Topics:

- Questions on convolution layers
- Visualization
• **Assignment 2**
  • Due in **2 days!!**

• **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
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**

- Simonyan et al, 2013

**Gradients**

- Hendrycks & Dietterich, 2019

**Robustness**
**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 Output Maps

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

Grad-CAM

What animal is in this picture? Cat

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

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

**Forward Pass**

\[
argmax S_c(I) - \lambda \|I\|_2^2
\]

**Backward Pass**

\[
\frac{\partial S_c}{\partial I} \frac{\partial I}{\partial I} = \alpha \frac{\partial S_c}{\partial I}
\]

Example Images

Note: You might have to squint!

Can improve results with various tricks:

- Clipping of small values & gradients
- Gaussian blurring
Improved Results

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

Rather than start from zero image, why not a real image?

And why not optimize the score of an arbitrary (incorrect!) class?

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

\[ \text{argmax} \ S_c(I) - \lambda \|I\|_2^2 \]

where \( c = \text{cat} \)

Example of Adversarial Noise

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!

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
Other Forms of Robustness Testing


\[ CE_c^f = \frac{\left( \sum_{s=1}^{5} E_{s,c}^f \right)}{\left( \sum_{s=1}^{5} E_{s,c}^{\text{AlexNet}} \right)} \]
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
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**
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

\[
L_{\text{content}} = (F^1_C - F^1_P)^2
\]
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!

\[
L_{\text{content}} = \sum_{\ell} (F^\ell_C - F^\ell_P)^2
\]
**Idea:** Can we have the content of one image and texture (style) of another image? 

Yes!
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

\[
G^\ell_S(i, j) = \sum_k F^\ell_S(i, k) F^\ell_S(j, k)
\]

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

\[
L_{style} = \sum_\ell \left( G^\ell_S - G^\ell_P \right)^2
\]

\[
L_{total} = \alpha L_{content} + \beta L_{style}
\]
Gradient Ascent on the Scores
Gradient Ascent on the Scores
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
Computer Vision Tasks

Classification
(Class distribution per image)

Semantic Segmentation
(Class distribution per pixel)

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

Instance Segmentation
(Class distribution per pixel with unique ID)
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)
Probability distribution over classes for this one pixel

Input & Output
Idea 1: Fully-Convolutional Network

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!
Converting FC Layers to Conv Layers

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: $H = 5$, $W = 5$
- Larger: $H = 7$, $W = 7$

Conv Kernel: $k_1 = 3$, $k_2 = 3$

Output

Fully Convolutional Layer Kernel

Georgia Tech
Why does this matter?
- We can stride the “fully connected” classifier across larger inputs!
- Convolutions work on arbitrary input sizes (because of striding)

Idea 2: “De”Convolution and UnPooling

Convolutional Neural Network (CNN)

Encoder

Image

Convolution + Non-Linear Layer

Pooling Layer

Convolution + Non-Linear Layer

Useful, lower-dimensional features

Decoder

(De)Convolution + Non-Linear Layer

(De)Convolution + Non-Linear Layer

(De)Convolution + Non-Linear Layer

(Un)Pooling Layer

Useful, lower-dimensional features

“Image”

We can develop learnable or non-learnable upsampling layers!
Example: Max pooling

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

\[ X(0:1, 0:1) = \begin{bmatrix} 100 & 150 \\ 100 & 200 \end{bmatrix} \quad \rightarrow \quad \text{max}(0:1,0:1) = 200 \]

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

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

Max Unpooling
Max Unpooling Example (one window)

Encoder

\[
X = \begin{bmatrix}
120 & 150 \\
100 & 50 \\
25 & 25 \\
110 & 10 \\
\end{bmatrix}
\]

2x2 max pool

Y = \[\begin{bmatrix}
150 & 150 \\
100 & 110 \\
\end{bmatrix}\]

Decoder

\[
X = \begin{bmatrix}
300 & 450 \\
100 & 250 \\
\end{bmatrix}
\]

2x2 max unpool

Y = \[\begin{bmatrix}
0 & 300 & - \\
0 & 0 & - \\
- & - & - \\
\end{bmatrix}\]
Contributions from multiple windows are summed.
We pull max indices from corresponding layers (requires symmetry in encoder/decoder).
How can we *upsample* using convolutions and learnable kernel?

**Normal Convolution**

- $H = 5$
- $W = 5$
- $k_1 = 3$
- $k_2 = 3$
- $H - k_1 + 1$
- $W - k_2 + 1$

**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)
Contributions from multiple windows are summed.
We can either learn the kernels, or take corresponding encoder kernel and rotate 180 degrees (no decoder learning)
We can start with a pre-trained trunk/backbone (e.g. network pretrained on ImageNet)!
U-Net

You can have skip connections to bypass bottleneck!

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