Key Ideas and Architectures in Deep Learning
Applications that (probably) use DL

Autonomous Driving

Scene understanding

/Segmentation
Applications that (probably) use DL

WordLens

Prisma
Outline of today’s talk

Image Recognition

- LeNet - 1998
- AlexNet - 2012
- VGGNet - 2014
- GoogLeNet - 2014
- ResNet - 2015

Fun application using CNNs

- Image Style Transfer
Revolution of Depth

152 layers

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: 25.8 (shallow)
ILSVRC'10: 28.2
Questions to ask about each architecture/paper

Special Layers
Loss function
Train faster?
Reduce Overfitting

Non-Linearity
Weight-update rule
Reduce parameters
Help you visualize?
LeNet5 - 1998
LeNet5 - Specs

MNIST - 60,000 training, 10,000 testing

Input is 32x32 image

8 layers

60,000 parameters

Few hours to train on a laptop
Modified LeNet Architecture - Assignment 3

Training

Input

Conv → ReLU → Maxpool → Conv → ReLU → Maxpool → FC → ReLU

Forward pass

Backpropagation - update weights

Softmax

Loss

Labels

Training phases:
- Forward pass: Input goes through convolution, ReLU activation, max pooling, and then another convolution, ReLU activation, and max pooling before passing through a fully connected layer with ReLU activation. The output is then passed through a softmax layer.
- Backpropagation: The error is calculated using the labels and the softmax output, then backpropagated through the network to update the weights.
Modified LeNet Architecture - Assignment 3

Testing

Input → Conv → ReLU → Maxpool → Conv → ReLU → Maxpool → FC → ReLU → Softmax → Output

Forward pass

Compare output with labels
Modified LeNet - CONV Layer 1

Input - 28 x 28

Output - 6 feature maps - each 24 x 24

Convolution filter - 5 x 5 x 1 (convolution) + 1 (bias)

How many parameters in this layer?
Modified LeNet - CONV Layer 1

Input - 32 x 32

Output - 6 feature maps - each 28 x 28

Convolution filter - 5 x 5 x 1 (convolution) + 1 (bias)

How many parameters in this layer?

\[(5\times5\times1+1)\times6 = 156\]
Modified LeNet - Max-pooling layer

Decreases the spatial extent of the feature maps, makes it translation-invariant

**Input** - 28 x 28 x 6 volume

Maxpooling with filter size 2 x 2

And stride 2

**Output** - ?
Modified LeNet - Max-pooling layer

Decreases the spatial extent of the feature maps

**Input** - 28 x 28 x 6 volume

Maxpooling with filter size 2 x 2 and stride 2

**Output** - 14 x 14 x 6 volume
LeNet5 - Key Ideas

**Convolution** - extract same features at different spatial locations with few parameters

**Spatial averaging** - sub-sampling to reduce parameters (we use max-pooling)

**Non-linearity** - Sigmoid (but we’ll use ReLU)

Multi-layer perceptron in the final layers

Introduced the **Conv -> Non-linearity -> Pooling** unit
LeNet5 Evaluation

Misclassifications

Accuracy

>97%
What happened from 1998-2012?

Neural nets were in incubation

More and more data was available - cheaper digital cameras

And computing power became better - CPUs were becoming faster

GPUs became a general-purpose computing tool (2005-6)

Creation of structured datasets - ImageNet (ILSVRC) 2010 (super important!)
A word about datasets - Network inputs

**ImageNet** (We’ll talk about object classification)

**CIFAR** - Object Classification

**Caltech** - Pedestrian detection benchmark

**KITTI** - SLAM, Tracking etc.

Remember : Your algo is only as good as your data!
How are networks evaluated? - Network outputs

Top-5 error
Top-1 error
Accuracy
AlexNet - 2012

Won the 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge)

Achieved a top-5 error rate of 15.4%, next best was 26.2%
AlexNet - Specs

ImageNet 1000 categories
1.2 million training images
50,000 validation images
150,000 testing images.

60M Parameters

Trained on two GTX 580 GPUs for five to six days.
AlexNet - Key Ideas

Used ReLU for the nonlinearity functions - \( f(x) = \max(0,x) \) - made convergence faster.

Used **data augmentation** techniques.

Implemented **dropout** to combat overfitting to the training data.

Trained the model using **batch stochastic gradient descent**.

Used **momentum** and **weight decay**.
Dropout

Dropout in Neural Networks

(a) Standard Neural Net

(b) After applying dropout.
**VGG Net - 2014**

“Simple and deep”

Top-5 error rate of 7.3% on ImageNet

16 layer CNN - Best result - Conf. D

138 M parameters

Trained on 4 Nvidia Titan Black GPUs for two to three weeks.

<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></td>
<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 × 224 RGB image)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
</tr>
<tr>
<td>LRN</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
</tr>
<tr>
<td>maxpool</td>
<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>conv3-128</td>
<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>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>conv3-256</td>
<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>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>conv3-512</td>
<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>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>conv3-512</td>
<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>
<td></td>
</tr>
<tr>
<td>FC-4096</td>
<td></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>
<td></td>
</tr>
<tr>
<td>FC-1000</td>
<td></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>
<td></td>
</tr>
</tbody>
</table>
VGG Net - Key Ideas

The use of only 3x3 sized filters. Used multiple times = greater receptive fields.

Decrease in spatial dimensions and increase in depth deeper into the network

Used scale jittering as one data augmentation technique during training

Used ReLU layers after each conv layer and trained with batch gradient descent

Reduced number of parameters - $3^2 (3^2)$ compared to $7^2$

Conclusion - Small RFs, deep networks are good. :-(
GoogLeNet / Inception - 2014

Winner of ILSVRC 2014 with a top 5 error rate of 6.7% (4M parameters compared to AlexNet’s 60M)

Trained on “a few high-end GPUs within a week”.
The Inception module
The Inception Module - A closer look
The Inception Module - A closer look
Inception module - Feature Map Concatenation
### Inception Parameter count

| type         | patch size/ 
<table>
<thead>
<tr>
<th></th>
<th>stride</th>
<th>output size</th>
<th>depth</th>
<th>#1×1</th>
<th>#3×3 reduce</th>
<th>#3×3 reduce</th>
<th>#5×5 reduce</th>
<th>#5×5</th>
<th>pool proj</th>
<th>params</th>
<th>ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>convolution</td>
<td>7×7/2</td>
<td>112×112×64</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.7K</td>
<td>34M</td>
</tr>
<tr>
<td>max pool</td>
<td>3×3/2</td>
<td>56×56×64</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>112K</td>
<td>360M</td>
</tr>
<tr>
<td>convolution</td>
<td>3×3/1</td>
<td>56×56×192</td>
<td>2</td>
<td>64</td>
<td></td>
<td>192</td>
<td></td>
<td></td>
<td></td>
<td>159K</td>
<td>128M</td>
</tr>
<tr>
<td>max pool</td>
<td>3×3/2</td>
<td>28×28×192</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>380K</td>
<td>304M</td>
</tr>
<tr>
<td>inception (3a)</td>
<td>28×28×256</td>
<td>2</td>
<td>64</td>
<td>96</td>
<td></td>
<td>128</td>
<td>16</td>
<td>32</td>
<td>32</td>
<td>364K</td>
<td>73M</td>
</tr>
<tr>
<td>inception (3b)</td>
<td>28×28×480</td>
<td>2</td>
<td>128</td>
<td>128</td>
<td></td>
<td>192</td>
<td>32</td>
<td>96</td>
<td>64</td>
<td>437K</td>
<td>88M</td>
</tr>
<tr>
<td>max pool</td>
<td>3×3/2</td>
<td>14×14×480</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>463K</td>
<td>100M</td>
</tr>
<tr>
<td>inception (4a)</td>
<td>14×14×512</td>
<td>2</td>
<td>192</td>
<td>96</td>
<td></td>
<td>208</td>
<td>16</td>
<td>48</td>
<td>64</td>
<td>580K</td>
<td>119M</td>
</tr>
<tr>
<td>inception (4b)</td>
<td>14×14×512</td>
<td>2</td>
<td>160</td>
<td>112</td>
<td></td>
<td>224</td>
<td>24</td>
<td>64</td>
<td>64</td>
<td>580K</td>
<td>119M</td>
</tr>
<tr>
<td>inception (4c)</td>
<td>14×14×512</td>
<td>2</td>
<td>128</td>
<td>128</td>
<td></td>
<td>256</td>
<td>24</td>
<td>64</td>
<td>64</td>
<td>580K</td>
<td>119M</td>
</tr>
<tr>
<td>inception (4d)</td>
<td>14×14×528</td>
<td>2</td>
<td>112</td>
<td>144</td>
<td></td>
<td>288</td>
<td>32</td>
<td>64</td>
<td>64</td>
<td>580K</td>
<td>119M</td>
</tr>
</tbody>
</table>

If we used (3 x 3, 512) convolution:

(3 x 3 x 512 x 512) parameters = 2.359 million parameters

Inception module: 437K parameters
Inception - Key Ideas

Used 9 Inception modules in the whole architecture

No use of fully connected layers! They use an average pool instead, to go from a 7x7x1024 volume to a 1x1x1024 volume - Saves a huge number of parameters.

Uses 12x fewer parameters than AlexNet.

During testing, multiple crops of the same image were created, fed into the network, and the softmax probabilities were averaged to give us the final solution.

Improved performance and efficiency through creatively stacking layers
Going deeper

Performance of ResNets versus plain-nets as depth is increased
Microsoft ResNet 2015

ResNet won ILSVRC 2015 with an incredible error rate of 3.6%
Humans usually hover around 5-10%
Trained on an 8 GPU machine for two to three weeks.

34-residual

34-plain

VGG
ResNet - A closer look

Plain net:
- Input: $x$
- Layers: weight layer, relu
- Output: $H(x)$
- Any two stacked layers

Residual net:
- Input: $x$
- Layers: weight layer, relu, weight layer
- Output: $H(x) = F(x) + x$
- Identity $x$
ResNets - Key Ideas

Residual learning

Interesting to note that after only the first 2 layers, the spatial size gets compressed from an input volume of 224x224 to a 56x56 volume.

Tried a 1202-layer network, but got a lower test accuracy, presumably due to overfitting.
Do I have to train from scratch every time?

If you have the data, the time and the power you should train from scratch.

But since ConvNets can take weeks to train - people make their pre-trained network weights available - Eg. Caffe Model Zoo

<table>
<thead>
<tr>
<th>Degree of similarity of pretrained data to your own</th>
<th>Do you have a lot of data and compute power?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low, Less</td>
<td>Low, More</td>
</tr>
<tr>
<td>High, Less</td>
<td>High, More</td>
</tr>
</tbody>
</table>

Initialize weights only from lower layers

Initialize/ Use weights from a higher layer

Train from scratch

Train from scratch
Do I have to train from scratch every time?

1. Use CNNs weights as initialization for your network - **Assignment 3**!
   
   **Fine-tune** the weights using your data + replace and retrain a classifier on top

2. Use CNN as a fixed feature extractor - Build SVM / some other classifier on top of it
A fun application - Style Transfer using ConvNets
\[ E_L = \sum (G^L - A^L)^2 \]

\[ \mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style} \]

\[ G_{ij}^L = \sum_k F_{ik}^L F_{jk}^L. \]

\[ \frac{\partial E_L}{\partial F^L} \]

\[ \frac{\partial E_L}{\partial F^{L-1}} \]

\[ \mathcal{L}_{content} = \sum (F^l - P^l)^2 \]

\[ \mathcal{L}_{style} = \sum_l w_l E_l \]

\[ \vec{a} = \]

\[ \vec{x} = \]

\[ \vec{p} = \]

\[ \vec{x} := \vec{x} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial \vec{x}} \]
Slide Credits and References

A brief overview of DL papers

http://iamaaditya.github.io

A course on CNNs http://cs231n.github.io/


Style transfer -
Slide Credits and References

Dropout (Recommended read)


ResNet Tutorial


Backpropagation Refresher (Useful read)

Thank you!