Deep Learning (CS324)

Convolutional Neural Networks* (Continued)

Prof. Jianguo Zhang

SUSTech

*Based on http://www.deeplearningbook.org chapter 9 + referenced papers

Online resources

- Build your own CNN inside your browser https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html
- Various neural networks in Keras https://transcranial.github.io/keras-js

Visualising ConvNets

Given a feature map (i.e., convolved image), what are the top 9 activations?
For the full set of experiments see https://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf

More cool visualisations here:https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

The grandaddy of CNNs: LeNet-5

- 7-level convolutional network by LeCun et al in 1998 (http://yann.lecun.com/exdb/lenet/)
- Applied to recognise hand-written numbers on cheques (32 x 32), but due to computational constraints, couldn't scale up to larger images

Original Image published in [LeCun et al., 1998]

The grandaddy of CNNs: LeNet-5

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

Approximately 60,000 parameters

Source: https://engmrk.com/lenet-5-a-classic-cnn-architecture/

The ImageNet challenge

The ImageNet project is a large visual database designed for use in visual object recognition software research. The ImageNet project runs an annual software contest, the **ImageNet Large Scale Visual Recognition Challenge** (ILSVRC)

The ImageNet challenge

The ImageNet challenge

AlexNet

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

AlexNet

- Outperformed all competitors and won the challenge by reducing **top-5 error** from 26% to **16.4%**
- **2nd place** went to a non-CNN model with **top-5 error** rate around **26.2%**
- Approximately 60 million parameters
- Paper: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

AlexNet

ReLUs instead of tanh (LeNet) + dropout

AlexNet

5 convolutional layers + 3 fully connected layers

AlexNet

Trained for 6 days simultaneously on two Nvidia Geforce GTX 580 GPUs:
kernels of the 2nd, 4th, and 5th conv layers are connected only to kernel maps in the previous layer residing on the same GPU. Kernels of the 3rd conv layer are connected to all kernel maps in the second layer.

source: https://www.learnopencv.com/understanding-alexnet/

AlexNet

96 convolutional kernels of size 11×11×3 learned by the first convolutional layer on the 224×224×3 input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2.

The effect of splitting the learning of the kernels over two GPUs can be observed in the difference between the top and bottom sets of kernel filters above: remember, these are learned and not hardcoded!

The ImageNet challenge

The ImageNet challenge

WE NEED TO GO
DEEPER
memegenerator.net

VGG16

- Runner-up at the 2014 ILSVRC (won by GoogLeNet) with a 7.3% error rate, developed by Oxford's **V**isual **G**eometry **G**roup
- **16 convolutional layers** (lots of 3x3 filters)
- Approximately 138 million parameters
- Trained on 4 GPUs for 3 weeks
- "Simple" yet effective (depth is the key)
- Paper: https://arxiv.org/abs/1409.1556

Source: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

VGG16: key observations

- Number of filters increases as we go deeper
- 3 x 3 convolutions stacked in groups of 2 or 3
- Note: 2 stacked 3 x 3 convolutions have the receptive field of a single 5 x 5 convolution

What about 3 stacked 3 x 3 convolutions?

VGG16: key observations

- Number of filters increases as we go deeper
- 3 x 3 convolutions stacked in groups of 2 or 3
- Note: 2 stacked 3 x 3 convolutions have the receptive field of a single 5 x 5 convolution
- Why stacking several small filters??
  1. More ReLU operations generate more non-linearity which means a more powerful model
  2. Fewer parameters, e.g.,
     (3 x 3) x 3 = 27 <u>versus</u> **(7 x 7) = 49**

The ImageNet challenge

The ImageNet challenge

GoogLeNet (Inception network V1)

- The winner of the 2014 ILSVRC was actually GoogLeNet (developed by Google), built around the **Inception** unit, with top-5 error rate of 6.7%
- Very close to human-level performance (5%)
- Key features: inception units, batch normalization, image distortions, RMSprop
- Parameters: ~5 million (V1) and ~23 million (V3)
- Papers: https://arxiv.org/abs/1409.4842 and https://arxiv.org/abs/1512.00567

GoogLeNet (Inception network V1)

- Let's first try to understand the problem the engineers behind the Inception unit tried to solve:

1. Very deep networks are prone to overfitting due to the large number of parameters
2. Naively stacking large convolution operations is computationally expensive: if two convolutional layers are chained, any uniform increase in the number of their filters results in a quadratic increase of computation

3) Salient parts in images can have very large variation in size

Source: https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

How to choose the kernel size then?

Source: https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

GoogLeNet (Inception network V1)

- Solution: make network **wider instead of deeper** having filters of **multiple sizes** operating at the **same level**

(a) Inception module, naïve version

(b) Inception module with dimension reductions

Source: https://arxiv.org/pdf/1409.4842v1.pdf

GoogLeNet (Inception network V1)

- Solution: make network **wider instead of deeper** having filters of **multiple sizes** operating at the **same level**

(a) Inception module, naïve version

(b) Inception module with dimension reductions

Use padding to make sure filter outputs have same size. Concatenate along depth.

Source: https://arxiv.org/pdf/1409.4842v1.pdf

GoogLeNet (Inception network V1)

- Solution: make network **wider instead of deeper** having filters of **multiple sizes** operating at the **same level**

(a) Inception module, naïve version

```
A = tensor of size (14, 14, 2)
B = tensor of size (16, 16, 3)
result = DepthConcat([A, B])
```

Use padding to make sure filter outputs have same size. Concatenate along depth.

Source: https://arxiv.org/pdf/1409.4842v1.pdf

GoogLeNet (Inception network V1)

- Solution: make network **wider instead of deeper** having filters of **multiple sizes** operating at the **same level**

(a) Inception module, naïve version

(b) Inception module with dimension reductions

This naive version of the Inception unit is very expensive!

Source: https://arxiv.org/pdf/1409.4842v1.pdf

GoogLeNet (Inception network V1)

- Solution: make network **wider instead of deeper** having filters of **multiple sizes** operating at the **same level**

(a) Inception module, naïve version

(b) Inception module with dimension reductions

Solution: use 1x1 convolutions to limit the number of channels

Source: https://arxiv.org/pdf/1409.4842v1.pdf

GoogLeNet (Inception network V1)

Source: https://www.jeremyjordan.me/convnet-architectures/

GoogLeNet (Inception network V1)

22 layers deep (VGG was 16) but “only” 5 million parameters (VGG had 138 million)

Source: https://www.jeremyjordan.me/convnet-architectures/

GoogLeNet (Inception network V1)

$$\frac{\partial \mathcal{L}}{\partial w^l} = \frac{\partial \mathcal{L}}{\partial a^L} \cdot \frac{\partial a^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial a^{L-2}} \cdot \ \dots \cdot \frac{\partial a^l}{\partial w^l}$$

Still, too deep to avoid the **vanishing gradient problem**

Source: https://www.jeremyjordan.me/convnet-architectures/

GoogLeNet (Inception network V1)

$$\frac{\partial \mathcal{L}}{\partial w^l} = \frac{\partial \mathcal{L}}{\partial a^L} \cdot \frac{\partial a^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial a^{L-2}} \cdot \ \dots \cdot \frac{\partial a^l}{\partial w^l}$$

Many terms < 1 lead to an extremely small gradient for first layers

Source: https://www.jeremyjordan.me/convnet-architectures/

GoogLeNet (Inception network V1)

Solution: intermediate classifiers

Source: https://www.jeremyjordan.me/convnet-architectures/

GoogLeNet (Inception network V1)

```
# The total loss used by the inception net during training.
total_loss = real_loss + 0.3 * aux_loss_1 + 0.3 * aux_loss_2
```

Source: https://www.jeremyjordan.me/convnet-architectures/

Inception network V3

Source: https://www.jeremyjordan.me/convnet-architectures/

Inception network V3

Factorise filters (like VGG); factorise n x n in 1 x n and n x 1;
wider nets; RMSprop; BatchNorm; etc.
(results in 23 million parameters)

Source: https://www.jeremyjordan.me/convnet-architectures/

ResNet: ILSVRC 2015 winner

K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

ResNet

- The residual module
  - Introduce *skip* or *shortcut* connections (existing before in various forms in literature)
  - Make it easy for network layers to represent the identity mapping

K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

ResNet

Deeper residual module (bottleneck)

- Directly performing 3x3 convolutions with 256 feature maps at input and output:
  256 x 256 x 3 x 3 ~ 600K operations
- Using 1x1 convolutions to reduce 256 to 64 feature maps, followed by 3x3 convolutions, followed by 1x1 convolutions to expand back to 256 maps:
  256 x 64 x 1 x 1 ~ 16K
  64 x 64 x 3 x 3 ~ 36K
  64 x 256 x 1 x 1 ~ 16K
  Total: ~70K

K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

ResNet

- Architectures for ImageNet:

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

Inception v4

C. Szegedy et al., [Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning](), arXiv 2016

Summary: ILSVRC 2012-2015

Team	Year	Place	Error (top-5)	External data
SuperVision – Toronto (AlexNet, 7 layers)	2012	-	16.4%	no
SuperVision	2012	1st	15.3%	ImageNet 22k
Clarifai – NYU (7 layers)	2013	-	11.7%	no
Clarifai	2013	1st	11.2%	ImageNet 22k
VGG – Oxford (16 layers)	2014	2nd	7.32%	no
GoogLeNet (19 layers)	2014	1st	6.67%	no
ResNet (152 layers)	2015	1st	3.57%	
Human expert*			5.1%	

http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

What's missing from the picture?

- Training tricks and details: initialization, regularization, normalization
- Training data augmentation
- Averaging classifier outputs over multiple crops/flips
- Ensembles of networks

- Officially, starting with 2015, image classification is not part of ILSVRC challenge, but people continue to benchmark on the data

Comparing architectures

https://culurciello.github.io/tech/2016/06/04/nets.html

Interpreting ResNets

- ResNets are collections of many paths of different length, and shorter paths predominantly contribute to training

(a) Conventional 3-block residual network

(b) Unraveled view of (a)

Figure 1: Residual Networks are conventionally shown as (a), which is a natural representation of Equation (1). When we expand this formulation to Equation (6), we obtain an *unraveled view* of a 3-block residual network (b). Circular nodes represent additions. From this view, it is apparent that residual networks have $O(2^n)$ implicit paths connecting input and output and that adding a block doubles the number of paths.

A. Veit, M. Wilber, S. Belongie, Residual Networks Behave Like Ensembles of Relatively Shallow Networks, NIPS 2016

Beyond ResNet: FractalNet

- Claim: key to good performance is not having skip connections (residuals), but having both shallow and deep paths

S. Larsson, M. Maire and G. Shakhnarovich, [FractalNet: Ultra-Deep Neural Networks without Residuals], ICLR 2017

Beyond ResNet: Stochastic depth

- At training time, in each mini-batch, randomly drop a subset of layers (bypass with identity function). At test time, use full network
  - Forces the network to learn better, speeds up convergence, improves accuracy

G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger, Deep Networks with Stochastic Depth, ECCV 2016

Beyond ResNet: Wide ResNet

- Reduce number of residual blocks, but increase number of feature maps in each block
  - More parallelizable, better feature reuse
  - 16-layer WRN outperforms 1000-layer ResNets, though with much larger # of parameters

Basic residual block Wide residual block

Image source

S. Zagoryuko and N. Komodakis, Wide Residual Networks, BMVC 2016

Beyond ResNet: ResNeXt

- Propose “cardinality” as a new factor in network design, apart from depth and width
- Claim that increasing cardinality is a better way to increase capacity than increasing depth or width

ResNet

ResNeXt, roughly same complexity

S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, Aggregated Residual Transformations for Deep Neural Networks, CVPR 2017

Beyond ResNet: ResNeXt

	setting	top-1 error (%)
ResNet-50	1×64d	23.9
ResNeXt-50	2×40d	23.0
ResNeXt-50	4×24d	22.6
ResNeXt-50	8×14d	22.3
ResNeXt-50	32×4d	**22.2**
ResNet-101	1×64d	22.0
ResNeXt-101	2×40d	21.7
ResNeXt-101	4×24d	21.4
ResNeXt-101	8×14d	21.3
ResNeXt-101	32×4d	**21.2**

Table 3. Ablation experiments on ImageNet-1K. (**Top**): ResNet-50 with preserved complexity ($\sim$4.1 billion FLOPs); (**Bottom**): ResNet-101 with preserved complexity ($\sim$7.8 billion FLOPs). The error rate is evaluated on the single crop of 224×224 pixels.

S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, Aggregated Residual Transformations for Deep Neural Networks, CVPR 2017

DenseNets

G. Huang, Z. Liu, and L. van der Maaten, [Densely Connected Convolutional Networks](), CVPR 2017 (Best Paper Award)

DenseNets

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

G. Huang, Z. Liu, and L. van der Maaten, Densely Connected Convolutional Networks, CVPR 2017 (Best Paper Award)

DenseNets

ImageNet validation error vs. number of parameters and test-time operations

G. Huang, Z. Liu, and L. van der Maaten, Densely Connected Convolutional Networks, CVPR 2017 (Best Paper Award)

Design principles

- Make networks parameter-efficient
  - Reduce filter sizes, factorize filters
  - Use 1x1 convolutions to reduce number of feature maps before more expensive operations
  - Minimize reliance on FC layers
- Reduce spatial resolution gradually, within each level of resolution replicate a given “block” multiple times
- Use skip connections and/or create multiple redundant paths through the network
- Play around with depth vs. width vs. “cardinality”

How to use a trained network for a new task?

- Take the vector of activations from one of the fully connected (FC) layers and treat it as an off-the-shelf feature
- Train a new classifier layer on top of the FC layer
- *Fine-tune* the whole network

Example: CNNs for image captioning

FC vectors from pre-trained network

O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. CVPR 2015

Reading list (incomplete)

- https://culurciello.github.io/tech/2016/06/04/nets.html
- Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86(11):
  2278–2324, 1998.
- A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012
- M. Zeiler and R. Fergus, Visualizing and Understanding Convolutional Networks,
  ECCV 2014
- K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015
- M. Lin, Q. Chen, and S. Yan, Network in network, ICLR 2014
- C. Szegedy et al., Going deeper with convolutions, CVPR 2015
- C. Szegedy et al., Rethinking the inception architecture for computer vision,
  CVPR 2016
- K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition, CVPR 2016
- G. Huang, Z. Liu, and L. van der Maaten, Densely Connected Convolutional Networks, CVPR 2017

Summary

- CNN use convolutions to exploit grid structure
- Parameters sharing and sparse connections allow to scale to larger inputs
- Pooling introduces invariance to transformations
- Modern architectures use several tricks to increase depth and reduce computational cost

Second assignment- Part 1 CNN

- Implement a reduced version of the VGG network using PyTorch (see next slide for structure)

Source: https://neurohive.io/en/popular-networks/vgg16/

Second assignment- Part 1 CNN

1. conv layer: k=3x3, s=1, p=1, in=3, out=64
2. maxpool: k=3x3, s=2, p=1, in=64, out=64
3. conv layer: k=3x3, s=1, p=1, in=64, out=128
4. maxpool: k=3x3, s=2, p=1, in=128, out=128
5. conv layer: k=3x3, s=1, p=1, in=128, out=256
6. conv layer: k=3x3, s=1, p=1, in=256, out=256
7. maxpool: k=3x3, s=2, p=1, in=256, out=256
8. conv layer: k=3x3, s=1, p=1, in=256, out=512
9. conv layer: k=3x3, s=1, p=1, in=512, out=512
10. maxpool: k=3x3, s=2, p=1, in=512, out=512
11. conv layer: k=3x3, s=1, p=1, in=512, out=512
12. conv layer: k=3x3, s=1, p=1, in=512, out=512
13. maxpool: k=3x3, s=2, p=1, in=512, out=512
14. linear, in=512, out=10

Second assignment- Part 1 CNN

1. conv layer: k=3x3, s=1, p=1, in=3, out=64
2. maxpool: k=3x3, s=2, p=1, in=64, out=64
3. conv layer: k=3x3, s=1, p=1, in=64, out=128
4. maxpool: k=3x3, s=2, p=1, in=128, out=128
5. conv layer: k=3x3, s=1, p=1, in=128, out=256
6. conv layer: k=3x3, s=1, p=1, in=256, out=256
7. maxpool: k=3x3, s=2, p=1, in=256, out=256
8. conv layer: k=3x3, s=1, p=1, in=256, out=512
9. conv layer: k=3x3, s=1, p=1, in=512, out=512
10. maxpool: k=3x3, s=2, p=1, in=512, out=512
11. conv layer: k=3x3, s=1, p=1, in=512, out=512
12. conv layer: k=3x3, s=1, p=1, in=512, out=512
13. maxpool: k=3x3, s=2, p=1, in=512, out=512
14. linear, in=512, out=10

Each “conv” layer is composed of:
2D-CONV + Bath Normalisation + ReLU

Second assignment- Part 1 CNN

1. conv layer: k=3x3, s=1, p=1, in=3, out=64
2. maxpool: k=3x3, s=2, p=1, in=64, out=64
3. conv layer: k=3x3, s=1, p=1, in=64, out=128
4. maxpool: k=3x3, s=2, p=1, in=128, out=128
5. conv layer: k=3x3, s=1, p=1, in=128, out=256
6. conv layer: k=3x3, s=1, p=1, in=256, out=256
7. maxpool: k=3x3, s=2, p=1, in=256, out=256
8. conv layer: k=3x3, s=1, p=1, in=256, out=512
9. conv layer: k=3x3, s=1, p=1, in=512, out=512
10. maxpool: k=3x3, s=2, p=1, in=512, out=512
11. conv layer: k=3x3, s=1, p=1, in=512, out=512
12. conv layer: k=3x3, s=1, p=1, in=512, out=512
13. maxpool: k=3x3, s=2, p=1, in=512, out=512
14. linear, in=512, out=10

Whenever not specified otherwise, use default pytorch parameters

- Next Topic: RNN