Project 4

• Part A was due last Thursday → checkpoint graded on completion
  – Update: slip days won’t be applied on part A
• Part B due next *Monday*
• Final submission includes both A & B
• if you have a bug in part A do fix it for Part B
Project 5

• Will be released Wednesday after Project 4 B is due.
• Go to this Thursday’s OH for PyTorch review!!!
• Very important that you go to this if you have never done anything with PyTorch.
Project 2 Class Choice Award Winner!

Fun with Filters and Frequencies!
Author: Skylar Sarabia

Runner ups: Anik Gupta, Erich Liang, Andrew Zhang
Vanilla (fully-connected) Neural Networks
Fully Connected Layer

Example: 200x200 image
40K hidden units

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..
Locally Connected Layer

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).
Locally Connected Layer

STATIONARITY? Statistics is similar at different locations

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters
Convolutional Layer

Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer

Ranzato
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer

Ranzato
Convolution Layer

One neuron, that looks at 5x5 region and outputs a sheet of activation map
**Convolution Layer**

**One neuron**, that looks at 5x5 region and outputs a sheet of activation map:

- **32x32 image**
- **3x5x5 filter**
- **convolve (slide) over all spatial locations**

**1x28x28 activation map**

Convolution Layer vs Image Filtering:
- >1 input and output channels
- Forget about convolution vs cross-correlation (Why?)

3x32x32 image

3x5x5 filter

convolve (slide) over all spatial locations

1x28x28 activation map

28

28

1
Convolution Layer:

2 neurons (aka filters)

3x32x32 image

3x5x5 filter

Add a second (green) filter:

convolve (slide) over all spatial locations

two 1x28x28 activation map
Convolution Layer

6 neurons (aka filters)

3x32x32 image

Can keep on adding new, say 6 filters, each 3x5x5

6x3x5x5 filters

Convolution Layer

6 activation maps, each 1x28x28

Stack activations to get a 6x28x28 output image!
Convolution Layer

3x32x32 image

Also 6-dim bias vector:

Convoluted Layer

6x3x5x5 filters

Stack activations to get a 6x28x28 output image!

28x28 grid, at each point a 6-dim vector
Convolution Layer: With many images

- Batch of images: 2x3x32x32
- Convolution Layer: 6x3x5x5 filters
- Also 6-dim bias vector:
- Batch of outputs: 2x6x28x28
**Convolution Layer**

- **Batch of images**: $N \times C_{in} \times H \times W$
- **Bias**: $C_{out}$-dim vector
- **Weights**: $C_{out} \times C_{in} \times K_w \times K_h$
- **Batch of outputs**: $N \times C_{out} \times H' \times W'$
The brain/neuron view of CONV Layer

E.g. with 5 filters, CONV layer consists of neurons arranged in a 3D grid (28x28x5)

There will be 5 different neurons all looking at the same region in the input volume
convolving the first filter in the input gives the first slice of depth in output volume.
CNNs in practice

Conv layers (they still need to go through non-linearity!!)

FC layers (MLPs)

Vectorize / flatten

output
What needs to be learned?

Conv layers (they still need to go through non-linearity!!)

Vectorize / flatten

FC layers (MLPs)

output
What needs to be learned?

Input:
B, C_in, H, W

Conv, Relu

Output:
B, C_out, H, W

W
C_out, C_in, K, K
We’re still doing matrix multiplications, just localized & shared

Recall one neuron in FC layer:

With Conv layer:

w takes the entire image!

Now w takes (overlapping) patches
CONVNET NITTY GRITTIES
Convolution Spatial Dimensions

Input: 7x7
Filter: 3x3

Q: How big is output?
Convolution Spatial Dimensions

Input: 7x7
Filter: 3x3
Q: How big is output?
Convolution Spatial Dimensions

Input: 7x7
Filter: 3x3

Q: How big is output?
Convolution Spatial Dimensions

Input: 7x7
Filter: 3x3

Q: How big is output?
Convolution Spatial Dimensions

Input: 7x7
Filter: 3x3
Output: 5x5
Convolution Spatial Dimensions

Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Output: W – K + 1

Problem:
Feature maps “shrink” with each layer!
Convolution Spatial Dimensions

Input: 7x7
Filter: 3x3
Output: 5x5

Problem: Feature maps “shrink” with each layer!

Solution: padding
Add zeros around the input

In general:
Input: W
Filter: K
Padding: P
In general:
Input: \( W \)
Filter: \( K \)
Padding: \( P \)
Output: \( W - K + 1 + 2P \)

Very common: “same padding”
Set \( P = (K - 1) / 2 \)
Then output size = input size
Receptive Fields

For convolution with kernel size $K$, each element in the output depends on a $K \times K$ receptive field in the input.
Receptive Fields

Each successive convolution adds $K - 1$ to the receptive field size.

With $L$ layers the receptive field size is $1 + L \times (K - 1)$

Careful – “receptive field wrt to the input”

vs “receptive field wrt the previous layer”
Receptive Fields

Each successive convolution adds $K - 1$ to the receptive field size
With $L$ layers the receptive field size is $1 + L \times (K - 1)$

**Problem:** For large images we need many layers for each output to “see” the whole image.
Receptive Fields

Each successive convolution adds $K - 1$ to the receptive field size.
With $L$ layers the receptive field size is $1 + L \times (K - 1)$

**Problem:** For large images we need many layers for each output to “see” the whole image.

**Solution:** Downsample inside the network.
Strided Convolution

Input: 7x7
Filter: 3x3
Stride: 2
Strided Convolution

Input: 7x7
Filter: 3x3
Stride: 2
Strided Convolution

Input: 7x7
Filter: 3x3  Output: 3x3
Stride: 2
**Strided Convolution**

Input: 7x7  
Filter: 3x3  
Stride: 2  
Output: 3x3

In general:
- Input: \(N\)
- Filter: \(K\)
- Stride: \(S\)
- With padding: \(P\)
- Output: \((N - K + 2P) / S + 1\)
Pooling Layers: Downsampling

Another way to reduce size while aggregating (pooling) information

Has a choice of:
- Kernel Size
- Stride
- Pooling function
Max Pooling

Introduces invariance to small spatial shifts
No learnable parameters!

Max pooling with 2x2 kernel size and stride 2

Single depth slice

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

64 x 224 x 224
Average Pooling

Introduces invariance to small spatial shifts
No learnable parameters!
Pooling Summary

**Input**: \(C \times H \times W\)

**Hyperparameters**:
- Kernel size: \(K\)
- Stride: \(S\)
- Pooling function (max, avg)

**Output**: \(C \times H' \times W'\) where
- \(H' = (H - K) / S + 1\)
- \(W' = (W - K) / S + 1\)

**Learnable parameters**: None!

Common settings:
- max, \(K = 2, S = 2\)
- max, \(K = 3, S = 2\) (AlexNet)
Normalization

Why do we need to normalize our data?

Extreme example:
- sometimes pixel range is $[0, 255]$, sometimes it’s $[0, 1]$ 😳

What is the problem?

Network activations will be completely different!!

You want the inputs to be in a similar range $\rightarrow$ low variance
How to normalize

\[ \hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}, \]

Is this differentiable?

Yes! so we can use it as an operator in our networks and backprop through it!

But now the data is always zero mean, unit variance…

Sol: Add scale and shift parameters: \( \gamma, \beta \)

\[ y = \gamma \hat{x} + \beta \]

Learning \( \gamma = \sigma, \beta = \mu \)

will recover the identity function (in expectation)

Slide modified from David Fouhey
Normalization

Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

\[
\hat{x} = \frac{x - E[x]}{\sqrt{\text{Var}[x]}}
\]
How to normalize

Next Q: from what do you compute the mean & variance?

- BatchNorm computes mean and var from each batch
- Depends on the batch, so need to keep a running average and store these.
- LayerNorm/InstanceNorm do not need this.
Batch Normalization for ConvNets

**Batch Normalization for fully-connected networks**

\[ x : N \times C \]

Normalize

\[ \mu, \sigma : 1 \times C \]
\[ \gamma, \beta : 1 \times C \]

\[ y = \frac{(x - \mu)}{\sigma} \gamma + \beta \]

**Batch Normalization for convolutional networks**

(Spatial Batchnorm, BatchNorm2D)

\[ x : N \times C \times H \times W \]

Normalize

\[ \mu, \sigma : 1 \times C \times 1 \times 1 \]
\[ \gamma, \beta : 1 \times C \times 1 \times 1 \]

\[ y = \frac{(x - \mu)}{\sigma} \gamma + \beta \]
Case Study: Lenet-5

Task: 10 digit classification

LeCun et al. 1998

Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-flatten-FC-output]
LeNet5 demo

LeCun et al. 1998
Case Study: AlexNet

Task: ImageNet 1000-class classification

<table>
<thead>
<tr>
<th>HxW</th>
<th>C</th>
<th>Input</th>
<th>Conv 1</th>
<th>Conv 2</th>
<th>Conv 3</th>
<th>Conv 4</th>
<th>Conv 5</th>
<th>FC 6</th>
<th>FC 7</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>227x227</td>
<td>3</td>
<td>227x227</td>
<td>55x55</td>
<td>27x27</td>
<td>13x13</td>
<td>13x13</td>
<td>13x13</td>
<td>1x1</td>
<td>1x1</td>
<td>1x1</td>
</tr>
<tr>
<td>227</td>
<td>73</td>
<td>227x227</td>
<td>55x55</td>
<td>27x27</td>
<td>13x13</td>
<td>13x13</td>
<td>13x13</td>
<td>1x1</td>
<td>1x1</td>
<td>1000</td>
</tr>
</tbody>
</table>

Each block is a HxWxC volume.

You transform one volume to another with convolution.
Input: 227x227x3 images

**First layer** (CONV1): 96 11x11 filters applied at stride 4

Q: What is the output volume size after Conv1?  
A: 55x55x96

Q: how many parameters in this layer?  
A: \((3 \times 11 \times 11) \times 96\)
Input: 227x227x3 images
After Conv1: 55x55x96

**Second layer:** Pool1: 3x3 at stride 2

**Q:** What is the output volume size after Pool1?  
**A:** (55-3)/2 + 1 = 27  
27x27x96

**Q:** how many parameters in this layer?  
**A:** 0!!!
Q: Which part of the network incurs high memory usage? Large # of parameters? high FLOPs?

Most of the memory usage is in the early convolution layers. Nearly all parameters are in the fully-connected layers. Most floating-point ops occur in the convolution layers.
class AlexNet(nn.Module):
    def __init__(self, num_classes: int = 1000, dropout: float = 0.5) -> None:
        super(AlexNet, self).__init__()
        _log_api_usage_once(self)
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(p=dropout),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(p=dropout),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )
VGGNet [Simonyan and Zisserman 2015]

Input: 224x224x3

**Simplified** design rules:

- All kernel size 3x3
- Always ReLu
- All max pool are 2x2, stride 2
- After pool, double the # of channels

<table>
<thead>
<tr>
<th>Table 2: <strong>Number of parameters</strong> (in millions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
</tr>
<tr>
<td>Number of parameters</td>
</tr>
</tbody>
</table>

The ReLU activation function is not shown for brevity.

<table>
<thead>
<tr>
<th>ConvNet Configuration</th>
<th>A</th>
<th>A-LRN</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 weight layers</td>
<td>11 weight layers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-64</td>
<td>conv3-64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LRN</td>
<td>LRN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-256</td>
<td>conv3-256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool</td>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-128</td>
<td>conv3-128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-128</td>
<td>conv3-128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-256</td>
<td>conv3-256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-256</td>
<td>conv3-256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool</td>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool</td>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool</td>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool</td>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool</td>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool</td>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FC-4096</td>
<td>FC-1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FC-4096</td>
<td>FC-4096</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>soft-max</td>
<td>soft-max</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
VGGNet [Simonyan and Zisserman 2015]

Input: 224x224x3

**Simplified** design rules:
- **All kernel size 3x3**
- Always ReLu
- All max pool are 2x2, stride 2
- After pool, double the # of channels

Two 3x3 conv has same receptive field as a single 5x5 conv, but has fewer parameters and takes less computation!

Option 1:
- Conv(5x5, C -> C)
- Params: $25C^2$
- FLOPs: $25C^2HW$

Option 2:
- Conv(3x3, C -> C) Conv(3x3, C -> C)
- Params: $18C^2$
- FLOPs: $18C^2HW$

Slide adapted from Johnson & Fouhey
Measuring performance over train/val sets
Measuring performance over train/val sets

![Graph showing accuracy over epoch for training and validation sets with different levels of overfitting.]

- **Training accuracy**
- **Validation accuracy:** little overfitting
- **Validation accuracy:** strong overfitting
Deeper network != Better performance

Q: is this over fitting? No! it’s a matter of optimization

ResNet [He et al. CVPR 2016]
Naively adding more layers $\neq$ better performance [He et al. CVPR 2016]

Deeper networks *should be* able to emulate shallow networks. How? By making the extra layers to learn the identity function
But they don’t seem to learn identity by default
ResNet [He et al. CVPR 2016]

34-layer plain

Input: 224x224x3

34-layer residual

ResNet: “Residual Learning”

ResNet Block:

\[ y = \mathcal{F}(x, \{W_i\}) + x. \]
ResNet [He et al. CVPR 2016]

Can be made very deep: ResNet-50, 101, 152
And converge well

Bottleneck design:
Reduce by 1x1
Conv by 3x3
Restore by 1x1
ImageNet Classification Challenge top-5 error

In 2021, Top-5 is .98 Florence [Yuan et al. 2021]


Error Rate

0 5 10 15 20 25 30

28.2 25.8 16.4 11.7 7.3 6.7 3.6 3 2.3 5.1

Shallow 8 Layer 8 Layer 19 Layer 22 Layer 152 Layer 152 Layer 152 Layer

Bottom line: CNN workflow

Input: \(H \times W \times C\)

Output: \(1 \times 1 \times F\)

Convert \(H \times W\) image into a \(F\)-dimensional vector

- What’s the probability this image has a cat? (\(F=1\))
- Which of 1000 categories is this image? (\(F=1000\))
- Where are the \(X, Y\) coordinates of 5 cat face keypoints? (\(F=28 \times 2 = 56\))

Slide modified from David Fouhey
Example: Image classification

Input: HxWxC

CNN

Output: 1 x 1 x F

P(image is class #1)

P(image is class #2)

P(image is class #F)

Slide modified from David Fouhey
Example: Keypoint detection

Input: HxWxC

CNN

Output:
1 x 1 x 2K

X coord of keypoint #1
Y coord of keypoint #1
X coord of keypoint #K
Y coord of keypoint #K

Slide modified from David Fouhey
How to train your network
Training diagnosis

Basic setup

1. Get your dataset
2. Design your CNN architecture (start by reusing (or simplified version of) AlexNet/ResNet)
3. Train + update weights with pytorch/tf…
Sanity Check #1

Make sure you can memorize a single data point: \((x_i, y_i)\).

- Train your CNN on only 1 image
- It **must** be able to get very low training loss

If you can’t do this, it means something is wrong in your loss function, label, network/pytorch setup

It should also overfit to a single data fairly quickly. If it takes many iterations to do this that’s also bad news bears
What if your data is not big enough?

- Horizontal Flip
- Color Jitter
- Image Cropping
Data Augmentation

- Apply transformations (on the fly) to increase training data
- Can mix multiple transformations at once
- Make sure you don’t change the meaning of the output
Sometimes labels need to be transformed too

For certain tasks, make sure to transform the output accordingly!!

Original  After flip + color aug

Right eye  Left eye  Defined wrt to the cat
Sometimes labels need to be transformed too

For certain tasks, make sure to transform the output accordingly!!

- Right eye
- Left eye

Defined wrt to the cat
Very common bug

For certain tasks, make sure to transform the output accordingly!!

Original

After flip + color aug

WRONG! Flipped without updating the label!!

Right eye  Left eye

Defined wrt to the cat
Very common bug

For certain tasks, make sure to transform the output accordingly!!

Original

After flip + color aug

Right eye  Left eye
Defined wrt to the cat

Correct
A good starting point

DeepPose: Human Pose Estimation via Deep Neural Networks
[Toshev and Szegedy 2014]
Regression Objective

This is what you implement in part 1. But this is not what the SoTA models do in practice.
Downsides of regression objective

Locally a lot of things look similar!!

With regression objective you have to commit to ONE location and only get one training signal on how correct that location was.
For K keypoints, train model to predict K many sheets (h x w) of scores of how likely the pixel is k-th keypoint
Problem

So far, we’ve only seen examples that output a vector representation out of an image.

How do we do dense (per-pixel) predictions?
Dense Prediction

Needed for many things:

Semantic Segmentation

GRASS, CAT, TREE, SKY

Keypoint detection

Does this pixel contain right wrist?

Image Synthesis!

(next lecture)
Idea 1: Dense prediction by Sliding Window

What is the Problem?

Very inefficient! Not reusing shared features between overlapping patches
Instead, make the whole network convolutional!

Think of the last few fully connected network as a convolutions with kernels (receptive field) that cover the entire image ➔ can turn it into 1x1 convs
Fully convolutional networks

- Design a network with only convolutional layers, make predictions for all pixels at once

Convert all FC into $1 \times 1$ convolutions


Slide courtesy of Lana Lezebnik
Fully convolutional networks

- Design a network with only convolutional layers, make predictions for all pixels at once.

- Ideally, we want convolutions at full image resolution, but implementing that naively is too expensive.
Fully convolutional networks

- Design a network with only convolutional layers, make predictions for all pixels at once
- Ideally, we want convolutions at full image resolution, but implementing that naively is too expensive
  - Solution: first downsample, then **upsample**

Source: Stanford CS231n
Slide courtesy of Lana Lezebnik
HOW TO UPSAMPLE WITH CONVNETS?
Simple solution

- Upsample, followed by a regular Convolution

Input: $B \times C_{\text{in}} \times H \times W$

After upsample, factor 2: $B \times C_{\text{in}} \times 2H \times 2W$

After conv: $B \times C_{\text{out}} \times 2H \times 2W$
How to Upsample

Bed of Nails

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Input: $C \times 2 \times 2$
Output: $C \times 4 \times 4$

Nearest Neighbor

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Input: $C \times 2 \times 2$
Output: $C \times 4 \times 4$

Bilinear Interpolation

| 1.00 | 1.25 | 1.75 | 2.00 |
| 1.50 | 1.75 | 2.25 | 2.50 |
| 2.50 | 2.75 | 3.25 | 3.50 |
| 3.00 | 3.25 | 3.75 | 4.00 |

Input: $C \times 2 \times 2$
Output: $C \times 4 \times 4$
Recall from Morphing Lecture: Inverse warping

Don’t splat! Do inverse warping.
You know this from project 3
Recall: Bilinear Interpolation

http://en.wikipedia.org/wiki/Bilinear_interpolation
Help interp2
Application to pose detection:
Predict heat maps

Heat map prediction

Fully Convolutional Networks

K+1 for K parts + background

Target: K+1 x H x W Gaussian around (x,y) for k-th keypoint in the k-th channel
You will implement this in project 5!

**L2 Training Loss**

- L2 loss on the target heatmap (peaky gaussian around the gt keypoint)

\[
L = \sum_{k=1}^{K+1} \sum_{(x,y)} \| b^k(x, y) - b^*_k(x, y) \|
\]

Target “belief map”:

- \( K+1 \times H \times W \)
- Gaussian around \((x,y)\) for k-th keypoint in the k-th channel
Log Loss Training Loss

• Log loss (or cross entropy loss) on the target heatmap probabilities
• The target must also sum to 1
• Mask RCNN just uses 1 at the target, 0 everywhere else.
• Experiment

Target “belief map” : $K+1 \times H \times W$
1 at Ground truth location (x,y) for k-th keypoint in the k-th channel
Convolutional Pose Machines

Base architecture for OpenPose

Convolutional Pose Machines ($T$-stage)

(a) Stage 1

(b) Stage $\geq 2$

(c) Stage 1

(d) Stage $\geq 2$

(e) Effective Receptive Field

[Wei et al CVPR 2016]
Convolutional Pose Machines

[Wei et al CVPR 201]
Convolutional Pose Machines
Results
OpenPose

Great opensource tool, builds on convolutional pose machine architecture, adapted to multiple people.
Other dense prediction tasks

- Depth estimation
- Surface normal estimation
- Colorization
- Image synthesis (next week)
- ....
Depth and normal estimation


Slide courtesy of Lana Lezebnik
Depth and normal estimation

D. Eigen and R. Fergus, Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture, ICCV 2015
Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer
René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, Vladlen Koltun, PAMI 2020
Colorization (self-supervised learning)

Grayscale image: $L$ channel

$$X \in \mathbb{R}^{H \times W \times 1}$$

Concatenate $(L, ab)$ channels

$$(X, \hat{Y})$$

Examples for segmentation: SegNet


Drop the FC layers, get better results
Other upsampling networks: U-Net

• Fuse upsampled higher-level feature maps with lower-level feature maps (via upsampling)
• Fuse by concatenation, predict at the end

O. Ronneberger, P. Fischer, T. Brox *U-Net: Convolutional Networks for Biomedical Image Segmentation*, MICCAI 2015

Slide courtesy of Lana Lezebnik
Summary of upsampling architectures

Slide courtesy of Lana Lezebnik