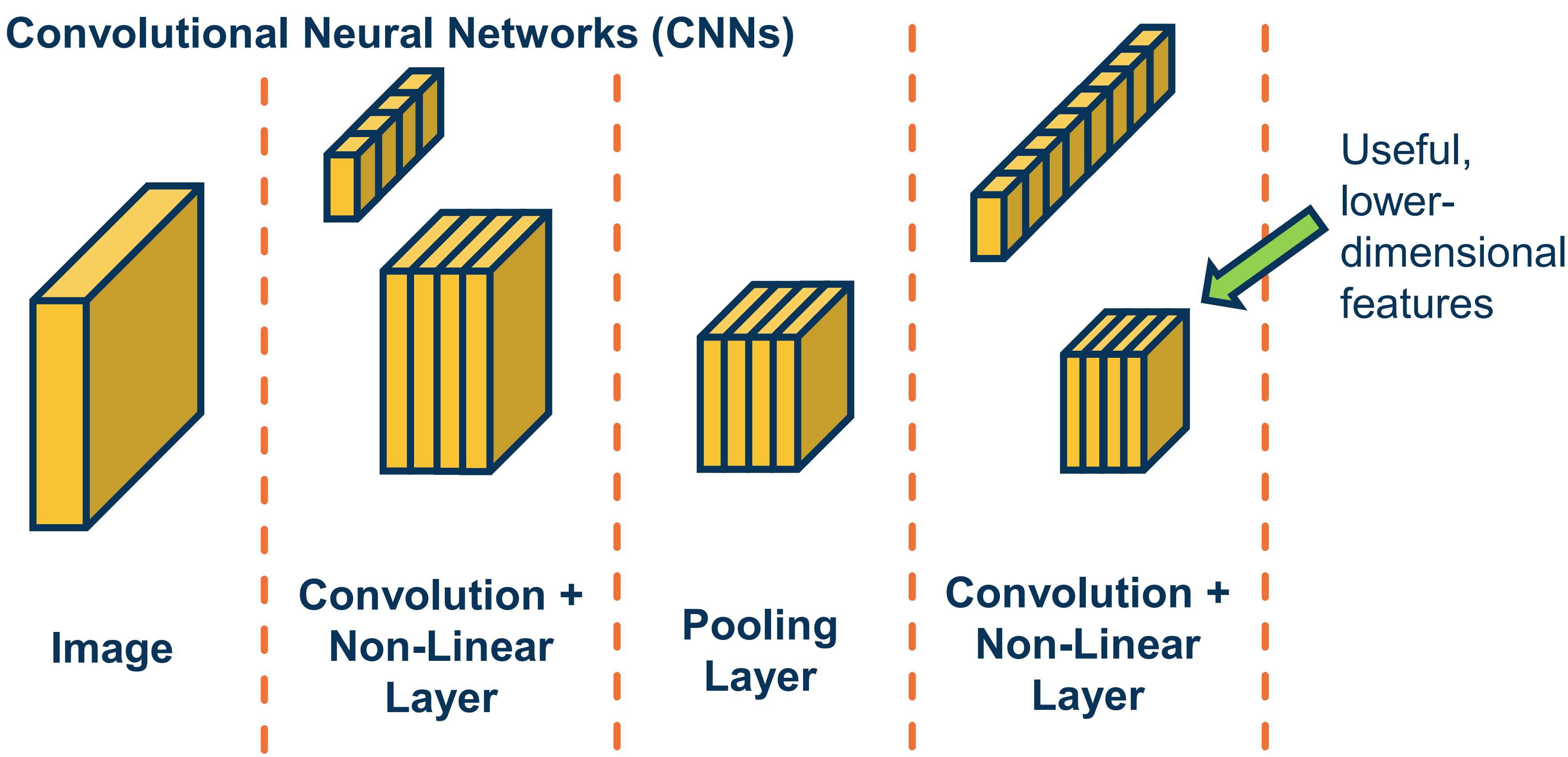
Convolution between upstream gradient and kernel!

(can implement by flipping kernel and cross- correlation)

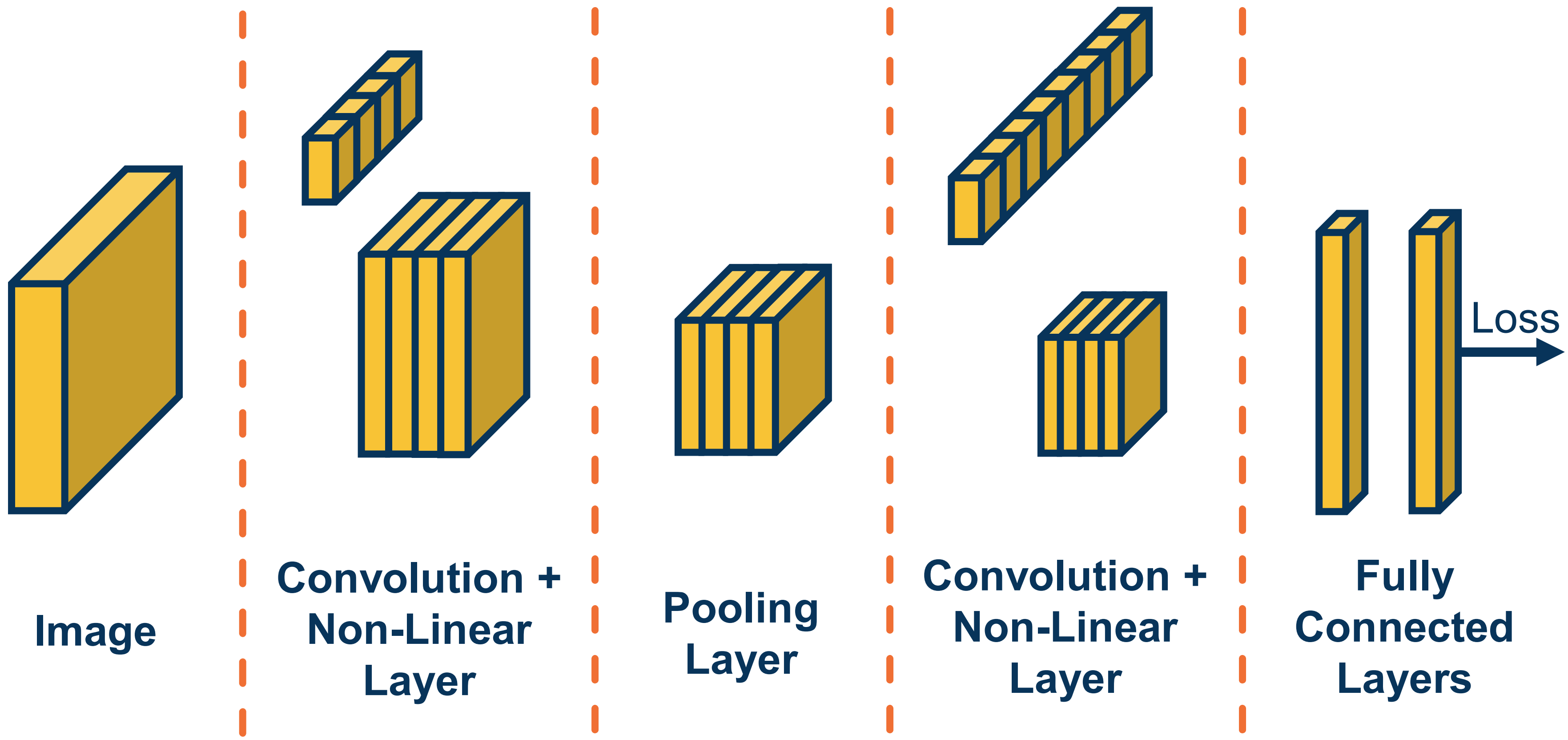
Again, all operations can be implemented via matrix multiplications (same as FC layer)!

Backwards is Convolution

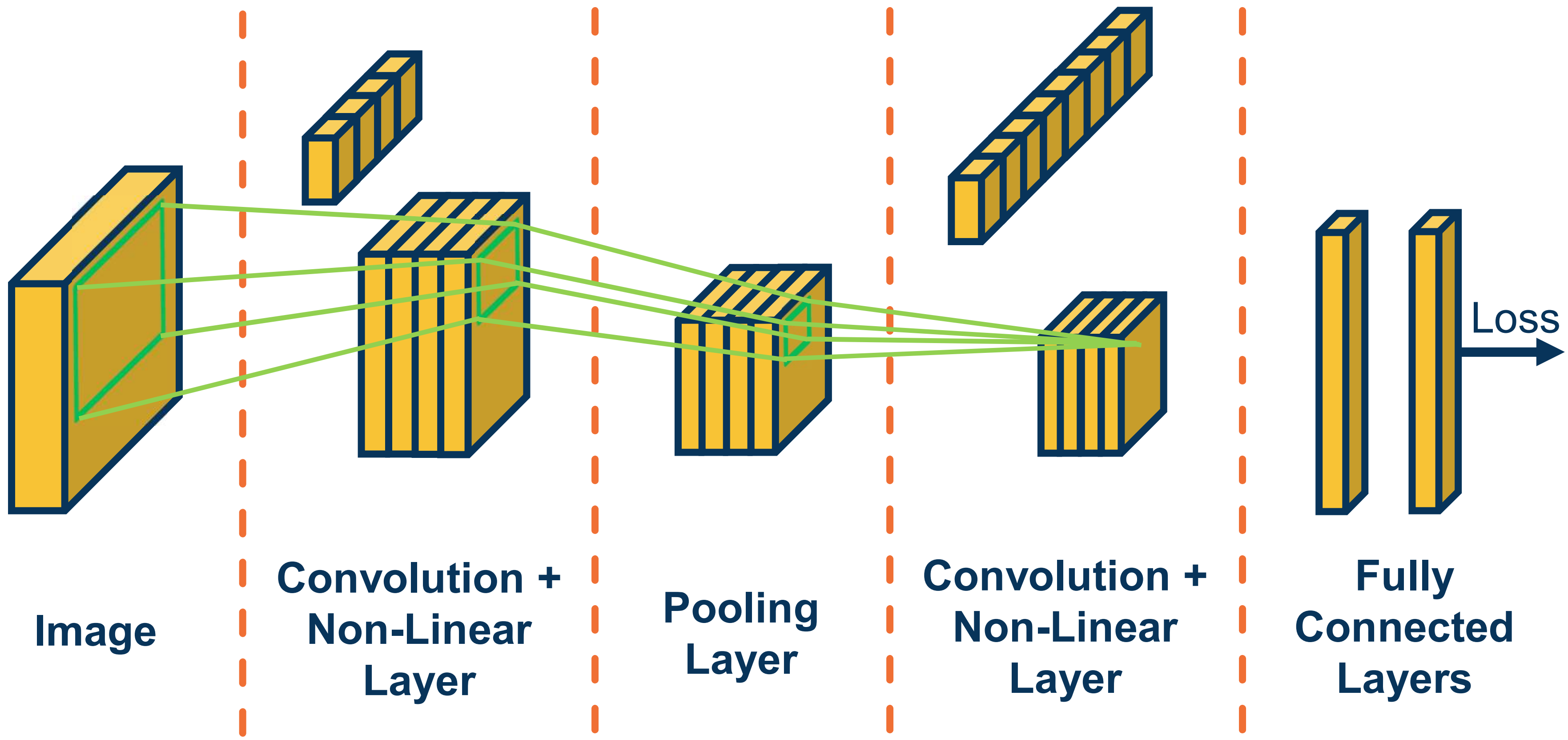
Alternating Convolution and Pooling



Georgia Tech



Adding a Fully Connected Layer



Receptive Fields

(not counting biases)

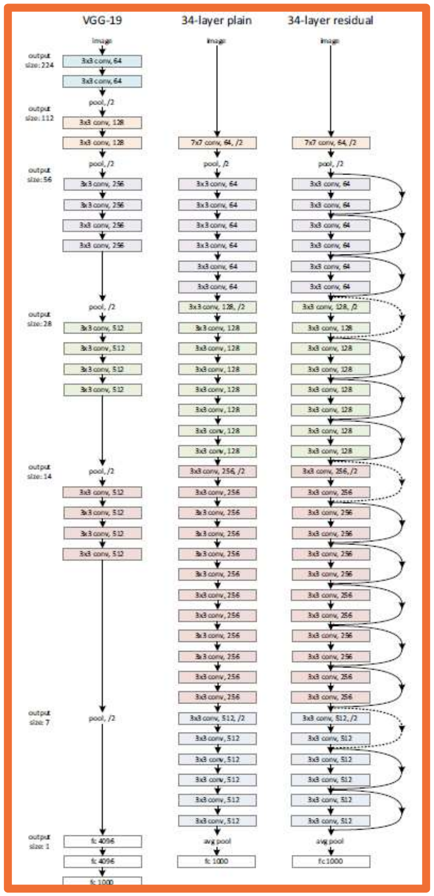
INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

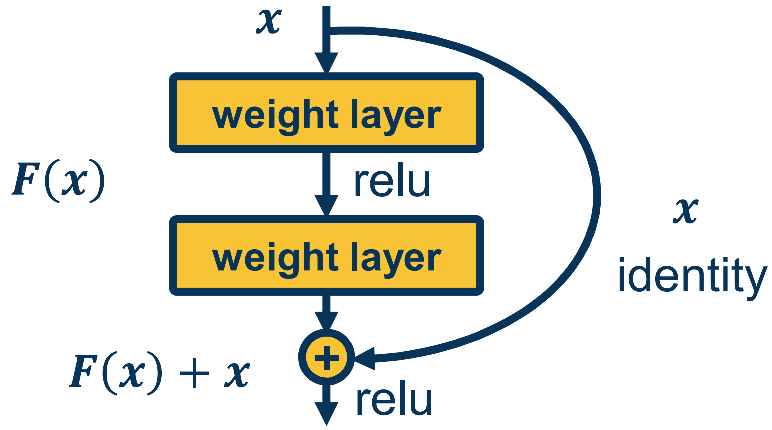
Most memory usage in convolution layers

Most parameters in FC layers

From: Simonyan & Zimmerman, Very Deep Convolutional Networks for Large-Scale Image Recognition
From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Parameters and Memory





Key idea: Allow information from a layer to propagate to any future layer (forward)

Same is true for gradients!

From: He et al., Deep Residual Learning for Image Recognition

Residual Blocks and Skip Connections

Step 3: (Continue to) train on new dataset

- **Finetune:** Update all parameters
- **Freeze** feature layer: Update only last layer weights (used when not enough data)

Replace last layer with new fully-connected for output nodes per new category

Finetuning on New Dataset

There is a large number of different low-labeled settings in DL research

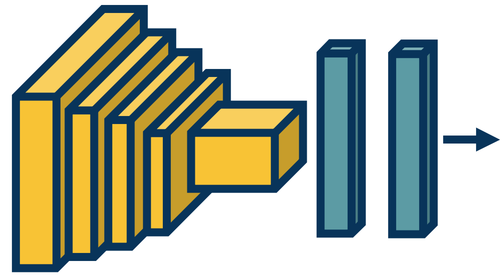
Setting	Source	Target	Shift Type
Semi-supervised	Single labeled	Single unlabeled	None
Domain Adaptation	Single labeled	Single unlabeled	Non-semantic
Domain Generalization	Multiple labeled	Unknown	Non-semantic
Cross-Task Transfer	Single labeled	Single unlabeled	Semantic
Few-Shot Learning	Single labeled	Single few-labeled	Semantic
Un/Self-Supervised	Single unlabeled	Many labeled	Both/Task

Non-Semantic Shift

Semantic Shift

Visualization of Neural Networks

Given a **trained** model, we'd like to understand what it learned.



Weights

Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Zeiler & Fergus, 2014

Activations

Gradients

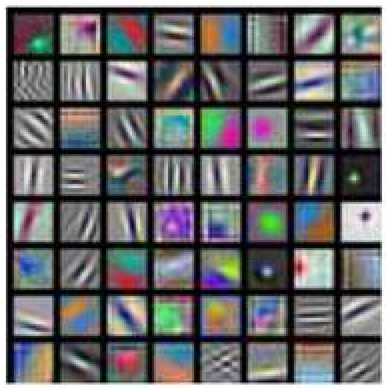
Simonyan et al, 2013

Robustness

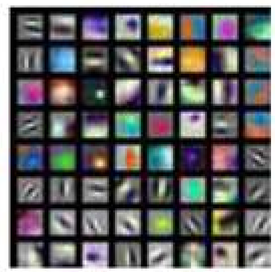
Hendrycks & Dietterich, 2019

FC Layer: Reshape weights for a node back into size of image, scale 0-255

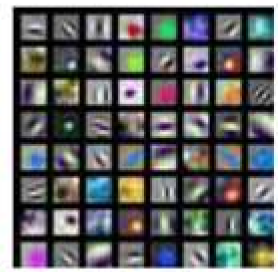
Conv layers:
For each kernel, scale values from 0-255 and visualize



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

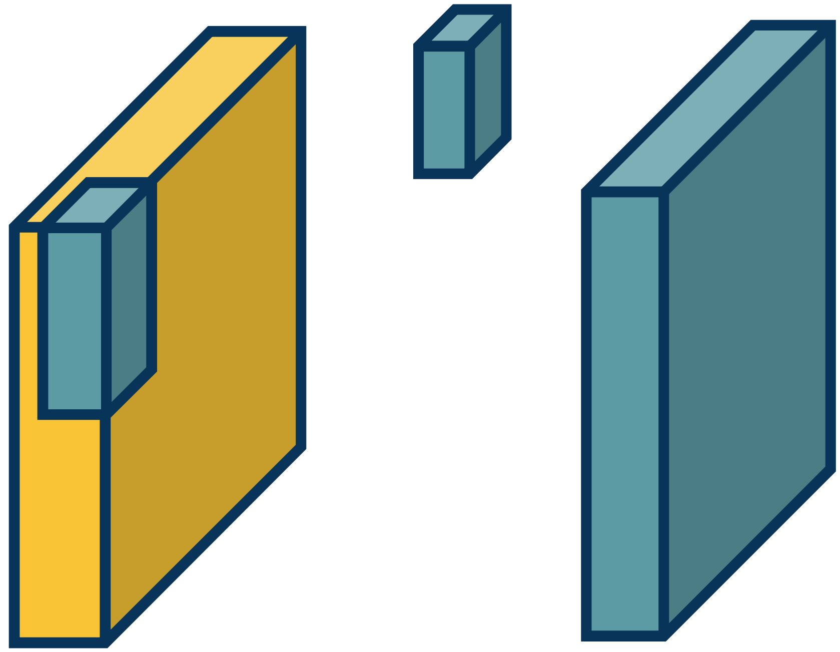
Problem: 3x3 filters difficult to interpret!

Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Visualizing Weights

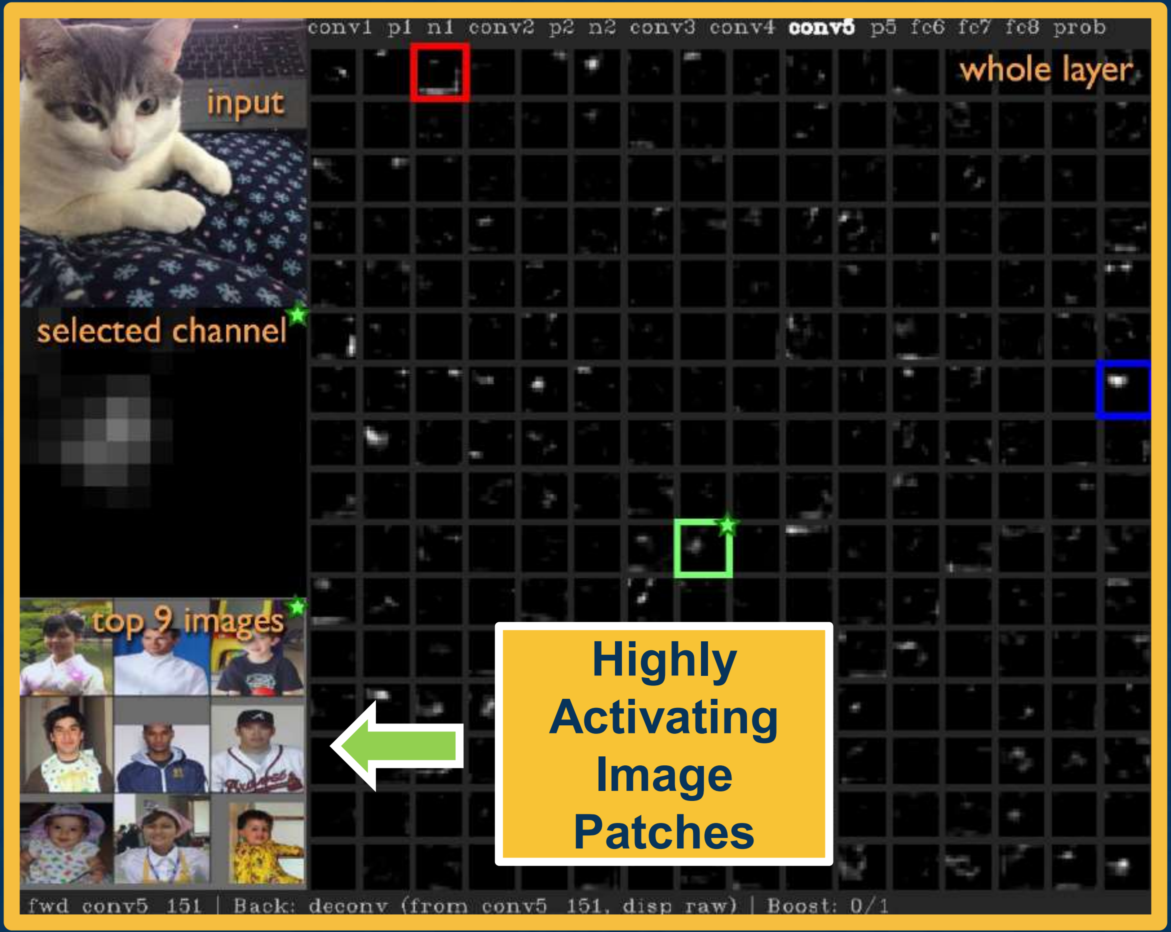
We can also produce **visualization output (aka activation/filter) maps**

These are **larger** early in the network.



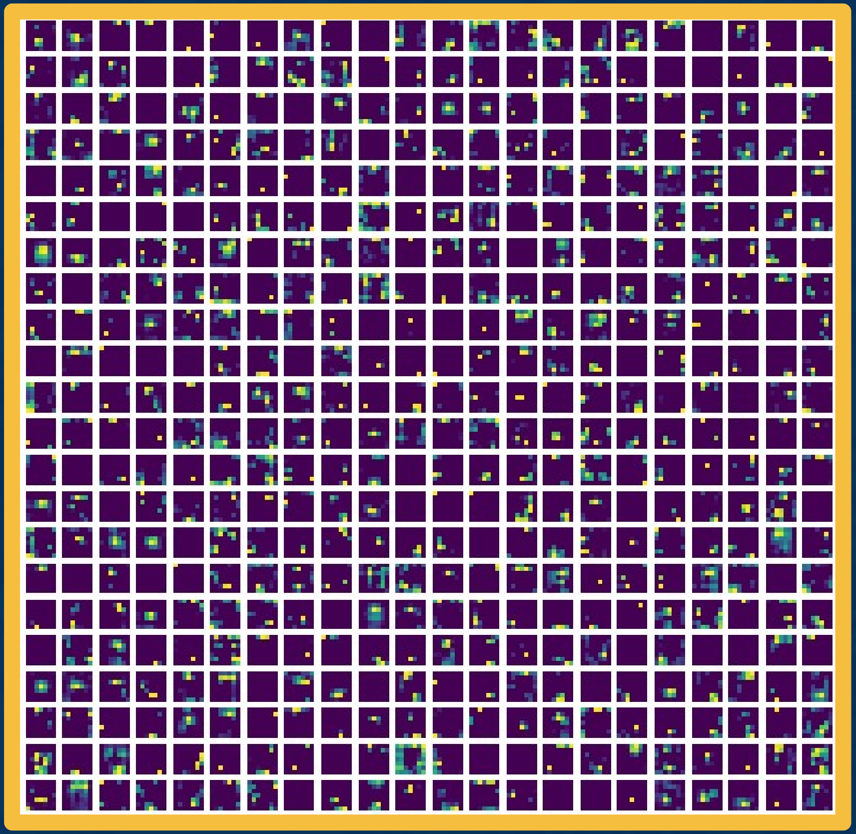
Visualizing Output Maps

Visualizing Output Maps



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

Activations – Small Output Sizes

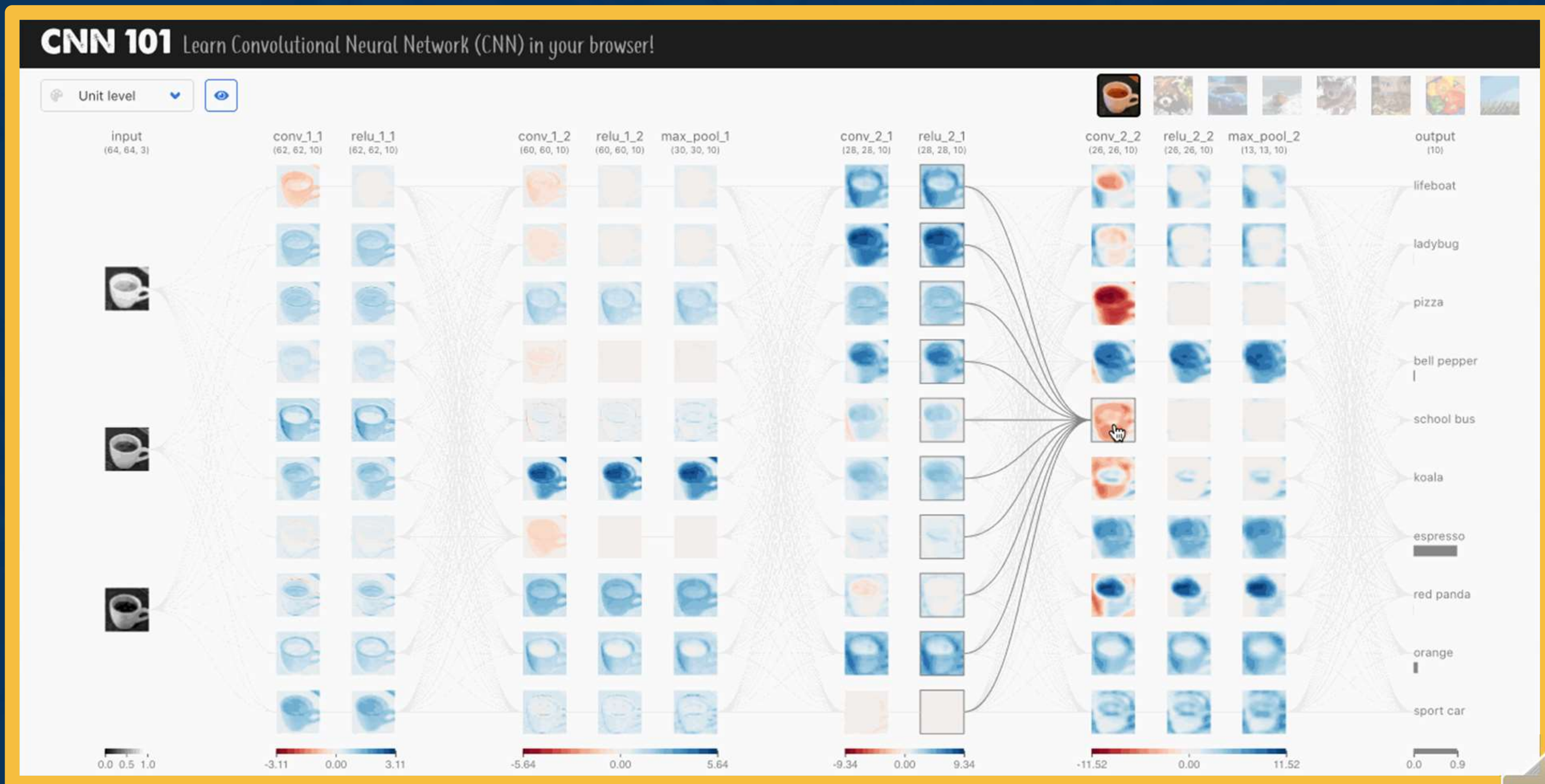


Activations of last conv layer in VGG network

Problem: Small conv outputs also hard to interpret

Georgia Tech

CNN101 and CNN Explainer



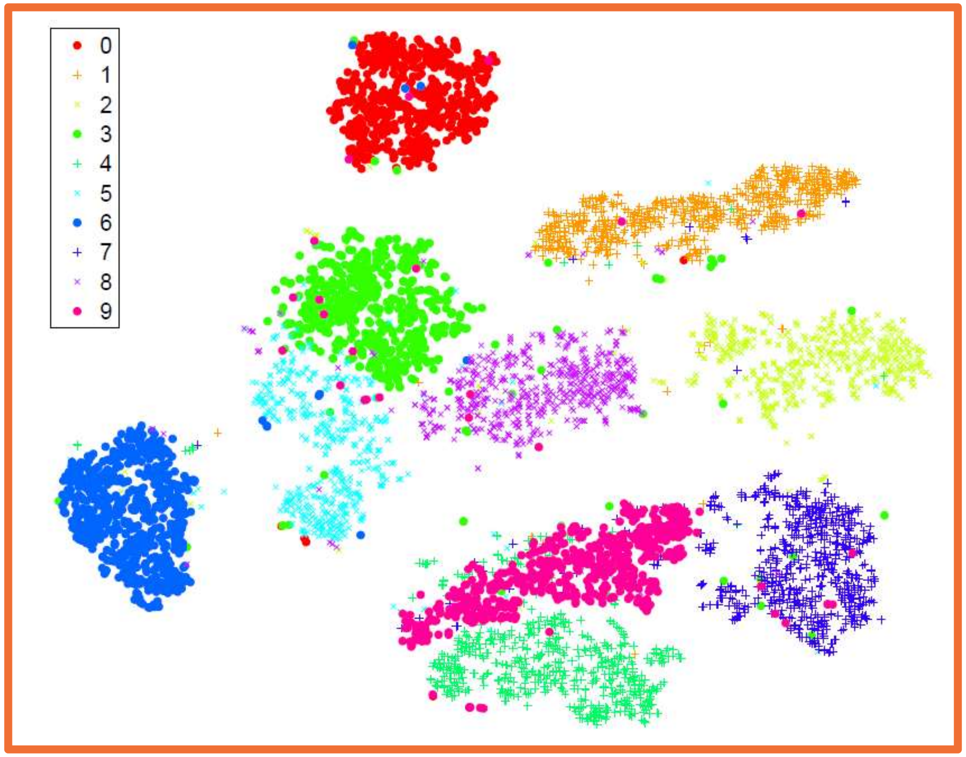
https://poloclub.github.io/cnn-explainer/ https://fredhohman.com/papers/cnn101

We can take the activations of any layer (FC, conv, etc.) and **perform dimensionality reduction**

- Often reduce to two dimensions for plotting
- E.g. using Principle Component Analysis (PCA)

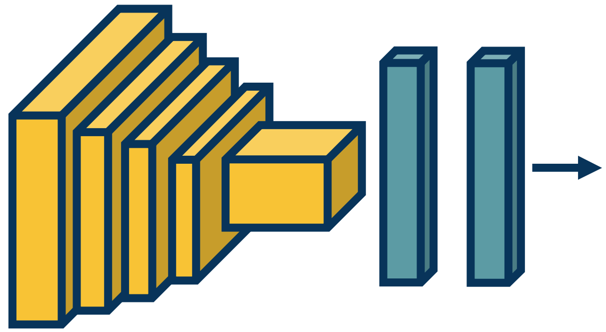
t-SNE is most common

- Performs non-linear mapping to preserve pair-wise distances



Van der Maaten & Hinton, "Visualizing Data using t-SNE", 2008.

Dimensionality Reduction: t-SNE



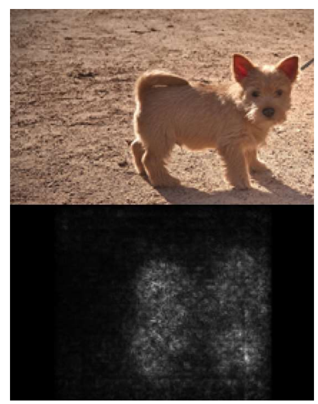
Weights

Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Zeiler & Fergus, 2014

Activations

Gradients



Simonyan et al, 2013

Robustness

Hendrycks & Dietterich, 2019

Visualizing Neural Networks

Summary & Caveats

While these methods provide **some** visually interpretable representations, they can be misleading or uninformative (Adebayo et al., 2018)

Assessing interpretability is difficult

- Requires **user studies** to show **usefulness**
- E.g. they allow a user to predict mistakes beforehand

Neural networks learn **distributed representation**

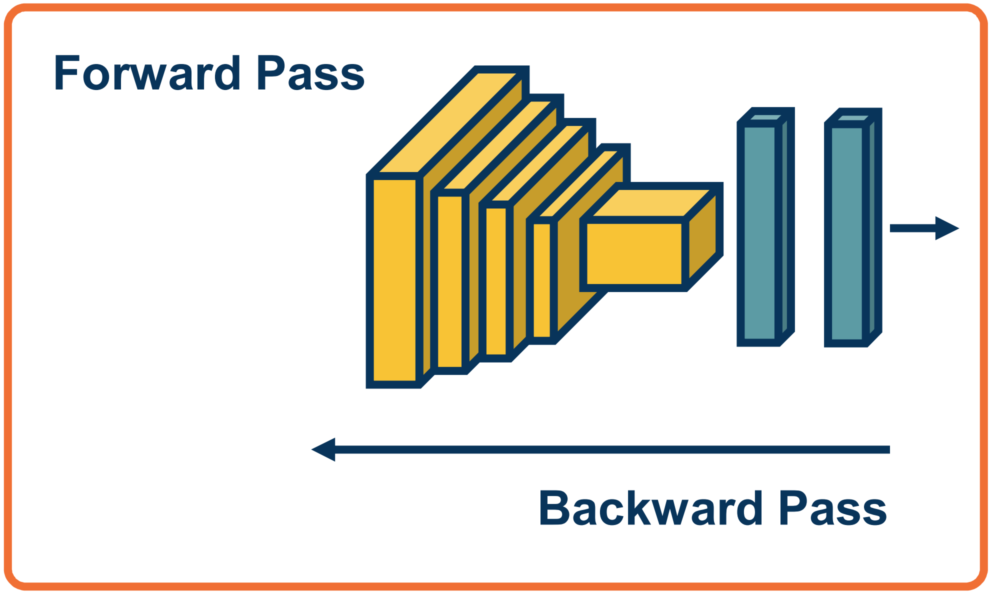
- (no one node represents a particular feature)
- This makes interpretation difficult

Adebayo et al., "Sanity Checks for Saliency Maps", 2018.

Gradient-Based Visualizations

Backwards pass gives us **gradients** for all layers: How the loss changes as we change different parts of the input

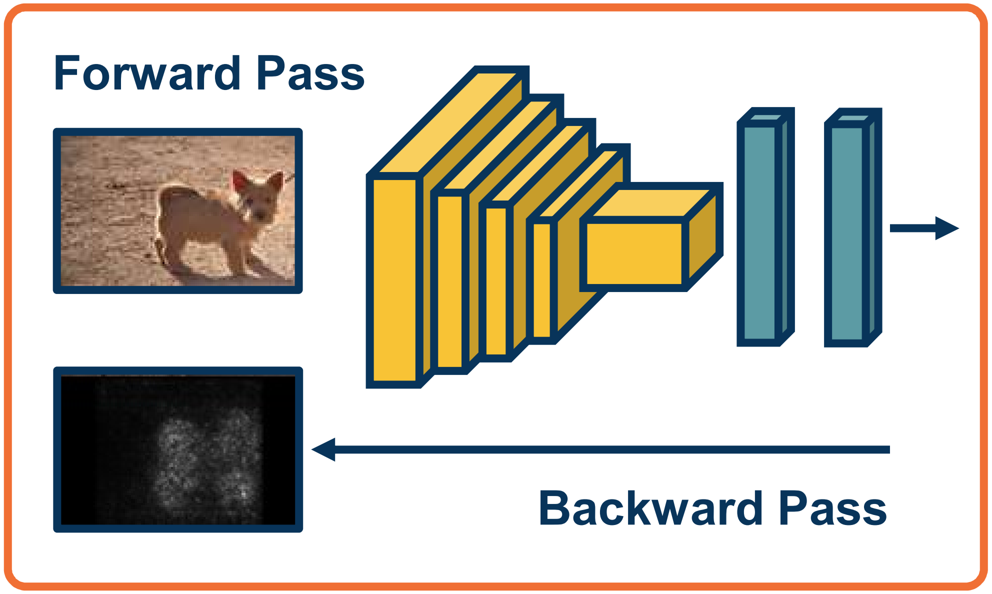
This can be **useful not just for optimization**, but also to understand what was learned



- Gradient of **loss** with respect to **all layers** (including input!)
- Gradient of **any layer** with respect to **input** (by cutting off computation graph)

Idea: We can backprop to the image

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



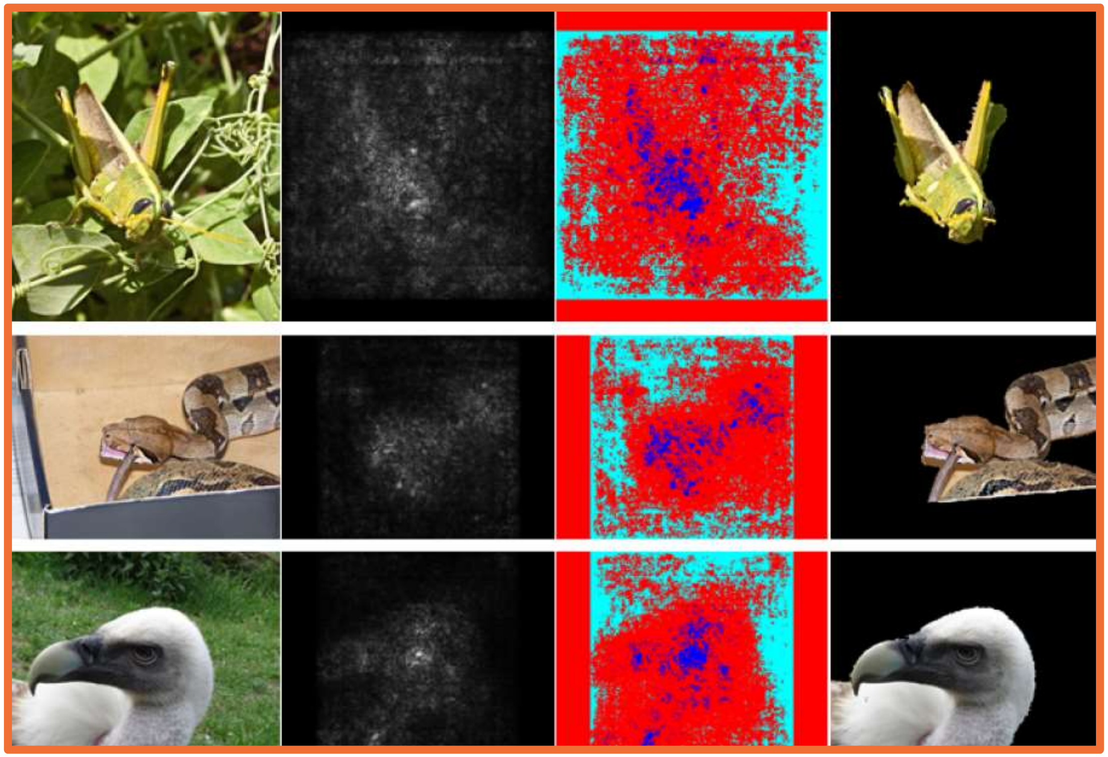
In practice:

- 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

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

Object Segmentation for Free!

Can be used to **detect dataset bias**

- E.g. snow used to misclassify as wolf

Incorrect predictions also informative

(a) Husky classified as wolf (b) Explanation

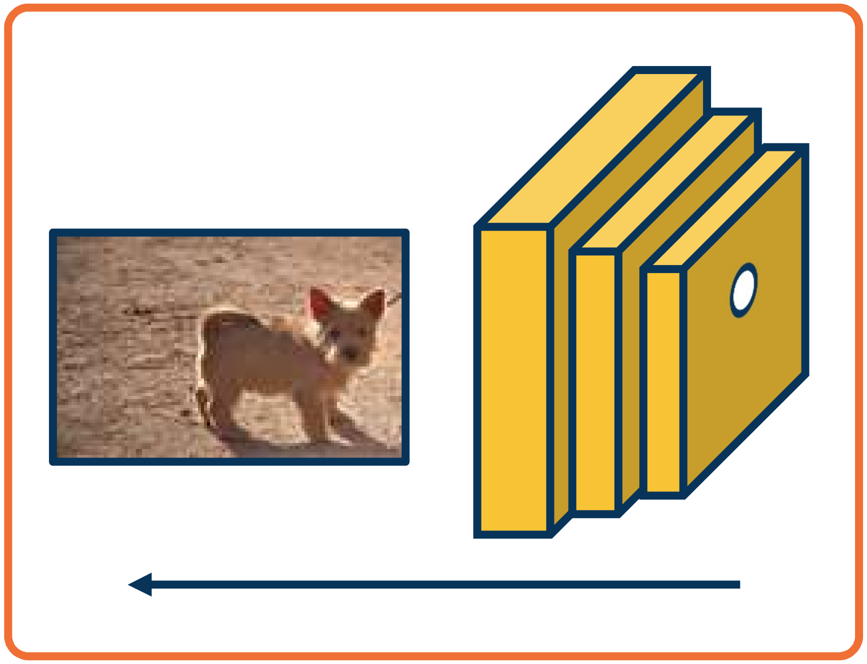
From: Ribeiro et al., "Why Should I Trust You?": Explaining the Predictions of Any Classifier

Detecting Bias

Rather than loss or scores, we can pick a neuron somewhere deep in the network and compute gradient of **activation** with respect to input

Steps:

- Pick a neuron
- Find gradient of its activation w.r.t. input image
- Can first find highest activated image patches using its corresponding neuron (based on receptive field)



From: Ribeiro et al., "Why Should I Trust You?": Explaining the Predictions of Any Classifier

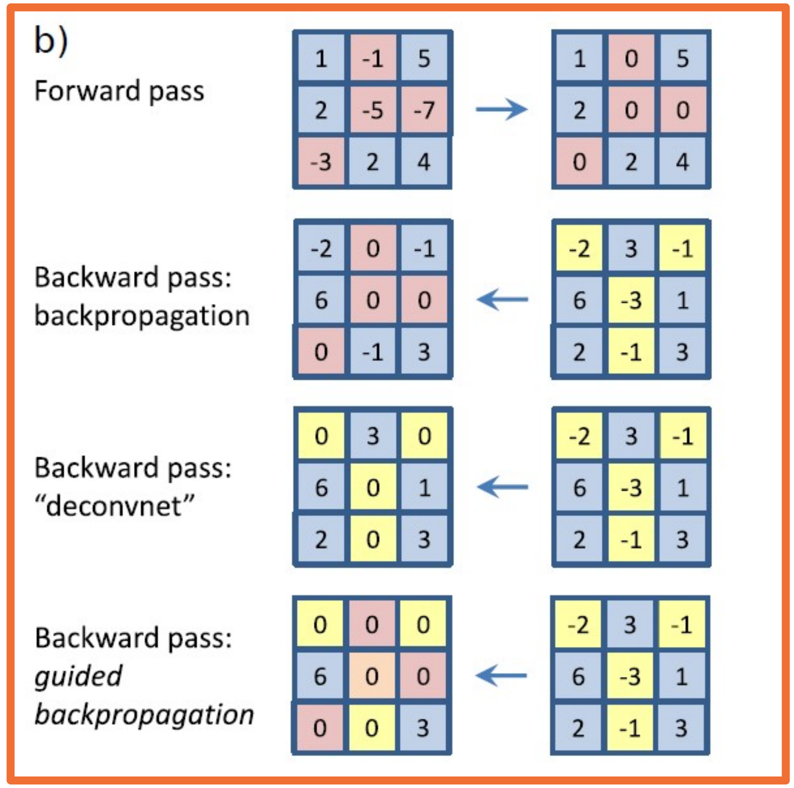
Gradient of Activation with respect to Input

Normal backprop not always best choice

Example: You may get parts of image that **decrease** the feature activation

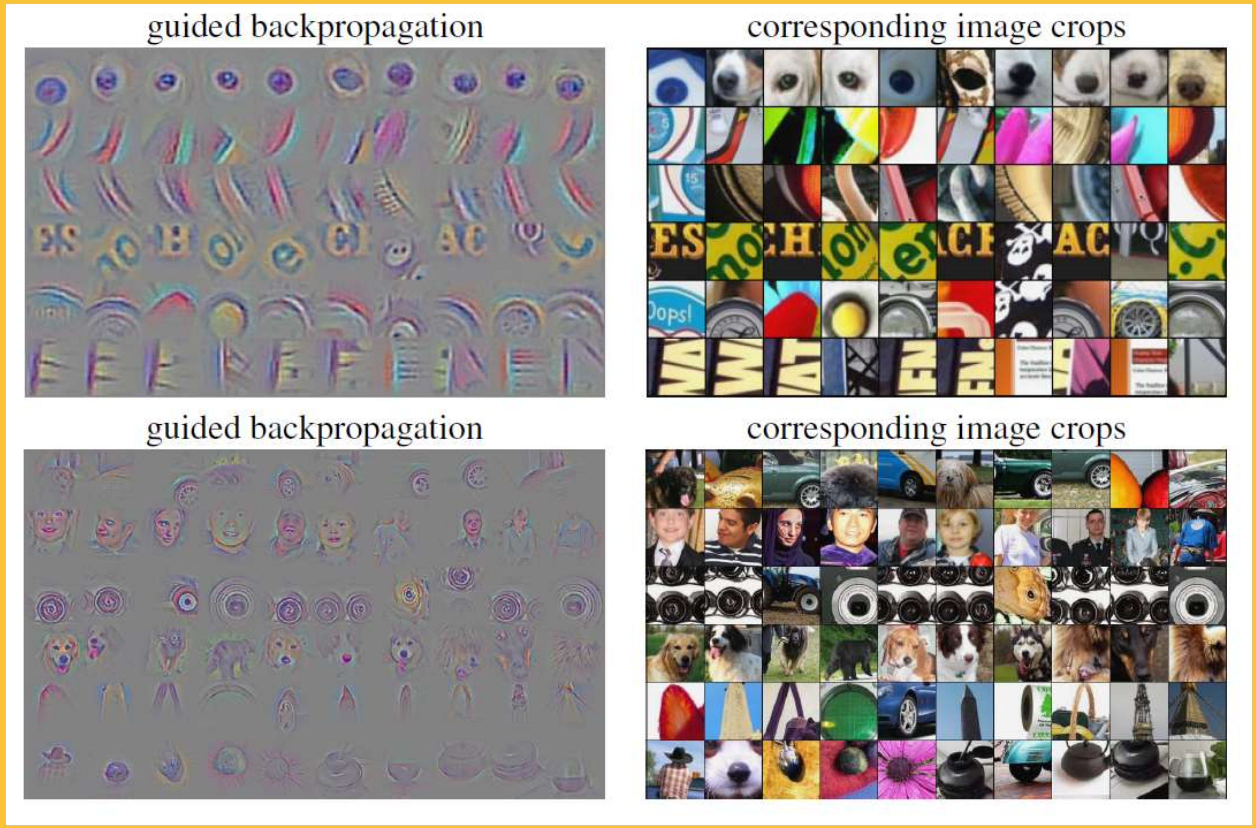
- There are probably lots of such input pixels

Guided backprop can be used to improve visualizations



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

Guided Backprop Results



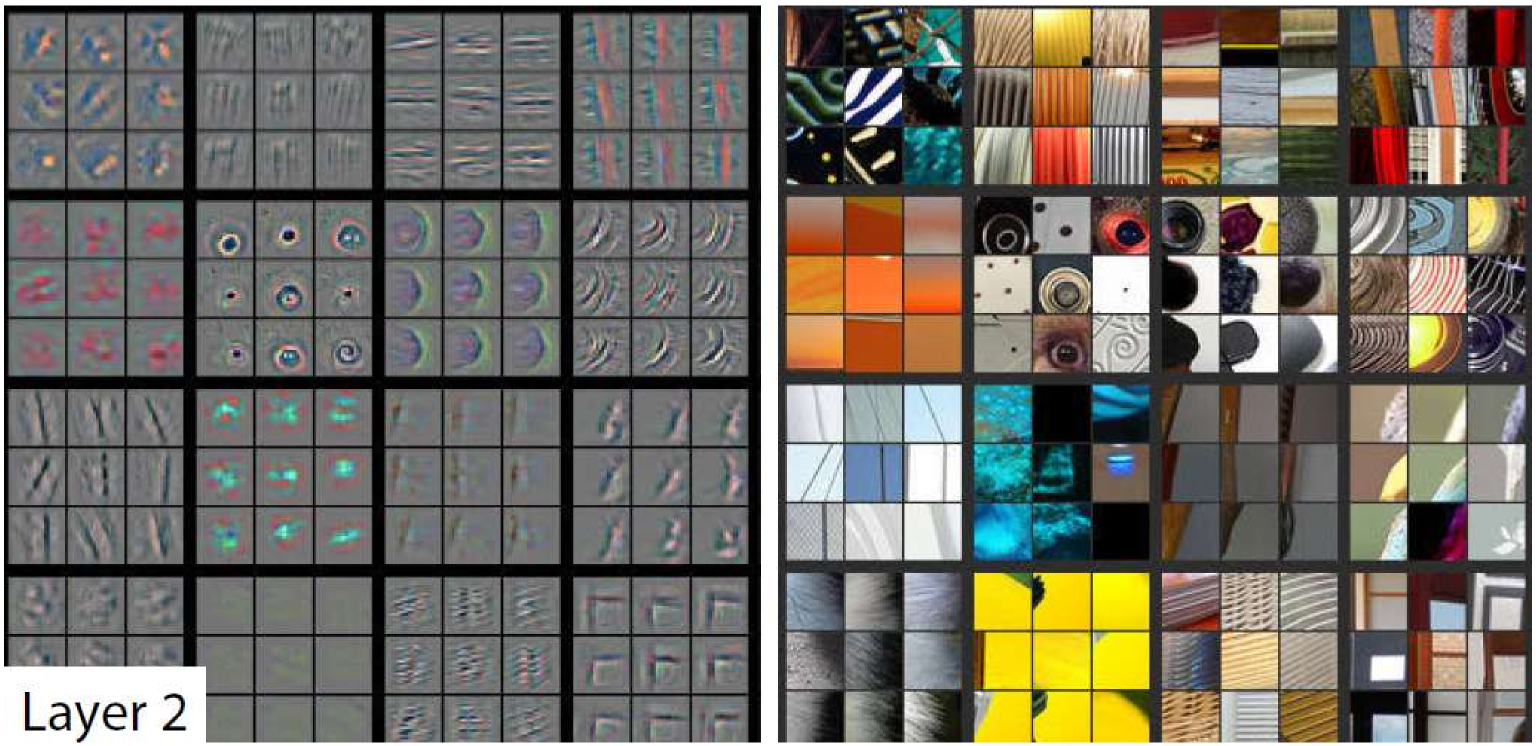
From: Springenberg et al., “Striving For Simplicity: The All Convolutional Net"

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.

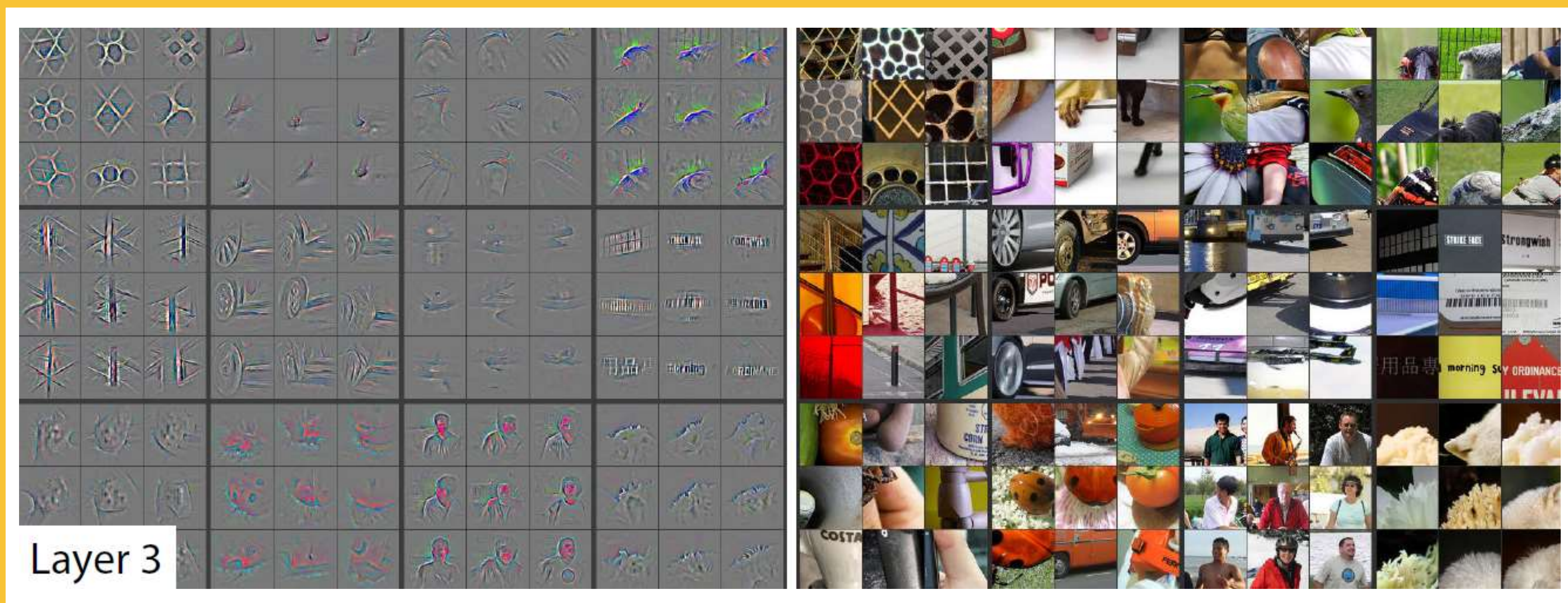
VGG Layer-by-Layer Visualization



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

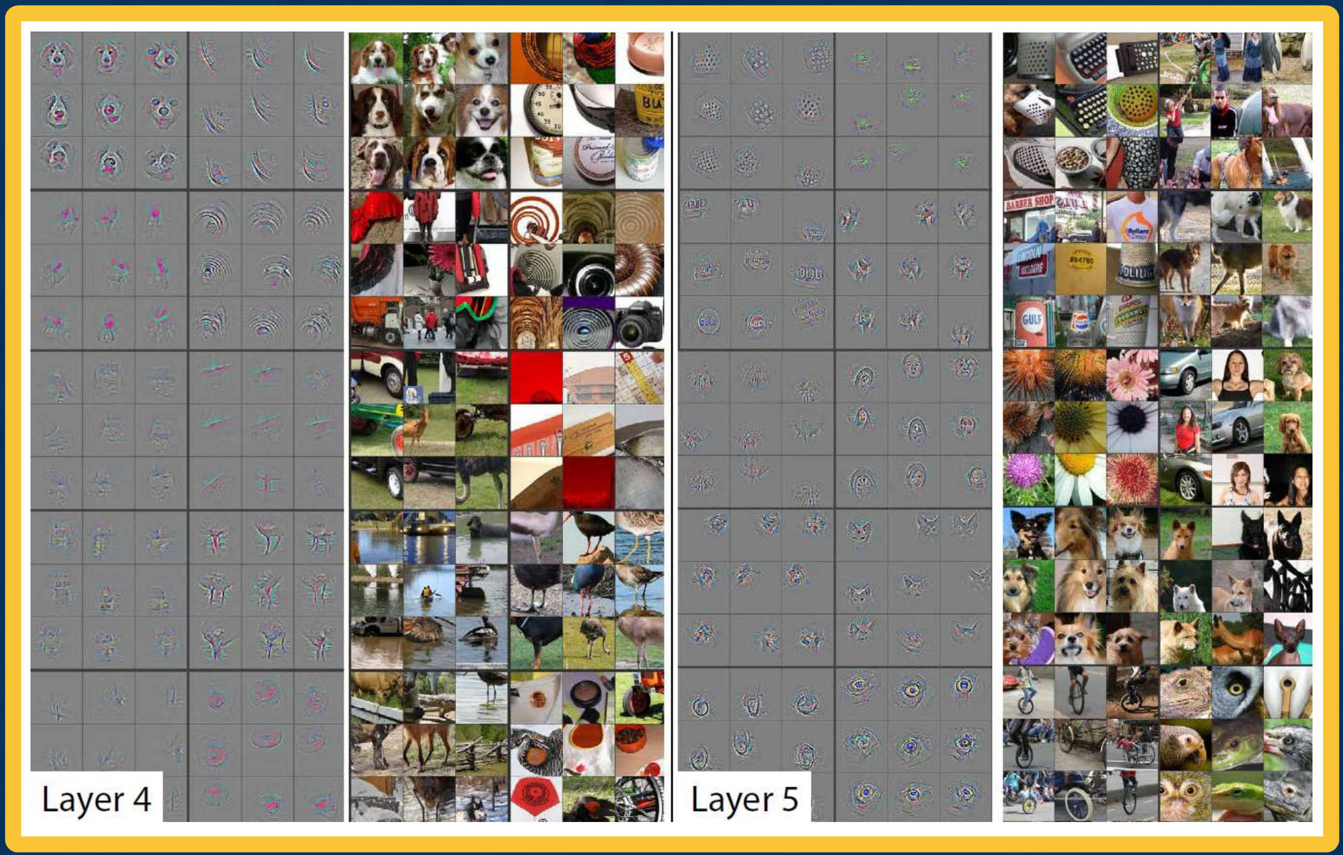
Georgia Tech

VGG Layer-by-Layer Visualization



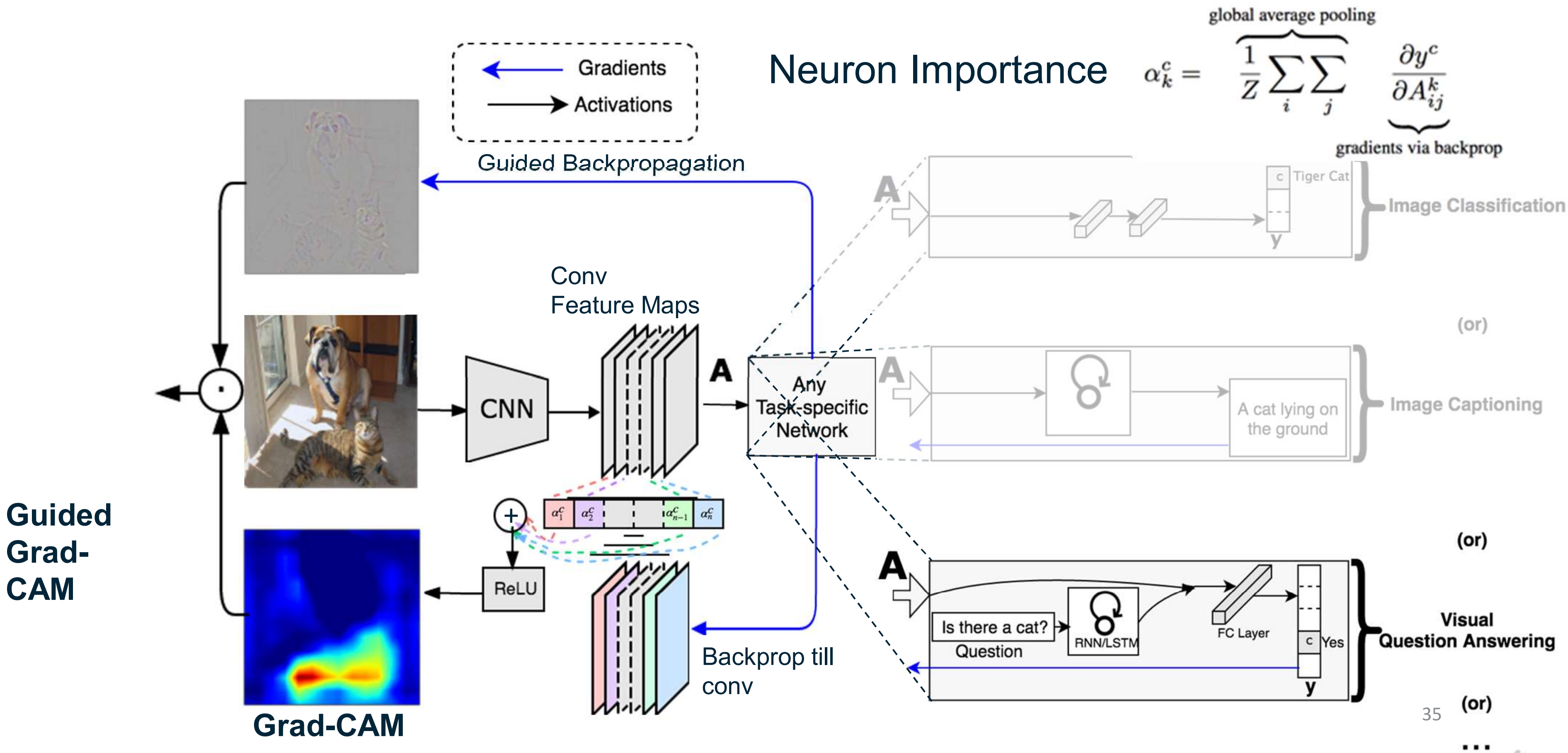
From: "Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014.

VGG Layer-by-Layer Visualization



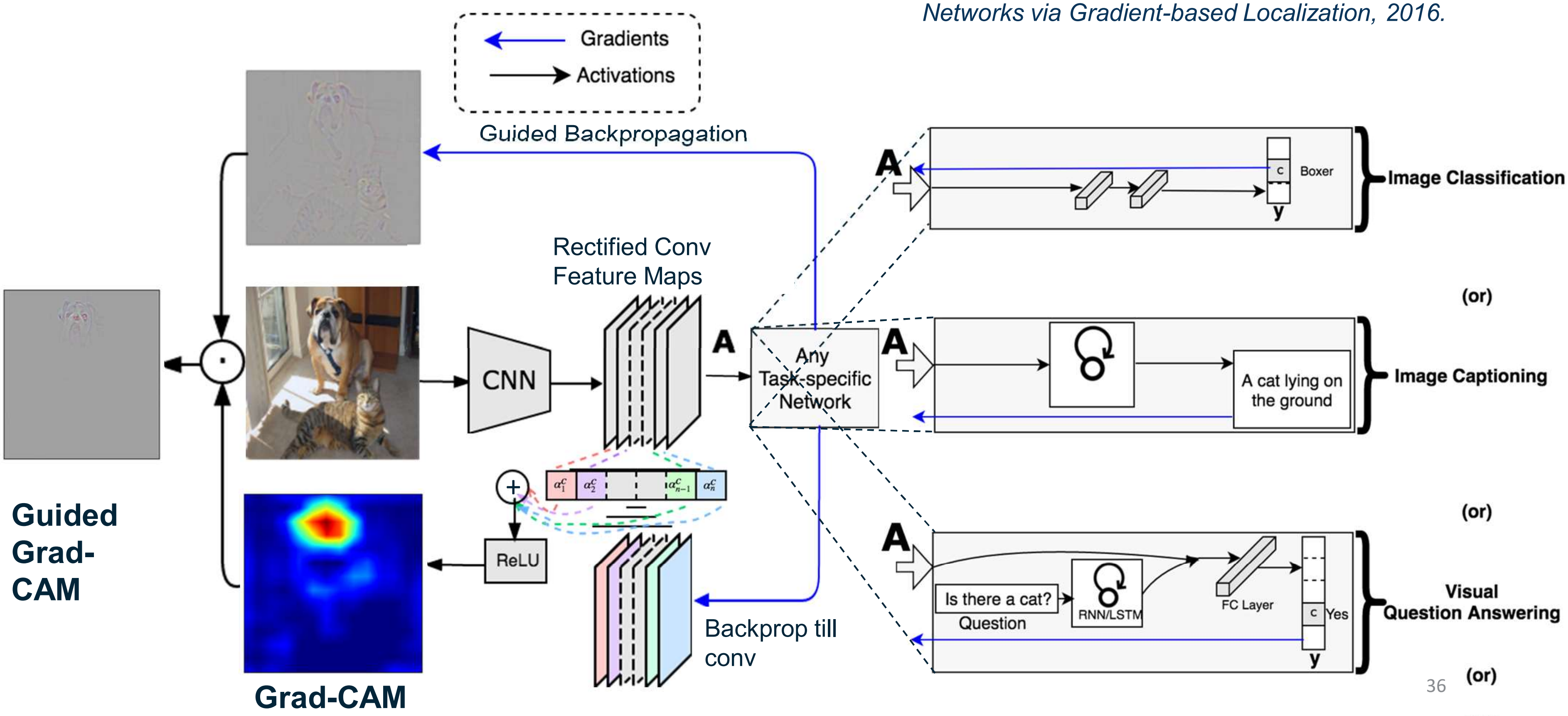
From: "Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014.

GradCAM

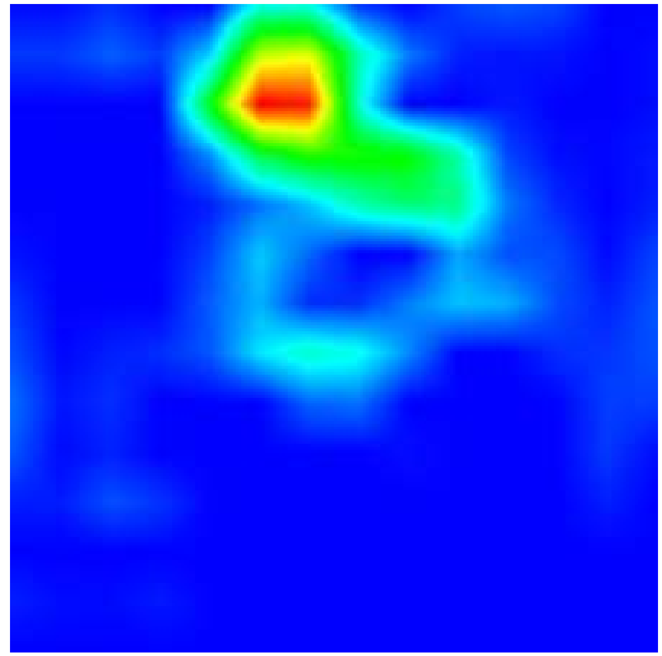


Selfvaraju et al., Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization, 2016.

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



Grad-CAM

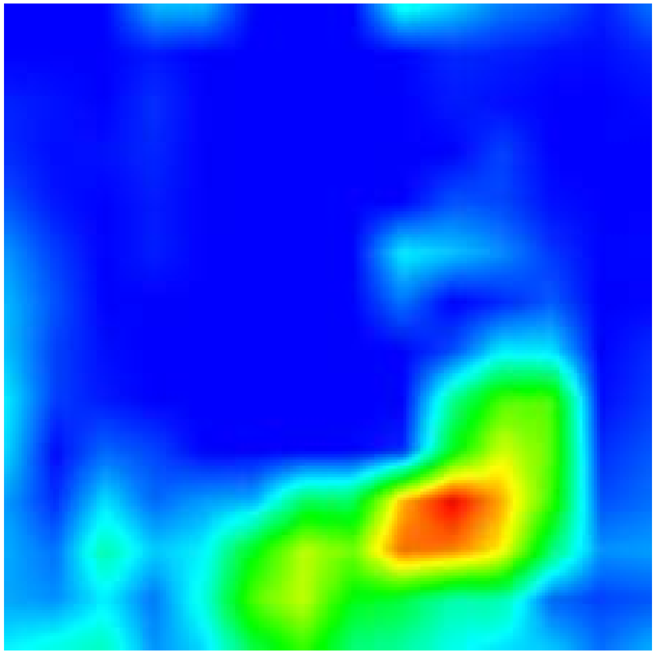


What animal is in this picture? Dog

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

Grad-CAM

Georgia Tech



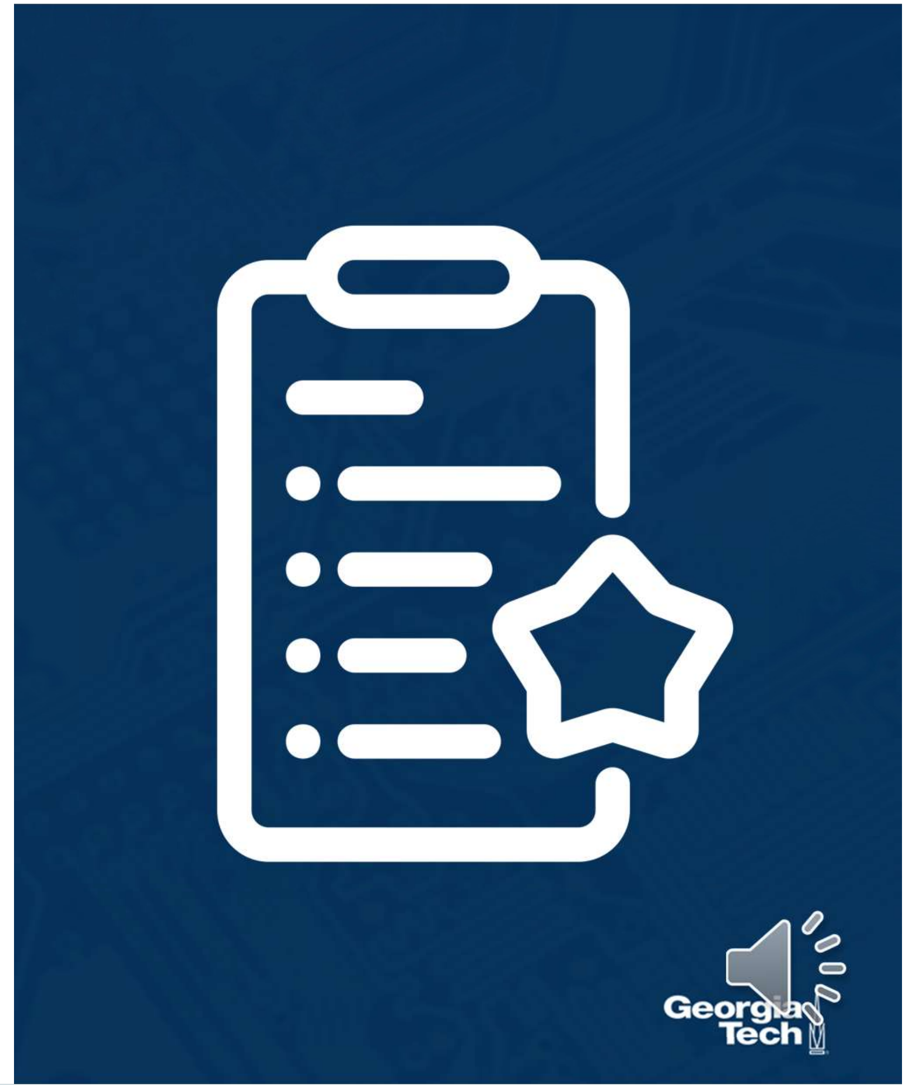
What animal is in this picture? Cat

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

Grad-CAM

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



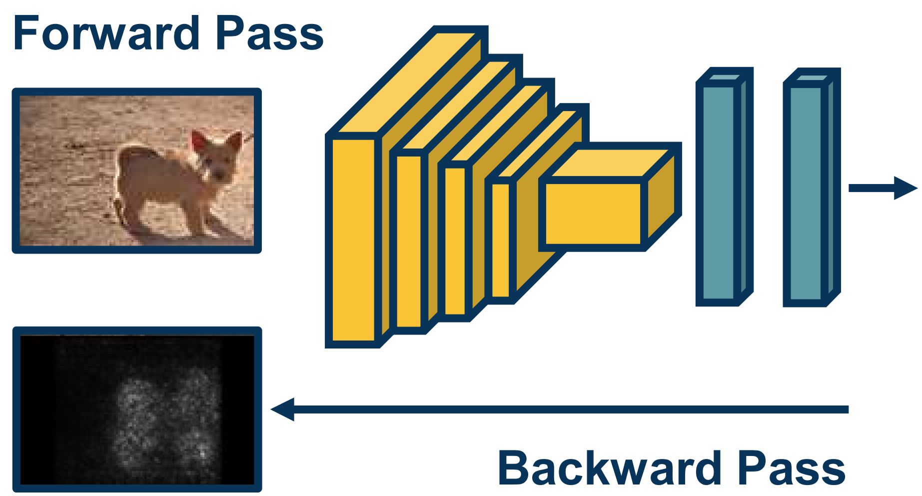
Optimizing the Input Images

Georgia Tech

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

Optimizing the Image

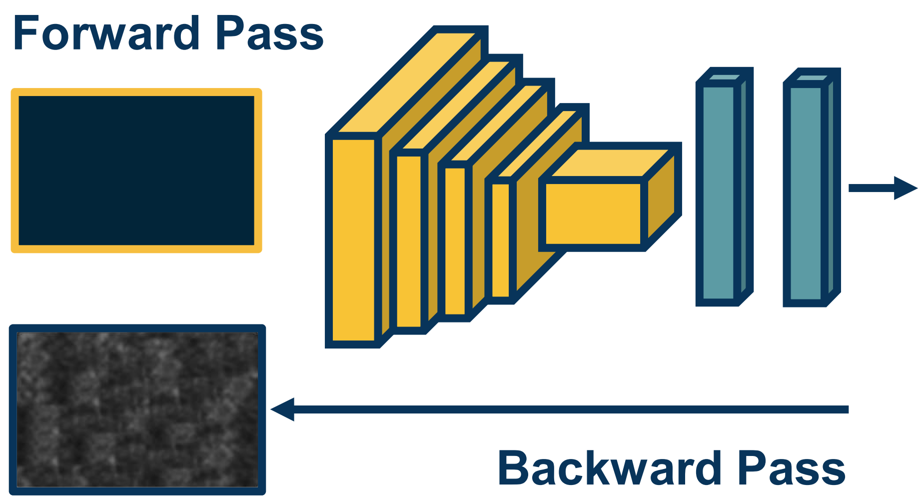
We can perform **gradient ascent** on image

- 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

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



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

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

Gradient Ascent on the Scores

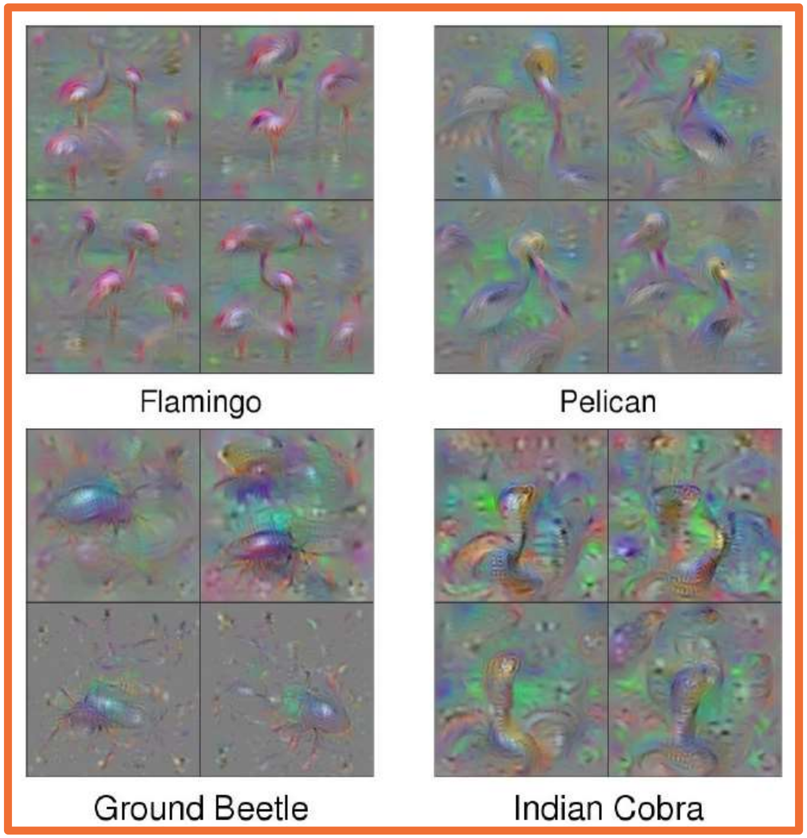
Example Images

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**:

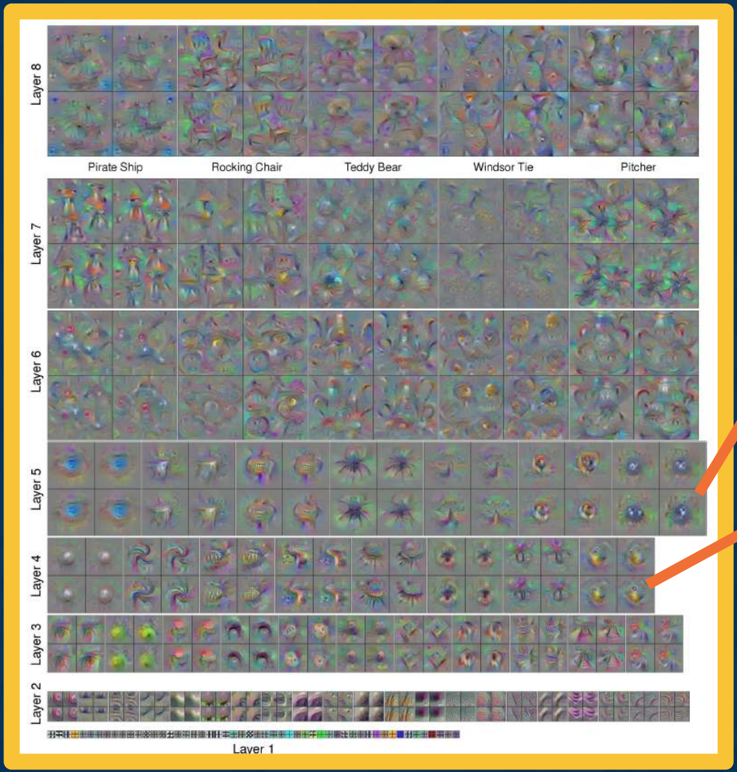
- Clipping of small values & gradients
- Gaussian blurring

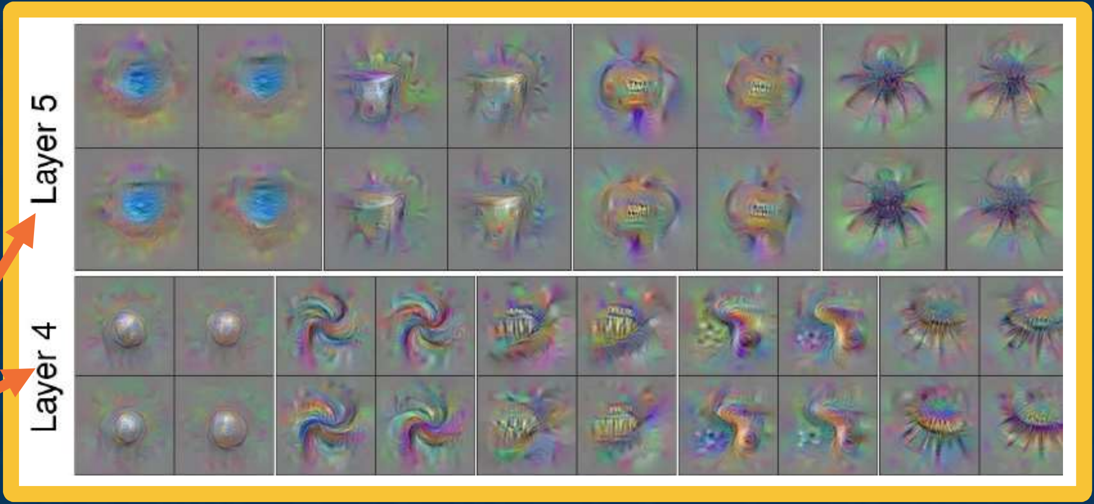


From: Yosinski et al., "Understanding Neural Networks Through Deep Visualization", 2015

Example Images

Improved Results



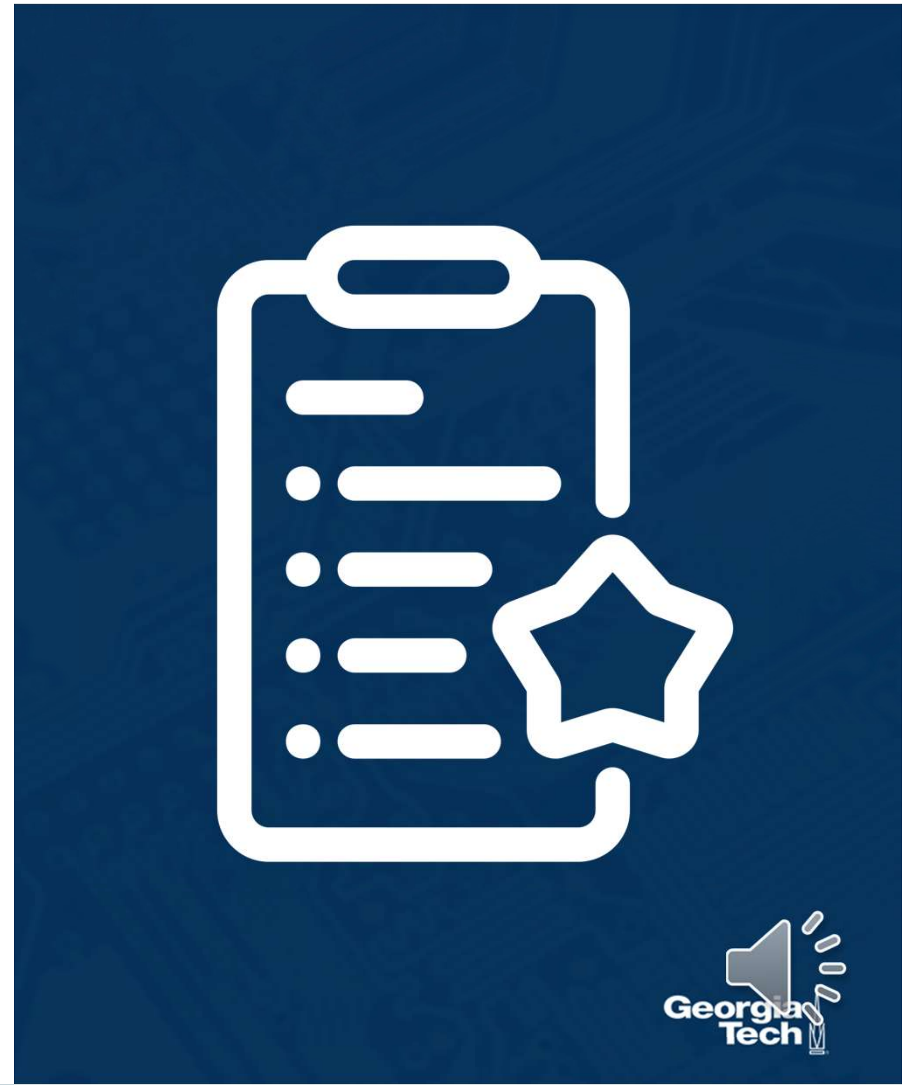


From: Yosinski et al., "Understanding Neural Networks Through Deep Visualization", 2015

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**



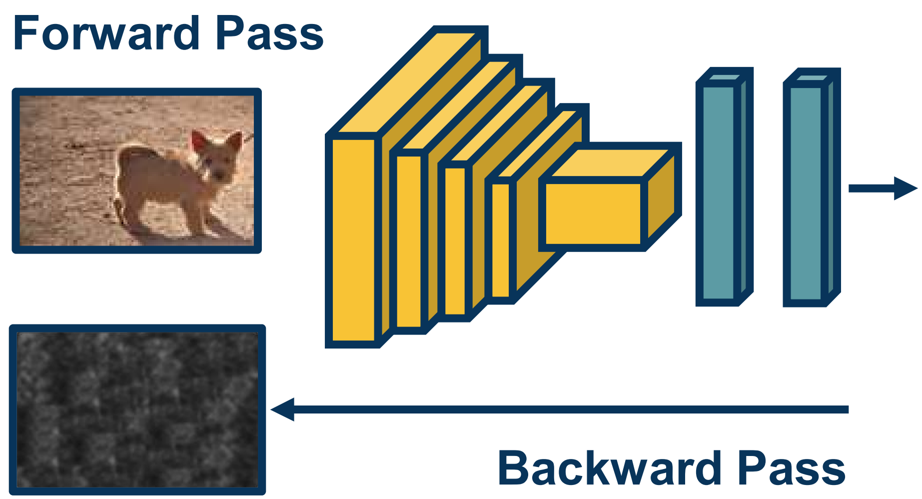
Testing Robustness

- 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!

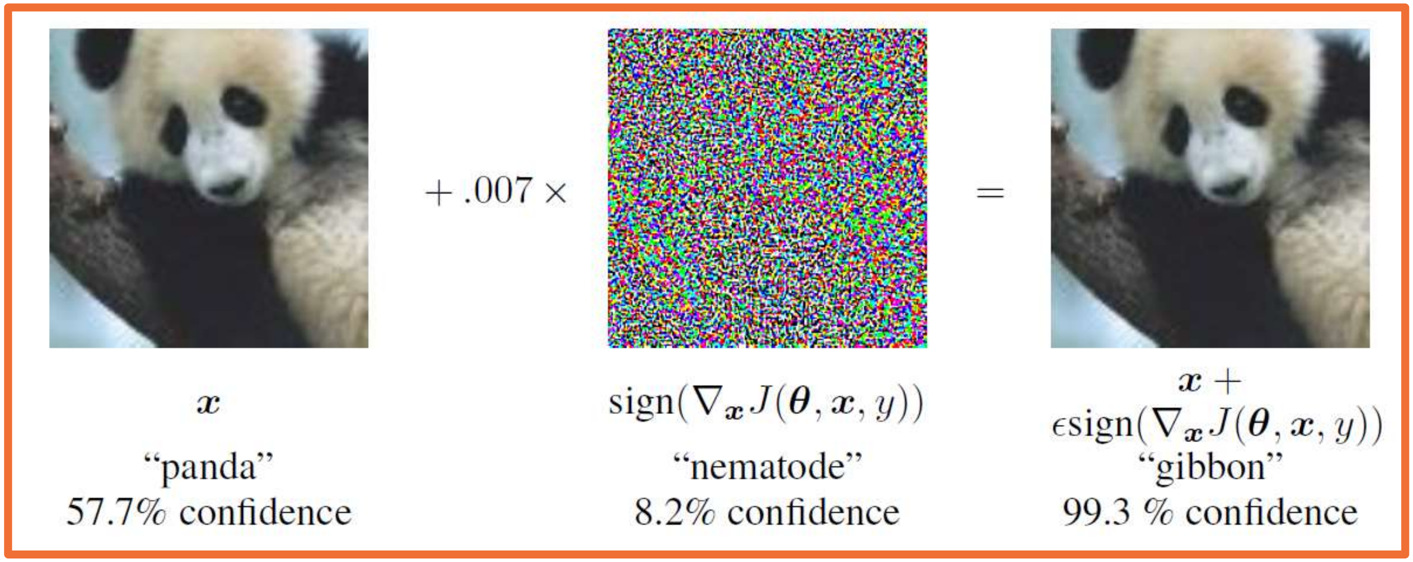
$$\mathbf{argmax}\, S_c(I) - \lambda \left|\left|I\right|\right|_2^2$$

where $c = cat$



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

Gradient Ascent on the Scores



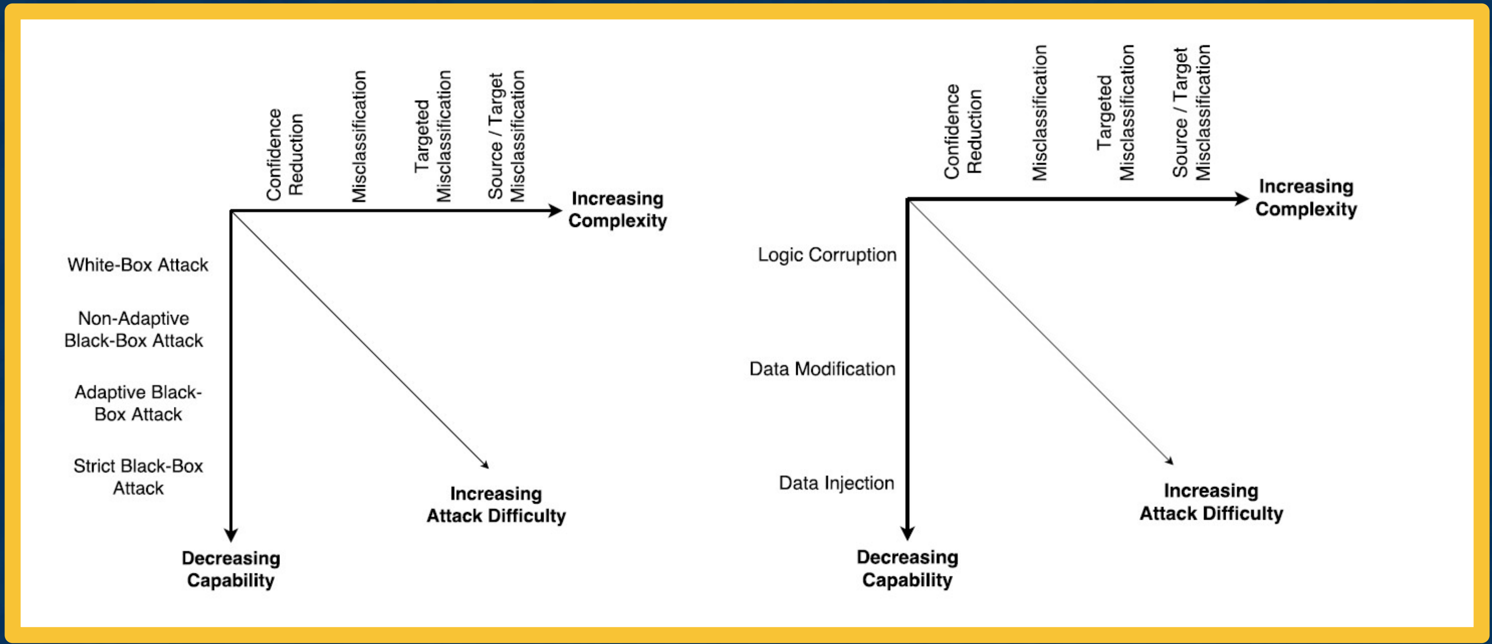
Note this problem is not specific to deep learning!

- 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

Example of Adversarial Noise

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 dversarial Attacks/Defenses

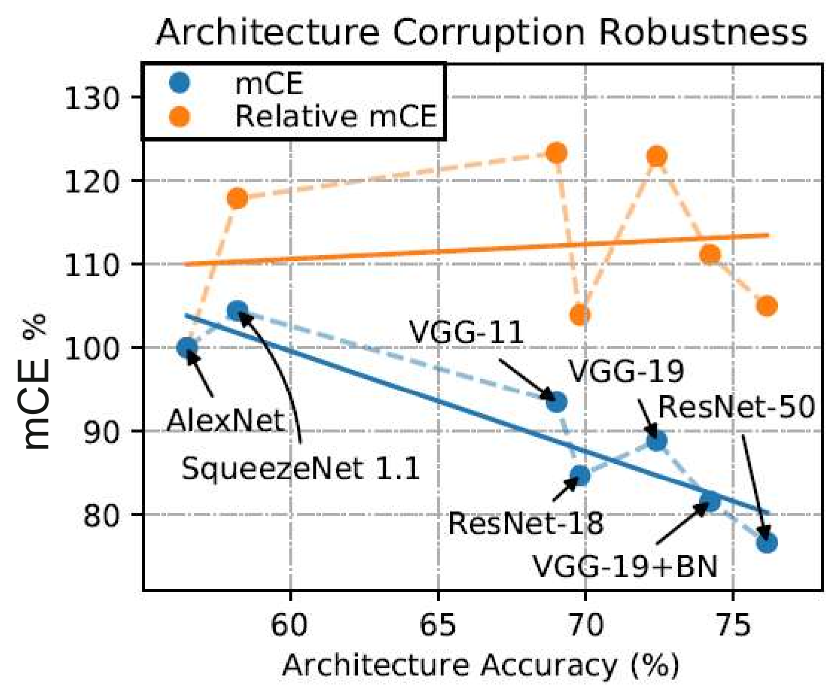
Similar to other security-related areas, it’s an active **cat-and-mouse game**

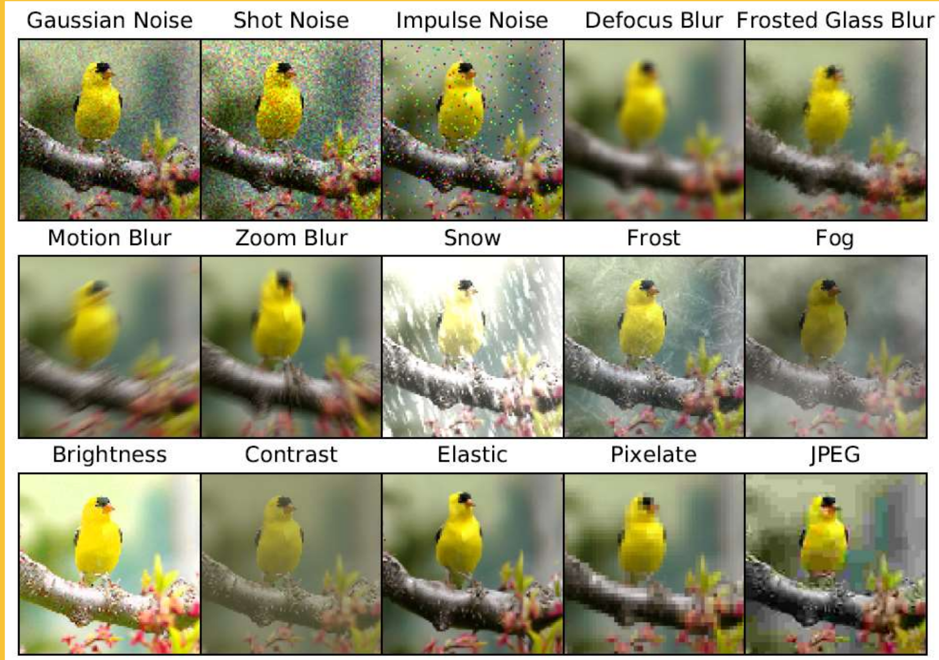
Several defenses such as:

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

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

Other Forms of Robustness Testing





$$\mathrm{CE}_c^f = \left(\sum_{s=1}^{5} E_{s,c}^f\right) \Big/ \left(\sum_{s=1}^{5} E_{s,c}^{\mathrm{AlexNet}}\right).$$

Hendrycks & Dietterich, "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations", 2019.

Georgia Tech

We can try to understand the **biases of CNNs**

- Can compare to those of humans

Example: Shape vs. Texture Bias

Geirhos, "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness", 2018.

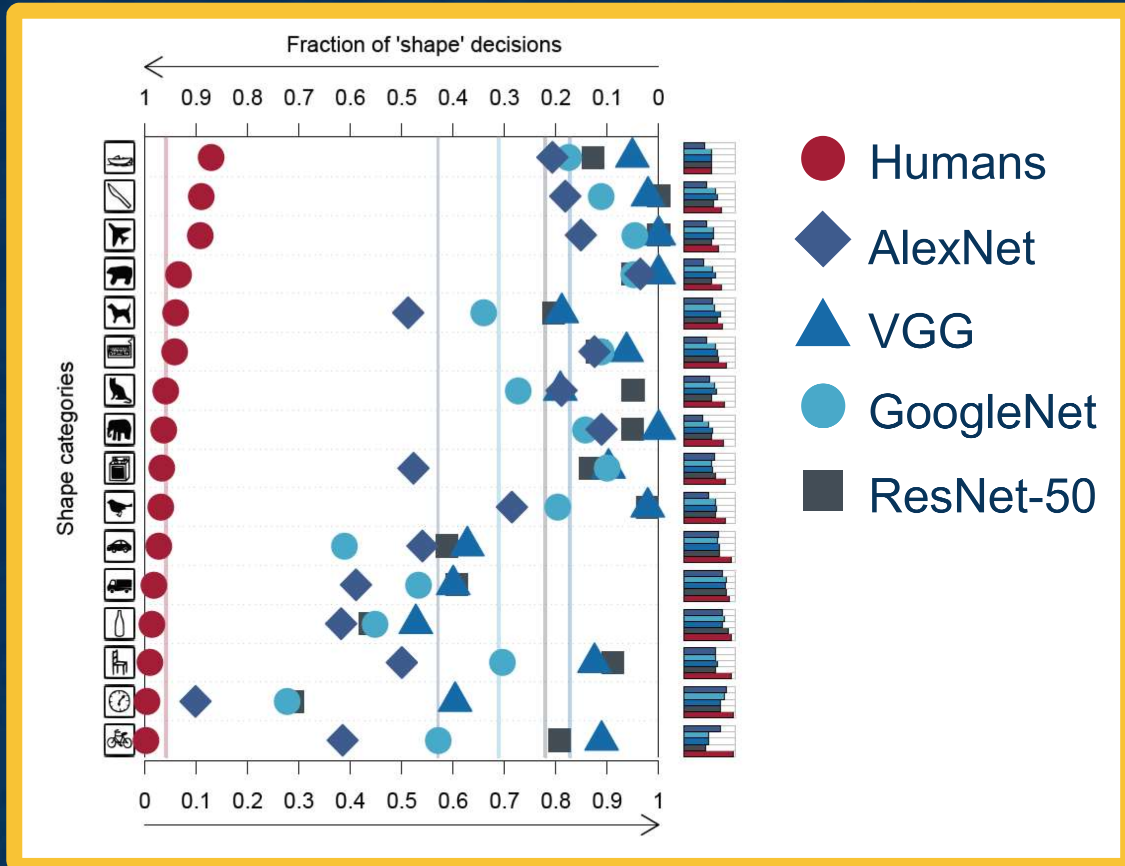
(a) Texture image
81.4% **Indian elephant**
10.3% indri
8.2% black swan

(b) Content image
71.1% **tabby cat**
17.3% grey fox
3.3% Siamese cat

(c) Texture-shape cue conflict
63.9% **Indian elephant**
26.4% indri
9.6% black swan

Analyzing Bias

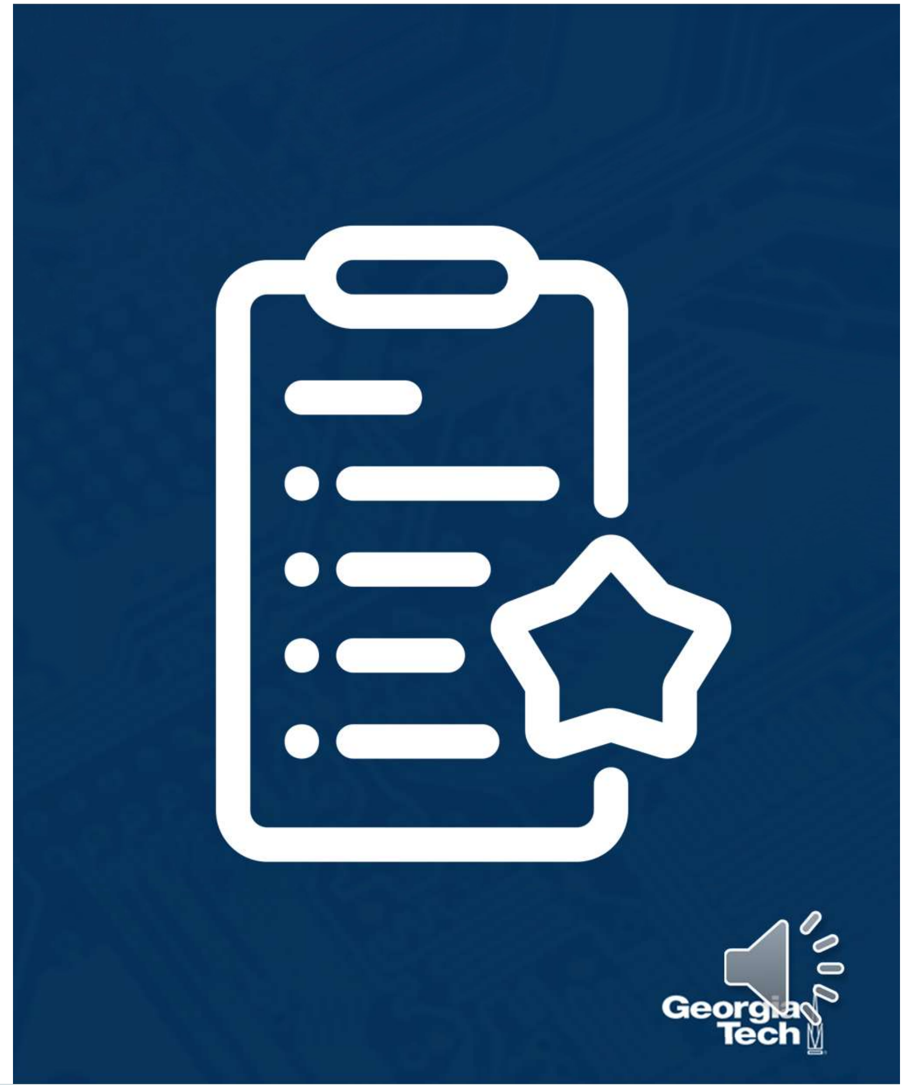
Shape vs. Texture Bias



Geirhos, "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness", 2018.

Summary

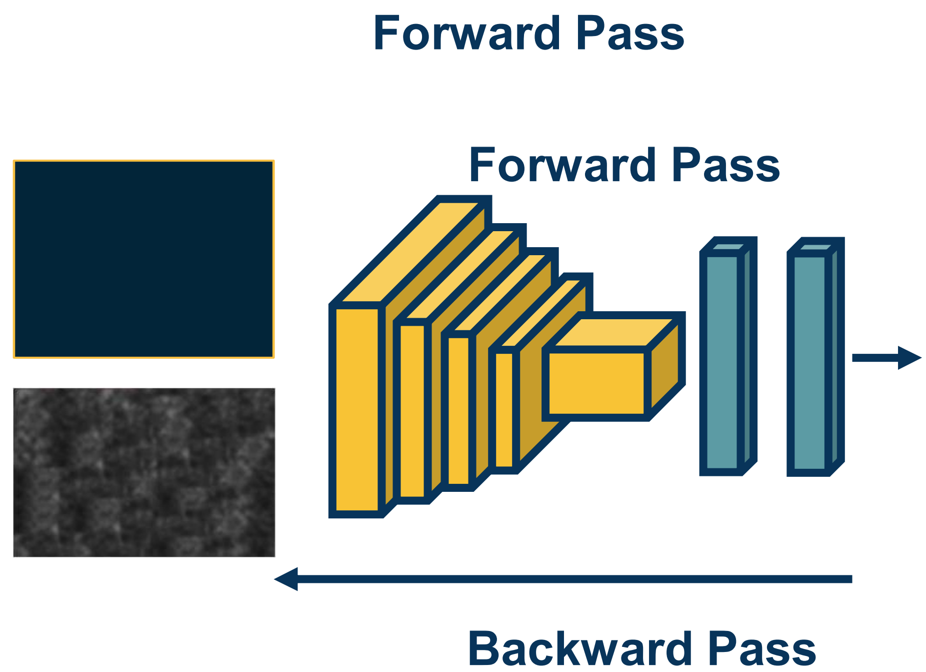
- Various ways to test the **robustness** and **biases** of neural networks
- Adversarial examples have **implications** for understanding and trusting them
- Exploring the **gain of different architectures** in terms of robustness and biases can also be used to understand what has been learned



Style Transfer

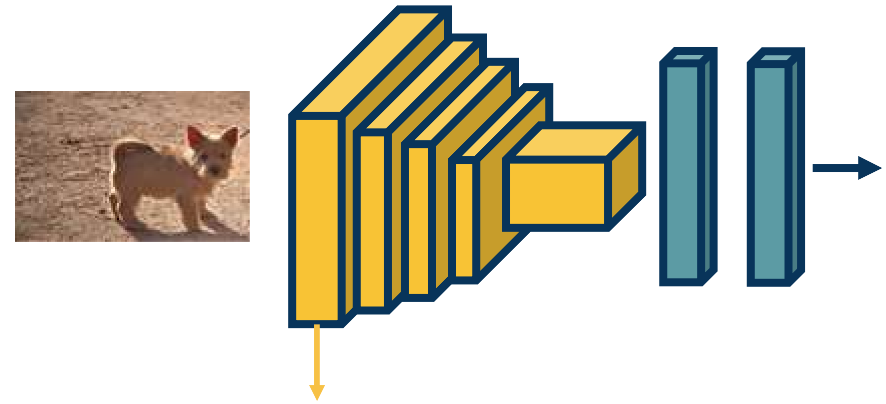
Generating Images with Content

- We can generate images through backprop
  - Regularization can be used to ensure we match image statistics
- **Idea:** What if we want to preserve the content of the image?
  - Match features at different layers!
  - We can have a loss for this

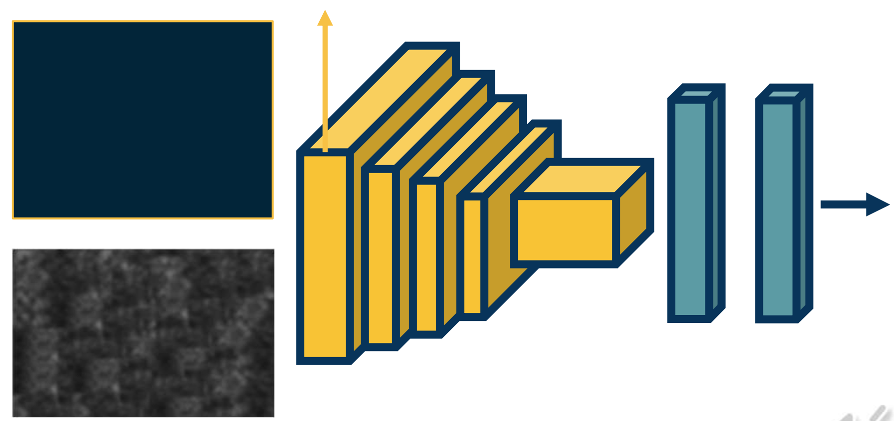


- We can generate images through backprop
  - Regularization can be used to ensure we match image statistics

- **Idea:** What if we want to preserve the content of a particular image C?
  - Match features at different layers!
  - We can have a loss for this



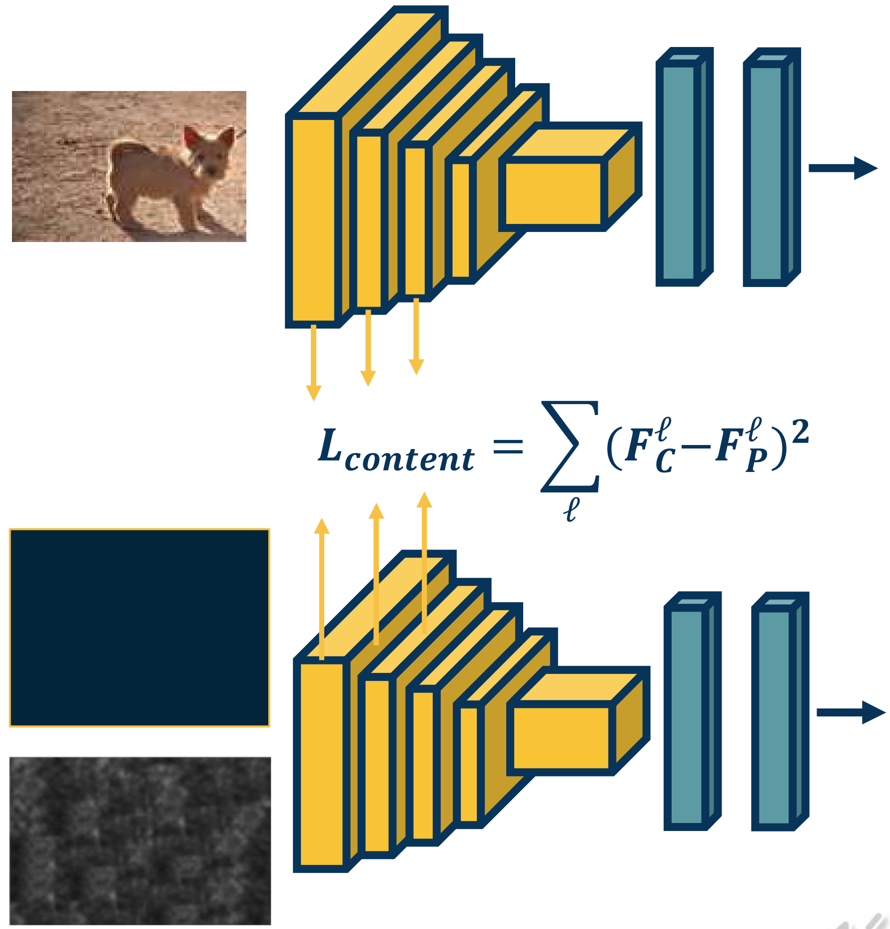
$$L_{content} = (F_C^1 - F_P^1)^2$$



Matching Features to Replicate Content

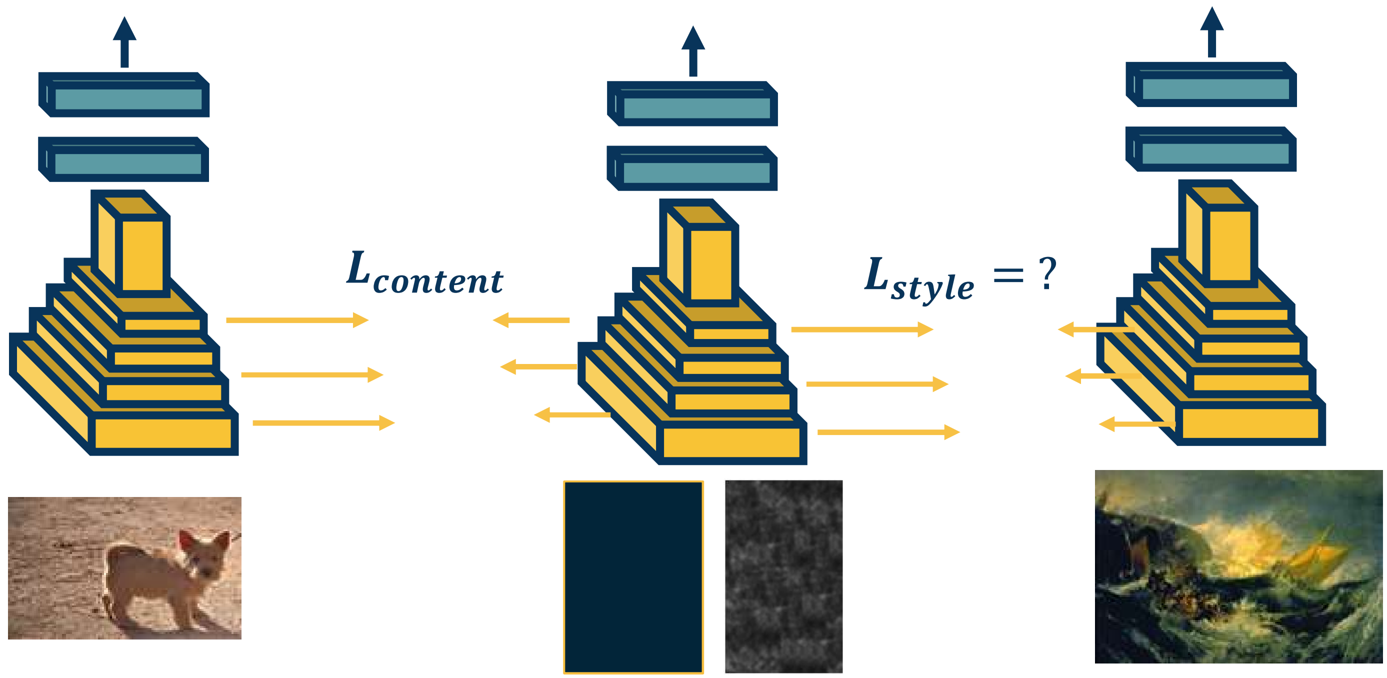
Multiple Content Losses

- How do we deal with multiple losses?
  - Remember, backwards edges going to same node *summed*
- We can have this content loss at many different layers and sum them too!



$$L_{content} = \sum_{\ell} (F_C^{\ell} - F_P^{\ell})^2$$

- **Idea:** Can we have the *content* of one image and *texture* (style) of another image?
  - **Yes!**



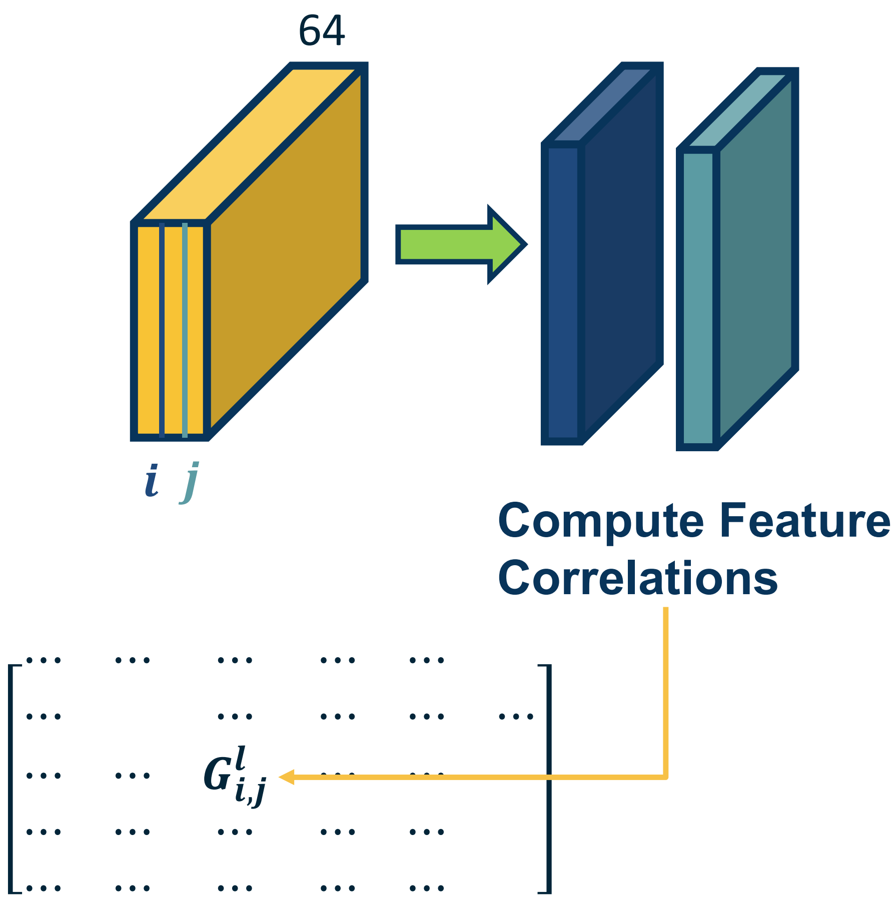
Replicating Content and Style

- How do we represent similarity in terms of textures?

- Long history in image processing!
  - Key ideas revolve around summary *statistics*
  - Should ideally remove most spatial information

- Deep learning variant: Feature correlations!
  - Called a Gram Matrix

Gradient Ascent on the Scores



$$G_S^{\ell}(i,j) = \sum_k F_S^{\ell}(i,k) F_S^{\ell}(j,k)$$

where i,j are particular **channels** in the output map of layer ℓ and k is the position (convert the map to a vector)

$$L_{style} = \sum_{\ell} \left(G_S^{\ell} - G_P^{\ell}\right)^2$$

$$L_{total} = \alpha L_{content} + \beta L_{style}$$

A

B

Gradient Ascent on the Scores

A

E

Gradient Ascent on the Scores

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