Instructor: Wei Xu
Ohio State University

[These slides were adapted from CS231n Computer Vision at Stanford and CS188 Intro to AI at UC Berkeley]
Face Detection

Face Detection, Viola & Jones, 2001

Image is public domain
Fast-forward to today: ConvNets are everywhere

[Faster R-CNN: Ren, He, Girshick, Sun 2015]
Self-driving Cars
Whale and Road Recognition

Whale recognition, Kaggle Challenge

Mnih and Hinton, 2010
And Many More ...

Fast-forward to today: ConvNets are everywhere

[Levy et al. 2016]

[Dieleman et al. 2014]

From left to right: public domain by NASA, usage permitted by ESA/Hubble, public domain by NASA, and public domain.

[Seranet et al. 2011]

[Ciresan et al.]
Image and Video Captioning

A white teddy bear sitting in the grass

A man in a baseball uniform throwing a ball

A woman is holding a cat in her hand

A man riding a wave on top of a surfboard

A cat sitting on a suitcase on the floor

A woman standing on a beach holding a surfboard

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]
Fast-forward to today: ConvNets are everywhere

[Toshev, Szegedy 2014]

Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Guo et al. 2014]
DeepDream and Style Transfer


Original image is CC0 public domain

Starry Night and Tree Roots by Van Gogh are in the public domain

Bokeh image is in the public domain

Stylized images copyright Justin Johnson, 2017; reproduced with permission

Gatys et al, “Controlling Perceptual Factors in Neural Style Transfer”, CVPR 2017
Image Classification & Retrieval

Fast-forward to today: ConvNets are everywhere

PASCAL Visual Object Challenge

(20 object categories)
[Everingham et al. 2006-2012]
ImageNet

22K categories and 14M images

- Animals
  - Bird
  - Fish
  - Mammal
  - Invertebrate
- Plants
  - Tree
  - Flower
  - Food
  - Materials
- Structures
  - Artifact
  - Tools
  - Appliances
  - Structures
- Person
- Scenes
  - Indoor
  - Geological Formations
  - Sport Activities

Deng, Dong, Socher, Li, Li, & Fei-Fei, 2009
The Image Classification Challenge:

- 1,000 object classes
- 1,431,167 images

Output:

- Scale
- T-shirt
- Steel drum
- Drumstick
- Mud turtle

Correct outputs: Scale, T-shirt, Steel drum, Drumstick, Mud turtle

Incorrect outputs: Scale, T-shirt, Giant panda, Drumstick, Mud turtle

Russakovsky et al. arXiv, 2014
ImageNet Large Scale Visual Recognition Challenge

The Image Classification Challenge:
1,000 object classes
1,431,167 images

Russakovsky et al. arXiv, 2014
ImageNet Large Scale Visual Recognition Challenge

**Year 2010**
- NEC-UIUC
  - Dense descriptor grid: HOG, LBP
  - Coding: local coordinate, super-vector
  - Pooling, SPM
  - Linear SVM
  - [Lin CVPR 2011]

**Year 2012**
- SuperVision
  - [Krizhevsky NIPS 2012]

**Year 2014**
- GoogLeNet
  - Pooling
  - Convolution
  - Softmax
  - Other
  - [Szegedy arxiv 2014]

- VGG
  - conv-64
  - conv-128
  - conv-256
  - conv-512
  - maxpool
  - [Simonyan arxiv 2014]

**Year 2015**
- MSRA
  - [He ICCV 2015]

---

Lion image by Swissfrog is licensed under CC BY 3.0

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.
Convolutional Neural Networks (CNN)

1998
LeCun et al.

Image Maps

Convolutions

Subsampling

10^6

# of transistors

10^7

# of pixels used in training

2012
Krizhevsky et al.

Input

Image Maps

Convolutions

Subsampling

Output

Fully Connected

10^9

# of transistors

10^14

GPUs

# of pixels used in training

Reproduced with permission.
Image Classification

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

---

cat
The Problem: Semantic Gap

An image is just a big grid of numbers between $[0, 255]$:

e.g. $800 \times 600 \times 3$

(3 channels RGB)
Challenges: Viewpoint variation

This image by Nikita is licensed under CC-BY 2.0

All pixels change when the camera moves!
Challenges: Illumination
Challenges: Deformation
Challenges: Occlusion
Challenges: Background clutter
Challenges: Intraclass variation
How do we do this?

Attempts have been made

John Canny, “A Computational Approach to Edge Detection”, IEEE TPAMI 1986

Find edges

Find corners

?
Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

Example training set

```python
def train(images, labels):
    # Machine learning!
    return model

def predict(model, test_images):
    # Use model to predict labels
    return test_labels
```
Linear Classifier

Image

Array of 32x32x3 numbers (3072 numbers total)

Array of 32x32x3 numbers (3072 numbers total)

\[ f(x, W) \]

10 numbers giving class scores

W

parameters or weights

Parametric Approach
Linear Classifier

Image

Array of $32 \times 32 \times 3$ numbers
(3072 numbers total)

$f(x, W) = Wx + b$

$10 \times 1$

$3072 \times 1$

$10 \times 1$

$10 \times 3072$

10 numbers giving class scores

$W$

parameters or weights

Array of $32 \times 32 \times 3$ numbers (3072 numbers total)
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)
Neural Networks

Linear score function:

2-layer Neural Network

\[ f = W x \]
\[ f = W_2 \max(0, W_1 x) \]
Convolutional Neural Networks (CNN)
Convolution Layer

32x32x3 image

- width: 32
- height: 32
- depth: 3
**Convolution Layer**

32x32x3 image

5x5x3 filter

**Convolve** the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolution Layer

32x32x3 image

5x5x3 filter

Filters always extend the full depth of the input volume

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolution Layer

32x32x3 image
5x5x3 filter \( w \)

1 number: the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

\[ w^T x + b \]
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
Consider a second, green filter.
For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps:

We stack these up to get a “new image” of size 28x28x6!
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

CONV, ReLU

e.g. 6 5x5x3 filters
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions

3 \times 32 \quad \rightarrow \quad \text{CONV, ReLU, e.g. 6 filters, size 5x5x3}

6 \times 28 \quad \rightarrow \quad \text{CONV, ReLU, e.g. 10 filters, size 5x5x6}

10 \times 24 \quad \rightarrow \quad \text{CONV, ReLU, e.g. 10 filters, size 5x5x6}

\ldots
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

[From recent Yann LeCun slides]
A closer look at spatial dimensions:

32x32x3 image

5x5x3 filter

convolve (slide) over all spatial locations

activation map
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

=> 5x5 output
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
applied with stride 2
=> 3x3 output!
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 3?
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied \textit{with stride 3}? doesn’t fit!
cannot apply 3x3 filter on 7x7 input with stride 3.
Output size:
\((N - F) / \text{stride} + 1\)

e.g. \(N = 7, F = 3\):

- stride 1 => \((7 - 3)/1 + 1 = 5\)
- stride 2 => \((7 - 3)/2 + 1 = 3\)
- stride 3 => \((7 - 3)/3 + 1 = 2.33 \)
In practice: Common to zero pad the border

E.g. input 7x7
3x3 filter, applied with stride 1
Pad with 1 pixel border => what is the output?

(recall:)
\[
\frac{N - F}{\text{stride}} + 1
\]
In practice: Common to zero pad the border

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

e.g. input 7x7
3x3 filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

7x7 output!
In practice: Common to zero pad the border

e.g. input 7x7
3x3 filter, applied with \textbf{stride 1}
\textbf{pad with 1 pixel} border \Rightarrow what is the output?

\textbf{7x7 output!}

in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. $F = 3$ \Rightarrow zero pad with 1
    $F = 5$ \Rightarrow zero pad with 2
    $F = 7$ \Rightarrow zero pad with 3
Remember back to…
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn’t work well.
Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?
Examples time:

Input volume: 32x32x3
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so 32x32x10
Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
Examples time:

Input volume: \textbf{32x32x3}

10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

each filter has \(5 \times 5 \times 3 + 1 = 76\) params \((+1\ \text{for bias})\)

\[ => 76 \times 10 = 760 \]
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters $K$,
  - their spatial extent $F$,
  - the stride $S$,
  - the amount of zero padding $P$.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - \[ W_2 = \frac{(W_1 - F + 2P)}{S} + 1 \]
  - \[ H_2 = \frac{(H_1 - F + 2P)}{S} + 1 \] (i.e. width and height are computed equally by symmetry)
  - \[ D_2 = K \]
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and $K$ biases.
- In the output volume, the $d$-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the $d$-th filter over the input volume with a stride of $S$, and then offset by $d$-th bias.
Common settings:

- $K = \text{(powers of 2, e.g. 32, 64, 128, 512)}$  
  - $F = 3$, $S = 1$, $P = 1$
  - $F = 5$, $S = 1$, $P = 2$
  - $F = 5$, $S = 2$, $P = \? \text{ (whatever fits)}$
  - $F = 1$, $S = 1$, $P = 0$

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters $K$
  - their spatial extent $F$
  - the stride $S$
  - the amount of zero padding $P$
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and $K$ biases.
- In the output volume, the $d$-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the $d$-th filter over the input volume with a stride of $S$, and then offset by $d$-th bias.
(btw, 1x1 convolution layers make perfect sense)

1x1 CONV with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)
two more layers to go: POOL/FC
Pooling layer
- makes the representations smaller and more manageable
- operates over each activation map independently:
MAX POOLING

Single depth slice

max pool with 2x2 filters and stride 2
• Accepts a volume of size $W_1 \times H_1 \times D_1$
• Requires three hyperparameters:
  ◦ their spatial extent $F$,
  ◦ the stride $S$,
• Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  ◦ $W_2 = (W_1 - F)/S + 1$
  ◦ $H_2 = (H_1 - F)/S + 1$
  ◦ $D_2 = D_1$
• Introduces zero parameters since it computes a fixed function of the input
• Note that it is not common to use zero-padding for Pooling layers
Common settings:

- \( F = 2, S = 2 \)
- \( F = 3, S = 2 \)

- Accepts a volume of size \( W_1 \times H_1 \times D_1 \)
- Requires three hyperparameters:
  - their spatial extent \( F \),
  - the stride \( S \),
- Produces a volume of size \( W_2 \times H_2 \times D_2 \) where:
  - \( W_2 = (W_1 - F)/S + 1 \)
  - \( H_2 = (H_1 - F)/S + 1 \)
  - \( D_2 = D_1 \)
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers
Fully Connected Layer (FC layer)
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks
Summary of ConvNets

- ConvNets stack CONV, POOL, FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like
  
  \[(\text{CONV-RELU})^N \text{-POOL?}]^M \text{-}(\text{FC-RELU})^K, \text{SOFTMAX}\]

  where $N$ is usually up to $\sim5$, $M$ is large, $0 \leq K \leq 2$.
  - but recent advances such as ResNet/GoogLeNet challenge this paradigm
Deep Learning

Revolution of Depth

ImageNet Classification top-5 error (%)

ILSVRC'15 ResNet 3.57
ILSVRC'14 GoogleNet 6.7
ILSVRC'14 VGG 7.3
ILSVRC'13 11.7
ILSVRC'12 AlexNet 16.4
ILSVRC'11 shallow 25.8
ILSVRC'10 shallow 28.2

(slots from Kaiming He’s recent presentation)
Deep Learning

Revolution of Depth

(Image from Kaiming He’s recent presentation)

Case Study: ResNet \cite{He2015}

ILSVRC 2015 winner (3.6% top 5 error)

2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet!
(even though it has 8x more layers)

(slide from Kaiming He’s recent presentation)
What’s going on inside ConvNets?

Input Image: 3 x 224 x 224

What are the intermediate features looking for?

Figure reproduced with permission.
First Layer: Visualize Filters

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

DenseNet-121:
64 x 3 x 7 x 7

Huang et al, “Densely Connected Convolutional Networks”, CVPR 2017
Visualize the filters/kernels (raw weights)

We can visualize filters at higher layers, but not that interesting

(These are taken from ConvNetJS CIFAR-10 demo)
4096-dimensional feature vector for an image (layer immediately before the classifier)

Run the network on many images, collect the feature vectors
Last Layer: Nearest Neighbors

Recall: Nearest neighbors in pixel space

Test image L2 Nearest neighbors in feature space

Figures reproduced with permission.
Last Layer: Dimensionality Reduction

Visualize the “space” of FC7 feature vectors by reducing dimensionality of vectors from 4096 to 2 dimensions

Simple algorithm: Principle Component Analysis (PCA)

More complex: t-SNE
Last Layer: Dimensionality Reduction

Van der Maaten and Hinton, “Visualizing Data using t-SNE”, JMLR 2008
Figure reproduced with permission.

See high-resolution versions at
http://cs.stanford.edu/people/karpathy/cnnembed/

https://cs.stanford.edu/people/karpathy/cnnembed/
https://projector.tensorflow.org/
Occlusion Experiments

Mask part of the image before feeding to CNN, draw heatmap of probability at each mask location

Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014
Maximally Activating Patches

Pick a layer and a channel; e.g. conv5 is 128 x 13 x 13, pick channel 17/128

Run many images through the network, record values of chosen channel

Visualize image patches that correspond to maximal activations

Figure copyright Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, 2015; reproduced with permission.
Intermediate features via (guided) backprop

Pick a single intermediate neuron, e.g. one value in 128 x 13 x 13 conv5 feature map

Compute gradient of neuron value with respect to image pixels

Images come out nicer if you only backprop positive gradients through each ReLU (guided backprop)

Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

Figure copyright Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, 2015; reproduced with permission.
Intermediate features via (guided) backprop

Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks". ECCV 2014
Figure copyright Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, 2015; reproduced with permission.
Visualizing CNN features: Gradient Ascent

\[(\text{Guided}) \text{ backprop:}\]
Find the part of an image that a neuron responds to

\[\text{Gradient ascent:}\]
Generate a synthetic image that maximally activates a neuron

\[I^* = \arg \max_I f(I) + R(I)\]

Neuron value \hspace{2cm} Natural image regularizer
Visualizing CNN features: Gradient Ascent

1. Initialize image to zeros

Repeat:
2. Forward image to compute current scores
3. Backprop to get gradient of neuron value with respect to image pixels
4. Make a small update to the image

\[
\arg\max_I \left[ S_c(I) - \lambda ||I||^2_2 \right]
\]

score for class c (before Softmax)
Visualizing CNN features: Gradient Ascent

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

Simple regularizer: Penalize L2 norm of generated image

Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.
Visualizing CNN features: Gradient Ascent

Use the same approach to visualize intermediate features

Figure copyright Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson, 2014. Reproduced with permission.
DeepDream: Amplify existing features

Rather than synthesizing an image to maximize a specific neuron, instead try to **amplify** the neuron activations at some layer in the network.

Choose an image and a layer in a CNN; repeat:

1. Forward: compute activations at chosen layer
2. Set gradient of chosen layer *equal to its activation*
3. Backward: Compute gradient on image
4. Update image

Mordvintsev, Olah, and Tyka, "Inceptionism: Going Deeper into Neural Networks", Google Research Blog. Images are licensed under CC-BY 3.0.
DeepDream: Amplify existing features

Rather than synthesizing an image to maximize a specific neuron, instead try to **amplify** the neuron activations at some layer in the network.

Choose an image and a layer in a CNN; repeat:

1. **Forward**: compute activations at chosen layer
2. **Set gradient of chosen layer** equal to its activation
3. **Backward**: Compute gradient on image
4. **Update image**

Equivalent to:

$$ I^* = \arg \max_I \sum_i f_i(I)^2 $$

Mordvintsev, Olah, and Tyka, “Inceptionism: Going Deeper into Neural Networks”, Google Research Blog. Images are licensed under CC-BY.
Facebook’s Blind Users

April 4, 2016

Using Artificial Intelligence to Help Blind People ‘See’ Facebook

By Shaomei Wu, Software Engineer and Hermes Pique, Software Engineer on iOS and Jeffrey Wieland, Head of Accessibility

https://www.youtube.com/watch?v=5fZLch5DjZc