

Convolutional Neural Networks

Deep Learning Course

Kevin Webster, Pierre Harvey Richemond

Course outline

1. Introduction to deep learning
2. Neural networks optimisation
3. Convolutional neural networks
4. Introduction to reinforcement learning - part 1
5. Introduction to reinforcement learning - part 2
6. Sequence models
7. Generative adversarial networks (GANs)
8. Variational autoencoders (VAEs)
9. Normalising flows
10. Theories of deep learning

CNN fundamentals

- Convolution, padding, strides and pooling

- Transposed convolutions

- Batch normalisation in convolutional networks

CNN architectures

- AlexNet

- VGG

- GoogLeNet/Inception

- ResNet

Capsule networks

Creative application: style transfer

CNN fundamentals

- Convolution, padding, strides and pooling

- Transposed convolutions

- Batch normalisation in convolutional networks

CNN architectures

- AlexNet

- VGG

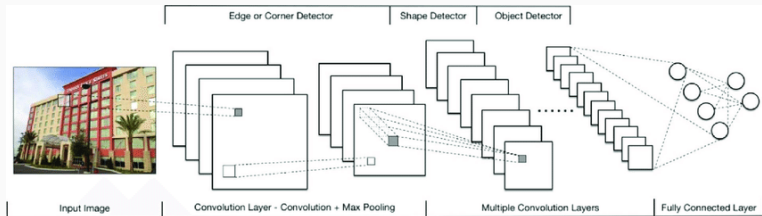
- GoogLeNet/Inception

- ResNet

Capsule networks

- Creative application: style transfer

Convolutional neural networks



- **Convolutional neural network, CNN or ConvNet** is a class of neural network with a special structure
- Breakthrough improvements in image processing
- Also used in NLP, audio waveform analysis and generation, and RL models for games

Convolution operation

The convolution operator

$$(f * w)(t) := \int_{-\infty}^{\infty} f(\tau)w(t - \tau)d\tau$$

can be described as a weighted average of the **input** f according to the weighting (or **kernel**) w at each point in time t .

The discrete convolution is given by

$$(f * w)(t) := \sum_{\tau=-\infty}^{\infty} f(\tau)w(t - \tau).$$

Note that when w has a finite support, a finite summation may be used.

Convolution operation

In convolutional neural networks with image inputs, a two dimensional convolution is used:

$$(I * K)(i, j) = \sum_m \sum_n I(m, n) K(i - m, j - n)$$

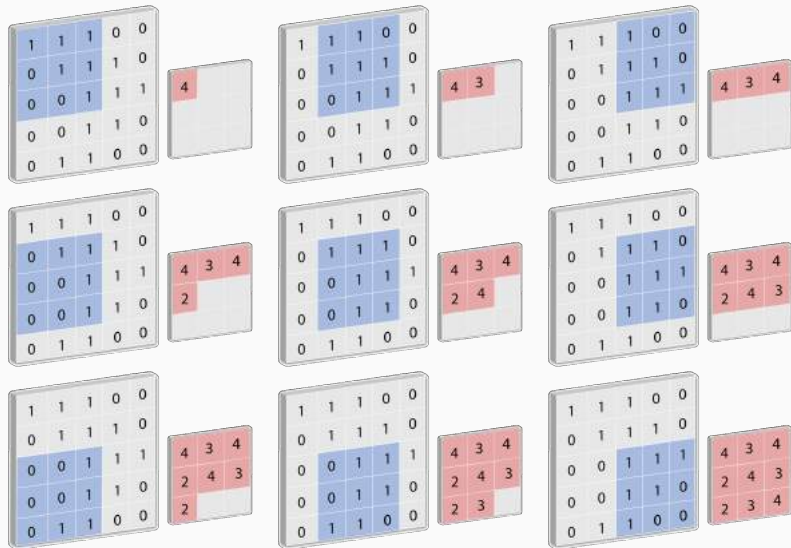
The kernel K will have a fixed size, outside which is can be assumed to be zero.

In practice, many libraries implement the cross-correlation operation, which is the same as above but changing the orientation of the arguments:

$$S(i, j) = \sum_m \sum_n I(i + m, j + n) K(m, n)$$

We will consider the above operation in CNNs and refer to it as a **convolution**.

Convolution operation



Convolution operation

- The **kernel** is also referred to as a **filter**
- The support of the kernel defines its **receptive field**
- Output of a convolution is often referred to as a **feature map**
- There are usually many feature maps defined for each layer
- Within each layer, there are several **channels** (e.g. there are 3 channels in the input layer for an RGB image)
- In practice, the kernels for each feature map are usually three dimensional tensors (esp. for images) and the convolution operation is performed across all channels in the input layer:

$$S^l(i, j) = \sum_m \sum_n \sum_c I(i + m, j + n, c) K^l(m, n, c)$$

- The convolution is usually combined with a shared bias (one per channel) and passed through an activation function

Convolution properties

- Design inspired by visual processing systems
- **Sparse interactions:** a feedforward network connects every input to every output. The sparse connectivity of CNNs reduces number of parameters significantly and improves efficiency
- **Parameter sharing**, or tied weights, means that the CNN does not need to learn a separate set of parameters for each location, but reuses one set in every location
- **Equivariant features:** due to the parameter sharing used in CNNs, the feature maps are equivariant with respect to translations. (However note that convolutions are not equivariant with respect to other transformations such as rotation.)
- Also enables some flexibility in input size

In general, we can also define the convolution operation for N -dimensional inputs. The kernel/filter is then a tensor of shape (n, m, k_1, \dots, k_N) , where

n = number of output maps

m = number of input maps

k_j = kernel size in dimension j

The output size o of a convolution along a dimension with input size i and kernel size k is given by

$$o = (i - k) + 1.$$

The output size is frequently controlled by padding the input with zeros, effectively increasing the input size.

- **'SAME'** (or half) padding adds $p = k - 1$ zeros to the input to constrain the output size to be the same for unit strides, $o = i$. For odd-sized kernels, add $p = \lfloor k/2 \rfloor$ zeros both sides of the input
- **'VALID'** padding is the standard terminology for when no padding is used
- **'FULL'** padding adds $p = 2(k - 1)$ zeros to the input ($k - 1$ zeros on both sides), resulting in an output size of $i + (k - 1)$

Convolutions may also use a **stride** s , which is the distance between consecutive positions of the kernel (the convolutions considered so far use $s = 1$).

In this case, the output size is given by

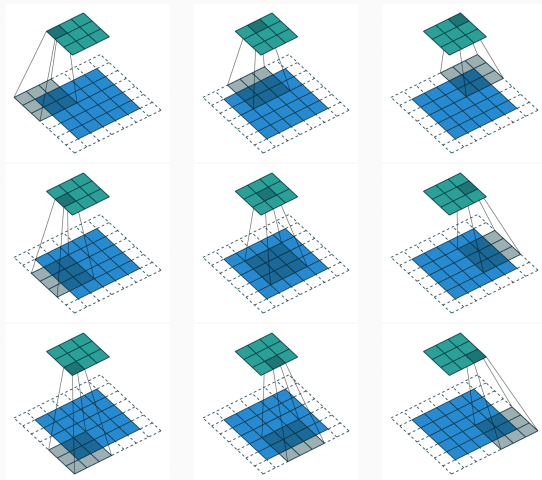
$$o = \left\lfloor \frac{i + p - k}{s} \right\rfloor + 1$$

Thus, strides are one way of *downsampling* within a convolutional neural network.

Note that when using strides $s > 1$, different input sizes can still lead to the same output size.

Strides

Example: 3×3 convolution with half ('SAME') padding $p = 2$, stride $s = 2$, $i = 5 \Rightarrow$ output size $o = 3$.



In typical CNNs, convolutional layers are interleaved with **pooling** layers. These layers compute a summary statistic over regions of the input space identified by a sliding window.

It can be thought of as being similar to the convolution operation, where the linear transformation is replaced with a pooling operation.

- **Max pooling** computes the maximum activation per channel in the window
- **Average pooling** computes the average activation per channel in the window
- **L^2 norm** computes the 2-norm of activations per channel in the window

Pooling

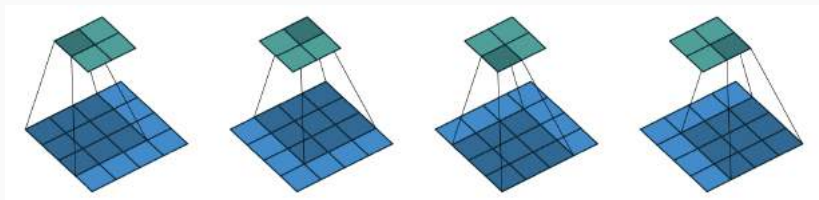
- The pooling operation introduces a degree of *translation invariance* to the input
- This is appropriate when we do not need to know the exact location of a feature, just that it is there
- It can be thought of as a prior assumption that the learned function must have translation invariance
- Pooling is often used with strided windows, leading to downsampling of the input
- We can also pool over the outputs of separate feature maps, allowing the network to learn which transformations to become invariant to
- Pooling output can also be computed as $o = \lfloor \frac{i-k}{s} \rfloor + 1$
- Pooling is useful for handling inputs of varying size

Transposed convolutions

- Also called *fractionally strided convolutions* or *deconvolutions*
- The typical convolution \rightarrow pooling combination leads to spatial downsampling.
- Sometimes we need to do the opposite: transposed convolutions provide one way to do this
- For example, we might want to construct a convolutional decoder as part of an autoencoder
- Transposed convolutions are intended to be the analogue to transposing the weight matrix in the fully connected case

Transposed convolutions

Consider the following as a simple example case:



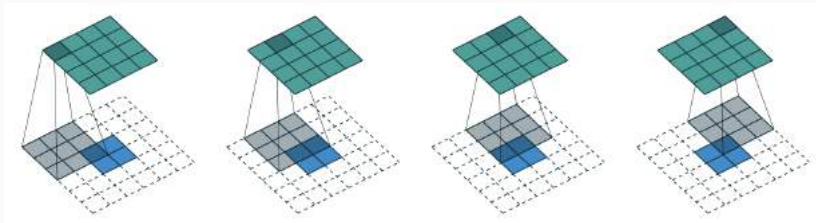
We can 'unroll' this operation to obtain the sparse matrix

$$\begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$

Transposed convolutions

- This 4×16 matrix flattens the input layer and the output is then reshaped to a 2×2 matrix
- In the backward pass, the error is backpropagated by multiplying the loss by the transpose of this matrix
- The connectivity pattern is compatible with the original convolution by construction
- Transposed convolutions essentially swap the forward and backward passes of a convolution

Transposed convolutions as convolutions



- Note that it is possible to emulate a transposed convolution with a direct convolution by adding zeros to the input
- The above convolution is equivalent to the transpose of the previous convolution example
- Note that the kernel size and stride are the same, but the input is padded with zeros
- The connectivity pattern is the same

Transposed convolutions as convolutions: unit strides

- In the case of unit strides ($s = 1$), consider a convolution with padding p , kernel size k and input size i
- It has an associated *transposed convolution* with $k' = k$, $s' = s$ and $p' = 2(k - 1) - p$. Its output size is given by

$$o' = i' + (k - 1) - p$$

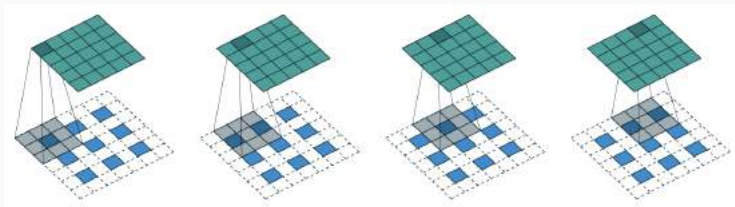
- No padding of the convolution input leads to full padding for the transposed convolution and vice versa
- Half ('SAME') padding of the convolution input leads to half padding for the transposed convolution

Transposed convolutions as convolutions: non-unit strides

- In the case of non-unit strides ($s > 1$), we can think of the associated transposed convolution as having $s < 1$
- Simplest case is when the convolution is such that s divides $(i + p - k)$
- The input to the associated transposed convolution adds $s - 1$ zeros between the input units. It has $k' = k$, $s' = 1$ and $p' = 2(k - 1) - p$ and output size

$$o' = s(i' - 1) + k - p$$

- Example with $i = 5$, $k = 3$, $s = 2$ and $p = 2$:

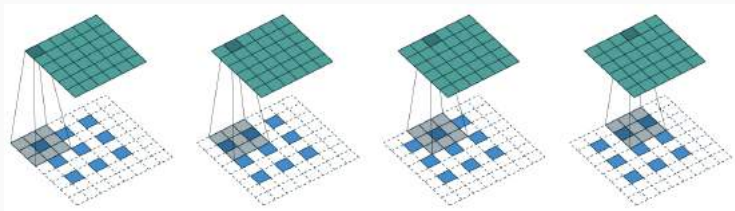


Transposed convolutions as convolutions: non-unit strides

- The case where s does not divide $(i + p - k)$ has to account for there being multiple input sizes that lead to the same output size
- This is captured by the parameter $a = (i + p - k) \bmod s$
- The transposed convolution again adds $s - 1$ zeros between input units
- It has $k' = k$, $s' = 1$ and $p' = 2(k - 1) - p$ and output size

$$o' = s(i' - 1) + a + k - p$$

- Example with $i = 6$, $k = 3$, $s = 2$ and $p = 1$:



- Recall BN normalises each feature map separately
- For CNNs we want to respect the structure and normalise over all spatial locations
- In particular, an input in a 2D ConvNet will have shape $[N, H, W, C]$, where
 - N is the number of examples in the minibatch
 - H and W are image height and width respectively
 - C is the number of channels
- Standard BN would compute $H \times W \times C$ means and stddevs to normalise each feature separately at each spatial location
- BN in ConvNets instead computes C means and stddevs and normalises jointly for every spatial location

CNN fundamentals

Convolution, padding, strides and pooling

Transposed convolutions

Batch normalisation in convolutional networks

CNN architectures

AlexNet

VGG

GoogLeNet/Inception

ResNet

Capsule networks

Creative application: style transfer

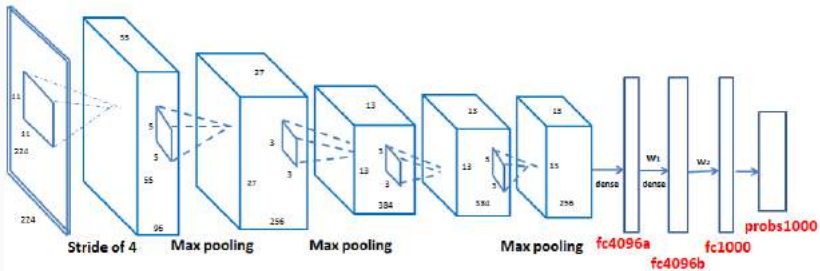
CNN architectures

- Certain CNN architectures have contributed to significant progress in image recognition
- ImageNet is a standard dataset to compare networks against each other
- ImageNet has been running an annual competition since 2010: the **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)**
- 1.2 million images with 1000 classes for recognition

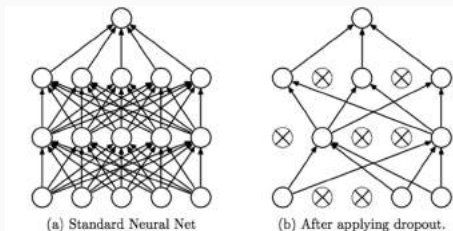


- AlexNet was published in 2012, and competed in the ImageNet competition in the same year
- Achieved top-5 error rate of 15.3%, more than 10.8% lower than that of the runner up
- One of the first deep networks to significantly outperform traditional methodologies in image classification
- Composed of 5 convolutional layers followed by 3 fully connected layers
- Used ReLU for the non-linear activations, instead of a tanh or sigmoid
- Reduced over-fitting by using a dropout layer after each fully connected layer

AlexNet

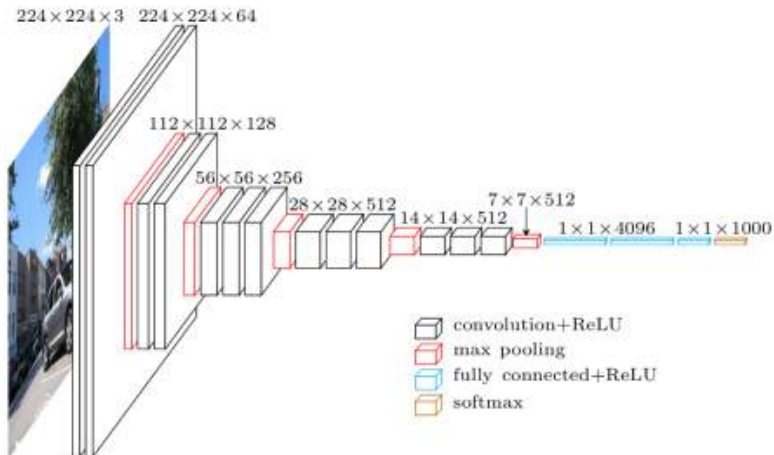


AlexNet architecture



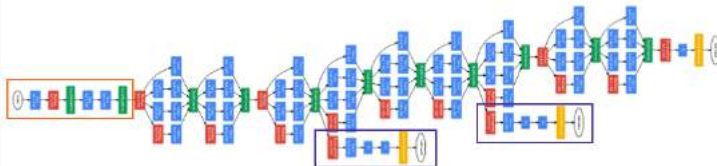
- Developed by the VGG (Visual Geometry Group) in Oxford
- Improves AlexNet by replacing the large filters in the first two layers by multiple 3x3 filters
- Multiple stacked smaller size kernels allows the network to learn more complex features
- Also reduces number of parameters
- VGG convolutional layers are followed by 3 fully connected layers
- Width of the network (number of channels) starts small at 64 and increases by a factor of 2 after every pooling layer
- Achieved the top-5 error rate of 7.3% on ImageNet in 2014

VGG-16

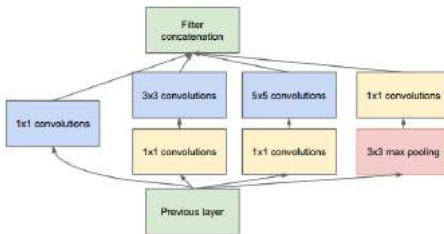


- VGG is computationally demanding due to large width of convolutional layers
- Insight of GoogLeNet is that many activations are zero or highly correlated
- Introduces the inception module that approximates a sparse CNN
- Uses convolutions of different sizes to capture details at various scales
- Replaces fully connected layers with global average pooling
- Achieves 6.67% top-5 error rate on ImageNet in 2014 and is much faster than VGG

GoogLeNet/Inception



GoogLeNet architecture, including auxiliary losses to mitigate vanishing gradients



Inception module

Residual networks

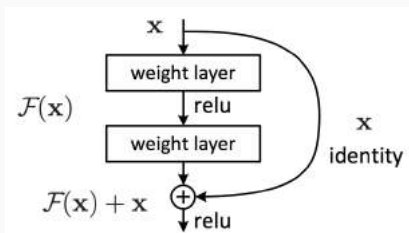
- Previous developments show increased depth increases capacity
- Backpropagated signal becomes vanishingly small with large depth
- Increased parameters also present optimisation problems
- Proposed solution is to introduce **residual blocks** into the architecture, that use shortcut connections to propagate the signal
- Assumption is that the optimal learned function is closer to identity than to zero
- Similar to GoogLeNet, uses a global average pooling followed by the classification layer
- ResNets were learned with network depth of as large as 152
- Similar to the VGGNet, consisting mostly of 3X3 filters
- ResNet-152 achieved 3.6% top-5 error rate on ImageNet in 2015, surpassing human performance

Residual block

- Residual blocks add the input to the output:

$$\mathbf{y} = \sigma(\mathcal{F}(\mathbf{x}) + \mathbf{x}),$$

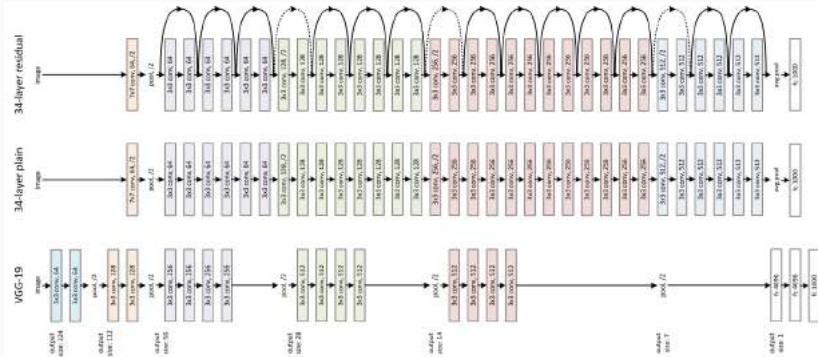
where $\mathcal{F}(\mathbf{x}) = W_2\sigma(W_1\mathbf{x} + b_1) + b_2$



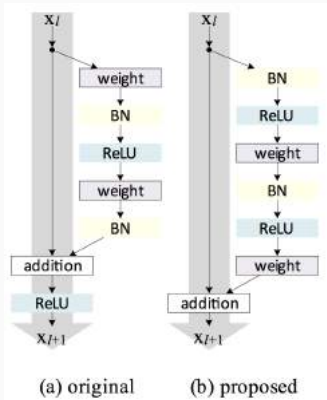
- Two layers are included inside the block, since one layer would be too close to a linear transformation:

$$\tilde{\mathbf{y}} = \sigma(W_1\mathbf{x} + b_1 + \mathbf{x})$$

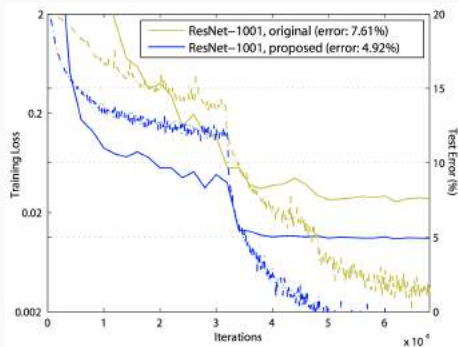
ResNet, plain, VGG-19



Revised ResNet (2016)



Original and revised ResNet blocks



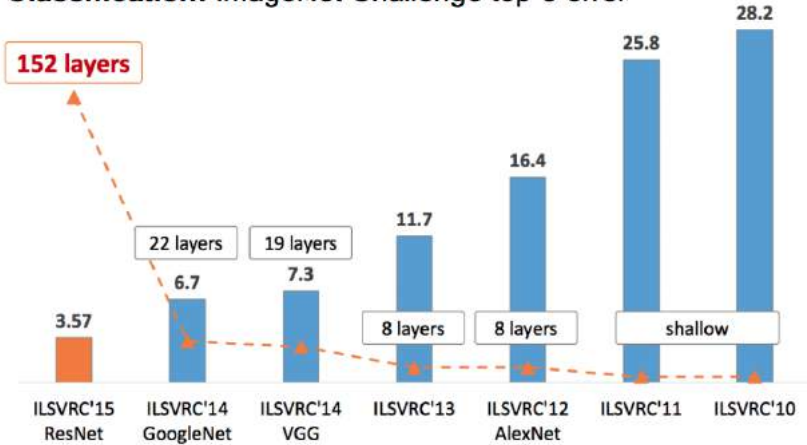
Training curves on CIFAR-10 of 1001-layer ResNets. Solid lines denote test error, and dashed lines denote training loss

CNN architectures and ILSVRC

Year	CNN	Developed by	Place	Top-5 error rate	No. of parameters
1998	LeNet(8)	Yann LeCun et al			60 thousand
2012	AlexNet(7)	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	1st	15.3%	60 million
2013	ZFNet()	Matthew Zeiler and Rob Fergus	1st	14.8%	
2014	GoogLeNet(19)	Google	1st	6.67%	4 million
2014	VGG Net(16)	Simonyan, Zisserman	2nd	7.3%	138 million
2015	<u>ResNet(152)</u>	Kaiming He	1st	3.6%	

ImageNet competition progress

Classification: ImageNet Challenge top-5 error



CNN fundamentals

- Convolution, padding, strides and pooling

- Transposed convolutions

- Batch normalisation in convolutional networks

CNN architectures

- AlexNet

- VGG

- GoogLeNet/Inception

- ResNet

Capsule networks

- Creative application: style transfer

'The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster.'

- *Geoffrey Hinton*

Motivation:

- Pooling operations lose information about where things are. It provides only a small amount of translational invariance and is a crude routing algorithm
- We would like to be able to represent precise relationships between parts of an object
- CNNs do not easily represent the same entity in different positions or poses
- We would like to represent entities and their relationship to other entities, independent of pose
- Compare to computer graphics, where images are constructed from internal hierarchical representations of geometric data

Capsule networks

- Neurons are grouped together within a layer to form a **capsule**.
- Intuitively, each capsule represents a (learned) entity from the data
- Let \mathbf{v}_j denote the vector output of a capsule, and \mathbf{u}_i denote vector outputs of capsules from the layer below
- Each capsule in the lower layer produces a ‘prediction vector’ $\hat{\mathbf{u}}_{j|i}$ that represents how the entity relates to capsules in the higher layer:

$$\hat{\mathbf{u}}_{j|i} = \mathbf{W}_{ij}\mathbf{u}_i$$

- The linear transform \mathbf{W}_{ij} is also learned, and represents the transformation between relative positions of entities (cf. computer graphics)

Capsule networks

- Given the lower layer's predictions, the higher-level capsule computes its input as

$$\mathbf{s}_j = \sum_i c_{ij} \hat{\mathbf{u}}_{j|i}$$

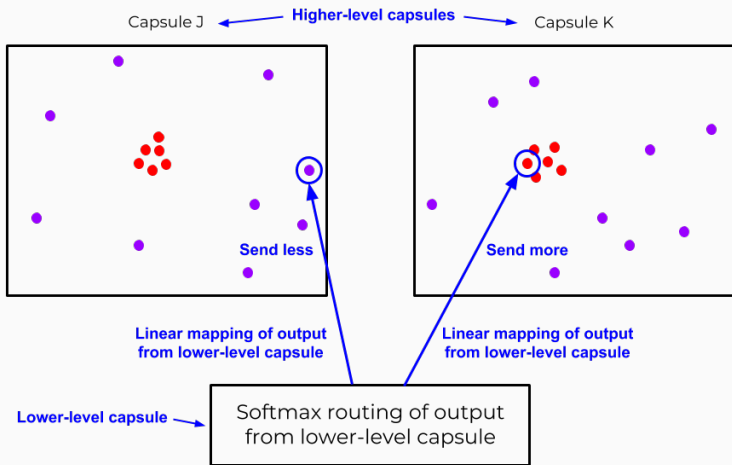
and is passed through the nonlinear squashing function to obtain

$$\mathbf{v}_j = \frac{||\mathbf{s}_j||^2}{1 + ||\mathbf{s}_j||^2} \frac{\mathbf{s}_j}{||\mathbf{s}_j||}$$

- The c_{ij} are coupling coefficients that are learned through a dynamic routing algorithm
- Probability of an entity's existence is captured by the length of the capsule vector
- The routing is intended to strengthen connections between entities that exist in the input: 'routing by agreement'

Capsule networks: dynamic routing

Dynamic routing based on agreement



Capsule networks: dynamic routing algorithm

Inputs $\hat{\mathbf{u}}_{j|i}, r, l$

for all capsules i in layer l and capsules j in layer $(l + 1)$: set $b_{ij} = 0$

for r iterations **do**

for all capsules i in layer l : $c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})}$

for all capsules j in layer $(l + 1)$: $\mathbf{s}_j = \sum_i c_{ij} \hat{\mathbf{u}}_{j|i}$

for all capsules j in layer $(l + 1)$: $\mathbf{v}_j = \frac{||\mathbf{s}_j||^2}{1+||\mathbf{s}_j||^2} \frac{\mathbf{s}_j}{||\mathbf{s}_j||}$

for all capsules i in layer l and capsules j in layer $(l + 1)$:

$$b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i} \mathbf{v}_j$$

return \mathbf{v}_j

CNN fundamentals

- Convolution, padding, strides and pooling

- Transposed convolutions

- Batch normalisation in convolutional networks

CNN architectures

- AlexNet

- VGG

- GoogLeNet/Inception

- ResNet

Capsule networks

Creative application: style transfer

Neural style transfer



source: [Gatys et al., 2015]

Neural style transfer



source: [Gatys et al., 2015]

Neural style transfer

- Aim of style transfer is to use one image for content and another for style
- Merge the images in such a way as to present the content of one image in the style of the other
- Style is interpreted as patterns, textures and colours
- Uses a pretrained CNN (original paper uses VGG-19 trained on ImageNet)
- Uses the features extracted by the convolutional layers
- Makes use of the increasing complexity of features and receptive field in deeper layers

Neural style transfer: content loss

- A given input image \mathbf{x} is encoded in each layer by the filter responses to that image
- A layer with N_l distinct filters has N_l feature maps each of size M_l , where M_l is the height times the width of the feature map
- Responses in a layer l can be stored in a matrix $F^l \in \mathbb{R}^{N_l \times M_l}$, where F_{ij}^l is the activation of the i -th filter at position j in layer l
- Start with a white noise image and perform gradient descent *on the image* with a loss function that encourages filter responses to be close:

$$\mathcal{L}_{\text{content}}(\mathbf{p}, \mathbf{x}) = \sum_l \frac{w_l}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2,$$

where \mathbf{p} is the original image, w_l is a layer weighting, and P^l is the feature representation of the original image in layer l .

Neural style transfer: style loss

- Style representation computes the correlations between the different filter responses
- Feature correlations are given by the Gram matrix $G^l \in \mathbb{R}^{N_l \times N_l}$, where G_{ij}^l is the inner product between the vectorised feature maps i and j (vectorised over all spatial locations) in layer l :

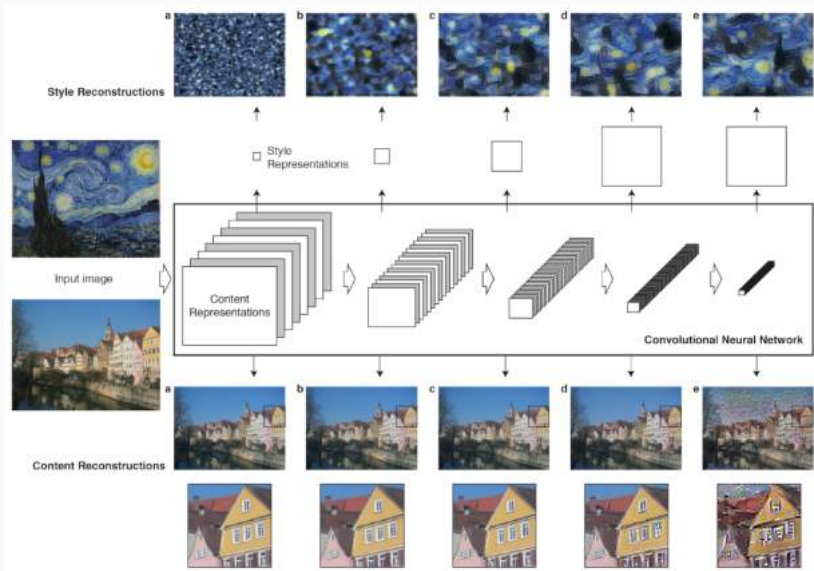
$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$$

- Again start with white noise image and perform gradient descent to match the Gram matrices of an original image \mathbf{a} and generation \mathbf{x} :

$$\mathcal{L}_{\text{style}}(\mathbf{a}, \mathbf{x}) = \sum_l \frac{w_l}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2,$$

where w_l is a weighting for each layer and A_l is the corresponding original image Gram matrix

Neural style transfer: content and style reconstructions



source: [Gatys et al., 2015]

Neural style transfer: mixing content and style

- To mix the content of one image with the style of another, jointly minimise the loss of corresponding content and style losses:




$$\mathcal{L}_{\text{total}}(\mathbf{p}, \mathbf{a}, \mathbf{x}) = \alpha \mathcal{L}_{\text{content}}(\mathbf{p}, \mathbf{x}) + \beta \mathcal{L}_{\text{style}}(\mathbf{a}, \mathbf{x}),$$

where α and β are the weighting factors for content and style reconstruction respectively

- Example images from the original paper were obtained by using a content loss weighted only on the second layer in the fourth block, and style loss on the first layer within each of the 5 blocks with equal weighting
- The ratio α/β was either 1×10^{-3} or 1×10^{-4}

Neural style transfer: weightings and style loss depth



-  Gatys, L. A., Ecker, A. S., and Bethge, M. (2015).
A neural algorithm of artistic style.
CoRR, abs/1508.06576.
-  He, K., Zhang, X., Ren, S., and Sun, J. (2015).
Deep residual learning for image recognition.
CoRR, abs/1512.03385.
-  He, K., Zhang, X., Ren, S., and Sun, J. (2016).
Identity mappings in deep residual networks.
CoRR, abs/1603.05027.



Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012).
Imagenet classification with deep convolutional neural networks.

In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA. Curran Associates Inc.



LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1989).
Backpropagation applied to handwritten zip code recognition.
Neural Computation, 1(4):541–551.



LeCun, Y., Kavukcuoglu, K., and Farabet, C. (2010).
Convolutional networks and applications in vision.
In *ISCAS 2010 - 2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems*, pages 253–256.



Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., and Yang, M.-H. (2017).

Universal style transfer via feature transforms.



Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015).

Imagenet large scale visual recognition challenge.

Int. J. Comput. Vision, 115(3):211–252.



Sabour, S., Frosst, N., and Hinton, G. E. (2017).

Dynamic routing between capsules.

CoRR, abs/1710.09829.



Simonyan, K. and Zisserman, A. (2014).

Very deep convolutional networks for large-scale image recognition.

CoRR, abs/1409.1556.



Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014).

Going deeper with convolutions.