Figure from Convolutional Pose Machines, Wei et al. CVPR 2016

CS194: Computer Vision and Comp. Photo
Angjoo Kanazawa/Alexei Efros, UC Berkeley, Fall 2021
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>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>55x55</td>
<td>96</td>
<td>27x27</td>
<td>256</td>
<td>13x13</td>
<td>384</td>
<td>13x13</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1x1</td>
<td>4096</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1x1</td>
<td>4096</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1x1</td>
<td>1000</td>
</tr>
</tbody>
</table>

Each block is a HxWxC volume.
You transform one volume to another with convolution
Case Study: AlexNet

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: \((3\times11\times11) \times 96\)

Hint: \((N-K)/S + 1\)
\((227-11)/4 + 1 = 55\)
Case Study: AlexNet

Input: 227x227x3 images
After Conv1: 55x55x96

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

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

Q: how many parameters in this layer?  
A: 0!!!
AlexNet

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

Memory (KB)

Params (K)

MFLOP
In pytorch

https://github.com/pytorch/vision/blob/main/torchvision/models/alexnet.py#L18

```python
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 2: Number of parameters (in millions).

<table>
<thead>
<tr>
<th>Network</th>
<th>A, A-LRN</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of parameters</td>
<td>133</td>
<td>133</td>
<td>134</td>
<td>138</td>
<td>144</td>
</tr>
</tbody>
</table>

ConvNet Configuration

<table>
<thead>
<tr>
<th></th>
<th>A</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>13 weight layers</td>
<td>16 weight layers</td>
<td>16 weight layers</td>
<td>19 weight layers</td>
</tr>
<tr>
<td>input (224 x 224 RGB image)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-64</td>
<td>conv3-64-LRN</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
</tr>
<tr>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
</tr>
<tr>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
</tr>
<tr>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
</tr>
<tr>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FC-4096</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FC-4096</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FC-1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<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

Naively adding more layers != better performance

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

Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

CVPR 2016]
Naively adding more layers != better performance

[He et al. CVPR 2016]

Deeper networks can 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

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

Error Rate

- 2010: 28.2
- 2011: 25.8
- 2012: 16.4
- 2013: 11.7
- 2014: 7.3
- 2014: 6.7
- 2015: 3.6
- 2016: 3
- 2017: 2.3
- Human: 5.1

- Lin et al
- Sanchez & Perronnin
- Krizhevsky et al (AlexNet)
- Zeiler & Fergus
- Simonyan & Zisserman (VGG)
- Szegedy et al (GoogLeNet)
- He et al (ResNet)
- Shao et al
- Hu et al (SENet)
- Russakovsky et al

Lecture 16 - EECS 442 WI 2021

March 16, 2021
How to train your network
Training diagnosis

What if your data is not big enough?
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!!

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!!

Right eye  Left eye
Defined wrt to the cat

Original

After flip + color aug

Correct
“You need a lot of a data if you want to train/use CNNs”
Transfer Learning

“You need a lot of data if you want to train the CNN

NOT ALWAYS
Transfer Learning with CNNs

1. Train on Imagenet
Transfer Learning with CNNs

1. Train on Imagenet

2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

i.e. swap the Softmax layer at the end
Transfer Learning with CNNs

1. Train on Imagenet

2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier
   i.e. swap the Softmax layer at the end

3. If you have medium sized dataset, “finetune” instead: use the old weights as initialization, train the full network or only some of the higher layers
   retrain bigger portion of the network, or even all of it.
Transfer Learning with CNNs

1. Train on Imagenet

2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier
   i.e. swap the Softmax layer at the end

3. If you have medium sized dataset, “finetune” instead: use the old weights as initialization, train the full network or only some of the higher layers
   retrain bigger portion of the network, or even all of it.

   tip: use only ~1/10th of the original learning rate in finetuning to player, and ~1/100th on intermediate layers
Project 5 starting point (default)

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

This is what you should do before bells & whistles. 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

Keypoint detection

Does this pixel contain right wrist?

GRASS, CAT, TREE, SKY

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

What is the Problem?

Very inefficient! Not reusing shared features between overlapping patches

Farabet et al., "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014
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 1x1 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

Source: Stanford CS231n
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
- Solution: first downsample, then **upsample**
HOW TO UPSAMPLE WITH CONVNETS?
There are a few ways

• I will begin by the first thing people tried
• Then a much simpler, natural solution that’s more used in practice
Upsampling in a deep network

- **Transposed** or **backwards-strided** convolution: “paint” in the output feature map with the learned filter
  - Multiply input value by filter, place result in the output, sum overlapping value

Animation: https://github.com/vdumoulin/conv_arithmetic
Upsampling in a deep network

- Filter \([w_1, w_2, w_3]\), output stride = 2

Animation: https://distill.pub/2016/deconv-checkerboard/

Slide courtesy of Lana Lezebnik
Upsampling in a deep network

- Filter \([w_1, w_2, w_3]\), output stride = 2

Animation: [https://distill.pub/2016/deconv-checkerboard/](https://distill.pub/2016/deconv-checkerboard/)

Slide courtesy of Lana Lezebnik
Upsampling in a deep network

- Filter $[w_1, w_2, w_3]$, output stride = 2

Animation: [https://distill.pub/2016/deconv-checkerboard/](https://distill.pub/2016/deconv-checkerboard/)

Slide courtesy of Lana Lezebnik
Upsampling in a deep network

- Filter $[w_1, w_2, w_3]$, output stride $= 2$

Animation: [https://distill.pub/2016/deconv-checkerboard/](https://distill.pub/2016/deconv-checkerboard/)

Slide courtesy of Lana Lezebnik
Upsampling in a deep network

- Filter \([w_1, w_2, w_3]\), output stride = 2

Animation: [https://distill.pub/2016/deconv-checkerboard/](https://distill.pub/2016/deconv-checkerboard/)

Slide courtesy of Lana Lezebnik
But this is generates checkerboard like artifacts!

Because of overlap.

You can try to prevent overlap (play around with the distill page), but it gets complicated with multiple layers.
Simpler, effective solution
(often used in practice)

- Upsample, followed by a regular Convolution

Input:
\[ B \times C_{in} \times H \times W \]

After upsample, factor 2:
\[ B \times C_{in} \times 2H \times 2W \]

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Input C x 2 x 2</th>
<th>Output C x 4 x 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bed of Nails</td>
<td>1 2 0 0</td>
<td>1 0 2 0</td>
</tr>
<tr>
<td>Nearest Neighbor</td>
<td>0 0 0 0</td>
<td>1 1 1 1</td>
</tr>
<tr>
<td>Bilinear Interpolation</td>
<td>3 4 4 4</td>
<td>3 3 4 4</td>
</tr>
<tr>
<td></td>
<td>0 0 0 0</td>
<td>3 3 4 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.50 2.75 3.25 3.50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.00 3.25 3.75 4.00</td>
</tr>
</tbody>
</table>

Input C x 2 x 2

Output C x 4 x 4
Recall from Morphing Lecture: Inverse warping

Get each pixel $g(x',y')$ from its corresponding location $(x,y) = T^{-1}(x',y')$ in the first image.

Q: what if pixel comes from “between” two pixels?
A: *Interpolate* color value from neighbors
   - nearest neighbor, bilinear, Gaussian, bicubic
   - Check out interp2 in Matlab / Python
Recall: Bilinear Interpolation

http://en.wikipedia.org/wiki/Bilinear_interpolation
Help interp2
Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

**Fully Convolutional Network**

Input: $3 \times H \times W$

High-res: $D_1 \times H/2 \times W/2$

Med-res: $D_2 \times H/4 \times W/4$

Low-res: $D_3 \times H/4 \times W/4$

High-res: $D_1 \times H/2 \times W/2$

Predictions: $H \times W$

**Downsampling:**
Pooling, strided convolution

**Upsampling:**
Various choices

Rule of thumb: every upsample is a factor of 2

---


Application to pose detection:
Predict heat maps

K+1 for K parts + background
Training Loss

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

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

- Mask RCNN applies softmax across a heatmap and trains with cross-entropy
Convolutional Pose Machines
Base architecture for OpenPose

(a) Stage 1

Convolutional Pose Machines
($T$-stage)

<table>
<thead>
<tr>
<th>P</th>
<th>Pooling</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Convolution</td>
</tr>
</tbody>
</table>

(b) Stage $\geq 2$

Loss $f_1$

Input Image $h \times w \times 3$

(c) Stage 1

9x9 C 2x P 9x9 C 2x P 9x9 C 2x P 5x5 C 9x9 C 1x1 C 1x1 C

(d) Stage $\geq 2$

Loss $f_2$

Input Image $h' \times w' \times 3$

(e) Effective Receptive Field

9 x 9 26 x 26 60 x 60 96 x 96 160 x 160 240 x 240 320 x 320 400 x 400
Convolutional Pose Machines

stage 1

stage 2

stage 3

R. Elbow  R. Shoulder  Neck  Head

R. Elbow  R. Elbow
Convolutional Pose Machines

![Diagram showing pose estimation over time for different body parts (left and right wrists and elbows)]
Results
OpenPose

Great opensource tool, builds on convolutional pose machine architecture, adapted to multiple people

Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, Yaser Sheikh ‘16-17
Examples for segmentation: SegNet

Drop the FC layers, get better results

Other upsampling networks: U-Net

• Like FCN, fuse upsampled higher-level feature maps with higher-res, lower-level feature maps
• Unlike FCN, fuse by concatenation, predict at the end

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

**Figure source**: Slide courtesy of Lana Lezebnik
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

Slide courtesy of Lana Lezebnik
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
\[ (X, \hat{Y}) \]