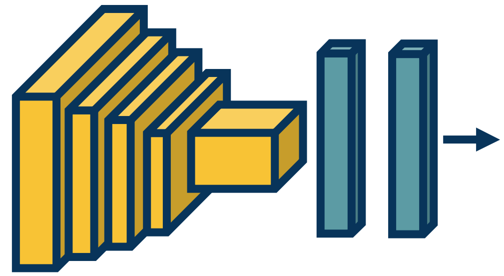
Topics:

- Visualization
- Advanced Architectures

CS 4644-DL / 7643-A

ZSOLT KIRA

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

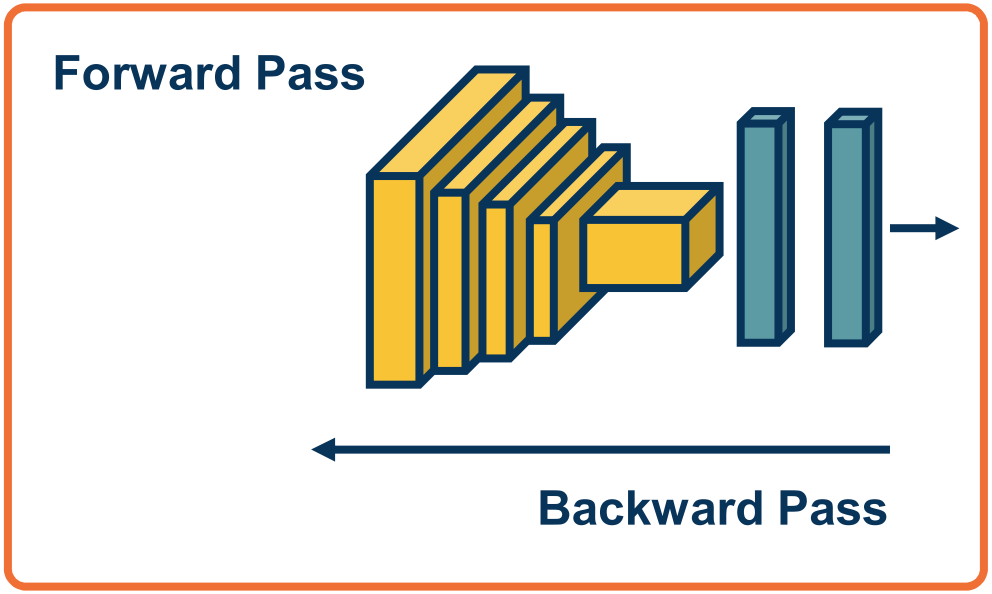
Robustness

Hendrycks & Dietterich, 2019

Visualizing Neural Networks

Given a **trained** model, we can perform:

- Freeze the model weights
- Forward pass given an input to get scores, softmax probabilities, loss and then
- Backwards pass to get gradients



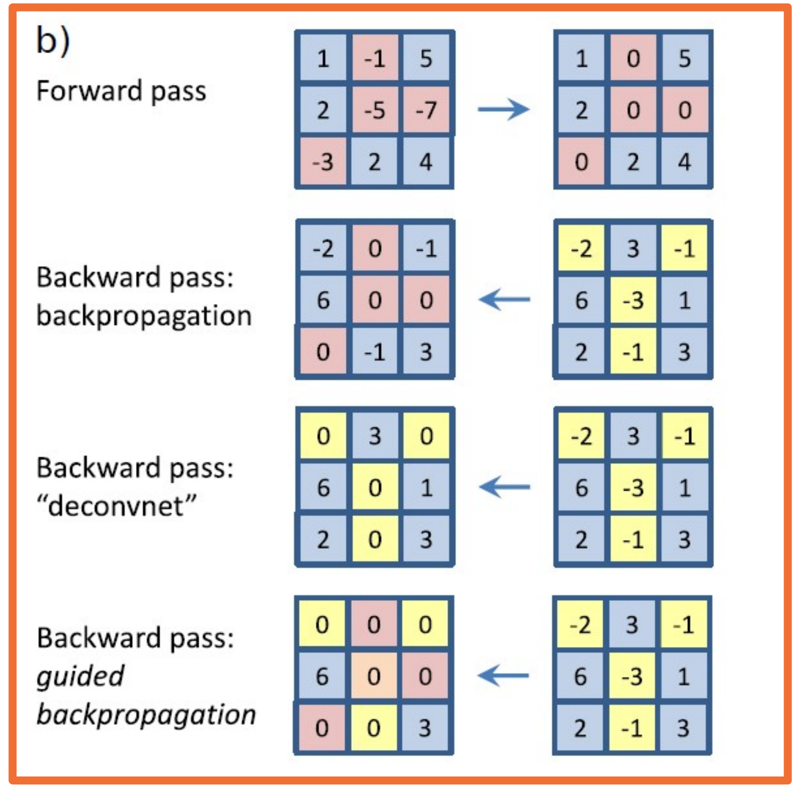
- Note: We are keeping parameters/weights **frozen**
  - Do not use gradients w.r.t. weights to perform updates
  - Instead use gradients to **analyze** what the network learned

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 Ne

Guided Backprop

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

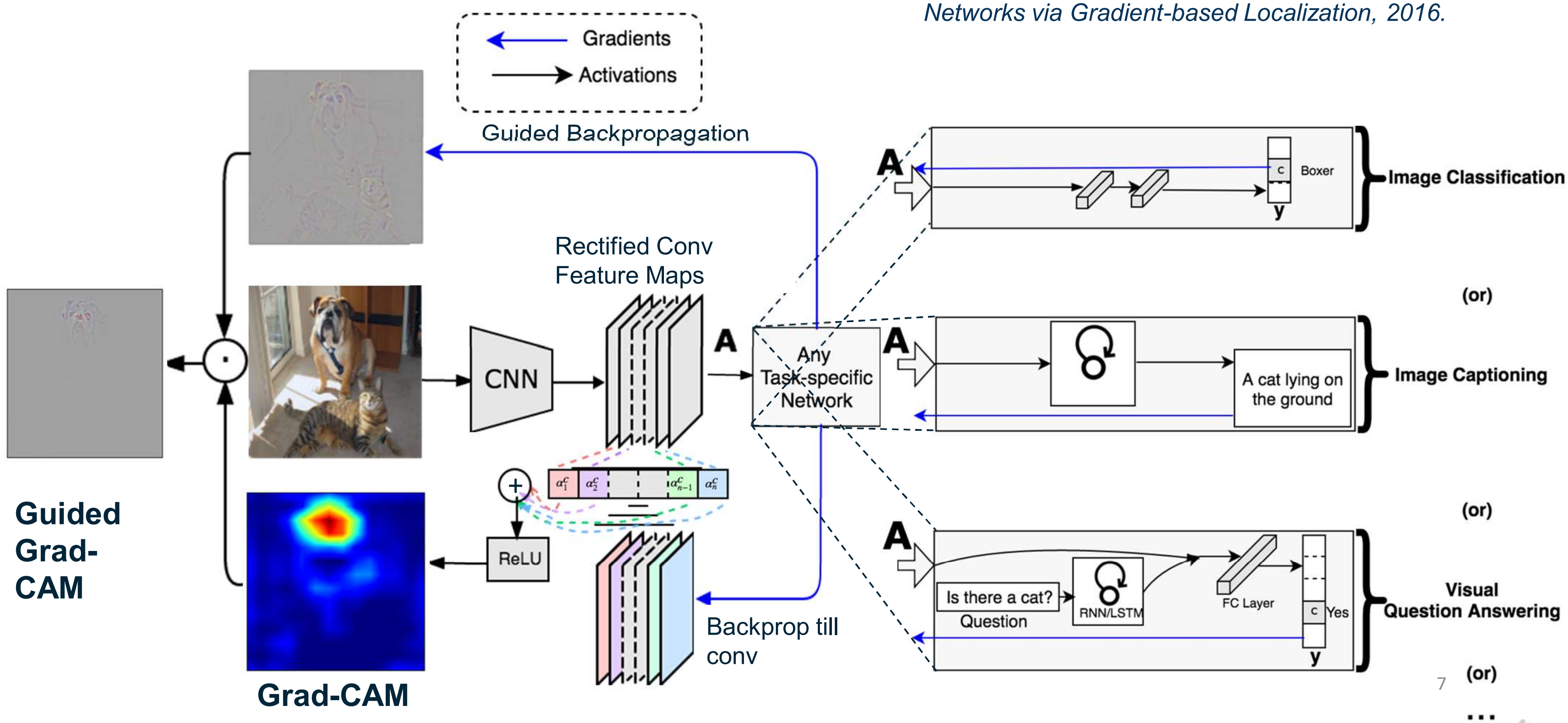
From: "Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014.

Georgia Tech

VGG Layer-by-Layer Visualization

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

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



Grad-CAM

Gradient Ascent on the Scores

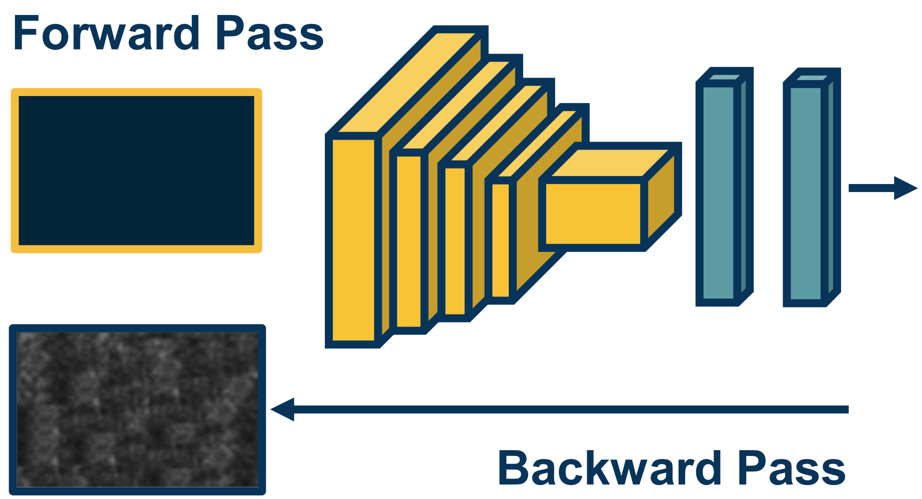
We can perform **gradient ascent** on image

- 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

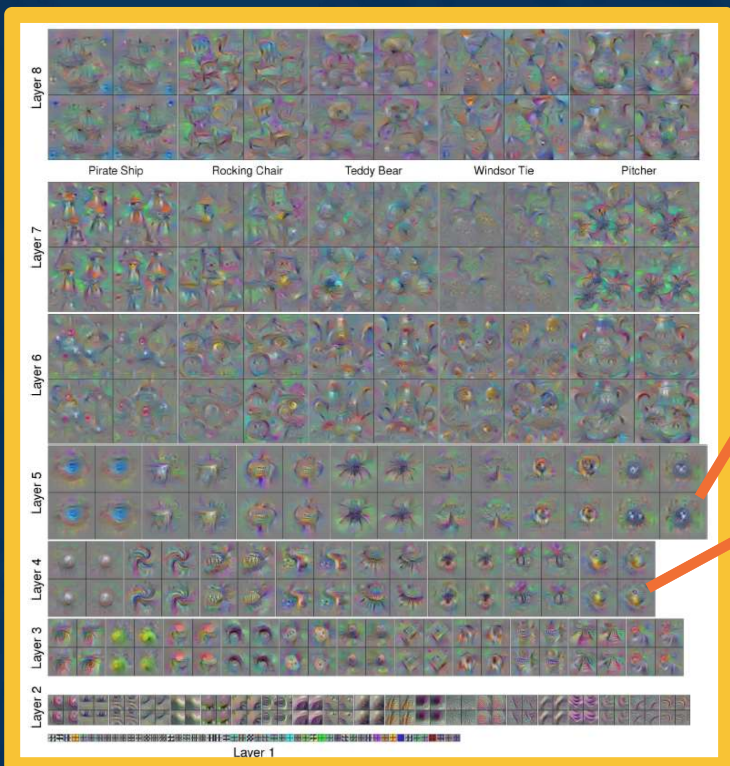
$$\arg\max S_c(I) - \lambda \, ||I||_2^2$$



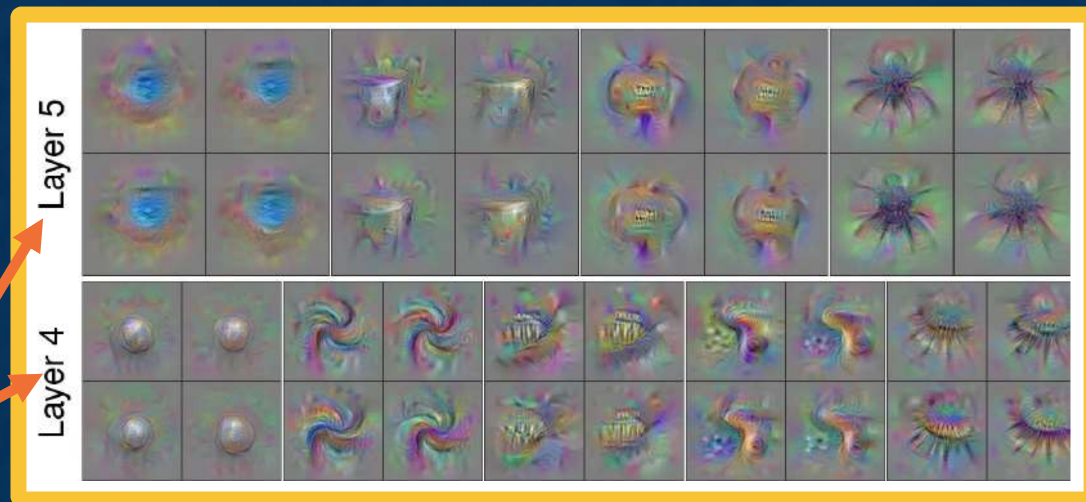
$$I = I + \alpha \frac{\partial S_c}{\partial I}$$

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

Improved Results



Note: Can generate input images to maximize any arbitrary activation!



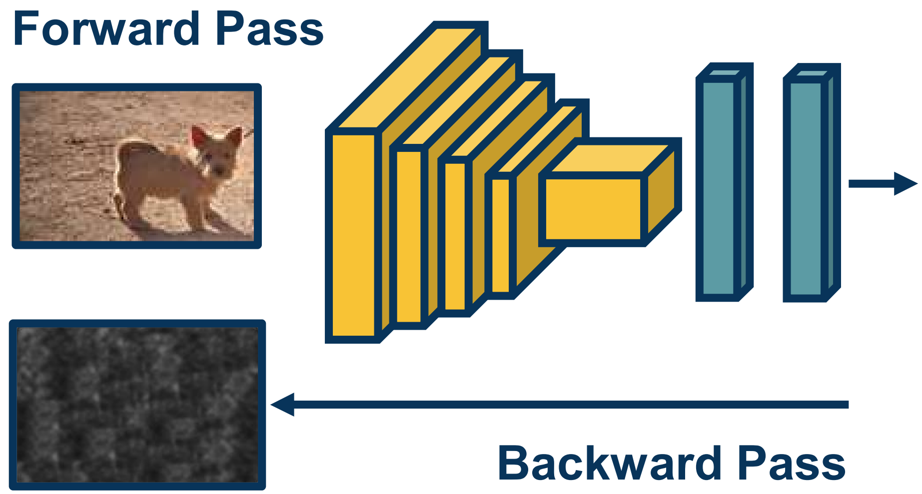
From: Yosinski et al., “Understanding Neural Networks Through Deep Visualization”, 2015

- 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!

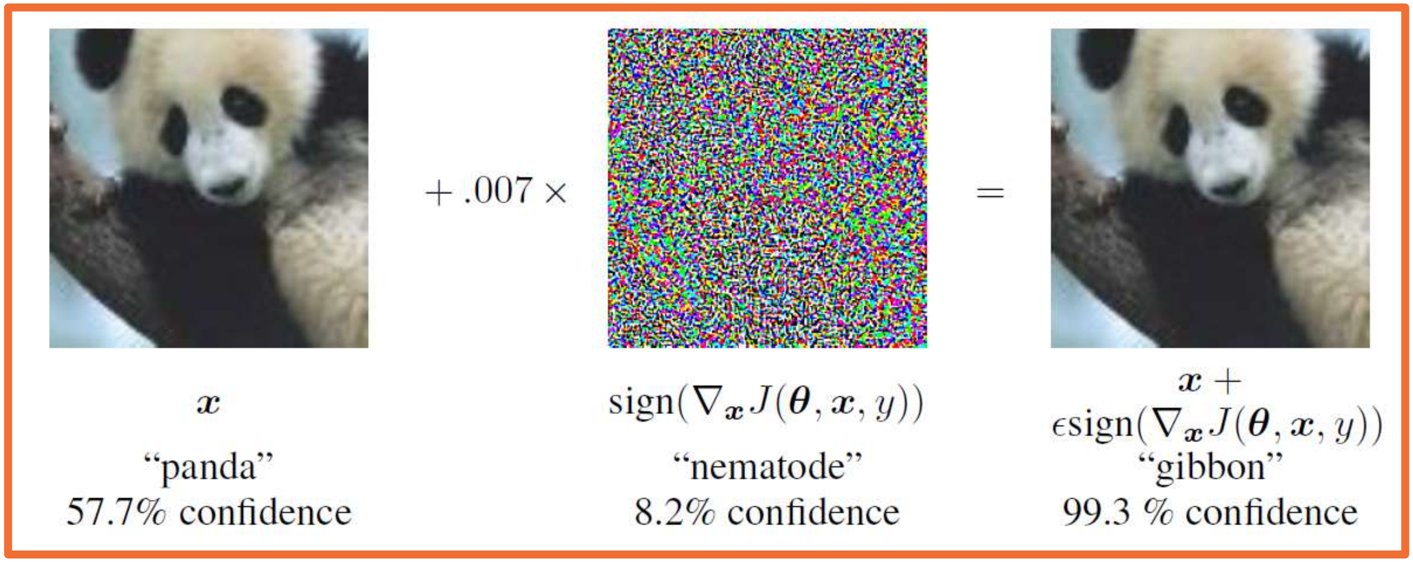
$$\boldsymbol{argmax}\, S_c(I) - \lambda\, \left|\left|I\right|\right|_2^2$$

where $c = cat$



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

Gradient Ascent on the Scores



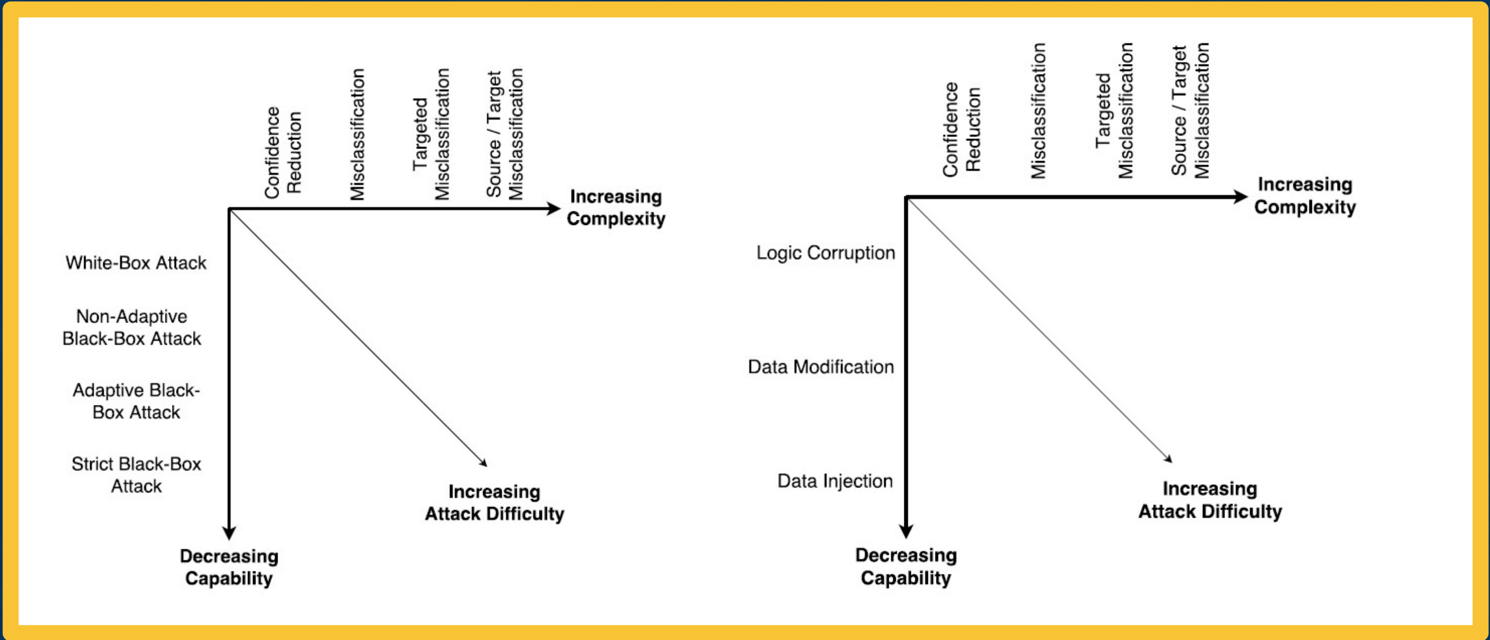
Note this problem is not specific to deep learning!

- 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

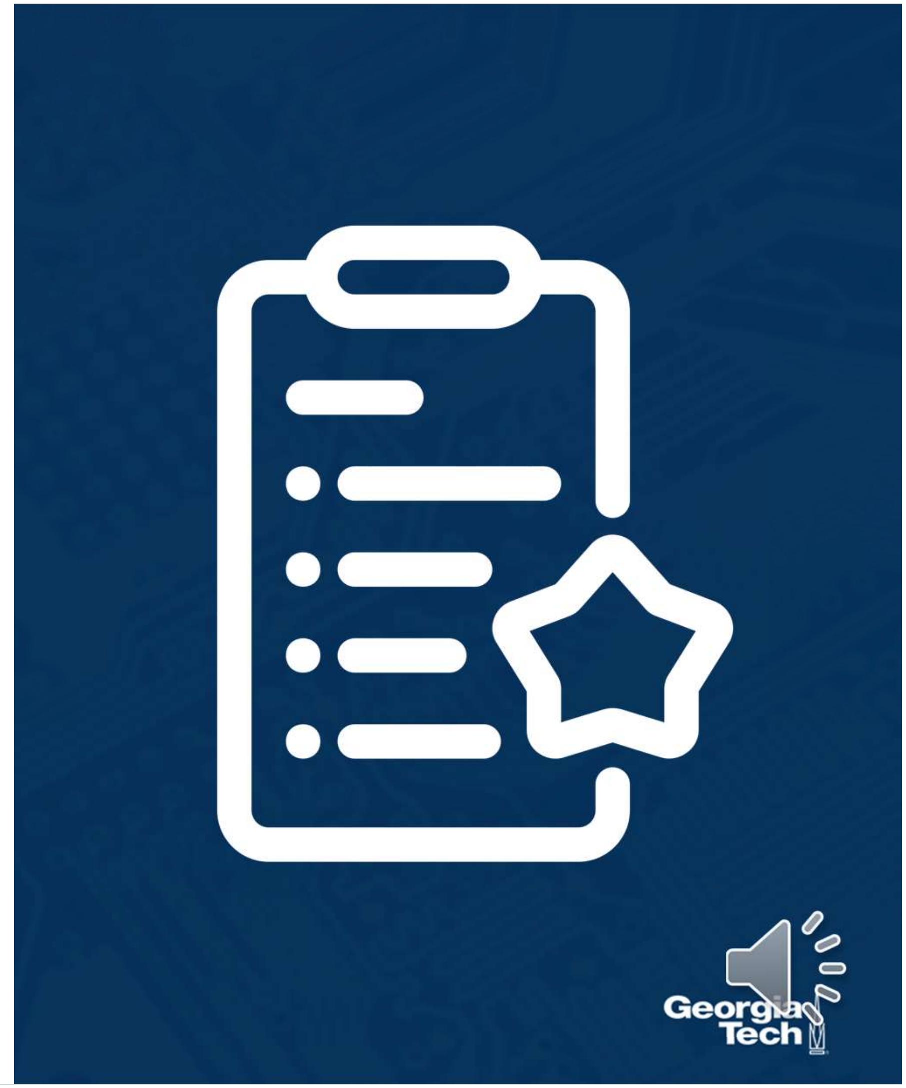
Summary of dversarial Attacks/Defenses

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

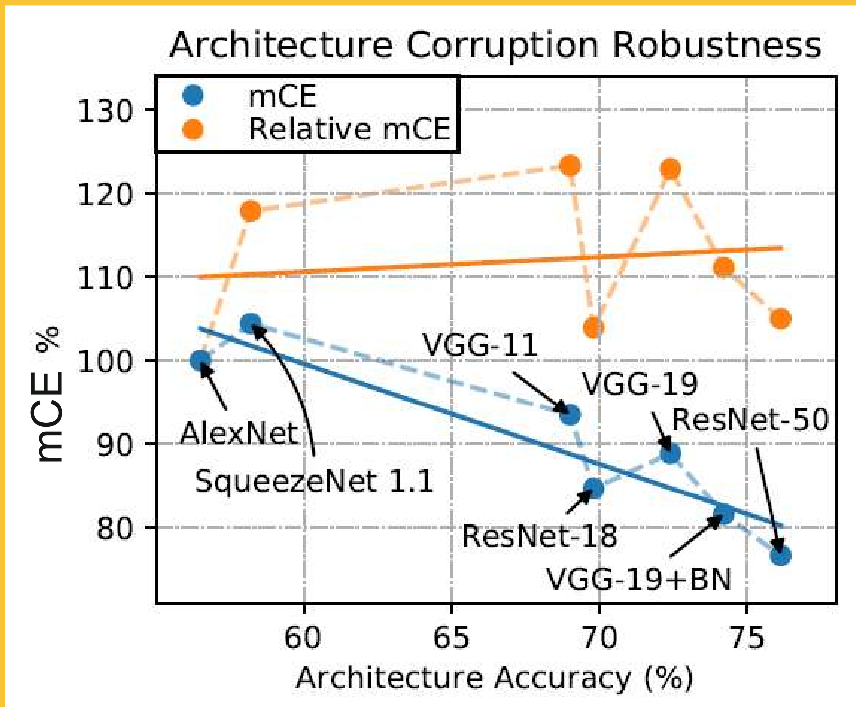
Several defenses such as:

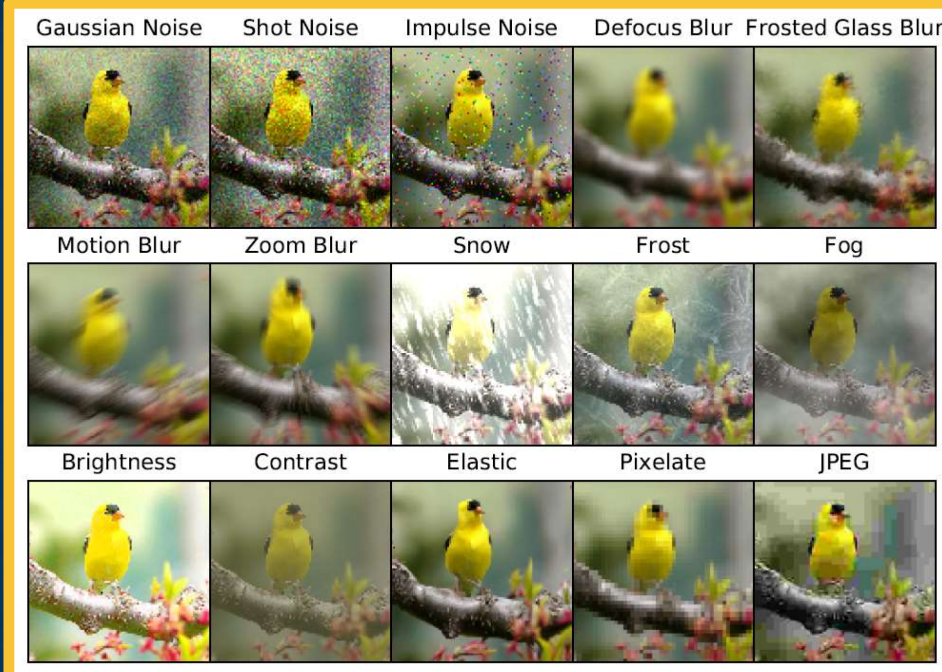
- Training with adversarial examples
- Perturbations, noise, or re-encoding of inputs

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

Shape vs. Texture Bias

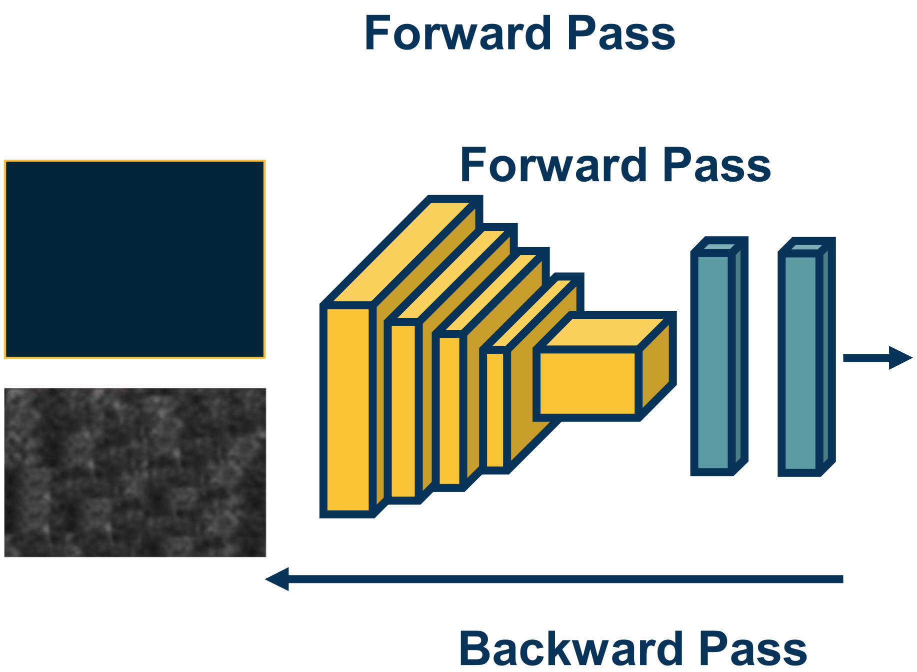
Geirhos, "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness", 2018.

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

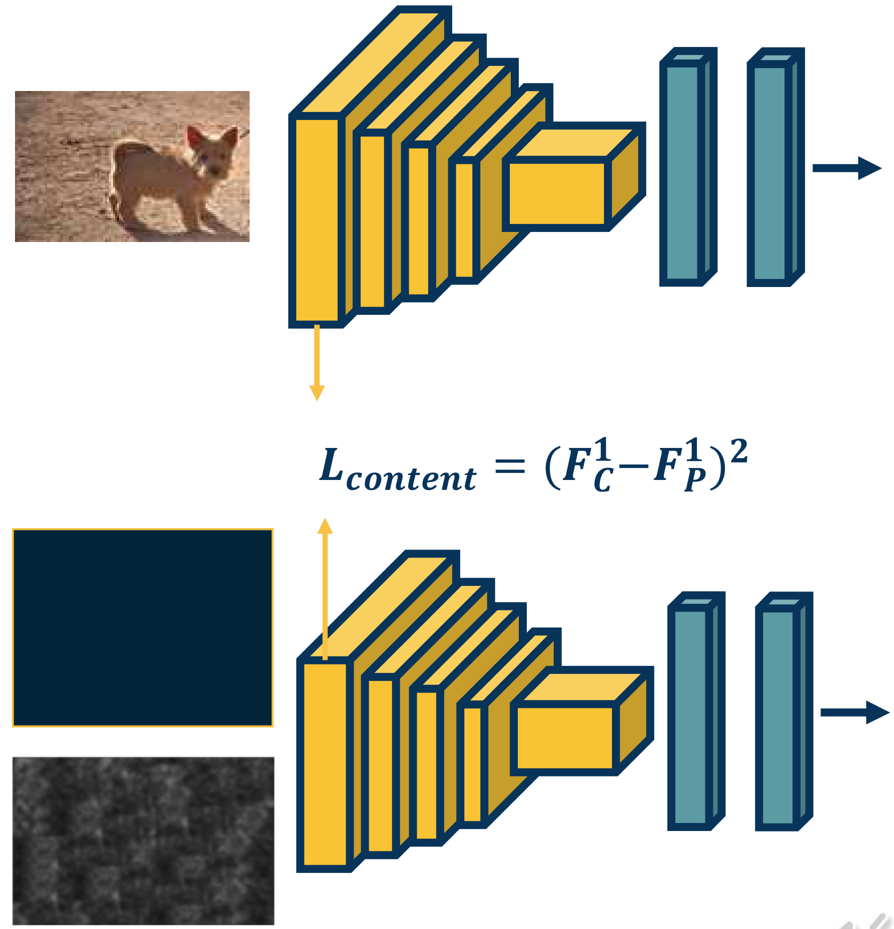
- We can generate images through backprop
  - Regularization can be used to ensure we match image statistics
- **Idea:** What if we want to preserve the content of the image?
  - Match features at different layers!
  - We can have a loss for this



Generating Images with Content

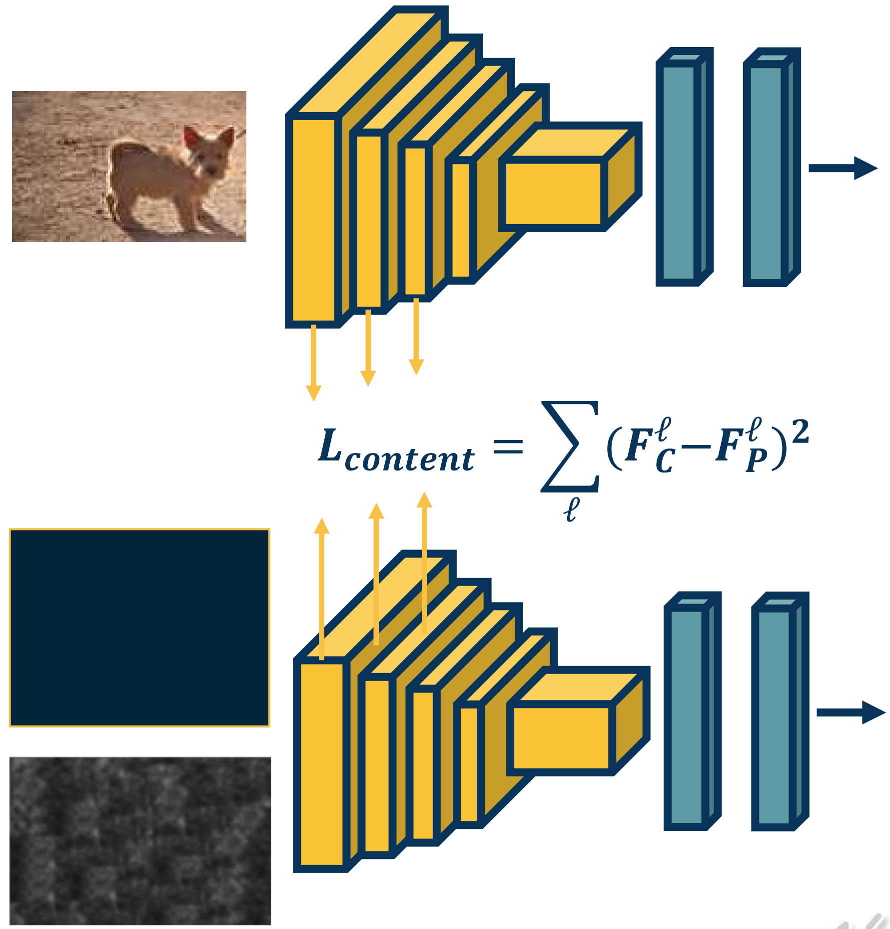
Matching Features to Replicate Content

- 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 a particular image C?
  - Match features at different layers!
  - We can have a loss for this

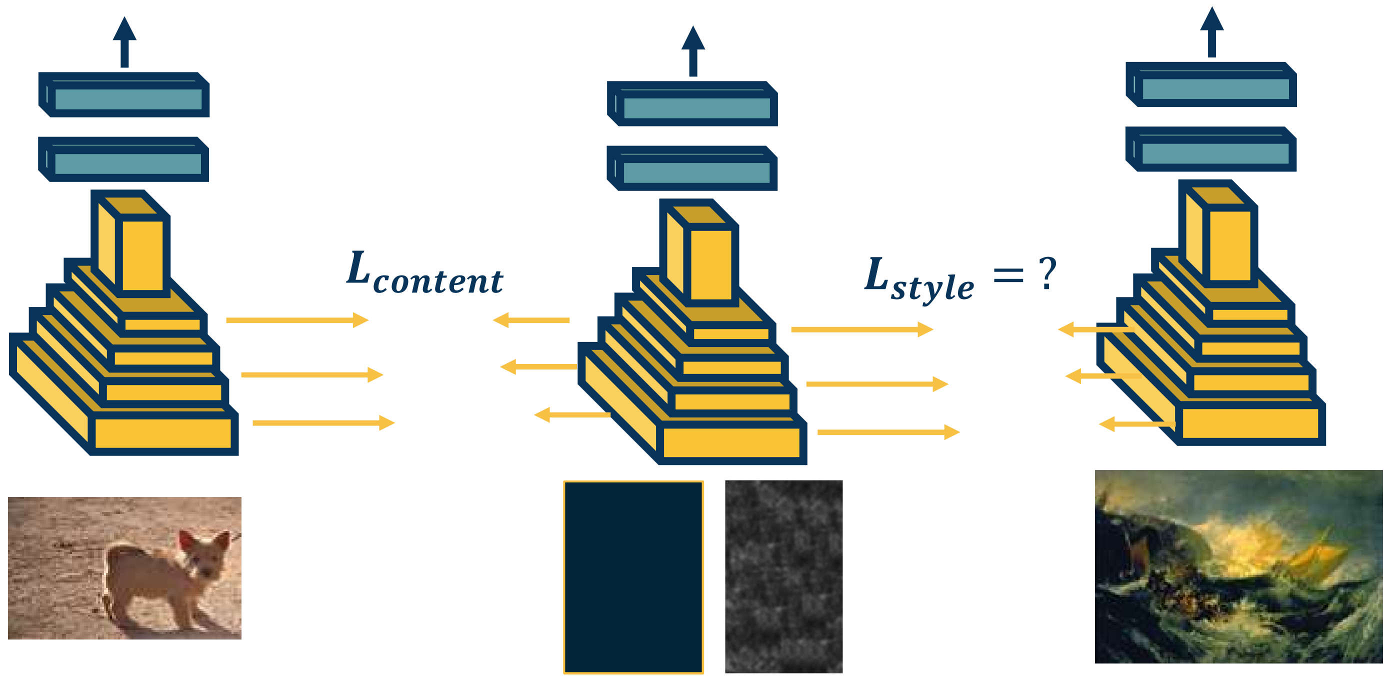


Multiple Content Losses

- 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!



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



Replicating Content and Style

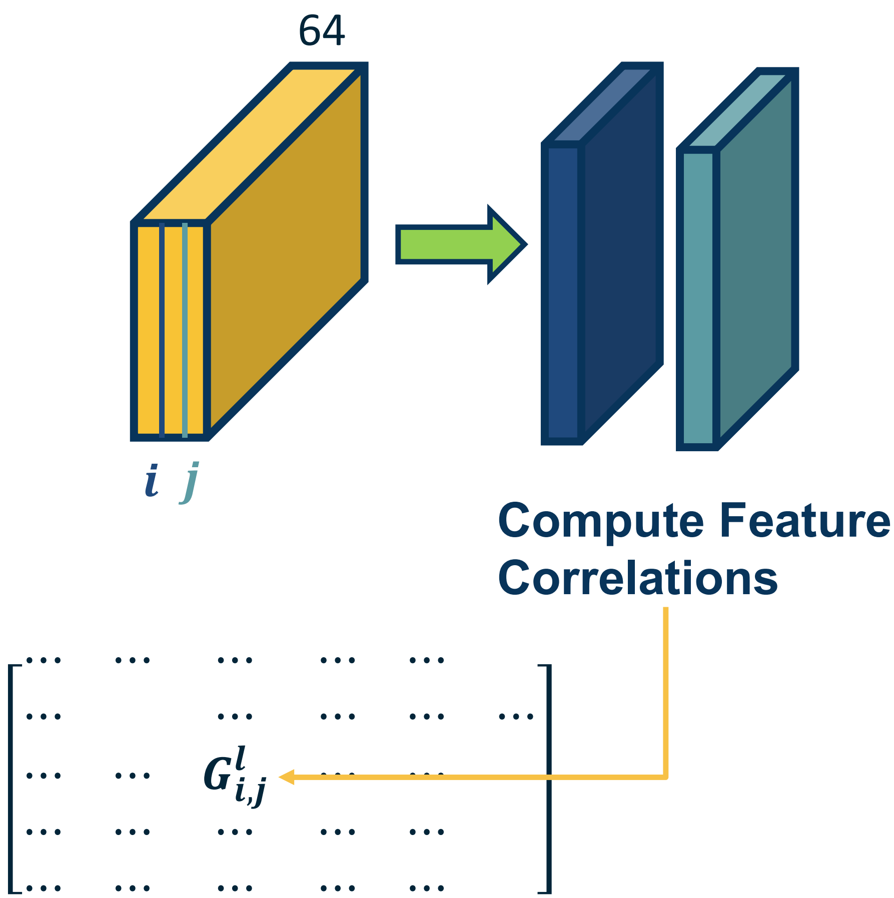
- How do we represent similarity in terms of textures?

- 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

Georgia Tech



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

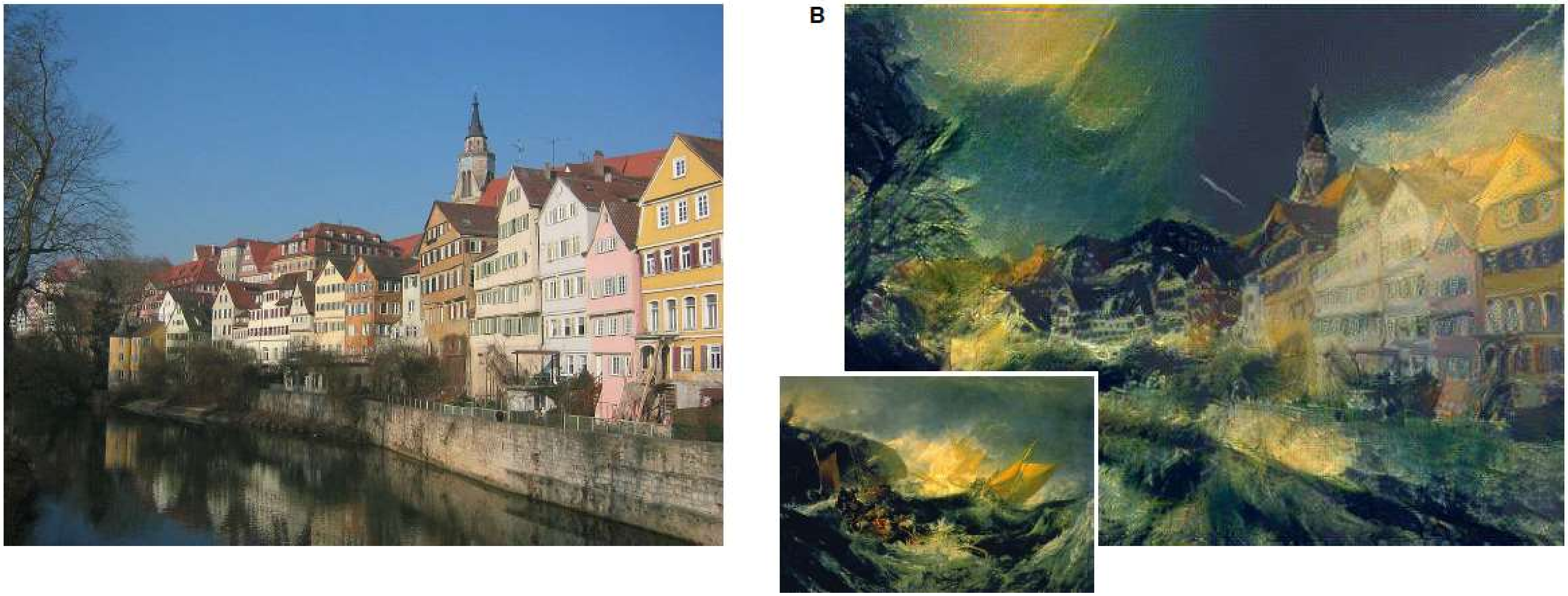
where i,j are particular **channels** in the output map of layer ℓ and k is the position (convert the map to a vector)

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

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

Gradient Ascent on the Scores

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

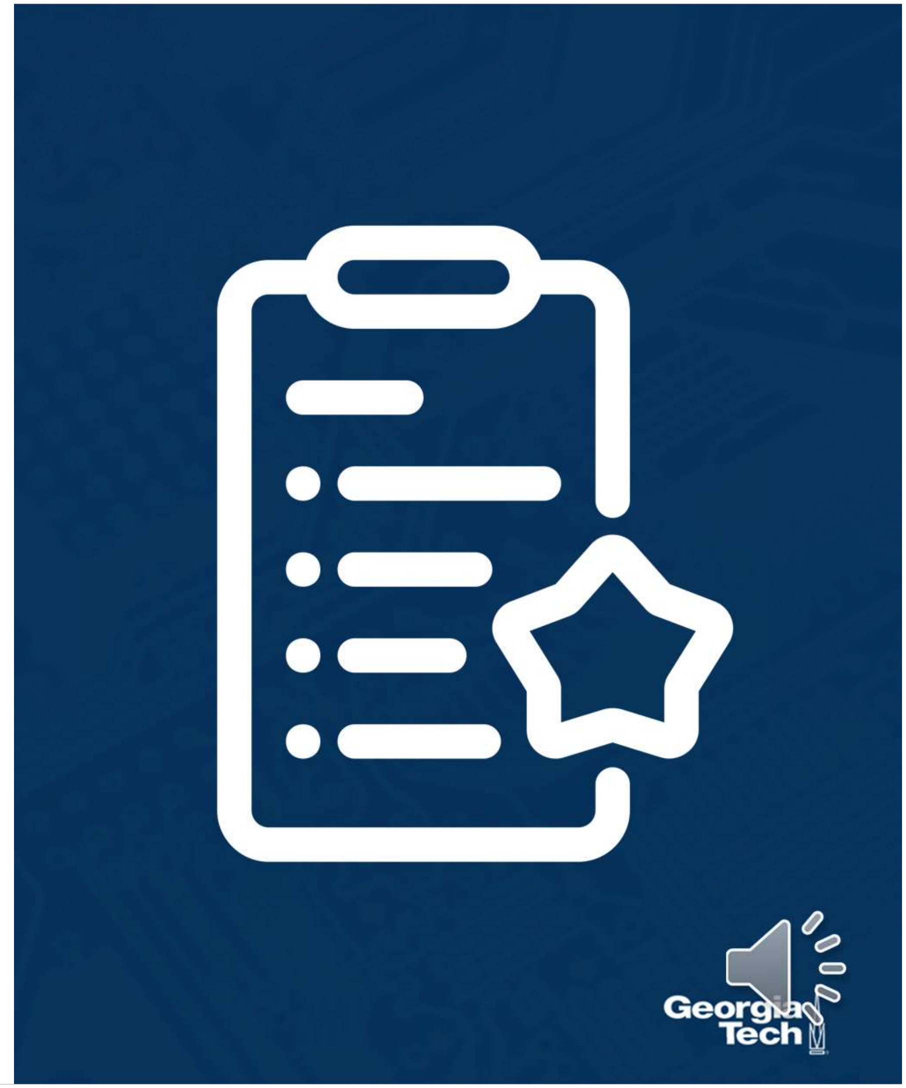
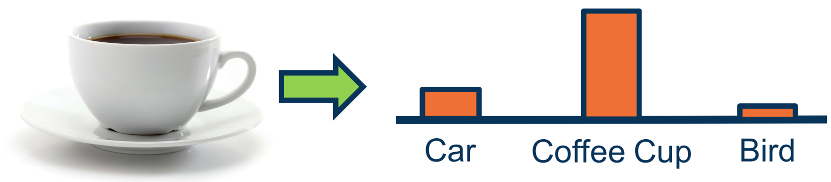
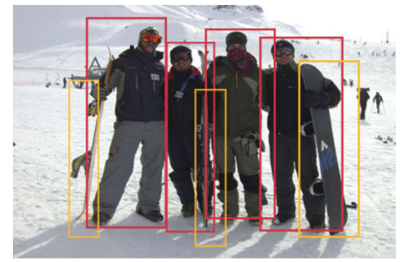


Image Segmentation Networks

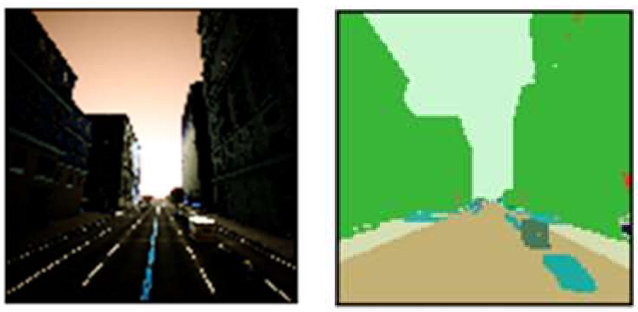
Georgia Tech



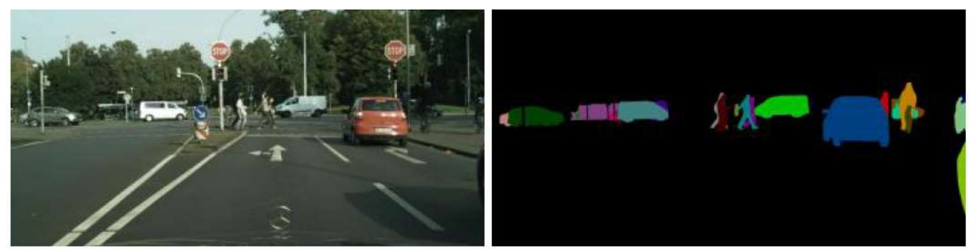
Classification
(Class distribution per image)



Object Detection
(List of bounding boxes with class distribution per box)



Semantic Segmentation
(Class distribution per pixel)

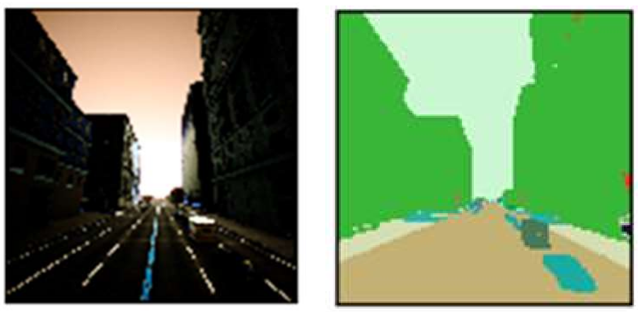


Instance Segmentation
(Class distribution per pixel with unique ID)

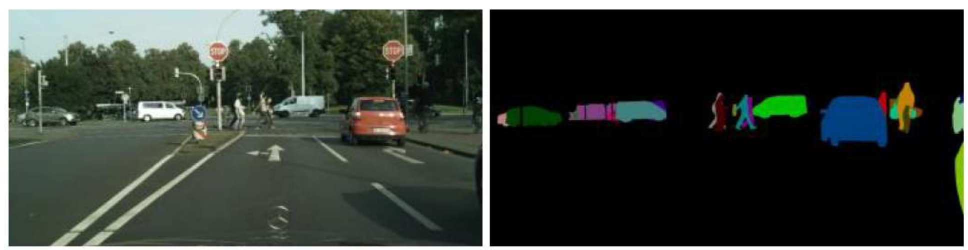
Computer Vision Tasks

Given an image, output another image

- Each output contains class distribution per pixel
- More generally an image-to-image problem

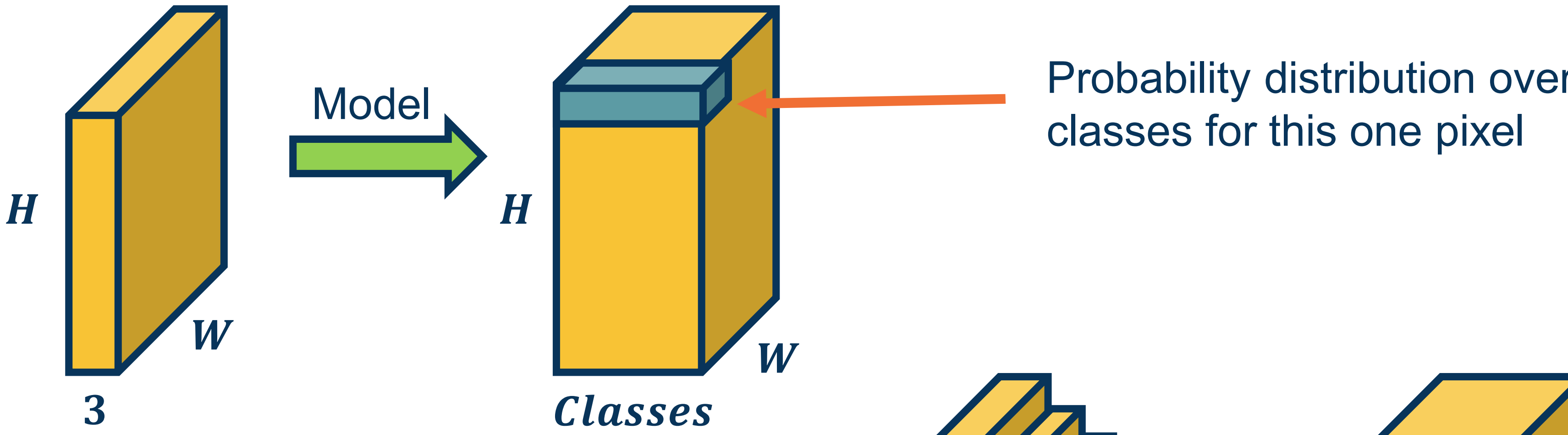


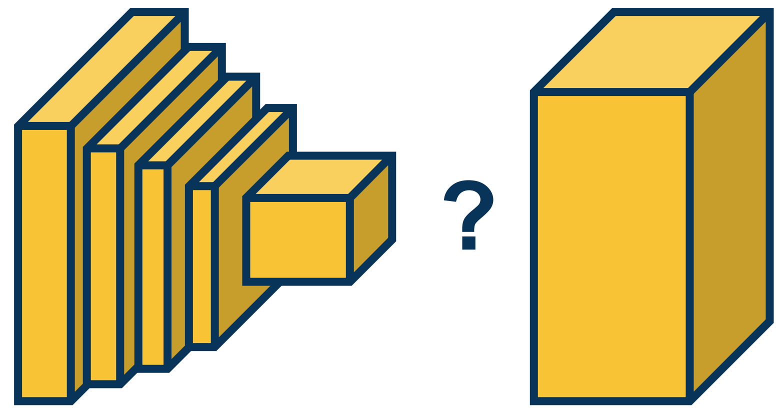
Semantic Segmentation
(Class distribution per pixel)



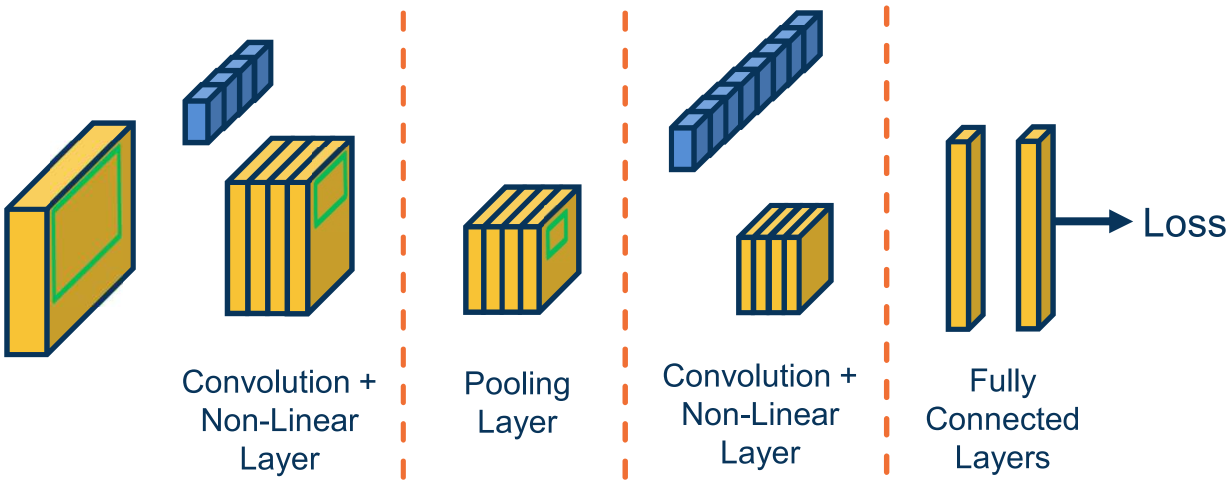
Instance Segmentation
(Class distribution per pixel with unique ID)

Segmentation Tasks





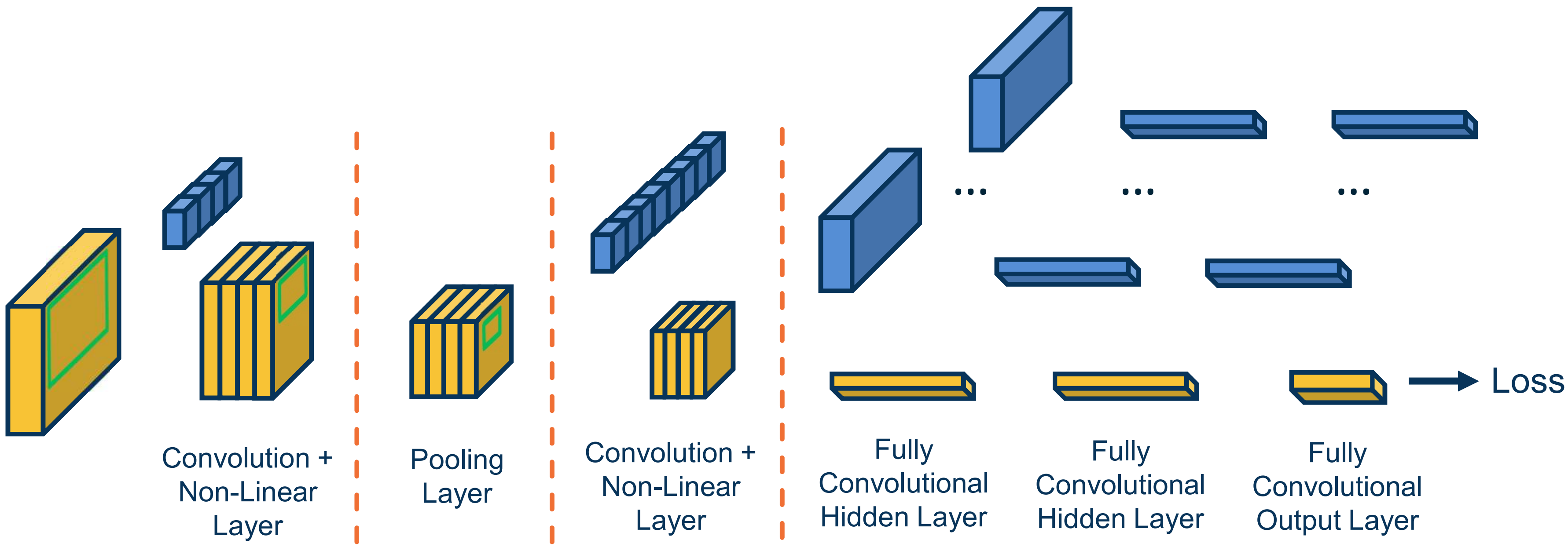
Input & Output



Fully connected layers no longer explicitly retain spatial information (though the network can still learn to do so)

Idea: Convert fully connected layer to convolution!

Idea 1: Fully-Convolutional Network



Each kernel has the size of entire input! (output is 1 scalar)

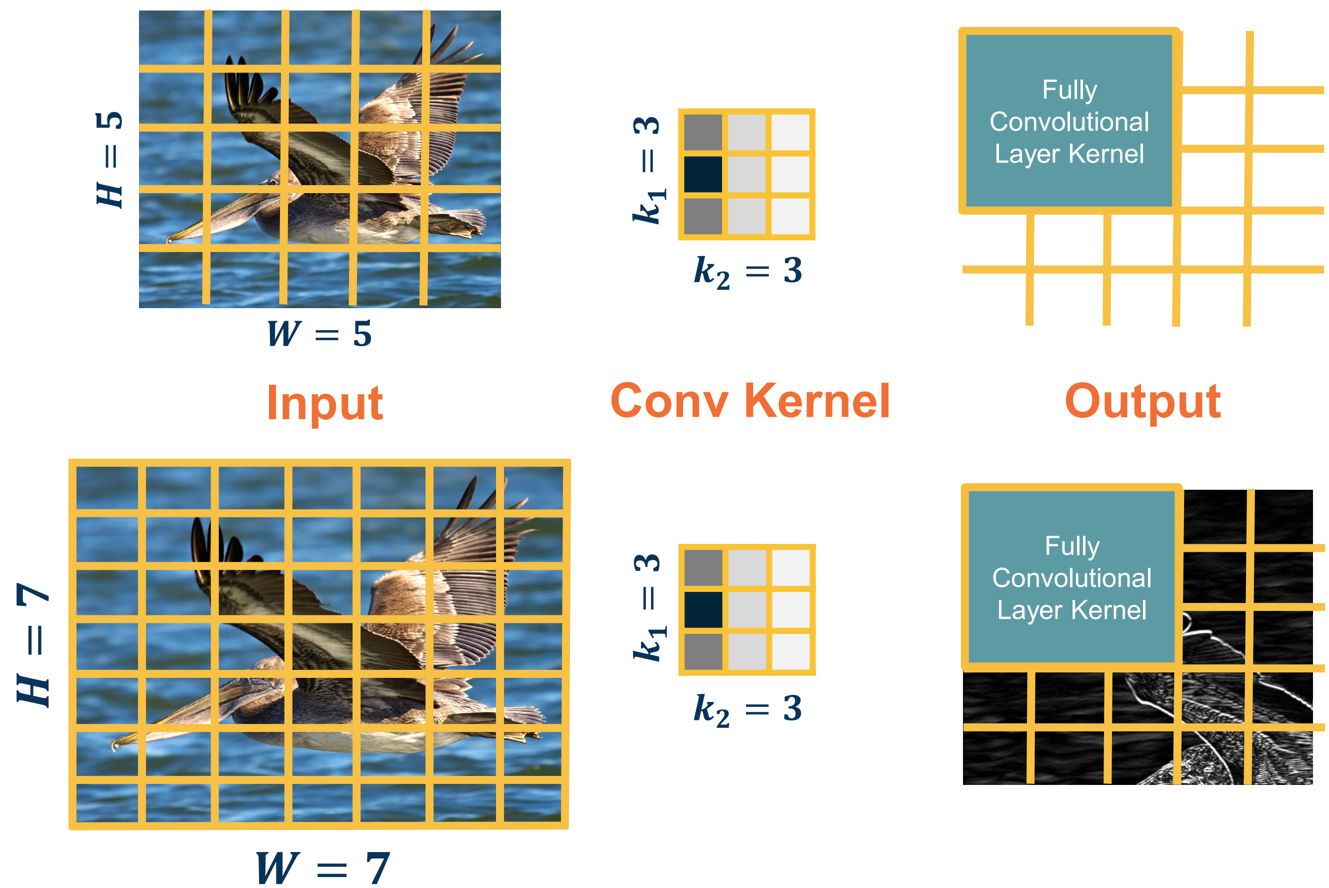
- This is equivalent to Wx+b!
- We have one kernel per output node

Converting FC Layers to Conv Layers

Same Kernel, Larger Input

Original:

Larger:

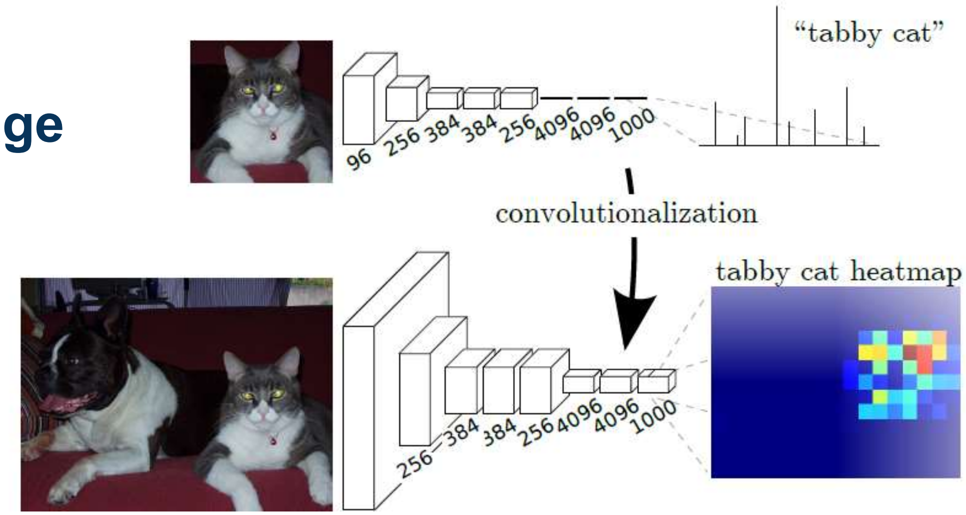


Why does this matter?

- We can stride the “fully connected” classifier across larger inputs!
- Convolutions work on arbitrary input sizes (because of striding)

Original sized image

Larger Image



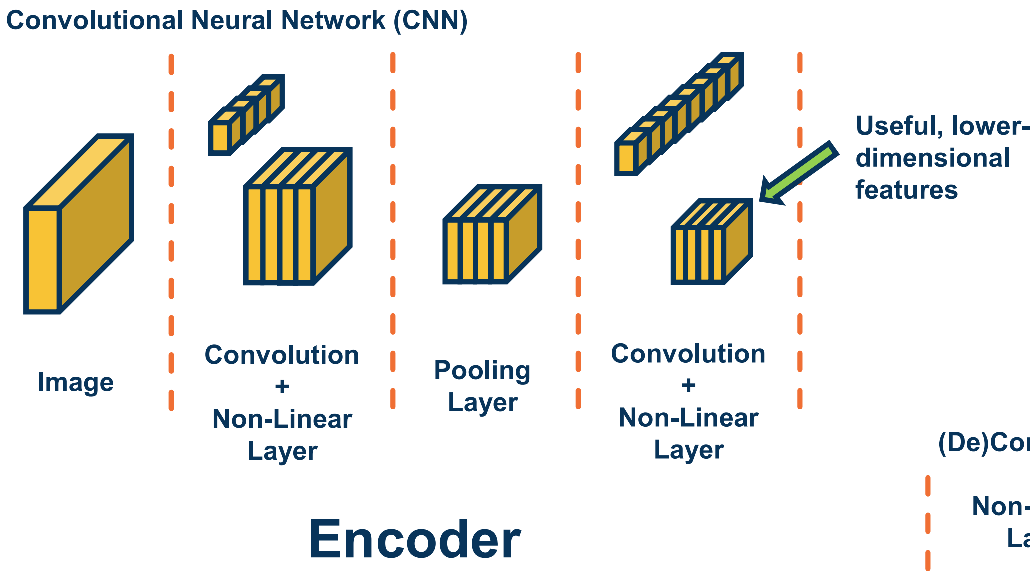
Larger Output Size!

Larger Output Maps

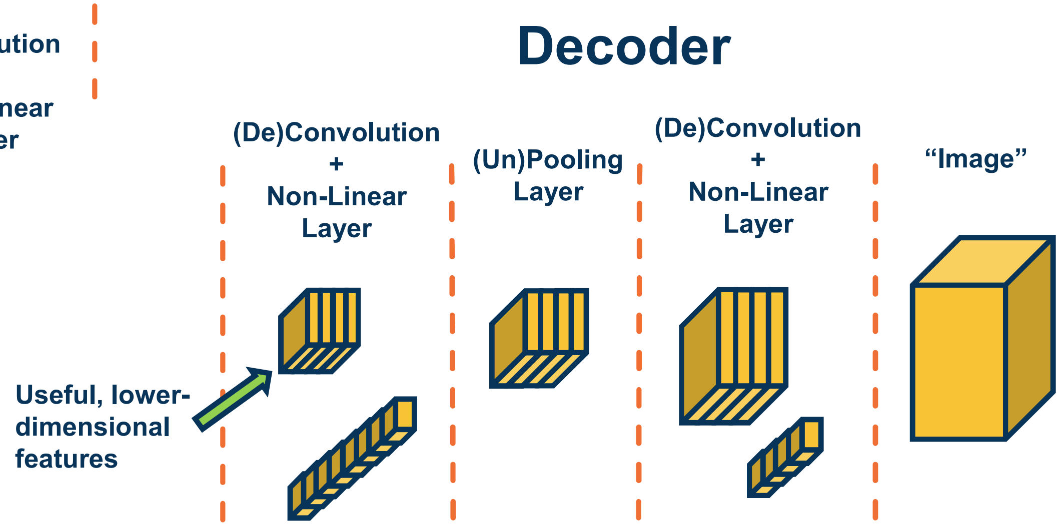
Long, et al., “Fully Convolutional Networks for Semantic Segmentation”, 2015

Inputting Larger Images

Idea 2: “De”Convolution and UnPooling



We can develop learnable or non-learnable upsampling layers!



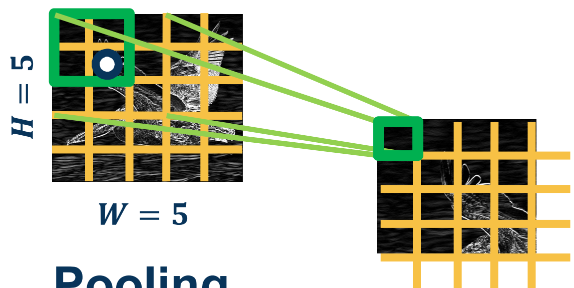
Max Unpooling

Example : Max pooling

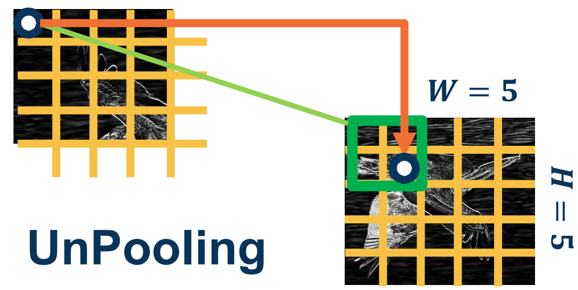
- Stride window across image but perform per-patch **max operation**

$$X(0:1,0:1) = \begin{bmatrix} 100 & 150 \\ 100 & 200 \end{bmatrix} \Rightarrow \max(0{:}1,0{:}1) = 200$$

Copy value to position chosen as max in encoder, fill reset of this window with zeros



Pooling



UnPooling

Idea: Remember max elements in encoder! Copy value from equivalent position, rest are zeros

$$X = \begin{bmatrix} 120 & 150 & 120 \\ 100 & 50 & 110 \\ 25 & 25 & 10 \end{bmatrix} \xrightarrow[\text{2x2 max pool}]{} Y = \begin{bmatrix} 150 & 150 \\ 100 & 110 \end{bmatrix}$$

Encoder

Decoder

$$X = \begin{bmatrix} 300 & 450 \\ 100 & 250 \end{bmatrix} \xrightarrow{\text{2x2 max unpool}} Y = \begin{bmatrix} 0 & 300 & - \\ 0 & 0 & - \\ - & - & - \end{bmatrix}$$

Max Unpooling Example (one window)

Georgia Tech

$$X_{enc} = \begin{bmatrix} 120 & 150 & 120 \\ 100 & 50 & 110 \\ 25 & 25 & 10 \end{bmatrix} \xrightarrow{\text{2x2 max pool}} Y_{enc} = \begin{bmatrix} 150 & 150 \\ 100 & 110 \end{bmatrix}$$

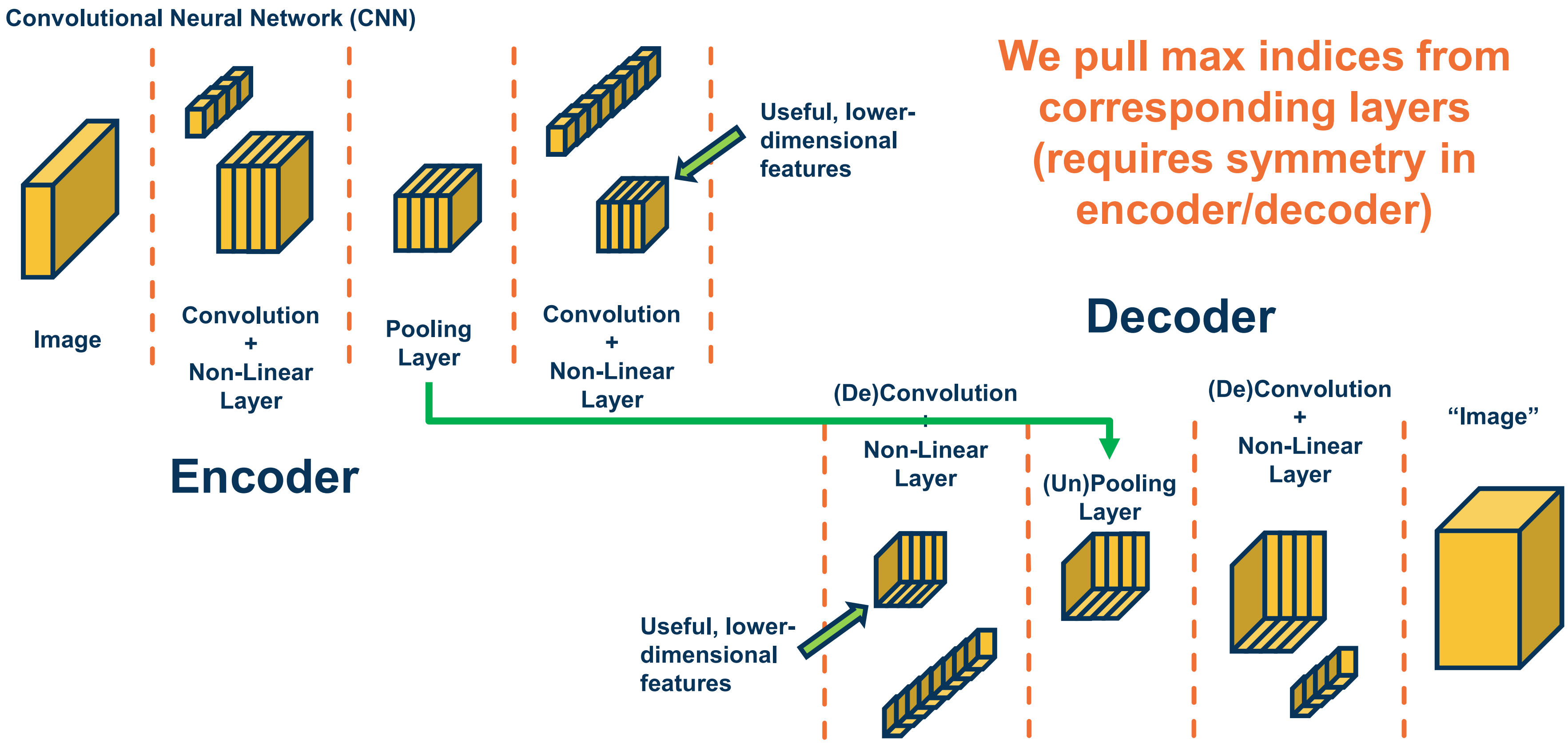
Encoder

Contributions from multiple windows are summed

Decoder

$$X_{dec} = \begin{bmatrix} 300 & 450 \\ 100 & 250 \end{bmatrix} \xrightarrow{\text{2x2 max unpool}} Y_{dec} = \begin{bmatrix} 0 & 300 + 450 & 0 \\ 100 & 0 & 250 \\ 0 & 0 & 0 \end{bmatrix}$$

Max Unpooling Example



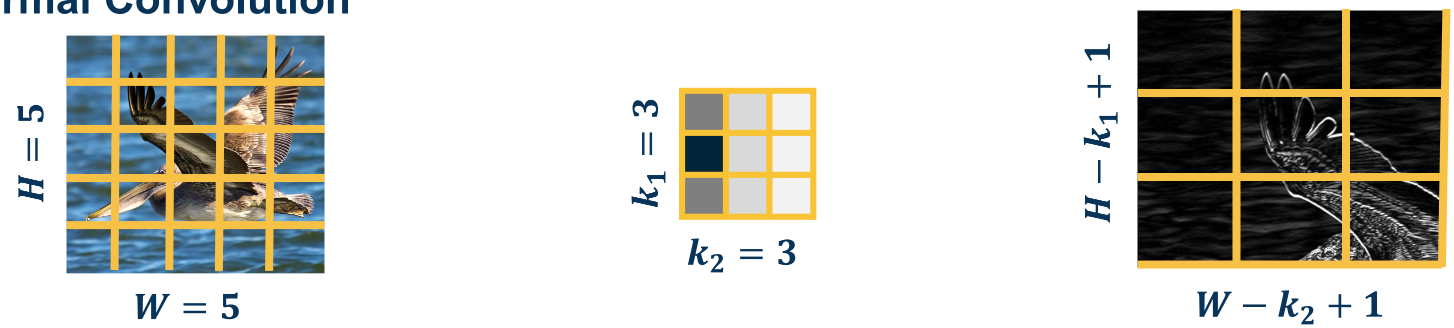
We pull max indices from corresponding layers (requires symmetry in encoder/decoder)

Symmetry in Encoder/Decoder

Georgia Tech

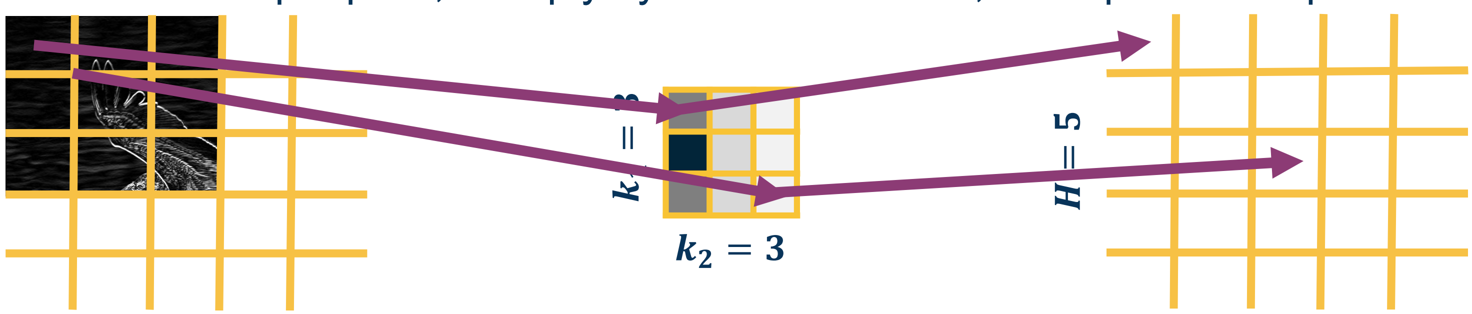
How can we *upsample* using convolutions and learnable kernel?

Normal Convolution



Transposed Convolution (also known as “deconvolution”, fractionally strided conv)

Idea: Take each input pixel, multiply by learnable kernel, “stamp” it on output



“De”Convolution (Transposed Convolution)

$$X = \begin{bmatrix} 120 & 150 & 120 \\ 100 & 50 & 110 \\ 25 & 25 & 10 \end{bmatrix} \qquad K = \begin{bmatrix} 1 & -1 \\ 2 & -2 \end{bmatrix}$$

Contributions from multiple windows are summed

$$\begin{bmatrix} 120 & -120 & 0 & 0 \\ 240 & -240 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

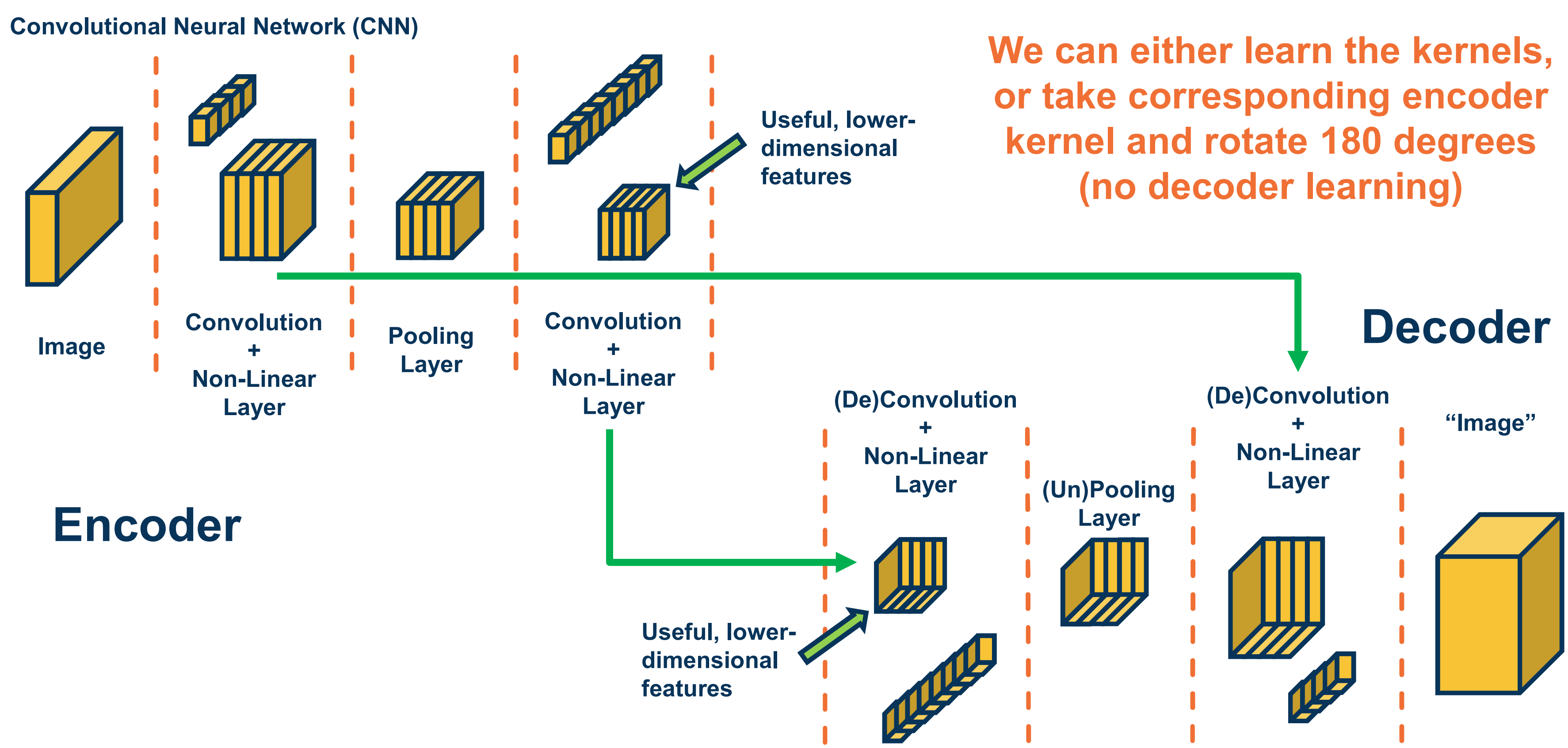
Incorporate
X(0,0)

$$\begin{bmatrix} 120 & -120+150 & -150 & 0 \\ 240 & -240+300 & -300 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Incorporate
X(1,0)

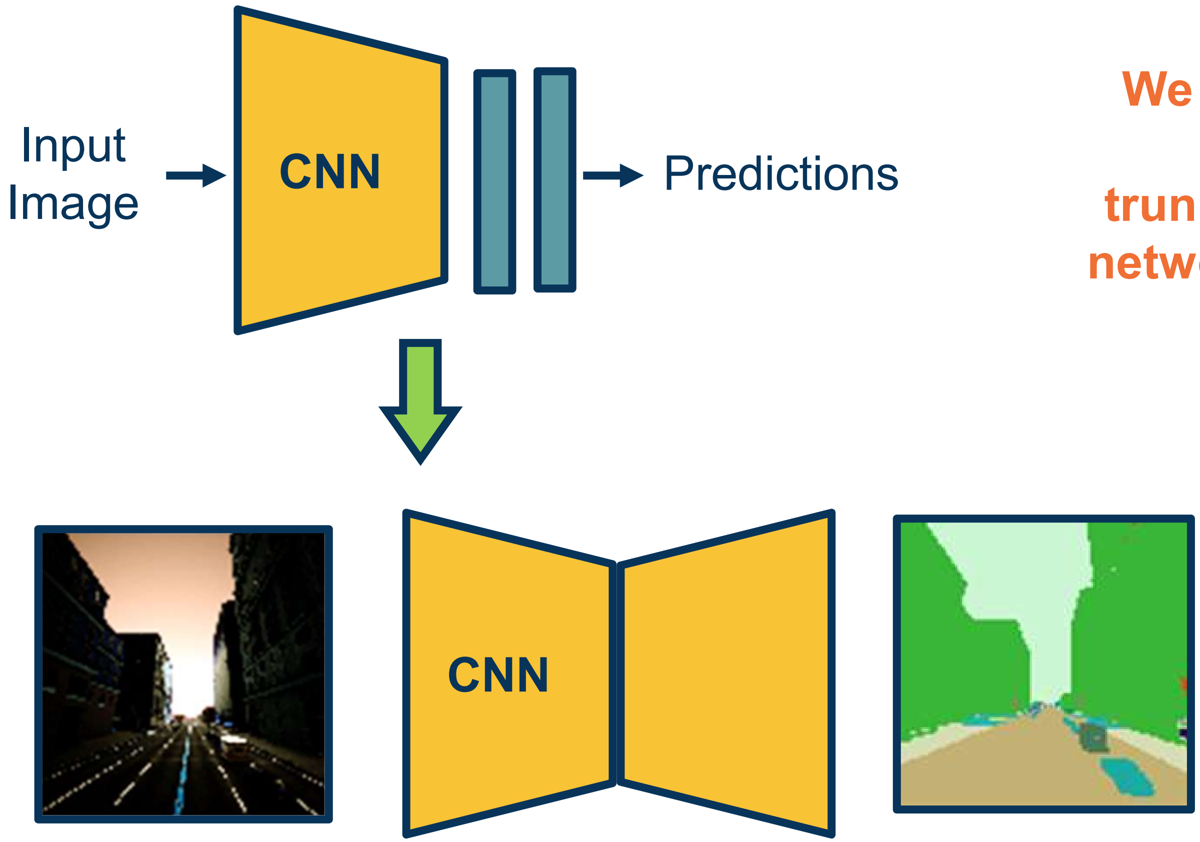
Transposed Convolution Example

Georgia Tech



Symmetry in Encoder/Decoder

Georgia Tech

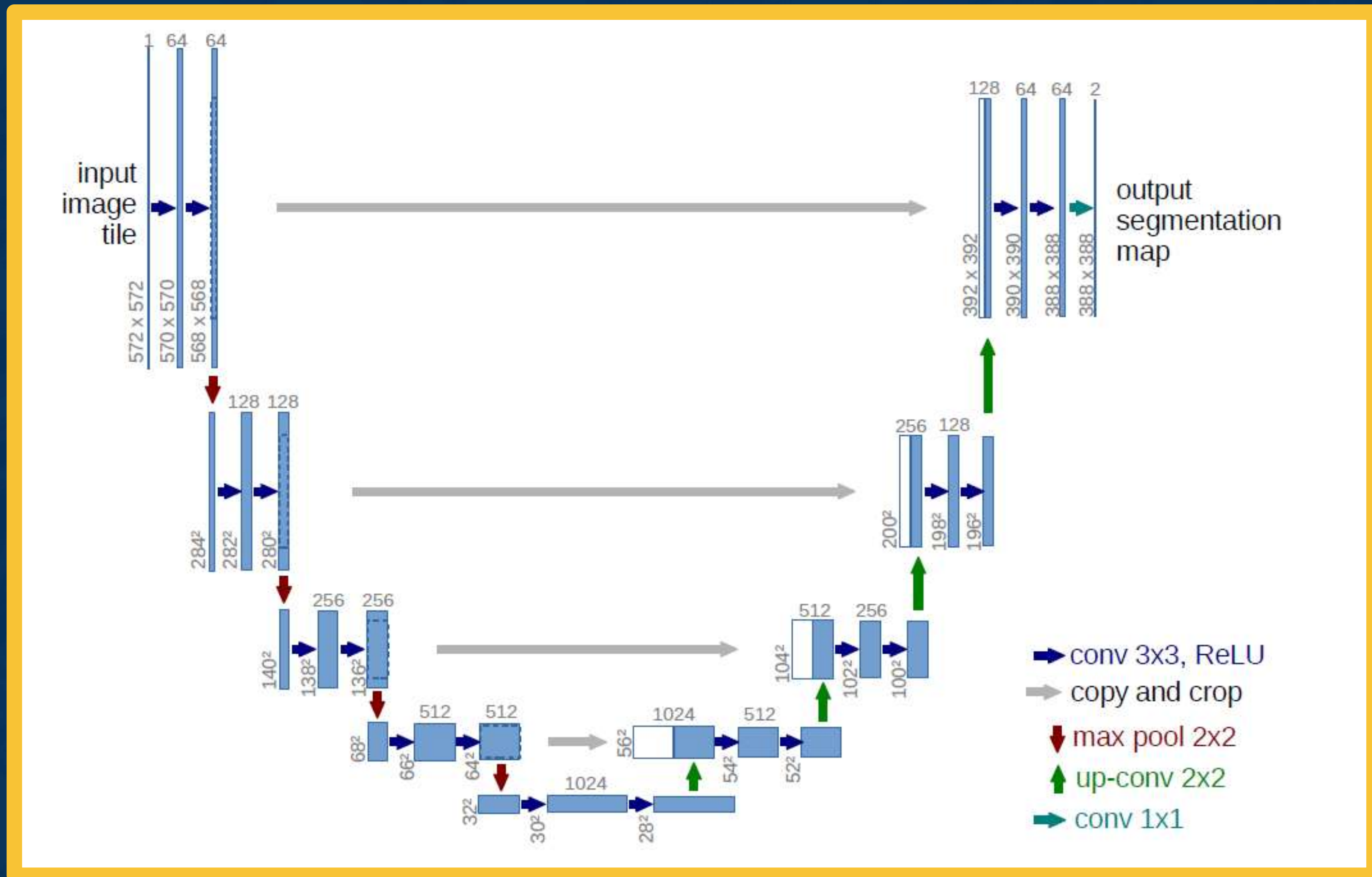


We can start with a pre-trained trunk/backbone (e.g. network pretrained on ImageNet)!

Transfer Learning

U-Net

You can have skip connections to bypass bottleneck!



Ronneberger, et al., "U-Net: Convolutional Networks for Biomedical Image Segmentation", 2015

Georgia Tech

Summary

- Various ways to get **image-like outputs,** for example to predict segmentations of input images
- Fully convolutional layers essentially apply the striding idea to the output classifiers, supporting arbitrary input sizes
  - (without output size depending on what the input size is)
- We can have various upsampling layers that actually increase the size
- Encoder/decoder architectures are popular ways to leverage these to perform general image-to-image tasks