Topics in AI (CPSC 532S):
Multimodal Learning with Vision, Language and Sound

Lecture 5: Convolutional Neural Networks (Part 2)
Logistics:

**Assignment 2** will be out tonight

- Data is reusable
  - get it on your Google Drive, will be used for Assignment 3 & 4
  - you **can** reduce the dataset size (e.g., 1/2 or 1/4 of data)
- Piazza (make questions public)
Logistics:

**Assignment 2** will be out tonight
- Data is reusable
  - get it on your Google Drive, will be used for Assignment 3 & 4
  - you **can** reduce the dataset size (e.g., 1/2 or 1/4 of data)
- Piazza (make questions public)

**Office Hours**
Logistics:

Assignment 2 will be out tonight
– Data is reusable
  get it on your Google Drive, will be used for Assignment 3 & 4
  you can reduce the dataset size (e.g., 1/2 or 1/4 of data)
– Piazza (make questions public)

Office Hours

Projects
– Some guidelines
– November 1 & 3 Project Proposals (two lectures week before the break)
Logistics:

**Assignment 2** will be out tonight
- Data is reusable
  - get it on your Google Drive, will be used for Assignment 3 & 4
  - you *can* reduce the dataset size (e.g., 1/2 or 1/4 of data)
- Piazza (make questions public)

**Office Hours**

**Projects**
- Some guidelines
- November 1 & 3 **Project Proposals** (two lectures week before the break)

**Paper presentations**
- List of papers will be made available in the next 1 week
Last time: Convolutional Layer
Last time: Convolutional Layer

* slide from Marc'Aurelio Renzato
Last time: Convolutional Layer
Last time: Convolutional Layer

* slide from Marc’Aurelio Renzato
Last time: Convolutional Layer
Last time: Convolutional Layer

* slide from Marc'Aurelio Renzato
Last time: Convolutional Layer
Last time: Convolutional Layer

* slide from Marc’Aurelio Renzato
Last time: Convolutional Layer

* slide from Marc’Aurelio Renzato
Last time: Convolutional Layer

* slide from Marc’Aurelio Renzato
Last time: Convolutional Layer
Last time: Convolutional Layer
Last time: Convolutional Layer

* slide from Marc’Aurelio Renzato
Last time: Convolutional Layer

Image

Response Map

* slide from Marc’Aurelio Renzato
Last time: Convolutional Layer

* clip from Marc’Aurelio Renzato
Last time: Convolutional Layer

* slide from Marc’Aurelio Renzato
**Last time:** Convolutional Layer

32 x 32 x 3 **image**  

5 x 5 x 3 **filter** \( W \)

**convolve (slide) over all spatial locations**

**activation map**

32 width  
3 depth  
28 width  
28 height  
1 depth

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Last time: Convolutional Neural Network (ConvNet)

- 32 height
- CONV, ReLU e.g. 6 5x5x3 filters
- 32 width
- 6 depth
- 28 height
- CONV, ReLU e.g. 10 5x5x6 filters
- 28 width
- 10 depth
- 24 height
- 24 width

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
What filters do networks learn?

[Zeiler and Fergus, 2013]
What filters do networks learn?

[Zeiler and Fergus, 2013]
Convolutional Layer: Closer Look at **Spatial Dimensions**

32 x 32 x 3 image

5 x 5 x 3 filter \((W)\)

convolve (slide) over all spatial locations

activation map

32 width

28 height

1 depth

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Convolutional Layer: Closer Look at **Spatial Dimensions**

7 width

7 x 7 input image (spatially)
3 x 3 filter

7 height

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Convolutional Layer: Closer Look at **Spatial Dimensions**

7 width

7 x 7 input image (spatially)

3 x 3 filter

7 height

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Convolutional Layer: Closer Look at **Spatial Dimensions**

7 width

7 x 7 input image (spatially)
3 x 3 filter

7 height

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, *cs231n Stanford*
Convolutional Layer: Closer Look at **Spatial Dimensions**

7 width

7 x 7 input image (spatially)
3 x 3 filter

7 height

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Convolutional Layer: Closer Look at **Spatial Dimensions**

7 width

7 x 7 input image (spatially)

3 x 3 filter

7 height

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Convolutional Layer: Closer Look at **Spatial Dimensions**

7 width

7 x 7 input image (spatially)
3 x 3 filter

=> 5 x 5 output

7 height

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, *cs213n Stanford*
Convolutional Layer: Closer Look at **Spatial Dimensions**

7 width

7 x 7 input image (spatially)
3 x 3 filter
(applied with **stride 2**)

7 height

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Convolutional Layer: Closer Look at **Spatial Dimensions**

- 7 width
- 7 x 7 input image (spatially)
- 3 x 3 filter
- (applied with **stride 2**)

- 7 height

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs213n Stanford
Convolutional Layer: Closer Look at **Spatial Dimensions**

- **7 width**
- **7 x 7 input image (spatially)**
- **3 x 3 filter**
- (applied with **stride 2**)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Convolutional Layer: Closer Look at **Spatial Dimensions**

- **7 width**
- **7 x 7 input image (spatially)**
- **3 x 3 filter**
  (applied with **stride 2**)

=> **3 x 3 output**

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Convolutional Layer: Closer Look at **Spatial Dimensions**

7 width

7 x 7 input image (spatially)
3 x 3 filter
(applied with **stride 3**)

7 height

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Convolutional Layer: Closer Look at **Spatial Dimensions**

7 width

7 x 7 input image (spatially)
3 x 3 filter
(applied with **stride 3**)

7 height

Does not fit! **Cannot apply** 3 x 3 filter on 7 x 7 image with stride 3

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Convolutional Layer: Closer Look at **Spatial Dimensions**

N x N input image (spatially)

F x F filter

**Output size:** \((N-F) / \text{stride} + 1\)
Convolutional Layer: Closer Look at **Spatial Dimensions**

N x N input image (spatially)

F x F filter

**Output size:** \((N-F) / \text{stride} + 1\)

**Example:** \(N = 7, F = 3\)

- stride 1 => \((7-3)/1+1 = 5\)
- stride 2 => \((7-3)/2+1 = 3\)
- stride 3 => \((7-3)/3+1 = \text{2.33}\)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Convolutional Layer: **Border padding**

- **7 width**
- **7 x 7 input image (spatially)**
- **3 x 3 filter**
  (applied with **stride 1**)
- **pad** with 1 pixel border

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs213n Stanford**
### Convolutional Layer: Border Padding

**7 width**

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

**7 height**

7 x 7 input image (spatially)
3 x 3 filter
(applied with **stride 1**)

**pad** with 1 pixel border

**Output size:** 7 x 7

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, *cs231n Stanford*
Convolutional Layer: **Border padding**

7 x 7 input image (spatially)
3 x 3 filter
(applied with **stride 3**)

**pad** with 1 pixel border

7 width

7 height
Convolutional Layer: **Border padding**

7 x 7 input image (spatially)
3 x 3 filter
(applied with **stride 3**)

**pad** with 1 pixel border

**Example:** $N = 7$, $F = 3$

- stride 1 $\Rightarrow (9-3)/1+1 = 7$
- stride 2 $\Rightarrow (9-3)/2+1 = 4$
- stride 3 $\Rightarrow (9-3)/3+1 = 3$
Convolutional Neural Network (ConvNet)

32 height

CONV, ReLU
  e.g. 6 5x5x3 filters

32 width

3 depth

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Convolutional Neural Network (ConvNet)

- **32 height**
- **32 width**
- **3 depth**

**CONV, ReLU**
e.g. **6 5x5x3 filters**

- **28 height**
- **28 width**
- **6 depth**

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Convolutional Neural Network (ConvNet)

32 width × 32 height filters

CONV, ReLU
e.g. 6 5x5x3 filters

28 width × 28 height filters

CONV, ReLU
e.g. 10 5x5x6 filters

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Convolutional Neural Network (ConvNet)

32 width, 32 height
CONV, ReLU
6 5x5x3 filters

3 depth

28 width, 28 height
CONV, ReLU
10 5x5x6 filters

6 depth

24 width, 24 height

10 depth

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Convolutional Neural Network (ConvNet)

32 height

32 width

CONV,
ReLU
e.g. 6 5x5x3
filters

28 height

28 width

CONV,
ReLU
e.g. 10 5x5x6
filters

24 height

24 width

3 depth

6 depth

10 depth

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Convolutional Neural Network (ConvNet)

With padding we can achieve no shrinking (32 -> 28 -> 24); shrinking quickly (which happens with larger filters) doesn’t work well in practice

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
**Convolutional Layer: 1x1 convolutions**

56 x 56 x 64 image

32 filters of size, 1 x 1 x 64

56 x 56 x 32 image

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Convolutional Layer **Summary**

Accepts a volume of size: $W_i \times H_i \times D_i$
Convolutional Layer **Summary**

Accepts a volume of size: $W_i \times H_i \times D_i$

Requires hyperparameters:

- Number of filters: $K$ (for typical networks $K \in \{32, 64, 128, 256, 512\}$)
- Spatial extent of filters: $F$ (for a typical networks $F \in \{1, 3, 5, \ldots\}$)
- Stride of application: $S$ (for a typical network $S \in \{1, 2\}$)
- Zero padding: $P$ (for a typical network $P \in \{0, 1, 2\}$)
Convolutional Layer Summary

Accepts a volume of size: $W_i \times H_i \times D_i$

Requires hyperparameters:
- Number of filters: $K$ (for typical networks $K \in \{32, 64, 128, 256, 512\}$)
- Spatial extent of filters: $F$ (for a typical networks $F \in \{1, 3, 5, \ldots\}$)
- Stride of application: $S$ (for a typical network $S \in \{1, 2\}$)
- Zero padding: $P$ (for a typical network $P \in \{0, 1, 2\}$)

Produces a volume of size: $W_o \times H_o \times D_o$
Convolutional Layer Summary

Accepts a volume of size: $W_i \times H_i \times D_i$

Requires hyperparameters:

- Number of filters: $K$ (for typical networks $K \in \{32, 64, 128, 256, 512\}$)
- Spatial extent of filters: $F$ (for typical networks $F \in \{1, 3, 5, \ldots\}$)
- Stride of application: $S$ (for a typical network $S \in \{1, 2\}$)
- Zero padding: $P$ (for a typical network $P \in \{0, 1, 2\}$)

Produces a volume of size: $W_o \times H_o \times D_o$

$$W_o = (W_i - F + 2P)/S + 1 \quad H_o = (H_i - F + 2P)/S + 1 \quad D_o = K$$
Convolutional Layer **Summary**

Accepts a volume of size: $W_i \times H_i \times D_i$

Requires hyperparameters:

- Number of filters: $K$ (for typical networks $K \in \{32, 64, 128, 256, 512\}$)
- Spatial extent of filters: $F$ (for a typical networks $F \in \{1, 3, 5, \ldots\}$)
- Stride of application: $S$ (for a typical network $S \in \{1, 2\}$)
- Zero padding: $P$ (for a typical network $P \in \{0, 1, 2\}$)

Produces a volume of size: $W_o \times H_o \times D_o$

$$W_o = (W_i - F + 2P)/S + 1 \quad H_o = (H_i - F + 2P)/S + 1 \quad D_o = K$$

Number of total learnable parameters: $(F \times F \times D_i) \times K + K$
Convolutional Neural Networks

VGG-16 Network
**CNNs: Reminder Fully Connected Layers**

An example of a fully connected layer taking as input a 32 x 32 x 3 image, stretching it to 3072 x 1, and then applying an activation function to each of the 10 neurons. The input is represented as a long block, the activation as a short block.

\[ W^T x + b, \text{ where } W \in \mathbb{R}^{10 \times 3072} \]

Each neuron looks at the full input volume.

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, *cs231n Stanford*
Convolutional Neural Networks

VGG-16 Network
The diagram illustrates the process of passing an input through a fully connected layer in a Convolutional Neural Network (CNN). The input is a $7 \times 7 \times 512$ image, which is transformed into a vector of size $25,088 \times 1$. Each neuron in the activation layer applies the weight matrix $W$ and bias $b$ to this input, defined by the equation $W^T x + b$, where $W \in \mathbb{R}^{25,088 \times 4,096}$. This operation effectively consists of $25,088 \times 4,096$ multiplications and additions per neuron. The result is a 4,096-dimensional vector representing a fully connected layer output. This diagram is adapted from Fei-Dei Li, Justin Johnson, Serena Yeung, and the CS231n course at Stanford University.
CNNs: Reminder Fully Connected Layers

\[ W^T x + b, \text{ where } W \in \mathbb{R}^{25,088 \times 4,096} \]

- Input
  - 25,088
  - (7 x 7 x 512 image -> stretches to 25,088 x 1)

- Activation
  - each neuron looks at the full input volume
  - 4,096

102,760,448 parameters!

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Convolutional Neural Networks

VGG-16 Network
Convolutional Neural Networks

VGG-16 Network
Invariance vs. Equivariance

Invariance: A mathematical object (or class of mathematical objects) remains unchanged after transformations of certain types applied to the objects.

Equivariance: Applying a transformation and then computing the function produces the same result as computing the function and then applying a transformation.

\[ g(f(x)) = f(g(x)) \]
Invariance vs. Equivariance

**Invariance:** A mathematical object (or class of mathematical objects) remains unchanged after transformations of certain types applied to the objects

\[ f(x) = f(g(x)) \]
**Invariance** vs. **Equivariance**

**Invariance:** A mathematical object (or class of mathematical objects) remains unchanged after transformations of certain types applied to the objects

\[ f(x) = f(g(x)) \]

**Equivariance:** Applying a transformation and then computing the function produces the same result as computing the function and then applying a transformation

\[ g(f(x)) = f(g(x)) \]
Revisit **Layers** we Learned About

**Fully Connected:**

- Not invariant to any transformations
- Not equivariant to any transformations

Convolutional:

- Not invariant to any transformations
- Convolution is translation equivariant

Note: convolution can "learn" not to be equivariant when padding is used.
Revisit **Layers** we Learned About

**Fully Connected:**

- **Not** invariant to any transformations
- **Not** equivariant to any transformations

Note: convolution can “learn” not to be equivariant when padding is used.
Revisit **Layers** we Learned About

**Fully Connected:**
- **Not** invariant to any transformations
- **Not** equivariant to any transformations

**Convolutional:**
Revisit **Layers** we Learned About

**Fully Connected:**
- **Not** invariant to any transformations
- **Not** equivariant to any transformations

**Convolutional:**
- **Not** invariant to any transformations
- Convolution is **translation equivariant**

**Note:** convolution can “learn” not to be equivariant when padding is used.
Revisit **Layers** we Learned About
Revisit **Layers** we Learned About

- **Weight**
  - 0 1 0
  - 0 0 0
  - 0 0 0

- **Bias**
  - 1
Pooling Layer

Let us assume the filter is an “eye” detector

How can we make detection spatially invariant (insensitive to position of the eye in the image)

* slide from Marc’Aurelio Renzato
Let us assume the filter is an “eye” detector.

How can we make detection spatially invariant (insensitive to position of the eye in the image)?

By “pooling” (e.g., taking a max) response over a spatial locations we gain robustness to position variations.

* slide from Marc’Aurelio Renzato
Pooling Layer

- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Pooling Layer

- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently

How many parameters?

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Pooling Layer

- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently

How many parameters? None!

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
**Pooling Layer**

- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Pooling Layer

- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently

How do we implement that in a computation graph?

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Max Pooling

activation map

max pool with 2 x 2 filter and stride of 2

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Average Pooling

activation map

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

<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

avg pool with 2 x 2 filter and stride of 2

<table>
<thead>
<tr>
<th>3.25</th>
<th>5.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Pooling Layer **Receptive Field**

If convolutional filters have size $K \times K$ and stride 1, and pooling layer has pools of size $P \times P$, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: $(P+K-1) \times (P+K-1)$
If convolutional filters have size $K \times K$ and stride 1, and pooling layer has pools of size $P \times P$, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: $(P+K-1) \times (P+K-1)$
Pooling Layer **Summary**

Accepts a volume of size: \( W_i \times H_i \times D_i \)

Requires hyperparameters:

- Spatial extent of filters: \( K \)
- Stride of application: \( F \)

Produces a volume of size: \( W_o \times H_o \times D_o \)

\[
W_o = W_i / F \\
H_o = H_i / F \\
D_o = D_i
\]

Number of total learnable parameters: 0

(you can do padding, but it’s a bit trickier)
Convolutional Neural Networks

VGG-16 Network
Improving **Single Model**

**Regularization**
- L2, L1
- Dropout / Inverted Dropout
- Data augmentation
Improving **Single Model**

### Regularization

- L2, L1
- Dropout / Inverted Dropout
- Data augmentation

**L2 Regularization:** Learn a more (dense) distributed representation

\[
R(W) = ||W||_2 = \sum_i \sum_j W_{i,j}^2
\]

**L1 Regularization:** Learn a sparse representation (few non-zero weight elements)

\[
R(W) = ||W||_1 = \sum_i \sum_j |W_{i,j}|
\]
Improving **Single Model**

**Regularization**
- L2, L1
- Dropout / Inverted Dropout
- Data augmentation

**L2 Regularization:** Learn a more (dense) distributed representation

\[
R(W) = \|W\|_2 = \sum_i \sum_j W_{i,j}^2
\]

**L1 Regularization:** Learn a sparse representation (few non-zero weight elements)

\[
R(W) = \|W\|_1 = \sum_i \sum_j |W_{i,j}|
\]
Regularization: Data Augmentation

Load image and label

Load image

cat

CNN

Compute Loss

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Regularization: Data Augmentation

Load image and label
Transform image

CNN

cat

[dog: 0.95]
[frisbee: 0.83]
[outdoor: 0.82]
[grass: 0.81]
[leap: 0.45]

Attributes

Visual Attributes by MIL

Visual representation by DCNN

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Regularization: Data Augmentation

Horizontal flips
Random crops & scales
Color Jitter

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Regularization: Data Augmentation

- Horizontal flips
- Random crops & scales
- Color Jitter

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
**Regularization:** Data Augmentation

**Horizontal flips**

**Random crops & scales**

**Color Jitter**

**Training:** sample random crops and scales
e.g., ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short size = L
3. Sample random 224x224 patch

**Testing:** average a fix set of crops
e.g., ResNet:

1. Resize image to 5 scales (224, 256, 384, 480, 640)
2. For each image use 10 224x224 crops: 4 corners + center, + flips

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Regularization: Data Augmentation

- Horizontal flips
- Random crops & scales
- Color Jitter

Random perturbations in contrast and brightness

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Regularization: Stochastic Depth

Effectively “dropout” but for layers

Stochastically with some probability **turn off** some layer (for each batch)

Effectively trains a collection of neural networks

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford

[Huang et al., ECCV 2016]
Common “Wisdom”: You need a lot of data to train a CNN

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Transfer Learning with CNNs

Common “Wisdom”: You need a lot of data to train a CNN

Solution: Transfer learning — taking a model trained on the task that has lots of data and adopting it to the task that may not

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Transfer Learning with CNNs

Common “Wisdom”: You need a lot of data to train a CNN

Solution: Transfer learning — taking a model trained on the task that has lots of data and adopting it to the task that may not

This strategy is PERVERSIVE.

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Transfer Learning with CNNs

Train on ImageNet

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Transfer Learning with CNNs

Train on **ImageNet**

Why on **ImageNet**?

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford

[ Yosinski et al., NIPS 2014 ]
[ Donahue et al., ICML 2014 ]
[ Razavian et al., CVPR Workshop 2014 ]
Transfer Learning with CNNs

Train on ImageNet

Why on ImageNet?
- Convenience, lots of data
- We know how to train these well

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Transfer Learning with CNNs

Train on ImageNet

Why on ImageNet?
- Convenience, lots of data
- We know how to train these well

However, for some tasks we would need to start with something else (e.g., videos for optical flow)

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Transfer Learning with CNNs

Train on ImageNet

Small dataset with C classes

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, [cs231n Stanford](https://cs231n.stanford.edu/)

[ Yosinski et al., NIPS 2014 ]
[ Donahue et al., ICML 2014 ]
[ Razavian et al., CVPR Workshop 2014 ]
Transfer Learning with CNNs

Train on **ImageNet**

- FC-1000
- FC-4096
- FC-4096
- MaxPool
- Conv-512
- Conv-512
- MaxPool
- Conv-512
- Conv-512
- MaxPool
- Conv-256
- Conv-256
- MaxPool
- Conv-128
- Conv-128
- MaxPool
- Conv-64
- Conv-64

Small dataset with C classes

- FC-1000
- FC-4096
- FC-4096
- MaxPool
- Conv-512
- Conv-512
- MaxPool
- Conv-256
- Conv-256
- MaxPool
- Conv-128
- Conv-128
- MaxPool
- Conv-64
- Conv-64

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford

[ Yosinski et al., NIPS 2014 ]
[ Donahue et al., ICML 2014 ]
[ Razavian et al., CVPR Workshop 2014 ]
Transfer Learning with CNNs

Train on ImageNet

Small dataset with C classes

Freeze these layers

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Transfer Learning with CNNs

Train on ImageNet

Small dataset with C classes

Re-initialize and train

 Freeze these layers

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford

[ Yosinski et al., NIPS 2014 ]
[ Donahue et al., ICML 2014 ]
[ Razavian et al., CVPR Workshop 2014 ]
Transfer Learning with CNNs

Train on **ImageNet**

**Small dataset** with C classes

- FC-1000
- FC-4096
- Image

Freeze these layers

Lower levels of the CNN are at task independent anyways

Re-initialize and train

- FC-1000
- FC-4096
- Image

- MaxPool
- Conv-512
- Conv-512

- MaxPool
- Conv-256
- Conv-256

- MaxPool
- Conv-128
- Conv-128

- MaxPool
- Conv-64
- Conv-64

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford

[ Yosinski et al., NIPS 2014 ]
[ Donahue et al., ICML 2014 ]
[ Razavian et al., CVPR Workshop 2014 ]
Transfer Learning with CNNs

Train on ImageNet

Small dataset with C classes

Re-initialize and train

Freeze these layers

Larger dataset

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford

[ Yosinski et al., NIPS 2014 ]
[ Donahue et al., ICML 2014 ]
[ Razavian et al., CVPR Workshop 2014 ]
Transfer Learning with CNNs

Train on ImageNet

Small dataset with C classes

Re-initialize and train

Larger dataset

Freeze these layers

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford

[ Yosinski et al., NIPS 2014 ]
[ Donahue et al., ICML 2014 ]
[ Razavian et al., CVPR Workshop 2014 ]
Transfer Learning with CNNs

- **Train on ImageNet**
  - FC-1000
  - FC-4096
  - FC-4096
  - MaxPool
  - Conv-512
  - Conv-512
  - MaxPool
  - Conv-512
  - Conv-512
  - MaxPool
  - Conv-256
  - Conv-256
  - MaxPool
  - Conv-128
  - Conv-128
  - MaxPool
  - Conv-64
  - Conv-64
  - Image

- **Small dataset with C classes**
  - Re-initialize and train
  - FC-1000
  - FC-4096
  - FC-4096
  - MaxPool
  - Conv-512
  - Conv-512
  - MaxPool
  - Conv-256
  - Conv-256
  - MaxPool
  - Conv-128
  - Conv-128
  - MaxPool
  - Conv-64
  - Conv-64
  - Image

- **Larger dataset**
  - Freeze these layers
  - FC-1000
  - FC-4096
  - FC-4096
  - MaxPool
  - Conv-512
  - Conv-512
  - MaxPool
  - Conv-256
  - Conv-256
  - MaxPool
  - Conv-128
  - Conv-128
  - MaxPool
  - Conv-64
  - Conv-64
  - Image

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford

[ Yosinski et al., NIPS 2014 ]
[ Donahue et al., ICML 2014 ]
[ Razavian et al., CVPR Workshop 2014 ]
Transfer Learning with CNNs

Train on ImageNet

Small dataset with C classes

Larger dataset

- Freeze these layers
- Re-initialize and train

- Freeze these layers
- Re-initialize and train

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford

[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]
[Yosinski et al., NIPS 2014]
Transfer Learning with CNNs

Dataset A: 500 classes

Layers fine-tuned
Layers fixed

[ Yosinski et al., NIPS 2014 ]
Transfer Learning with CNNs

Dataset A: 500 classes
Dataset B: (different) 500 classes

- Layers fine-tuned
- Layers fixed

[Yosinski et al., NIPS 2014]
Model **Ensemble**

**Training:** Train multiple independent models

**Test:** Average their results

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Model Ensemble

**Training:** Train multiple independent models

**Test:** Average their results

*~ 2% improved performance in practice*

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford*
Model Ensemble

**Training:** Train multiple independent models

**Test:** Average their results

~ 2% improved performance in practice

**Alternative:** Multiple snapshots of the single model during training!

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**
Model **Ensemble**

**Training:** Train multiple independent models

**Test:** Average their results

~ 2% improved performance in practice

**Alternative:** Multiple snapshots of the single model during training!

**Improvement:** Instead of using the actual parameter vector, keep a moving average of the parameter vector and use that at test time (Polyak averaging)

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**
CPU vs. GPU (Why do we need Azure?)

Data from https://github.com/jcjohnson/cnn-benchmarks

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Frameworks: Super quick overview

1. Easily **build computational graphs**

2. Easily **compute gradients** in computational graphs

3. **Run it all efficiently** on a GPU (weap cuDNN, cuBLAS, etc.)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
## Core DNN Frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffe</td>
<td>(UC Berkeley)</td>
</tr>
<tr>
<td>Caffe 2</td>
<td>(Facebook)</td>
</tr>
<tr>
<td>Puddle</td>
<td>(Baidu)</td>
</tr>
<tr>
<td>Torch</td>
<td>(NYU/Facebook)</td>
</tr>
<tr>
<td>PyTorch</td>
<td>(Facebook)</td>
</tr>
<tr>
<td>CNTK</td>
<td>(Microsoft)</td>
</tr>
<tr>
<td>Theano</td>
<td>(U Montreal)</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>(Google)</td>
</tr>
<tr>
<td>MXNet</td>
<td>(Amazon)</td>
</tr>
</tbody>
</table>

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
### Core DNN Frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffe</td>
<td>UC Berkeley</td>
</tr>
<tr>
<td>Caffe 2</td>
<td>Facebook</td>
</tr>
<tr>
<td>Torch</td>
<td>NYU/Facebook</td>
</tr>
<tr>
<td>PyTorch</td>
<td>Facebook</td>
</tr>
<tr>
<td>Theano</td>
<td>U Montreal</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>Google</td>
</tr>
<tr>
<td>Caffe 2</td>
<td>Facebook</td>
</tr>
<tr>
<td>Theano</td>
<td>U Montreal</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>Google</td>
</tr>
</tbody>
</table>

### Wrapper Libraries

<table>
<thead>
<tr>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keras</td>
</tr>
<tr>
<td>TFLearn</td>
</tr>
<tr>
<td>TensorFlow</td>
</tr>
<tr>
<td>CNTK</td>
</tr>
<tr>
<td>MXNet</td>
</tr>
</tbody>
</table>

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
**Frameworks:** PyTorch vs. TensorFlow (v1)

Dynamic vs. Static computational graphs
Dynamic vs. **Static** computational graphs

With static graphs, framework can **optimize** the graph for you before it runs.

*slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford*
**Frameworks:** PyTorch vs. TensorFlow (v1)

**Dynamic** vs. Static computational graphs

Graph building and execution is intertwined. Graph can be different for every sample.

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
PyTorch: Three levels of abstraction

Tensor: Imperative ndarray, but runs on GPU

Variable: Node in a computational graph; stores data and gradients

Module: A neural network layer; may store state or learnable weights

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford