Statistical Machine Learning

https://cvml.ist.ac.at/courses/SML_W20

Christoph Lampert (with material by Andrea Palazzi and others)

IST AUSTRIA

Institute of Science and Technology

Fall Semester 2020/2021

Lecture 11

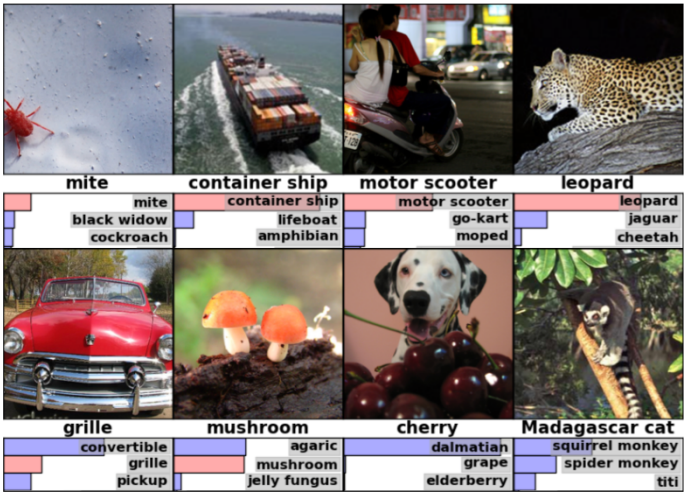
Overview (tentative)

Date		no.	Topic
Oct 05	Mon	1	A Hands-On Introduction
Oct 07	Wed	2	Bayesian Decision Theory, Generative Probabilistic Models
Oct 12	Mon	3	Discriminative Probabilistic Models
Oct 14	Wed	4	Maximum Margin Classifiers, Generalized Linear Models
Oct 19	Mon	5	Estimators; Overfitting/Underfitting, Regularization, Model Selection
Oct 21	Wed	6	Bias/Fairness, Domain Adaptation
Oct 26	Mon	-	no lecture (public holiday)
Oct 28	Wed	7	Learning Theory I, Concentration of Measure
Nov 02	Mon	8	Learning Theory II
Nov 04	Wed	9	Learning Theory III, Deep Learning I
Nov 09	Mon	10	Deep Learning II
Nov 11	Wed	11	Deep Learning III
Nov 16	Mon	12	project presentations
Nov 18	Wed	13	buffer

Convolutional (Neural) Networks – CNNs, ConvNets

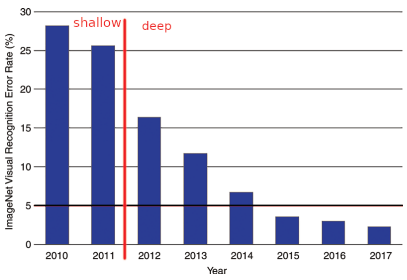
Example: Image Classification

One of the most popular benchmark tasks where Deep Networks excel is image classification,



ImageNet Large Scale Visual Recognition Challenge (ILSVRC):

- 1.2 million train images
- 1000 object categories



Example

Image Classification

- input $\mathcal{X} \subset \mathbb{R}^{150528}$: RGB images at resolution 224×224 $\quad (224 \times 224 \times 3 = 150528)$
- output $\mathcal{Y} = \{1, \ldots, K\}$ with $K = 1000$ object categories

What models can we build?

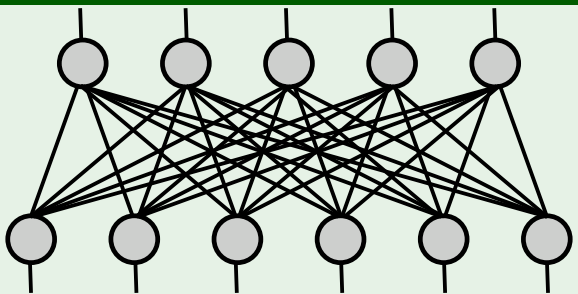
- linear model: $h(x) = Wx$ with $W \in \mathbb{R}^{1000 \times 150528}$
  $\rightarrow$ ≈150,000,000 parameters, 600MB RAM
- network with 1 hidden layer of size 150528: $h(x) = W_2\sigma(W_1x)$ $\quad W_1 \in \mathbb{R}^{150528 \times 150528}$
  $\rightarrow$ ≈22,000,000,000 parameters, 88GB RAM
- deep network with 50 hidden layer of size 150528
  $\rightarrow > 1$ trillion parameters, 4 TB RAM

For high-dimensional inputs, such as images, the number of parameters quickly gets excessively large. One has to, either

- make hidden layers very narrow, or think of something else.

Weight matrices are so big, because every neuron in layer $l+1$ is connected to every neuron in layer l:

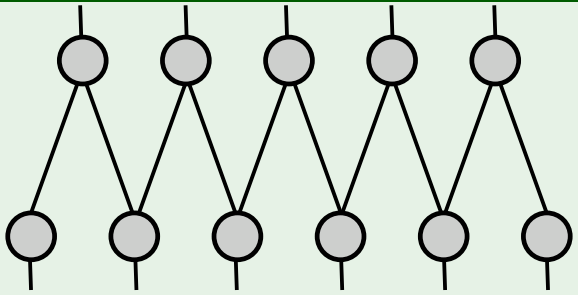
Fully-Connected Layer



- dense $W \in \mathbb{R}^{d_{out} \times d_{in}}$, $\rightarrow$ here: $5 \times 6 = 30$ entries
- if we increase width of all layers, number of free parameters grows quadratically

Weight matrices are much smaller, if every neuron in layer $l+1$ is connected only to subset of neurons in layer l:

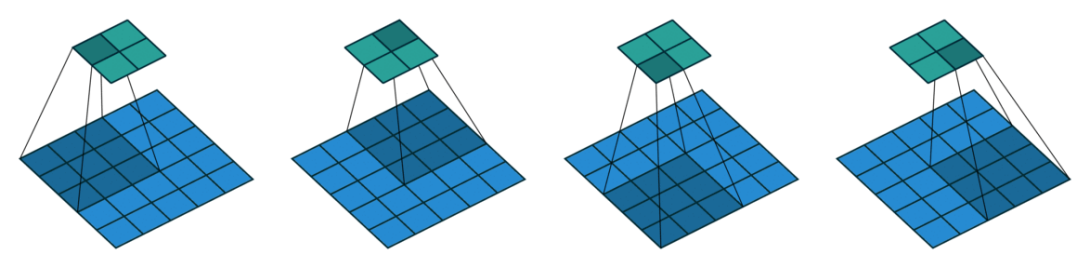
Example: Layers with Restricted Receptive Field (1D)



- sparse $W \in \mathbb{R}^{d_{out}} \times d_{in}$ with at most k non-zero entries at fixed positions per row
  $\rightarrow$ here $k = 2$: $5 \times 2 = 10$ entries
- if we increase width of all layers (but keep k fixed), number of free parameters grows linearly

For multi-dimensional inputs, e.g. images, neuron are connected locally:

Example: Layers with Restricted Receptive Field (2D)

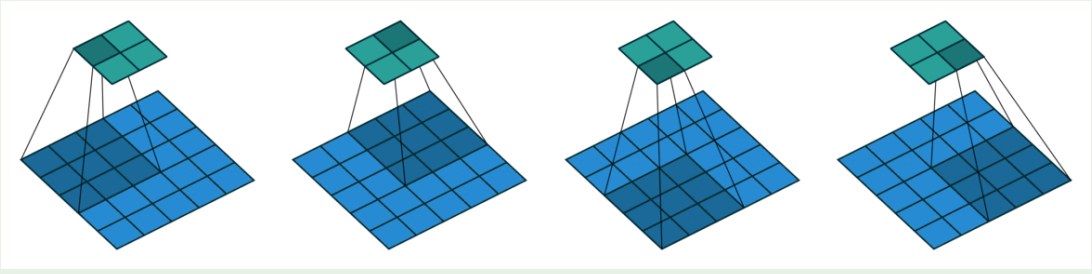


Many properties of images are local, and a small receptive field suffices to identify them:

- is this region bright or dark?
- is this region a smooth or rough region?
- is there a vertical/horizontal/diagonal line here?
- is there a corner here?

For multi-dimensional inputs, e.g. images, neuron are connected locally:

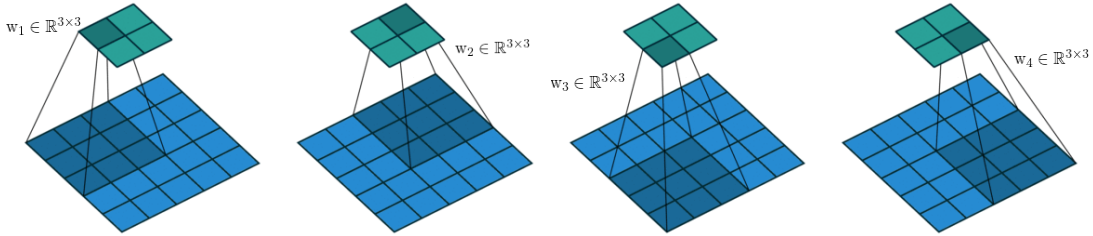
Example: Layers with Restricted Receptive Field (2D)



Later layers can combine local information from previous layers:

- is there a corner? check for a vertical line next to a horizontal line
- is there a stop sign? find eight red matching corners
- etc.

Weight Sharing



Example property: for each 3×3 region, is there a vertical line here?

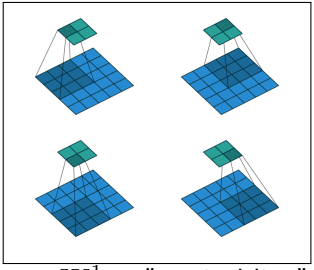
- $W = (w_1, w_2, w_3, w_4) \in \mathbb{R}^{4\times(3\times3)}$
- $w_1 = w_2 = w_3 = w_4 = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$ why learn individual weights at all then?

Weight sharing:

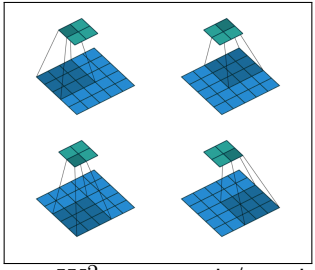
- use same weights for each neuron in layer (they have different receptive fields)
- number of free parameters = size of receptive field: $3 \times 3 = 9$, regardless of d_{l-1} and d_l

With weight sharing, the weight matrix between layer l and layer $l+1$ learns learns one local property for each neuron of layer $l+1$, i.e. for all local regions of layer l.

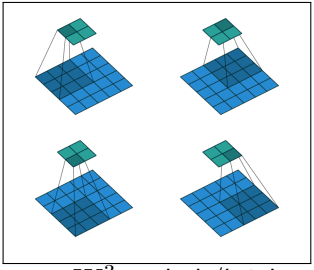
Ultimately, one visual property is not enough, we will need multiple
$\rightarrow$ multiple weight matrices for each receptive field $\rightarrow$ multiple output values for each neuron



e.g. W^1 = "vertical line"



e.g. W^2 = smooth/rough



e.g. W^3 = dark/bright

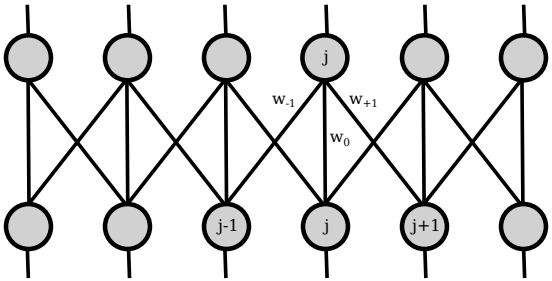
(of course, these are just examples, in reality, the network *learns* W^1, W^2, W^3)

Example: for 1D data, receptive fields are short intervals

- neuron j in layer l takes as input neurons $(j-K, j-K+1, \ldots, j+K-1, j+K)$ from layer $l-1$ and computes

$$y^l[j] = \sum_{k=-K}^{K} w[k] a^{l-1}[j-k] \qquad (1)$$

with weights $w = (w[-K], w[-K+1], \ldots, w[K]) \in \mathbb{R}^{2K+1}$

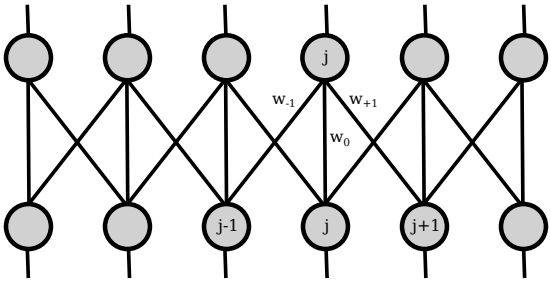


Example: for 1D data, receptive fields are short intervals

- neuron j in layer l takes as input neurons $(j-K, j-K+1, \ldots, j+K-1, j+K)$ from layer $l-1$ and computes

$$y^l[j] = \sum_{k=-K}^{K} w[k] a^{l-1}[j-k] \qquad (1)$$

with weights $w = (w[-K], w[-K+1], \ldots, w[K]) \in \mathbb{R}^{2K+1}$



Observation: (1) is a convolution operation between a^{l-1} and $\hat{w}$ (a flipped version of w)

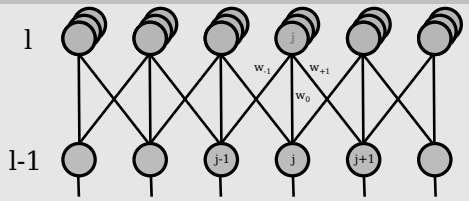
$$y^l[j] = \sum_{k=-K}^{K} w[k] a^{l-1}[j-k] = a^{l-1} * \hat{w} \qquad \text{with } \hat{w}[k] = w[-k]$$

Layers of this type are called convolutional layers with convolution kernel w

Example (1D data with C output channels)

- $a^{l-1} \in \mathbb{R}^{d_{l-1}} \quad w \in \mathbb{R}^{K \times C} \quad a^{l} \in \mathbb{R}^{d_l \times C}$

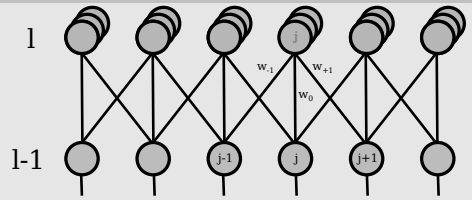
$$y^l[j;c] = \sum_{k=-K}^{K} w[k;c] a^{l-1}[j-k]$$



Example (1D data with C output channels)

- $a^{l-1} \in \mathbb{R}^{d_{l-1}} \quad w \in \mathbb{R}^{K \times C} \quad a^l \in \mathbb{R}^{d_l \times C}$

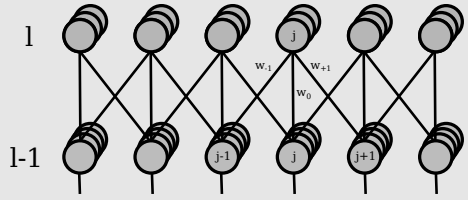
$$y^l[j;c] = \sum_{k=-K}^{K} w[k;c] a^{l-1}[j-k]$$



Example (1D data with C input and C' output channels)

- $a^{l-1} \in \mathbb{R}^{d_{l-1} \times C} \quad w \in \mathbb{R}^{K \times C' \times C}$
  $a^l \in \mathbb{R}^{d_l \times C'}$

$$y^l[j,c'] = \sum_{c=1}^{C} \sum_{k=-K}^{K} w[k;c',c] a^{l-1}[j-k;c]$$



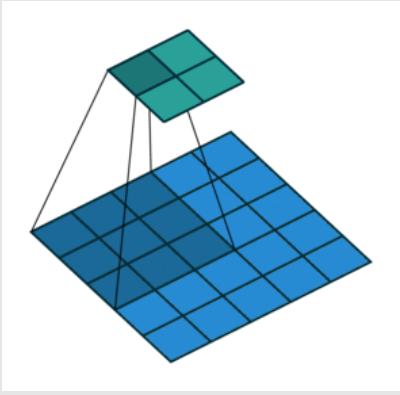
Example (2D data with layers arranged as 2D arrays)

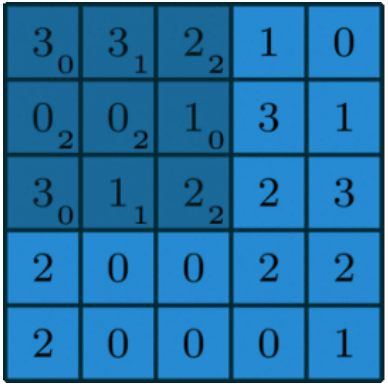
Receptive fields are small $(K \times L)$-rectangular regions:

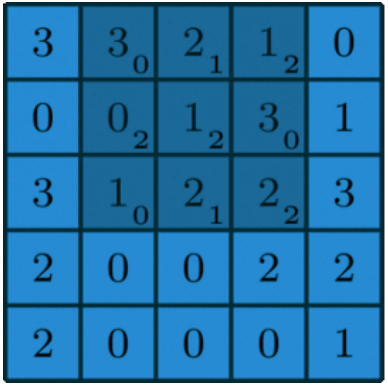
- neuron (i, j) in layer l takes as input neurons $(i-K, i-K+1, \ldots, i+K-1, i+K) \times (j-L, j-L+1, \ldots, j+L-1, j+L)$ from layer $l-1$ and computes

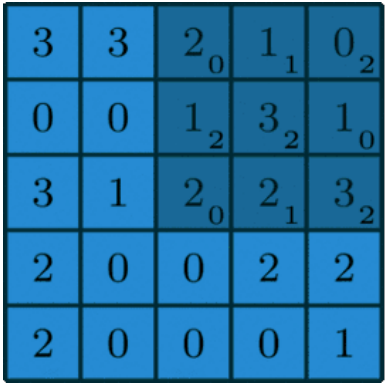
$$y_j^l = \sum_{k=-K}^{K} \sum_{l=-L}^{K} w[k,l] a^{l-1}[i-l, j-k]$$

for weights $(w[k,l])_{\substack{k=-K,\ldots,K \\ l=-L,\ldots,L}}$

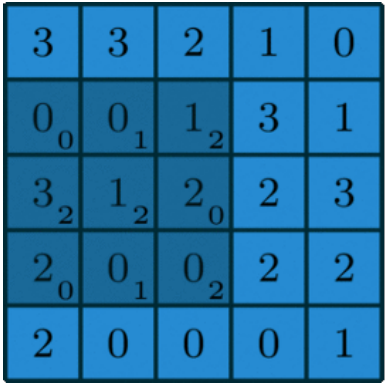



3 3 2 1 0
0 0 1 3 1
3 1 2 2 3
2 0 0 2 2
2 0 0 0 1
12.0 12.0 17.0
10.0 17.0 19.0
9.0 6.0 14.0

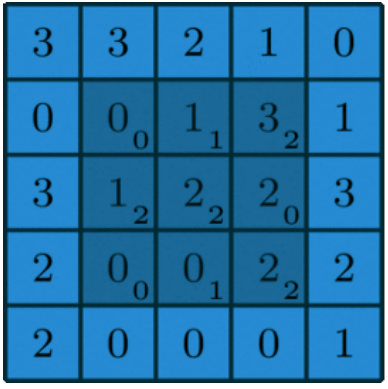

3 3 2 1 0
0 0 1 3 1
3 1 2 2 3
2 0 0 2 2
2 0 0 0 1
0 1 2
2 1 0
2 2 0
12.0 12.0 17.0
10.0 17.0 19.0
9.0 6.0 14.0


3 3 2 1 0
0 0 1 3 1
3 1 2 2 3
2 0 0 2 2
2 0 0 0 1
0 1 2
2 2 0
0 1 2

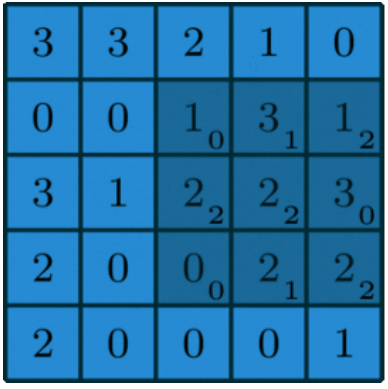

12.0 12.0 17.0
10.0 17.0 19.0
9.0 6.0 14.0


3 3 2 1 0
0 0 1 3 1
3 1 2 2 3
2 0 0 2 2
2 0 0 0 1


12.0 12.0 17.0
10.0 17.0 19.0
9.0 6.0 14.0

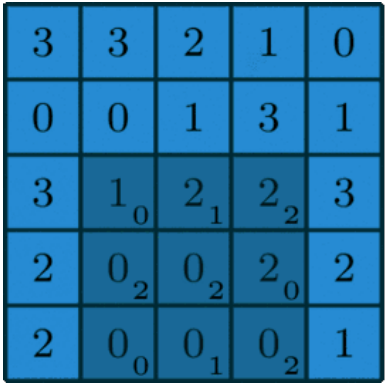

3 3 2 1 0
0 0 1 3 1
3 1 2 2 3
2 0 0 2 2
2 0 0 0 1
0 0
1 1
3 2
1 2
2 2
2 0
0 0
0 1
2 2


12.0 12.0 17.0
10.0 17.0 19.0
9.0 6.0 14.0


3 3 2 1 0
0 0 1 3 1
3 1 2 2 2
2 0 0 2 1
2 0 0 0 1
12.0 12.0 17.0
10.0 17.0 19.0
9.0 6.0 14.0


3 3 2 1 0
0 0 1 3 1
3 1 2 2 3
2 0 0 2 2
2 0 0 1 2
0 1 2
2 0 1
2 0 2


12.0 12.0 17.0
10.0 17.0 19.0
9.0 6.0 14.0


3 3 2 1 0
0 0 1 3 1
3 1 0 2 1 2 2 3
2 0 2 0 2 2 0 2
2 0 0 0 1 0 2 1

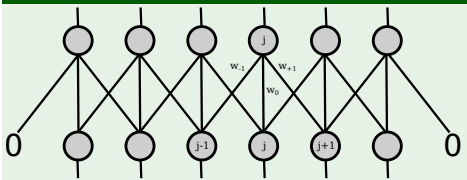

12.0 12.0 17.0
10.0 17.0 19.0
9.0 6.0 14.0


3 3 2 1 0
0 0 1 3 1
3 1 2 0 2 1 3 2
2 0 0 2 2 2 2 0
2 0 0 0 0 1 1 2


12.0 12.0 17.0
10.0 17.0 19.0
9.0 6.0 14.0

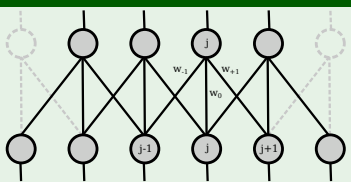
How to treat boundary cases?

Zero padding



Set inputs of "missing" neurons to 0.

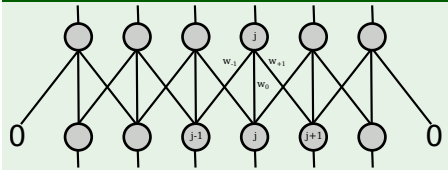
"Valid" padding



Don't allow neurons where receptive field would fallside of the input layer

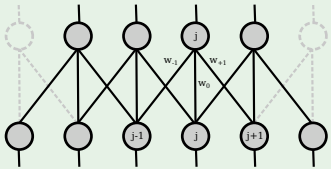
Observation: for convolutional layers, we cannot chose their sizes arbitrarily

Zero padding



- size of layer l is the same as for layer $l-1$
- the number of channels for each layer is arbitrary

"Valid" padding



- in 1D, for filters size $(2K+1)$

$$n_l = n_{l-1} - K$$

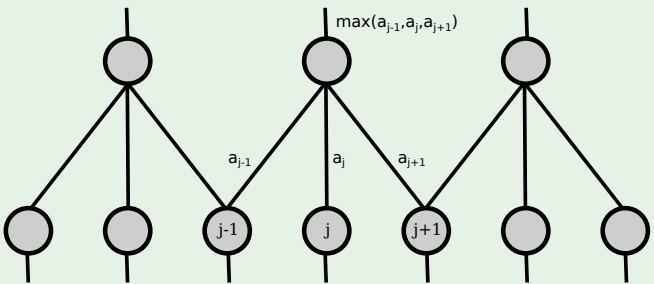
- in 2D, for filters size $(2K+1) \times (2L+1)$

$$(n_l^1, n_l^2) = (n_{l-1}^1 - K, n_{l-1}^2 - L)$$

- the number of channels for each layer is arbitrary

Other ways to combine local information besides convolutions?

Max Pooling

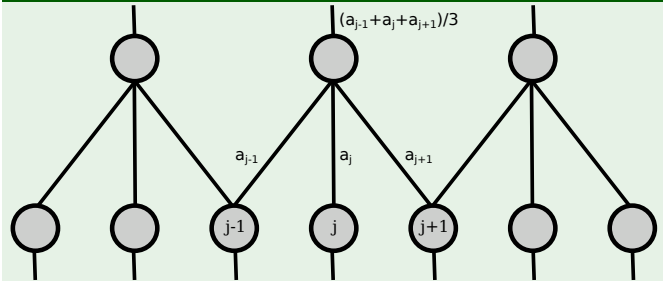


Non-linear pooling layer

- input: receptive field, jumps K neurons each time (="stride")
- output: maximum over all inputs
- no parameters to learn
- $n_l \approx n_{l-1}/K$

Other ways to combine local information besides convolutions?

Average Pooling

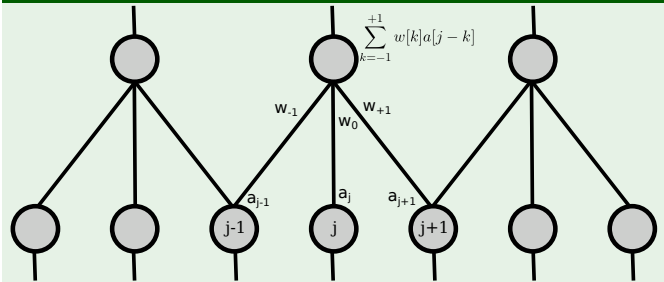


Linear pooling layer

- input: receptive field, stride K
- output: maximum over all inputs
- no parameters to learn
- $n_l \approx n_{l-1}/K$

Other ways to combine local information besides convolutions?

Strided Convolutions



Ordinary convolutional layer

- input: receptive field, stride K
- output: convolution over inputs
- filter kernel to learn
- $n_l \approx n_{l-1}/K$

The most famous ConvNet in the world:

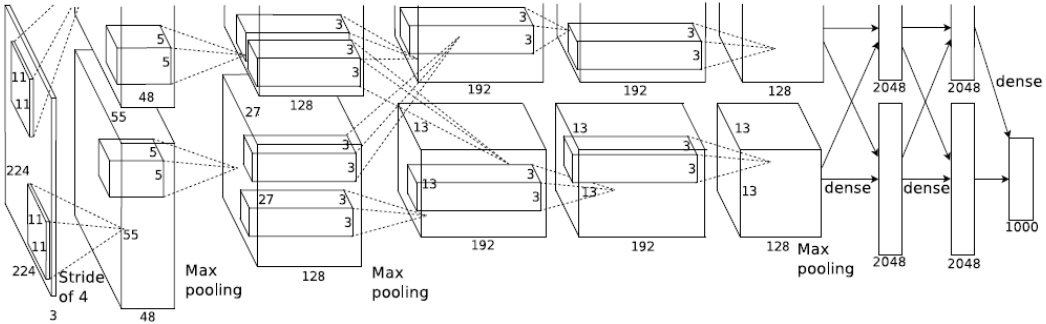


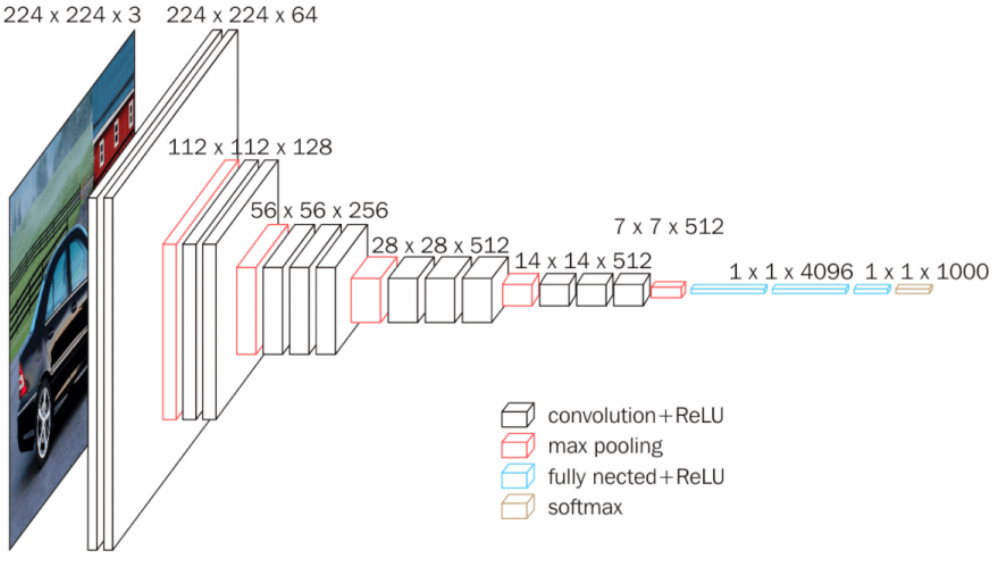
Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input 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.

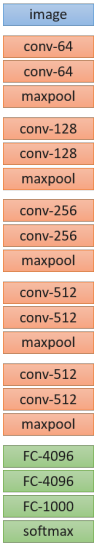
The **VGG** ConvNet architecture was proposed in 2014 by the Oxford Vision Geometry Group.

It quickly became popular for a variety of reasons:

- great performance (in the sense of accuracy) in benchmark tasks
- pre-trained weights were released
- a wow-effect because the network was much deeper than previous ones
- general smart (simple but elegant) architectural choices

VGG16 Architecture

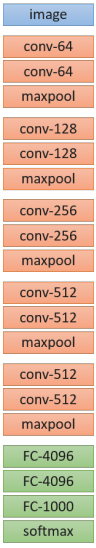




Architecture:

- Input: fixed size 224x224 RGB images
- Convolutional filters: 3x3 receptive field (smallest size that makes sense)
- Max pooling: over 2x2 pixel window with stride 2
- ReLu activation in all hidden layers
- Fully connected layers with 4096 neurons each followed by ReLu
- Output: dimension 1000 (number of ImageNet classes) with softmax activation

VGG16 Architecture



Architecture:

- Input: fixed size 224x224 RGB images
- Convolutional filters: 3x3 receptive field (smallest size that makes sense)
- Max pooling: over 2x2 pixel window with stride 2
- ReLu activation in all hidden layers
- Fully connected layers with 4096 neurons each followed by ReLu
- Output: dimension 1000 (number of ImageNet classes) with softmax activation

Properties:

- memory usage per image: 93MB (forward pass), 186MB (forward+backward)
- number of learnable parameters: 138 million
  - almost all in fully-connected part
  - quiz: one single layer is responsible for 100 millions parameters. Can you spot it?

Fully-convolutional networks

Common structure for image classification networks, including AlexNet and VGG:

- first: some convolutional and local pooling layers
  - weight are convolution kernels, $W \in \mathbb{R}^{K \times C \times C'}$ (1d) or $W \in \mathbb{R}^{K \times L \times C \times C'}$ (2d)
    $\rightarrow$ not restricted to a specific input size, number of channels must be fixed
  - parametrizes a function

$$\phi : \mathbb{R}^{W \times H \times 3} \rightarrow \mathbb{R}^{\lfloor \frac{W-a}{b} \rfloor \times \lfloor \frac{H-c}{d} \rfloor \times C'} \qquad \text{for (almost) any choice of } W, H$$

- then: some fully connected layers
  - weight are fixed size, $W \in \mathbb{R}^{n_l \times n_{l+1}}$
    $\rightarrow$ input and output must have specific number of neurons
  - parametrizes a function

$$c : \mathbb{R}^{\lfloor \frac{W-a}{b} \rfloor \times \lfloor \frac{H-c}{d} \rfloor \times C'} \rightarrow \mathbb{R}^{n_{out}} \qquad \text{for a single choice of } W, H$$

Common structure for image classification networks, including AlexNet and VGG:

- first: some convolutional and local pooling layers
  - weight are convolution kernels, $W \in \mathbb{R}^{K \times C \times C'}$ (1d) or $W \in \mathbb{R}^{K \times L \times C \times C'}$ (2d)
    $\rightarrow$ not restricted to a specific input size, number of channels must be fixed
  - parametrizes a function

$$\phi : \mathbb{R}^{W \times H \times 3} \rightarrow \mathbb{R}^{\lfloor \frac{W-a}{b} \rfloor \times \lfloor \frac{H-c}{d} \rfloor \times C'} \qquad \text{for (almost) any choice of } W, H$$

- then: some fully connected layers
  - weight are fixed size, $W \in \mathbb{R}^{n_l \times n_{l+1}}$
    $\rightarrow$ input and output must have specific number of neurons
  - parametrizes a function

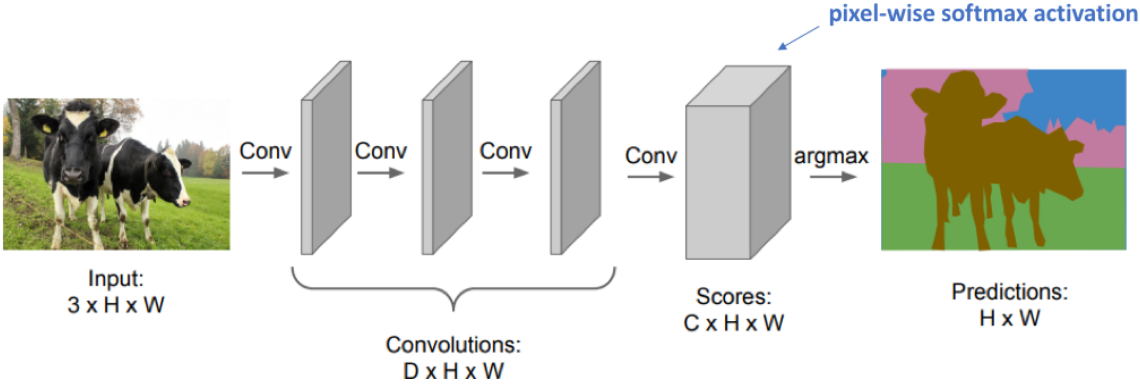
$$c : \mathbb{R}^{\lfloor \frac{W-a}{b} \rfloor \times \lfloor \frac{H-c}{d} \rfloor \times C'} \rightarrow \mathbb{R}^{n_{out}} \qquad \text{for a single choice of } W, H$$

What, if there were no fully connected layers at the end, only convolutional ones?

- arbitrary input sizes possible, output size would vary depending on input size

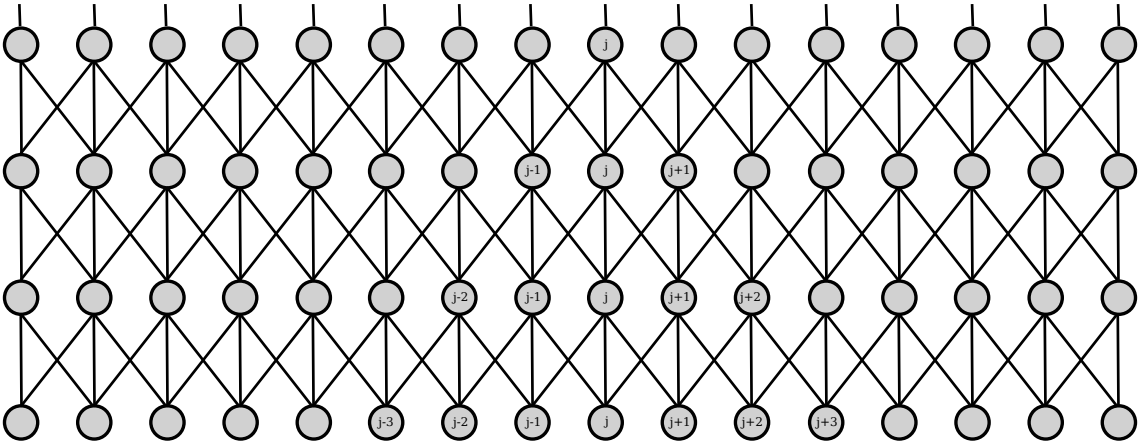
"Fully-convolutional" networks

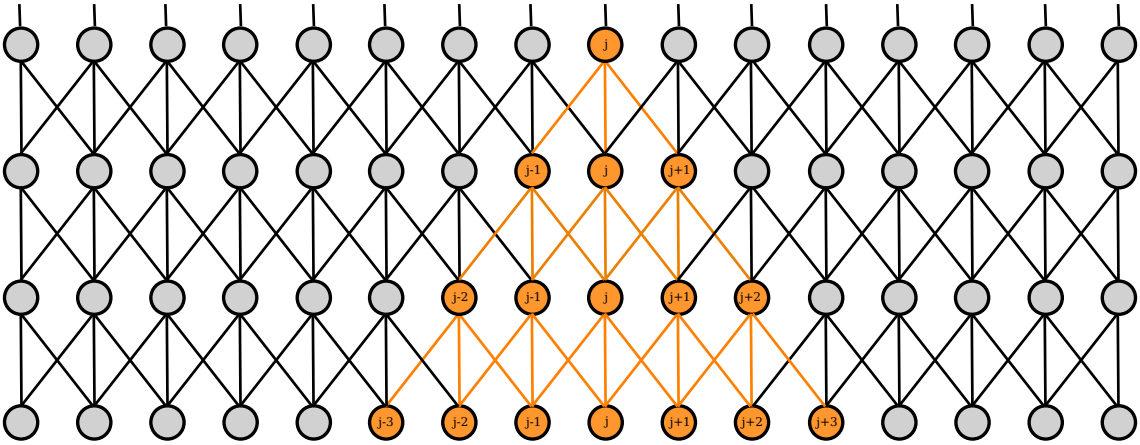
Example: Image Segmentation



Fully-convolutional network without spatial pooling:

- can provide per-pixel segmentations for images of arbitrary sizes $W \times H \times 3$
- disadvantages: for good quality, at least one of the following is needed:
  - very large convolution kernel sizes
  - very many layers


j
j-1
j
j+1
j-2
j-1
j
j+1
j+2
j-3
j-2
j-1
j
j+1
j+2
j+3

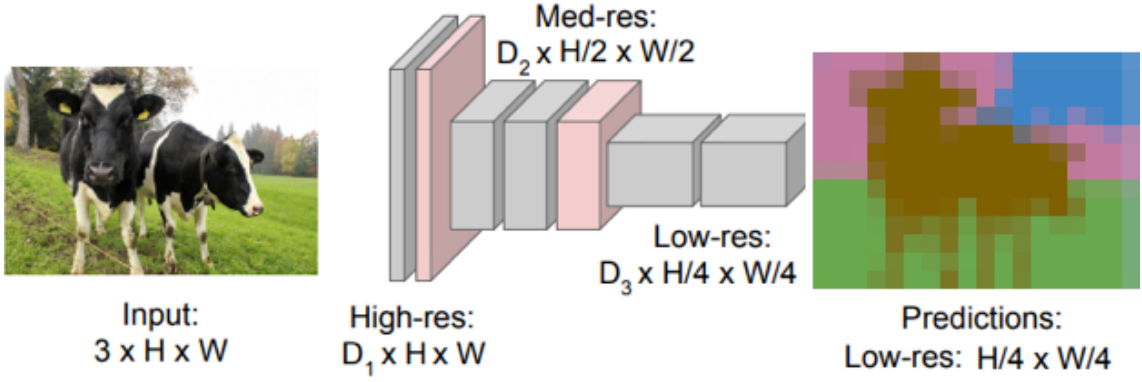


Size of input region that influences an output neuron ("field of view") grows like $O(KL)$,

- K: size of convolution kernel
- L: number of layers

Unless K or L are large, output decisions are based on small subset of input information.

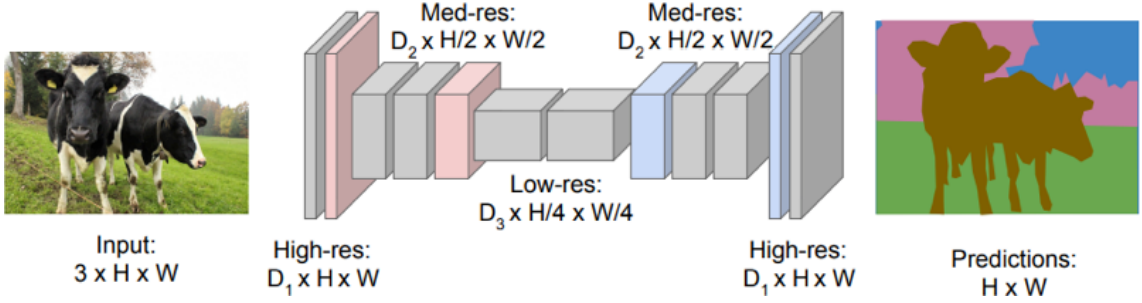
Example: Image Segmentation



Fully-convolutional network without spatial pooling:

- more efficient, because field of view grows much quicker
- disadvantages: output is lower-resolution than input

Example: Image Segmentation



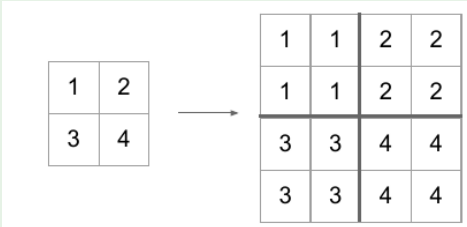
Fully-convolutional network with downscaling (spatial pooling) and upscaling

- efficient: reasonable depth and kernel size, reasonably small intermediate layers
- high quality: wide field of view

What's the "upscaling" part?

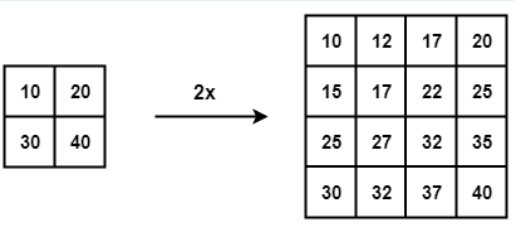
Image Upscaling

Nearest Neighbor



- duplicate pixel values
- efficient, no parameters
- output looks blocky
  $\rightarrow$ better after one more convolution

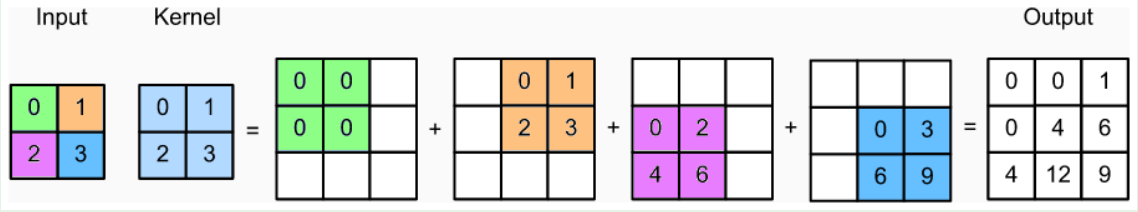
Bilinear Scaling



- bilinear interpolation
- efficient, no parameters
- output looks washed out
  $\rightarrow$ better after one more convolution

Images: https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba

Transpose Convolution (=~~de-convolution~~) ← misnomer



- linear operation: output is sum of kernel multiplied with input values
- kernel entries can be learned
- output looks better than *nearest* or *bilinear*, but sometimes shows some regular artifacts
- formula in matrix notation looks like transpose of a convolution operation

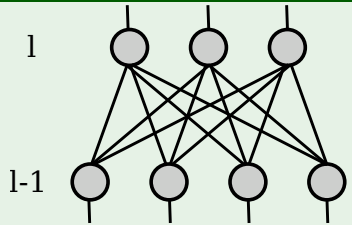
---

Images: https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba

Converting Fully-Connected into Convolutional Layers

An $n_{in} \to n_{out}$ fully-connected layer is equivalent to a convolutional one with kernel size 1 (or 1×1), but $C' = n_{in}$ input channels and $C = n_{out}$ output channels:

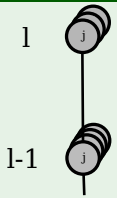
Fully-Connected



- units are neurons of a layer
- for $i = 1, \ldots, n_{out}$:

$$y^l[i] = \sum_{j=1}^{n_{in}} w[i,j] a^{l-1}[j]$$

equivalent ("1×1") convolution



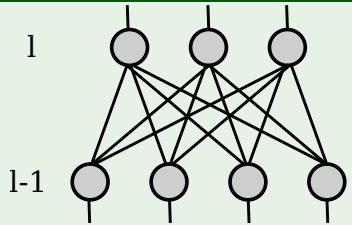
- units are channels of a single neuron n
- for $c = 1, \ldots, C$:

$$y^l[n, c'] = \sum_{c=1}^{C} \sum_{k=0}^{0} w[k; c', c] a^{l-1}[n; c]$$

Converting Fully-Connected into Convolutional Layers

An $n_{in} \to n_{out}$ fully-connected layer is equivalent to a convolutional one with kernel size 1 (or 1×1), but $C' = n_{in}$ input channels and $C = n_{out}$ output channels:

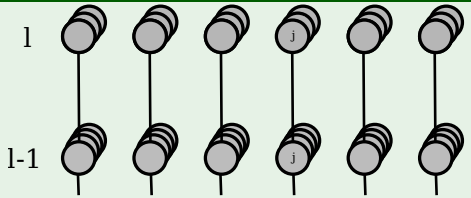
Fully-Connected



- units are neurons of a layer
- for $i = 1, \ldots, n_{out}$:

$$y^l[i] = \sum_{j=1}^{n_{in}} w[i,j] a^{l-1}[j]$$

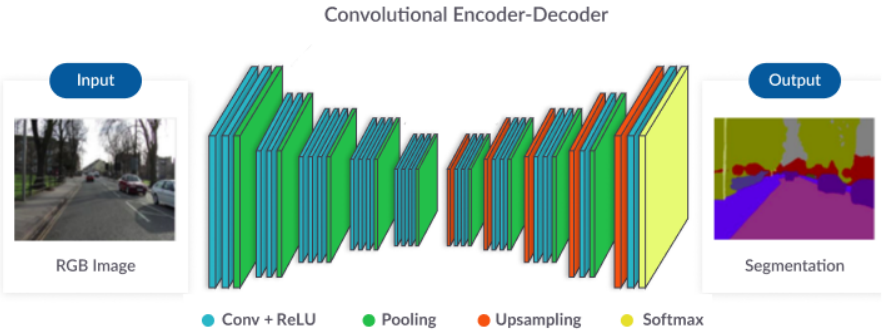
equivalent ("1×1") convolution



- units are channels of a single neuron n
- for $c = 1, \ldots, C$:

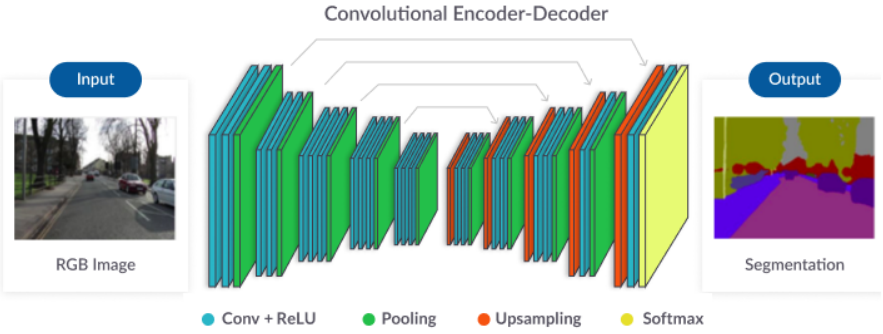
$$y^l[n, c'] = \sum_{c=1}^{C} \sum_{k=0}^{0} w[k; c', c] a^{l-1}[n; c]$$

- in a bigger layer, each neuron is processed separately



In image segmentation, pixel-precise outputs can be important, e.g. road markings.

This is a problem for wide-narrow-wide (encoder-decoder) architectures, because all information from high-res input to high-res output has to flow through low-res intermediate layers.

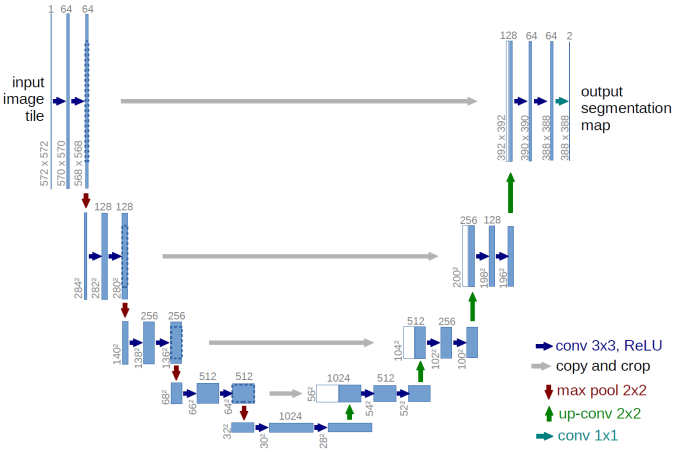


In image segmentation, pixel-precise outputs can be important, e.g. road markings.

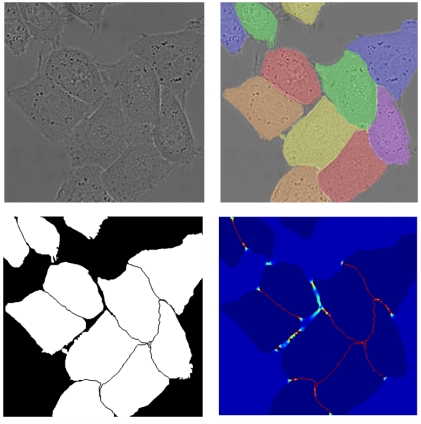
This is a problem for wide-narrow-wide (encoder-decoder) architectures, because all information from high-res input to high-res output has to flow through low-res intermediate layers.

Solution: introduce **skip connection**, directly from early to late layers of same resolution.

Image: Abhishek Kumar, https://medium.com/abhishekkakiak/semantic-segmentation-segnet-a54af19b6d6



Originally introduced for biological image analysis, now also popular elsewhere.



Long-range skip connections provide short-cuts from early to late layers.

Advantage in forward pass:

- information from early layers doesn't arrive "washed out" at later layers
- reduced risk of "vanishing signal" effect

Advantage in backwards pass:

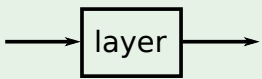
- early layers get more direct gradient signal from loss layer
- reduced risk of "vanishing gradient" effect

These advantage are not specific to image segmentation. Other applications can benefit.

Connections do not even have to be long-range. Short-range skip connections are also useful.

Residual blocks are a re-interpretation of short-range skip connections.

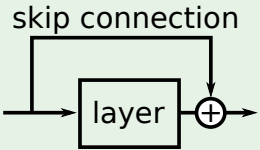
Skip Connection



- output is connected to input by computational layer

Residual blocks are a re-interpretation of short-range skip connections.

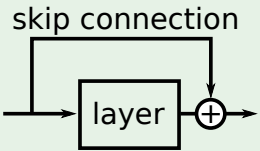
Skip Connection



- output is connected to input by computational layer
- skip connection provides shortcut that just copies values
- result of layer and shortcut are added

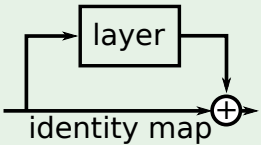
Residual blocks are a re-interpretation of short-range skip connections.

Skip Connection



- output is connected to input by computational layer
- skip connection provides shortcut that just copies values
- result of layer and shortcut are added

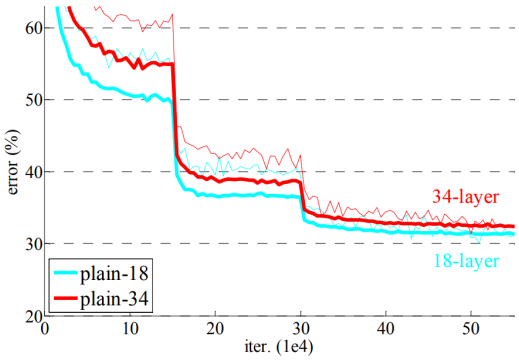
Residual block



- output is connected to input by identity function
- computational layer can provide correction term
- result of identity and layer are added

Puzzling observation:

- increasing the number of layers of a ConvNet (18 to 34) does not lead to lower training error
- puzzling, because the deeper model can encode stricly more functions. For example, extra layers could just learn the identity function.
- note: this is different from *overfitting*, which would be a statement about the test error
- possible reason: optimization is harder for deeper network



Analysis: it's hard for a network layer $h_l(x) = W_l x$ to learn an identity function ($W_l = \mathrm{Id}$) when initialized with random $W_l \approx 0$.

Analysis: it's hard for a network layer $h_l(x) = W_l x$ to learn an identity function ($W_l = \mathrm{Id}$) when initialized with random $W_l \approx 0$.

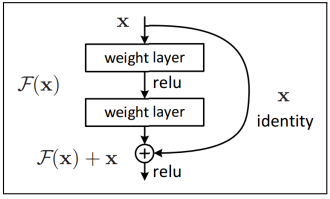
Solution:

- make the identity function explicit

$$h_l(x) = x + W_l x$$

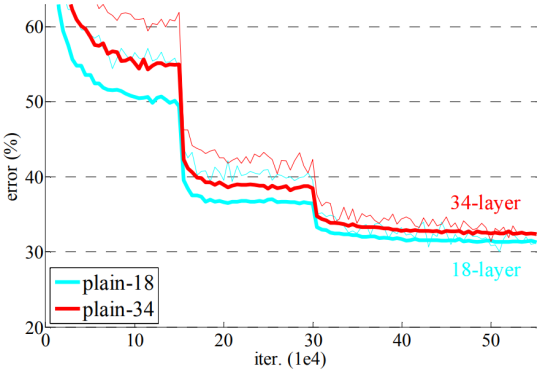
- $W_l x$ acts only as a correction term
- initializing $W_l \approx 0$ is fine

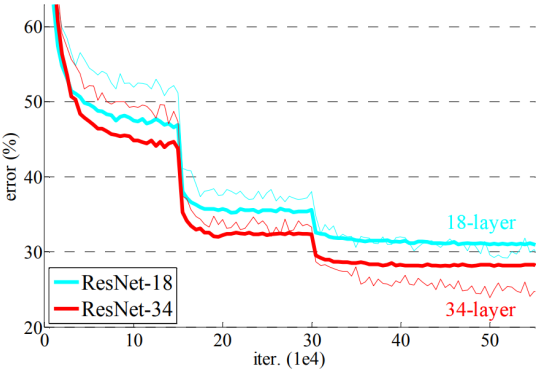
Specific choice slightly more complex:



"Residual Block"

If input-output dimensions not the same, use linear $W_l x$ instead of identity x in the shortcut.





Observation: with residual blocks (ResNets)

- optimization becomes easier
- deeper networks achieve smaller training error (and test error)
- to drive home its point, paper trains a 1202-layer network (but that overfits)

Unusual but popular layer type that helps with optimization. Assume SGD training with mini batches of size B. Batch normalization (BatchNorm, BN) acts on all examples of a mini batch jointly, but independently on each input/output dimension.

Batch Normalization Layer – Training Time

input $x_1, \ldots, x_B$ with each $x_i \in \mathbb{R}$: mini batch of inputs
input ϵ: small constant, e.g. $\epsilon = 10^{-8}$
input γ: scaling parameter $\leftarrow$ trainable
input β: shift parameters $\leftarrow$ trainable

$\mu \leftarrow \frac{1}{B} \sum_{i=1}^{B} x_i$ mini-batch mean
$\sigma^2 \leftarrow \frac{1}{B} \sum_{i=1}^{B} (x_i - \mu)^2$ mini-batch variance
for $i = 1, \ldots, B$ **do**
  $\hat{x}_i \leftarrow \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$ normalization
  $y_i \leftarrow \gamma \hat{x}_i + \beta$ scale and shift
end for

output $y_1, \ldots, y_B$ with each $y_i \in \mathbb{R}$: mini batch of outputs

BN performs a linear scale-and-shift normalization of each input dimension

Any batch of examples is transformed to have comparable output distribution

- intermediate $\hat{x}$s have mean $= 0$, std.dev. 1
- output ys have *learned* mean β, std.dev. γ

Batch Normalization Layer – Prediction Time

Same as at training time, but:

- mean and variance are not estimated from mini batch, but precomputed averages from training set are used

Prediction-time BN applies a fixed linear transformation
$\leftarrow$ can be absorbed by next layer weights

Advantages: including BN layers

- stabilizes training, e.g. smoother objective function
- allows larger learning rates $\rightarrow$ faster convergence
- enables training with sigmoid activations

Disadvantages: in practice

- BN only works with large enough batch sizes
- it's annoying having to compute μ and σ at training time and storing it to be used at prediction time
- it's error prone to have a layer behave differently during training than during prediction time

Discussion:

- original paper argues with "internal covariate shift"
- later analyses pretty much refutes this

Deep Learning: Other Architectures

Sequence-to-Sequence Models

- input: sequence of symbols, e.g. natural language text
- output: sequence of symbols, e.g. natural language text
- methods: recurrent neural networks (RNNs) or transformer networks

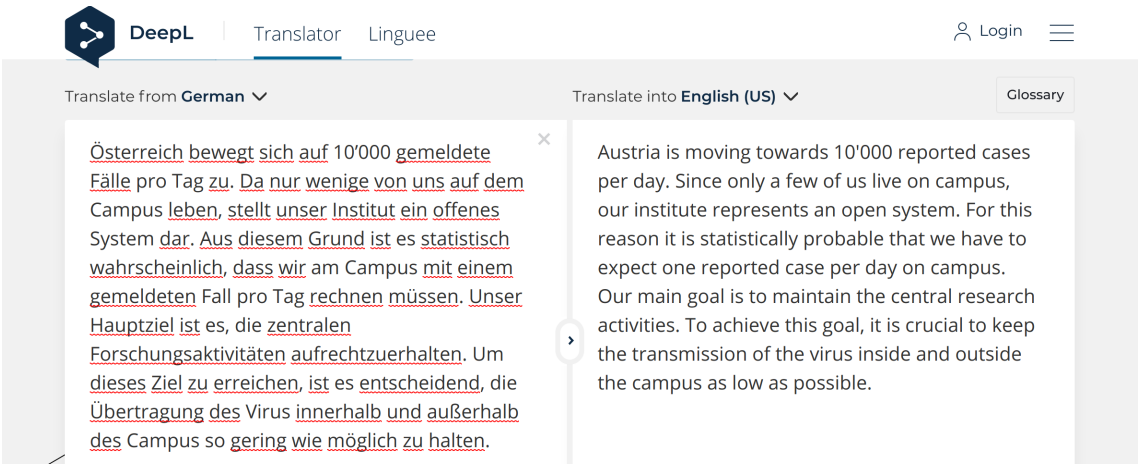
Deep Generative Models

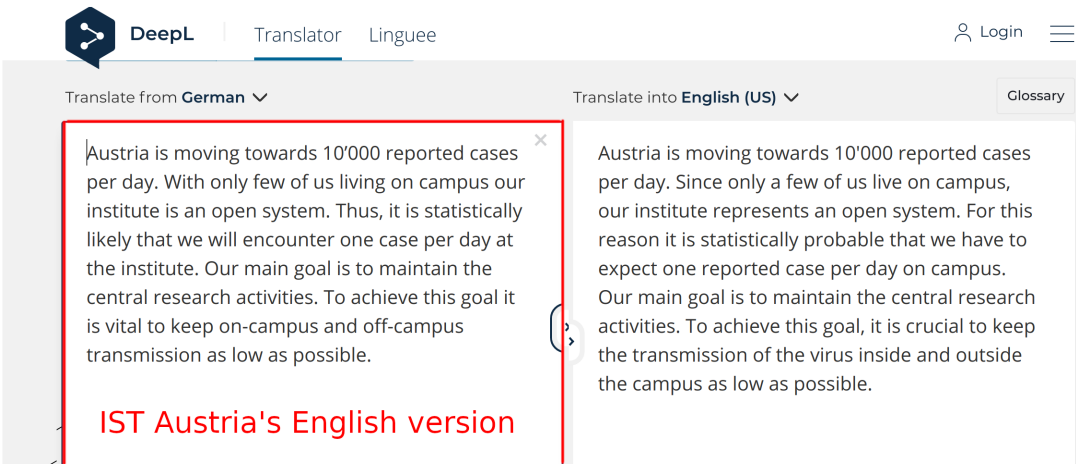
- input: a small amount of information, or even none
- output: complex output, e.g. an image or natural language text
- methods: variational autoencoder (VAE) or generative adversarial networks (GANs)

Graph Classification or Labelings

- input: a graph
- output: per-graph or per-node labels
- method: graph neural networks (GNNs) or graph convolutional networks (GCNs)

Deep Learning: Sequence-to-Sequence Models


DeepL
Translator
Linguee
Login
Translate from German
Translate into English (US)
Glossary
Österreich bewegt sich auf 10’000 gemeldete Fälle pro Tag zu. Da nur wenige von uns auf dem Campus leben, stellt unser Institut ein offenes System dar. Aus diesem Grund ist es statistisch wahrscheinlich, dass wir am Campus mit einem gemeldeten Fall pro Tag rechnen müssen. Unser Hauptziel ist es, die zentralen Forschungsaktivitäten aufrechtzuerhalten. Um dieses Ziel zu erreichen, ist es entscheidend, die Übertragung des Virus innerhalb und außerhalb des Campus so gering wie möglich zu halten.
Austria is moving towards 10'000 reported cases per day. Since only a few of us live on campus, our institute represents an open system. For this reason it is statistically probable that we have to expect one reported case per day on campus. Our main goal is to maintain the central research activities. To achieve this goal, it is crucial to keep the transmission of the virus inside and outside the campus as low as possible.


DeepL
Translator
Linguee
Login
Translate from German
Translate into English (US)
Glossary
Austria is moving towards 10’000 reported cases per day. With only few of us living on campus our institute is an open system. Thus, it is statistically likely that we will encounter one case per day at the institute. Our main goal is to maintain the central research activities. To achieve this goal it is vital to keep on-campus and off-campus transmission as low as possible.
IST Austria's English version
Austria is moving towards 10'000 reported cases per day. Since only a few of us live on campus, our institute represents an open system. For this reason it is statistically probable that we have to expect one reported case per day on campus. Our main goal is to maintain the central research activities. To achieve this goal, it is crucial to keep the transmission of the virus inside and outside the campus as low as possible.

Recurrent neurons implement feedback.

In each time step, t, a neuron A produces two outputs

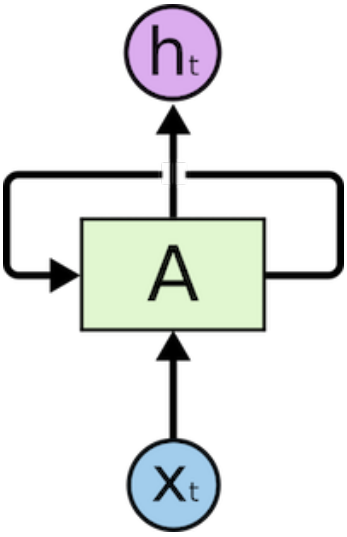
- prediction $h_t = A_h(x_t, z_{t-1})$
- state vector $z_t = A_z(x_t, z_{t-1})$

and it consumes two inputs:

- ordinary input: x_t, e.g. from a previous layer
- its own state vector z_{t-1} from one time step earlier (with $z_0 = 0$)

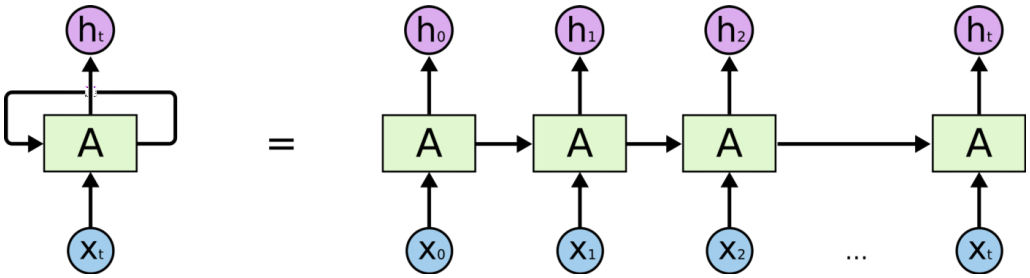
Result:

- for an input sequence $x_1, \ldots, x_T$
- output: predictions $h_1, \ldots, h_T$ with $h_t = h(x_1, \ldots, x_t)$, i.e. each output h_t depends not only on h_t but on the complete sequence of inputs so far



Given all inputs $x_1, \ldots, x_T$ and target outputs $y_1, \ldots, y_T$. How to train this?

- imagine to "unroll" the feedback loop
- one "ordinary" neuron per time step arranged in a sequence
- all neurons have shared weights
- supervised training using backpropagation



Backpropagation through time (BPTT)

Recurrent Neural Networks

More complex variant of the vanilla recurrent neuron:

- LSTMs: each neuron has a memory cell and switches for read/write/erase
- GRUs: simplified LSTM neuron, no memory but some switching behavior

Extensions of the vanilla recurrent neural network architecture:

- encoder-decoder architecture:
  - encoder networks reads all inputs and transforms them inputs into a state vector
  - decoder inputs the state vector and outputs a sequence of predictions
- attention mechanism assign time-dependent weights to different parts of the inputs
- beam search: predict multiple possible outputs at each time step instead of single one

Most recent trend:

- transformer networks: forget about recurrence etc., just use attention
  [Vaswani *et al.*, "Attention Is All You Need", NeurIPS 2017] – 13955 citations as of 07/11/2020
- e.g., at the core of powerful language models, e.g. GPT-3 https://en.wikipedia.org/wiki/GPT-3

Deep Learning: Deep Generative Models

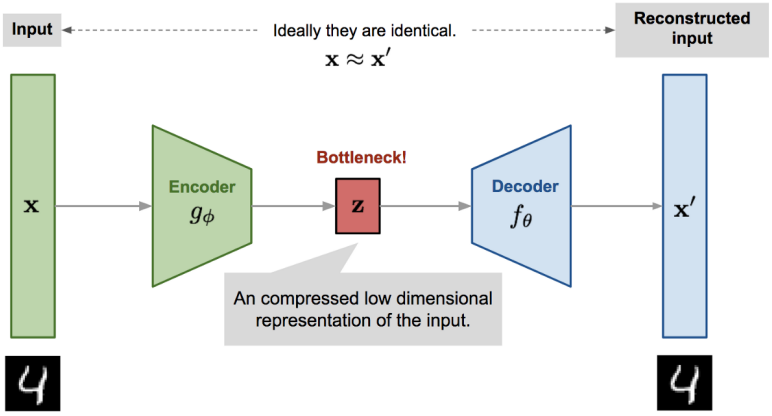
Traditional predictive models

- $x \in \mathcal{X}$: complex input, e.g. an image, $h(x) \in \mathcal{Y}$: simple output, e.g. a label

(Deep) generative models:

- $x \in \mathcal{X}$: any or no input, $h(x) \in \mathcal{Y}$: complex output, e.g. a natural image

None of these people exist, the images are outputs of a Generative Adversarial Network (GAN).

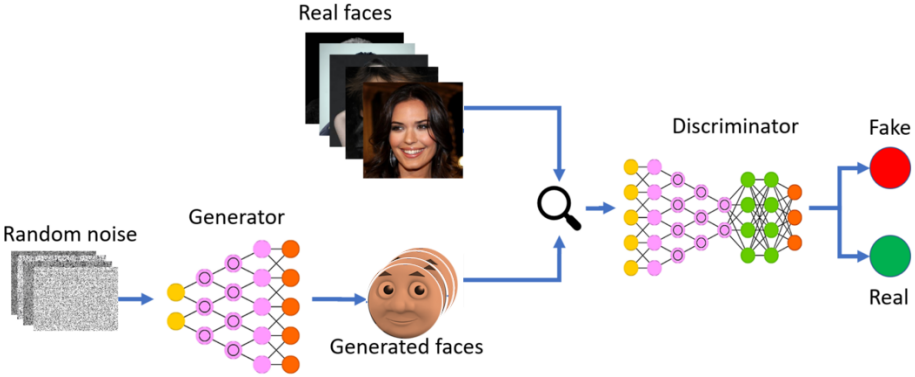


- encoder-decoder architecture: (large) input $\overset{\text{encoder}}{\rightarrow}$ (small) code $\overset{\text{decoder}}{\rightarrow}$ output
- trained by auto-encoding $\mathcal{L} = \|\text{input} - \text{output}\|^2 + \text{regularizer}$
- produce new outputs by decoding a randomly produced code

Image: https://medium.com/iitg-ai/variational-autoencoders-c780c4acaf4f

Generative Adversarial Networks (GANs)

[Goodfellow *et al*. "Generative adversarial nets", NeurIPS 2014] — 24566 citations as of 10/11/2020



- generator-discriminator architecture:
  - generator: a decoder-type network that transform random codes into outputs (e.g. images)
  - discriminator: a classifier that tries to distinguished between real and generated data
- generator and discriminator networks are trained jointly (competing with each other)
- after training, discard discriminator, decoder produces new outputs

Image: https://medium.com/sigmoid/a-brief-introduction-to-gans-and-how-to-code-them-2620ee465c30

Deep Learning: Graph Models

Task: graph classification

- input: input data $x_1, \ldots, x_n \in \mathcal{X}$, where each $x_i = (V_i, E_i)$ is a graph
- output: label $y_1, \ldots, y_n$, e.g. *"Is molecule x toxic?"*

Naive Method:

- transform $x \in \mathcal{X}$ into a feature representation $\phi(x) \in \mathbb{R}^D$
- train a neuron network on dataset $\{(\phi(x_1), y_1), \ldots, (\phi(x_n), y_n)\}$

Example (DeepTox [Mayr *et al.*, "DeepTox: Toxicity Prediction using Deep Learning", Frontiers in Environmental Science 2016]**)**

Large number of (often sparse) features from the literature:

- presence or absence of a library of substructures
- molecular weight,
- charge descriptors,
- geometric descriptors, ...

Architecture and hyperparameters chosen by extensive model selection.

Graph Convolutional Networks (GCNs)

Task: node classification from training data $\{(x^1, y^1)\, dots\, (x^m, y^m)\}$ with

- each $x^i = (V^i, E^i)$ is a graph
  - assume that each node V^i_j has an attribute vector $\phi^i_j \in \mathbb{R}^D$ attached to it
  - for any graph x^i, let $A_i \in \mathbb{R}^{n_i \times n_i}$ be its adjacency matrix
- each $y_i \in \{1, \ldots, K\}^{V^i}$ is a labeling of the nodes in x_i

Method:

- define a neural network with layers

$$H^l(x_i) = \sigma_l(\overbrace{\tilde{A}_i}^{\in \mathbb{R}^{n_i \times n_i}} \overbrace{H_i^{l-1}}^{\in \mathbb{R}^{n_i \times d_{l-1}}} \overbrace{W^l}^{\in \mathbb{R}^{d_{l-1} \times d_l}}) \in \mathbb{R}^{n_i \times d_l} \qquad \text{with} \quad H^0 = \Phi^i = (\phi^i_j)_{j=1,\ldots,|V^i|}$$

  - $\tilde{A}_i = D_i^{-1} A_i$ with $D_i = \text{diag}(d_i^1, \ldots, d_i^{n_i})$ for $d_i = \sum_j a_{ij}$ (normalized adjacency matrix)
  - W_l are learnable weights
- $\tilde{A}_i$ term propagates information between adjacent nodes in graph

Deep Learning: Summary

Viewpoint 1: Deep Learning is the present and the future of AI.

Regardless of the hype, deep learning has catapulted machine learning to the next level of usefulness and societal relevance. So far, there is no end in sight to its success.

Viewpoint 2: Deep Learning is nothing special.

Deep learning is simply a specific choice of hypothesis class that allows learning a classifier and a feature mapping together.

Viewpoint 3: Deep Learning is a powerful tool.

Many research areas have adopted deep learning into their toolboxes for solving problems of actual interest, let it be computer graphics, bio-imaging, computational physics, or others.

Viewpoint 4: Deep Learning will not be the final answer.

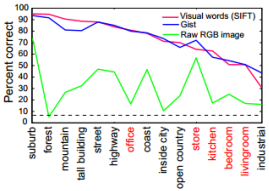
Relevant open questions, such as determining cause vs. effect, or learning on quantum computers, are not solved by deep learning and will probably require other techniques.

Significance of Results

Significance of Results

5. Experiments

In this section we present experiments for indoor scene recognition performed on the dataset described in section 2. We show that the model and representation proposed in this paper give significant improvement over a state of the art model for this task. We also perform experiments using different versions of our model and compare manual segmentations to segmentations obtained by running a segmentation algorithm.



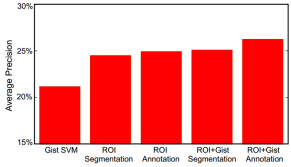


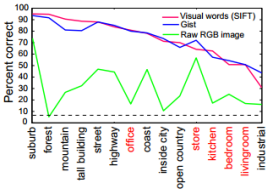
Figure 6. Multiclass average precision performance for the baseline and four different versions of our model.

How to tell if reported differences are due to chance?

Significance of Results

5. Experiments

In this section we present experiments for indoor scene recognition performed on the dataset described in section 2. We show that the model and representation proposed in this paper give significant improvement over a state of the art model for this task. We also perform experiments using different versions of our model and compare manual segmentations to segmentations obtained by running a segmentation algorithm.



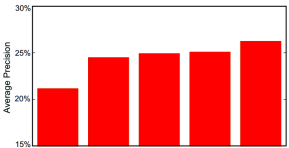


Figure 6. Multiclass average precision performance for the baseline and four different versions of our model.

How to tell if reported differences are due to chance?

Accuracy on a Single Dataset

- two-sample significance tests
- paired significance tests

Multiple Datasets

- non-parametric paired significance tests

Classification rate (%)	
Proposed method	Proposed method with CA
99.00	99.50

Two classifiers are evaluated on the same test set.

- classifier 1 has error rate $e_1 \in [0, 1]$
- classifier 2 has error rate $e_2 \in [0, 1]$

Are these significantly different, or due to chance?

Classification rate (%)	
Proposed method	Proposed method with CA
99.00	99.50

Two classifiers are evaluated on the same test set.

- classifier 1 has error rate $e_1 \in [0, 1]$
- classifier 2 has error rate $e_2 \in [0, 1]$

Are these significantly different, or due to chance?

Impossible to say (or even estimate), unless we know how many test samples!

How many examples do you guess?

Name	Number of samples (training)	Classification rate (%)	
		Proposed method	Proposed method with CA
10K (FT)	900 (100)	99.00	99.50

Two classifiers are evaluated on the same test set.

- classifier 1 has error rate $e_1 \in [0, 1]$
- classifier 2 has error rate $e_2 \in [0, 1]$

Are these significantly different, or due to chance?

Impossible to say (or even estimate), unless we know how many test samples!

How many examples do you guess? **Okay, that's a start...**

- true error rate of classifier f is $p \in [0, 1]$ $\quad \rightarrow$ Bernoulli variable
- estimate from m test samples: $\hat{p} = \frac{1}{m} \sum_i [\![f(x_i) \neq y_i]\!]$
- variance of estimate from m test samples: $V = \frac{1}{m} \hat{p}(1 - \hat{p})$
- report mean $\pm$ standard error of the mean: $\hat{p} \pm \sqrt{\frac{\hat{p}(1-\hat{p})}{m}}$

Name	Number of samples (training)	Classification rate (%)	
		Proposed method	Proposed method with CA
10K (FT)	900 (100)	99.00	99.50

- true error rate of classifier f is $p \in [0, 1]$ $\quad \rightarrow$ Bernoulli variable
- estimate from m test samples: $\hat{p} = \frac{1}{m} \sum_i [\![f(x_i) \neq y_i]\!]$
- variance of estimate from m test samples: $V = \frac{1}{m} \hat{p}(1 - \hat{p})$
- report mean $\pm$ standard error of the mean: $\hat{p} \pm \sqrt{\frac{\hat{p}(1-\hat{p})}{m}}$

Name	Number of samples (training)	Classification rate (%)	
		Proposed method	Proposed method with CA
10K (FT)	900 (100)	99.00 ± 0.33	99.50 ± 0.24

Not particularly convincing... better than nothing, but also not a proper test of significance.

Procedure:

- state the target hypothesis and the null hypothesis
  - H: `"method A and method B differ in quality"`
  - H_0: `"method A and method B are equally good"`
- compute the p-value and interpret it

Definition (p-value)

The p-value is the probability of obtaining a test result at least as extreme as the results actually observed, under the assumption that the null hypothesis is correct.

Procedure:

- state the target hypothesis and the null hypothesis
  - H: `"method A and method B differ in quality"`
  - H_0: `"method A and method B are equally good"`
- compute the p-value and interpret it

Definition (p-value)

The p-value is the probability of obtaining a test result at least as extreme as the results actually observed, under the assumption that the null hypothesis is correct.

Observation: even for difficult setting the p-value can often be computed because it conditions on the null hypothesis being correct.

Example

For $n = 800$ samples, we observe that method A makes $n_A = 8$ mistakes (99% accuracy) and method B make $n_B = 4$ mistakes (99.5% accuracy). *If* both methods were equally good, what's the probability of observing this outcome, or an even more extreme one?

Paired test

For a sequence of experiments we always observe two sets of outcomes A, B. Are the differences between them due to chance?

2×2 contingency table:

	g is right	g is wrong
f is right	a	b
f is wrong	c	d

binomial test: ignore a and d, analyze b and c.

- null hypothesis: f and g are equally good. we'd expect $b \approx c$
- probability of seeing (b, c) split or more extreme in $b + c$ differences:

$$p\text{-value} \;=\; 2\frac{1}{2^{b+c}} \sum_{i=0}^{\min(b,c)} \binom{b+c}{i}$$

- `scipy.stats.binom_test( min(b,c), n=b+c, p=0.5 )`

Paired test

For a sequence of experiments we always observe two sets of outcomes A, B. Are the differences between them due to chance?

2×2 contingency table:

	g is right	g is wrong
f is right	a	b
f is wrong	c	d

binomial test: ignore a and d, analyze b and c.

- null hypothesis: f and g are equally good. we'd expect $b \approx c$
- probability of seeing (b, c) split or more extreme in $b + c$ differences:

$$p\text{-value} \;=\; 2\frac{1}{2^{b+c}} \sum_{i=0}^{\min(b,c)} \binom{b+c}{i}$$

- `scipy.stats.binom_test( min(b,c), n=b+c, p=0.5 )`

Example:

792	0
4	4

$p = 0.125$

787	5
9	0

$p \approx 0.39$

8920	0
40	40

$p \approx 10^{-12}$

8875	40
80	0

$p \approx 0.0003$

Significance Using Multiple Datasets

Standard procedure in Machine Learning research:

- develop a new method
- compare it to results of previous methods on standard benchmarks

	HCRF	Our
SIFT-flow	31.22%	**27.73%**
MSRC-21	78.89%	**81.11%**
VOC 2008	20.13 %	**30.12%**
VOC 2009	42.43 %	**43.37%**
VOC 2010	30.13 %	**32.14%**

Are the differences just due to chance?

Significance Using Multiple Datasets

Standard procedure in Machine Learning research:

- develop a new method
- compare it to results of previous methods on standard benchmarks

	HCRF	Our
SIFT-flow	31.22%	**27.73%**
MSRC-21	78.89%	**81.11%**
VOC 2008	20.13 %	**30.12%**
VOC 2009	42.43 %	**43.37%**
VOC 2010	30.13 %	**32.14%**

Are the differences just due to chance?

- Idea 1: mean/std.err.: 40.6 ± 20.4 vs. 42.9 ± 19.8

Significance Using Multiple Datasets

Standard procedure in Machine Learning research:

- develop a new method
- compare it to results of previous methods on standard benchmarks

	HCRF	Our
SIFT-flow	31.22%	**27.73%**
MSRC-21	78.89%	**81.11%**
VOC 2008	20.13 %	**30.12%**
VOC 2009	42.43 %	**43.37%**
VOC 2010	30.13 %	**32.14%**

"HCRF" method is better	1
"Our" method is better	4
both methods are equal	0

Are the differences just due to chance?

- Idea 1: mean/std.err.: 40.6 ± 20.4 vs. 42.9 ± 19.8
- Idea 2: sign test (like binomial before): `binom_test(1,5)=0.375`

Significance Using Multiple Datasets

Standard procedure in Machine Learning research:

- develop a new method
- compare it to results of previous methods on standard benchmarks

	HCRF	Our
SIFT-flow	31.22%	**27.73%**
MSRC-21	78.89%	**81.11%**
VOC 2008	20.13 %	**30.12%**
VOC 2009	42.43 %	**43.37%**
VOC 2010	30.13 %	**32.14%**

"HCRF" method is better	1
"Our" method is better	4
both methods are equal	0

Are the differences just due to chance?

- Idea 1: mean/std.err.: 40.6 ± 20.4 vs. 42.9 ± 19.8
- Idea 2: sign test (like binomial before): `binom_test(1,5)=0.375`
- Idea 3: take differences into account, not just the sign
  - H_0: differences have a symmetric distribution around zero

Wilcoxon signed rank test

Given: real values $a_1, \ldots, a_m$ and $b_1, \ldots, b_m$

- drop all cases with $a_i = b_i$, call remaining points $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$ again
- compute $\delta_i = |a_i - b_i|$ and $s_i = \text{sign}(a_i - b_i)$ for $i = 1, \ldots, k$
- sort elements from smallest to larges δ_i
- compute rank, R_i, of each δ_i, ties get average of covered ranks
- compute statistics (sum of signed ranks)

$$W = \sum_{i=1}^{k} s_i R_i$$

- compare value to table, $W_{\text{critical},k}$ (large k: Gaussian approximation)

"HCRF" vs. "Our" example (5 datasets):

- $A =$ | 31.22 | 78.89 | 20.13 | 42.43 | 30.13 | $B =$ | 27.73 | 81.11 | 30.12 | 43.37 | 32.14 |
- `scipy.stats.wilcoxon(` A, B `) = 0.35`

Wilcoxon signed rank test

Given: real values $a_1, \ldots, a_m$ for method A, and $b_1, \ldots, b_m$ for method B

dataset	A [%]	B [%]	abs.diff	sign
1	99.50	99.00	0.50	$+$
2	99.25	100.00	0.75	$-$
3	98.25	99.88	1.63	$-$
4	97.50	96.75	0.75	$+$
5	94.00	97.00	3.00	$-$
6	99.75	98.38	1.37	$+$
7	94.25	97.50	3.25	$-$
8	97.00	98.25	1.25	$-$
9	95.13	99.63	4.50	$-$
10	93.75	99.75	6.00	$-$
11	94.13	99.00	4.87	$-$
12	95.75	98.75	3.00	$-$

Mean/std.dev.:

- a: 96.5 ± 2.14
- b: 98.6 ± 1.20

Sign test:

-

$a_i < b_i$	$a_i = b_i$	$a_i > b_i$
9	0	3

- `binom_test(3,12)` ≈ 0.146

Signed rank test:

- `wilcoxon(A,B)` ≈ 0.017

REPRODUCIBILITY

Statisticians issue warning on *P* values

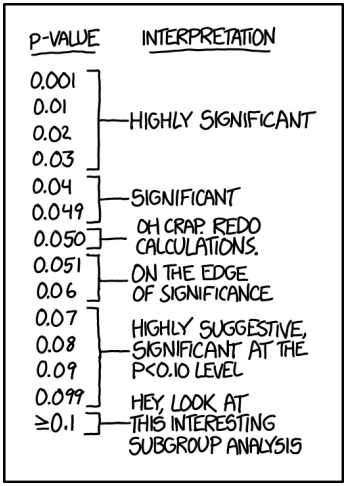
Statement aims to halt missteps in the quest for certainty.

BY MONYA BAKER

Misuse of the *P* value — a common test for judging the strength of scientific evidence — is contributing to the number of research findings that cannot be reproduced, the American Statistical Association (ASA) warned on 8 March. The group has taken the unusual step of issuing principles to guide use of the *P* value, which it says cannot determine whether a hypothesis is true or whether results are important.

cannot indicate the importance of a finding; for instance, a drug can have a statistically significant effect on patients' blood glucose levels without having a therapeutic effect.

Giovanni Parmigiani, a biostatistician at the Dana Farber Cancer Institute in Boston, Massachusetts, says that misunderstandings about what information a *P* value provides often crop up in textbooks and practice manuals. A course correction is long overdue, he adds. "Surely if this happened twenty years ago, biomedical research could be in a better place now."



Sources: Nature doi:10.1038/nature.2016.19503; xkcd.com

PASCAL VOC 2012 *val* set	[1] (Img+Obj)	CCNN [6]	EM-Adapt (re-impl. of [6])	[44] (stage1)	MIL+ILP +SP-sppxl† [33]	SEC (proposed)
background		71.7	67.2	68.5*	77.2	**82.2**
aeroplane		30.7	29.2	25.5*	37.3	**61.7**
bike		30.5	17.6	18.0*	18.4	**26.0**
bird		26.3	28.6	25.4*	25.4	**60.4**
boat		20.0	22.2	20.2*	**28.2**	25.6
bottle		24.2	29.6	36.3*	31.9	**45.6**
bus		39.2	47.0	46.8*	41.6	**70.9**
car		33.7	44.0	47.1*	48.1	**63.2**
cat		50.2	44.2	48.0*	50.7	**72.2**
chair		17.1	14.6	15.8*	12.7	**20.9**
cow		29.7	35.1	37.9*	45.7	**52.9**
diningtable		22.5	24.9	21.0*	14.6	**30.6**
dog		41.3	41.0	44.5*	50.9	**62.8**
horse		35.7	34.8	34.5*	44.1	**56.8**
motorbike		43.0	41.6	46.2*	39.2	**63.5**
person		36.0	32.1	40.7*	37.9	**57.0**
plant		29.0	24.8	30.4*	28.3	**32.2**
sheep		34.9	37.4	36.3*	44.0	**60.6**
sofa		23.1	24.0	22.2*	19.6	**32.3**
train		33.2	38.1	38.8*	37.6	**44.8**
tv/monitor		33.2	31.6	36.9*	35.0	**42.3**
average	32.2*	33.6	33.8	35.3*	36.6	**50.7**

Often, we want to compare with more than one other method.

- simply making many pairwise comparisons will increase risk of some coming up as significant just by chance

Bonferroni correction:

- target level: α, e.g. 0.05
- number of comparisons K, e.g. 5
- make each test with level α/K, e.g. 0.01

Overall by **union bound**:

$$\Pr\{\text{at least one tests } 1,\dots,K \text{ is a false positive}\} \leq \Pr\sum_{k=1}^{K}\{\text{test } k \text{ is a false positive}\} \leq \Pr\sum_{k=1}^{K}\frac{\alpha}{K} = \alpha$$