Deep learning

Convolutional Networks[1]

Hamid Beigy

Sharif University of Technology

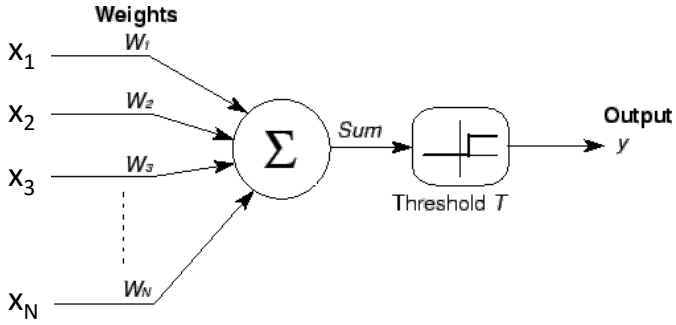
November 7, 2025

[1] Some slides are taken from B. Raj's slides

Table of contents

Introduction

1. Consider a Perceptron with threshold activation function



$$y = \begin{cases} 1 & \text{if } w^T x \geq b \\ 0 & \text{otherwise} \end{cases}$$

2. What do the weights tell us?
3. The Perceptron fires if the inner product between the weights and the inputs exceeds a threshold.

The weight as a template

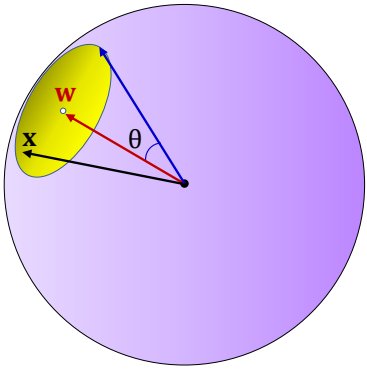
1. In Perceptron, we have

$$\mathbf{w}^\top x \geq b$$

$$\cos(\theta) \geq \frac{b}{\|\mathbf{x}\|\|\mathbf{w}\|}$$

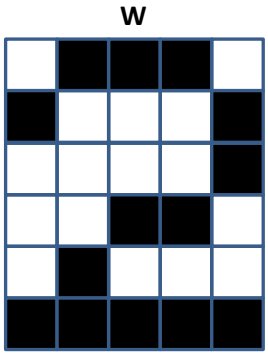
$$\theta < \cos^{-1}\left(\frac{b}{\|\mathbf{x}\|.\|\mathbf{w}\|}\right)$$

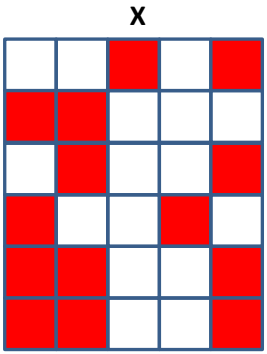
2. The Perceptron fires if the input vector is close enough to the weight vector.

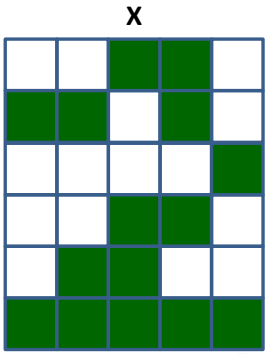


The weights as a correlation filter

1. If the correlation between the weight pattern and the inputs exceeds a threshold, Perceptron fires.
2. The Perceptron is a correlation filter.

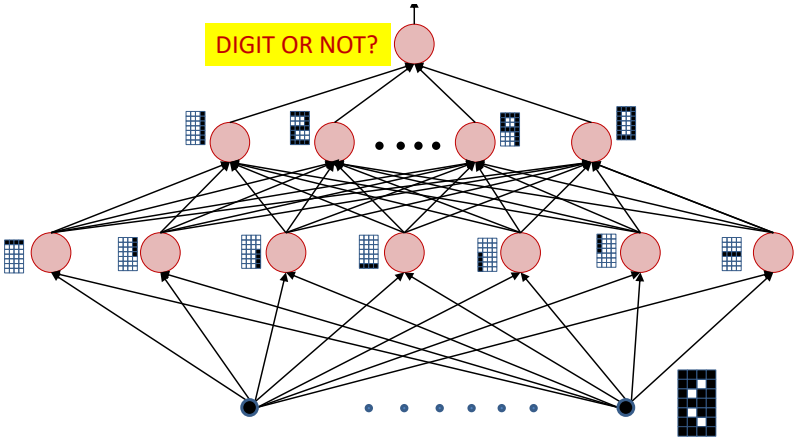






MLP as a cascade of feature detectors

1. Consider a classification task in which, we want to classify digit images from non-digit images.

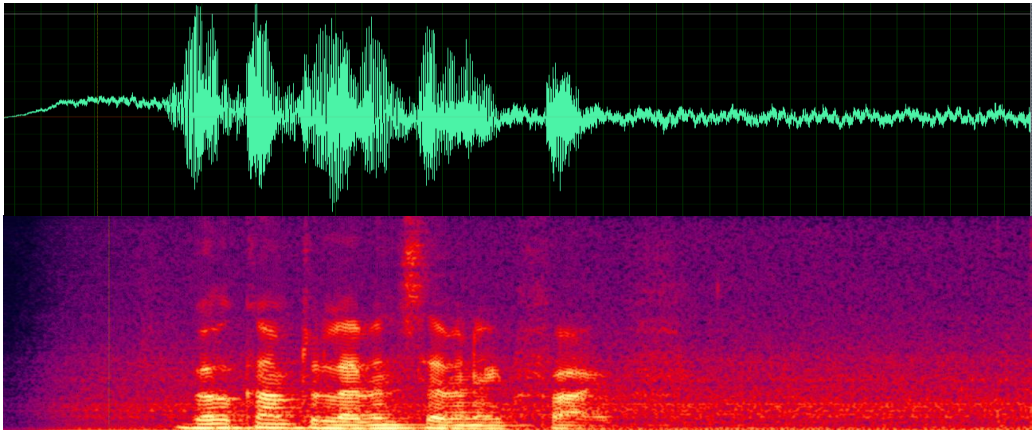


2. The network is a cascade of feature detectors
3. Higher level neurons compose complex templates from features represented by lower-level neurons.

Example

Finding a word in a signal

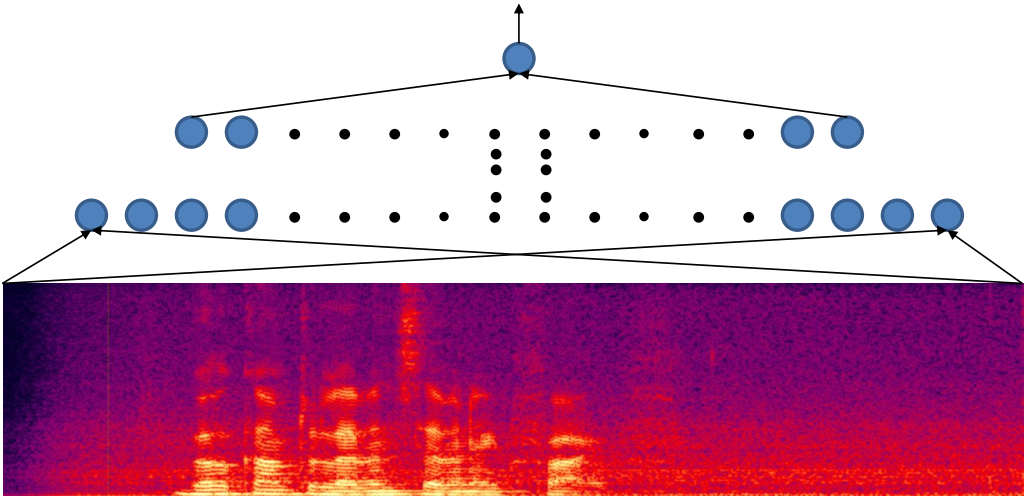
1. Does the following signal contain the word Hello?



2. Can MLP be used to solve the above problem?

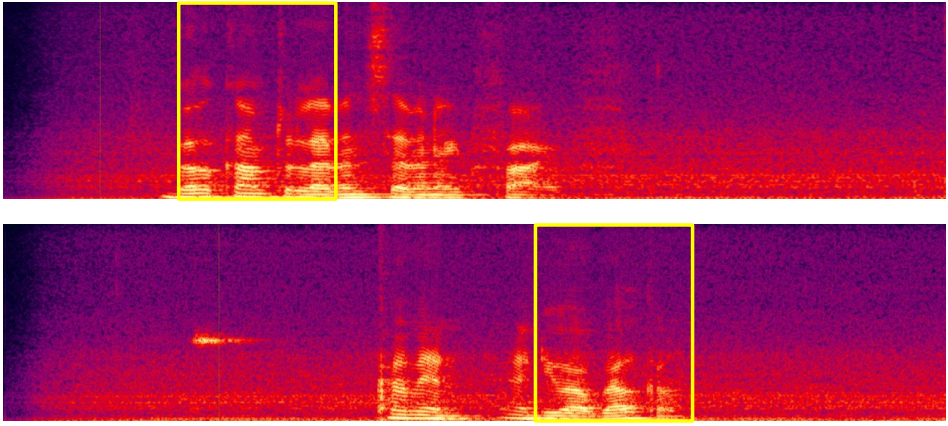
The first solution: using a MLP for the entire signal

1. We can use a large network.



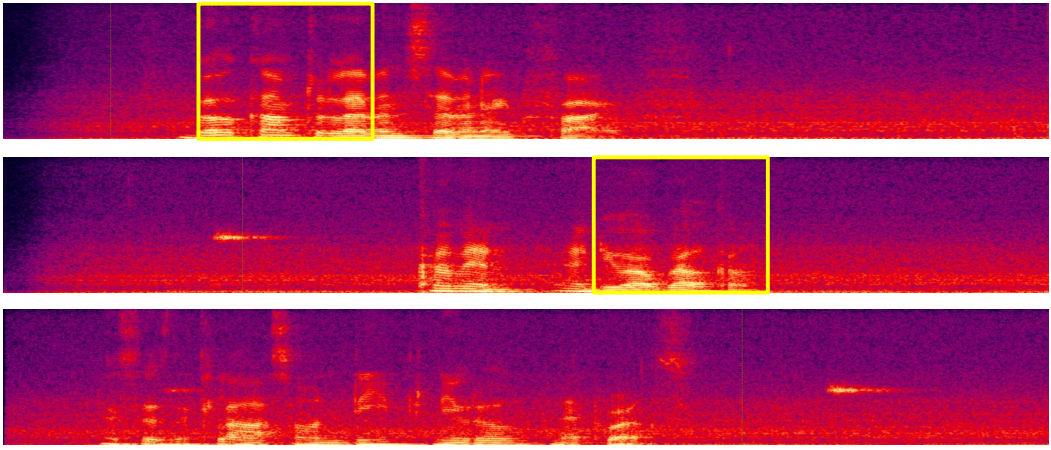
Problem with the first solution

1. Network that finds a Hello in the top recording will not find it in the lower one
   - Unless trained with both
   - Will require a very large network and a large amount of training data to cover every case



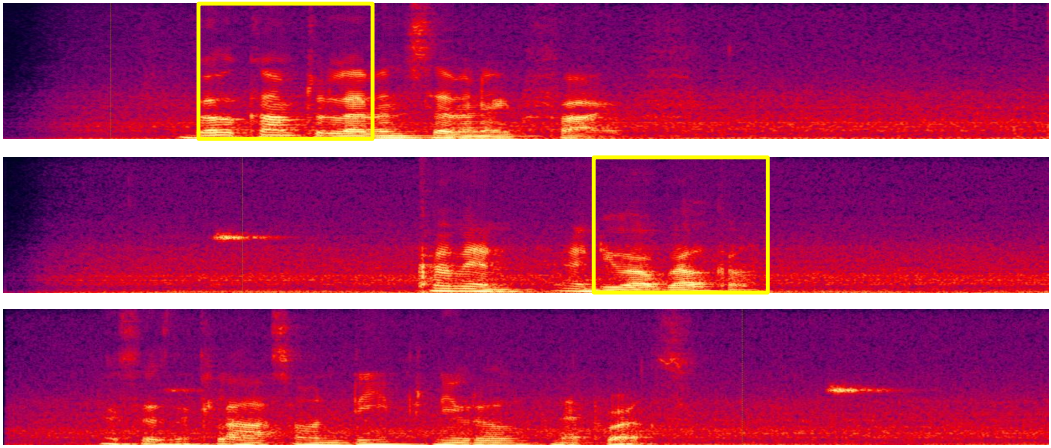
Finding Hello

1. We need a simple network that will fire regardless of the location of Hello and not fire when there is none.



The need for shift invariance

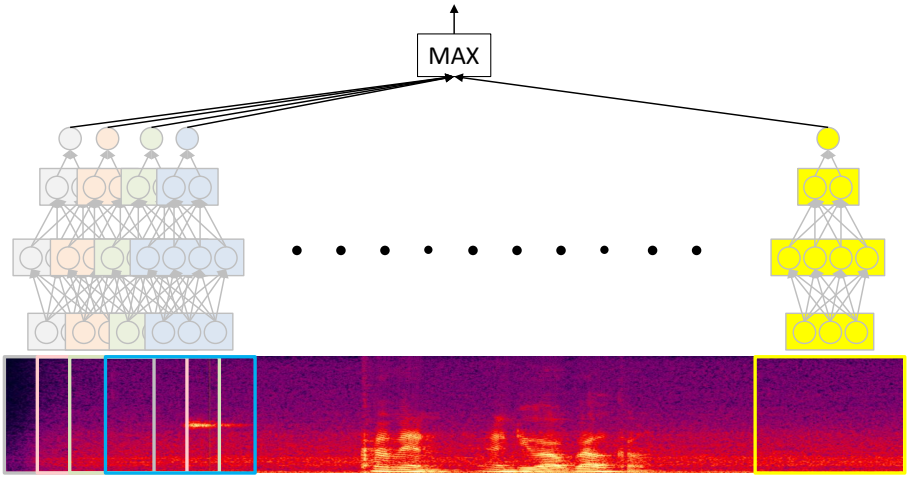
1. In many problems the location of a pattern is not important (Only the presence of the pattern).



2. Conventional MLPs are sensitive to the location of the pattern.
3. Network must be shift invariant

The second solution: scanning signal

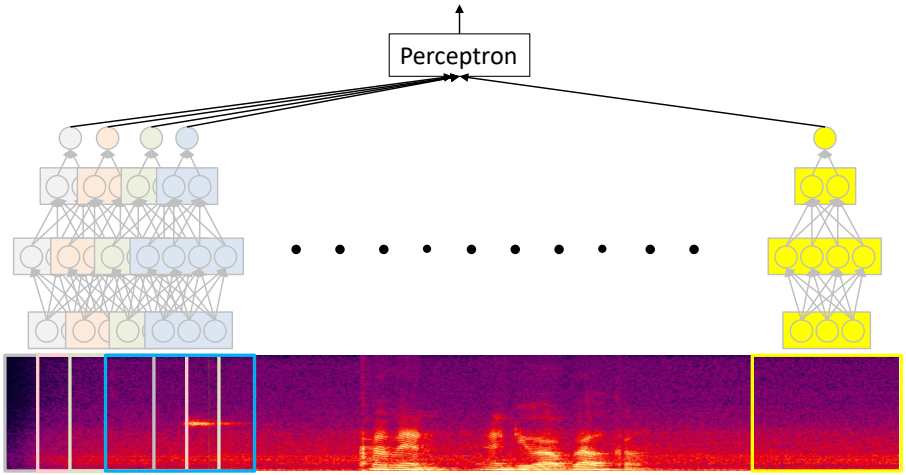
1. Use a sliding window and scan for the target word.



2. Does Hello occur in this recording?
   - We have classified many windows individually
   - Hello may have occurred in any of them.
3. Maximum of all the outputs (Equivalent of Boolean OR)

The second solution: scanning signal

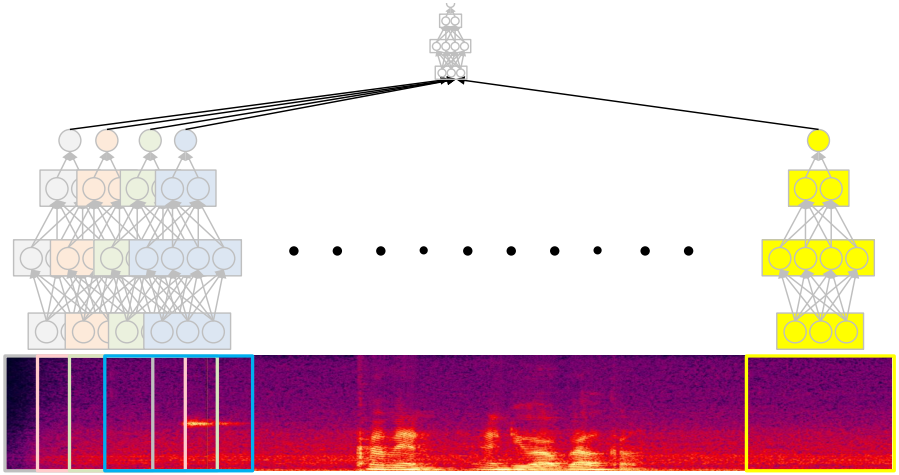
1. Use a sliding window and scan for the target word.



2. Does Hello occur in this recording?
3. Use a Perceptron to decide.

The second solution: scanning signal

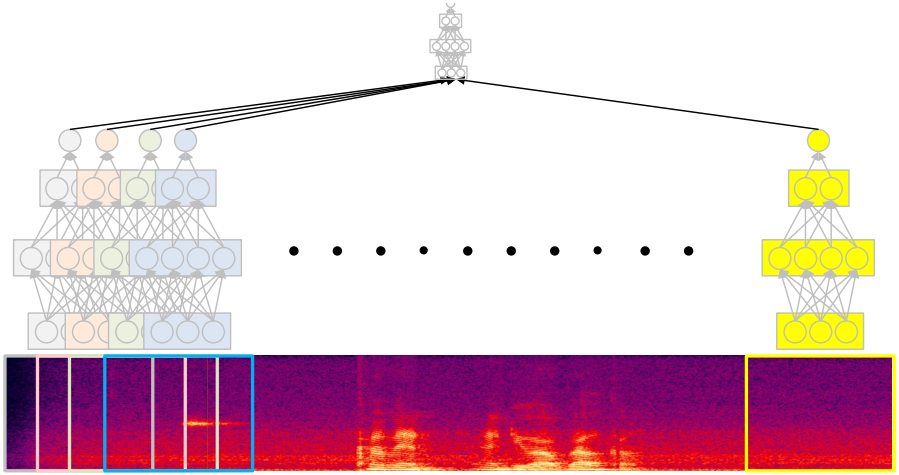
1. Use a sliding window and scan for the target word.



2. Does Hello occur in this recording?
3. Use a MLP to decide.

The second solution: scanning signal

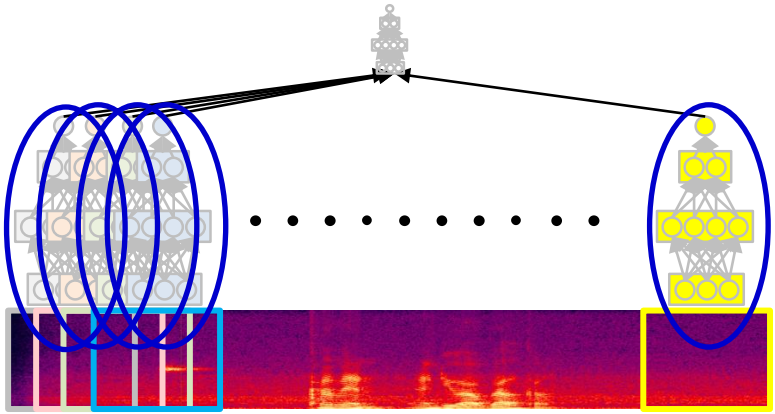
1. The entire operation can be viewed as one large network.



2. With many subnetworks, one per window
3. Restriction: All subnets are identical (why?)

The second solution: scanning signal

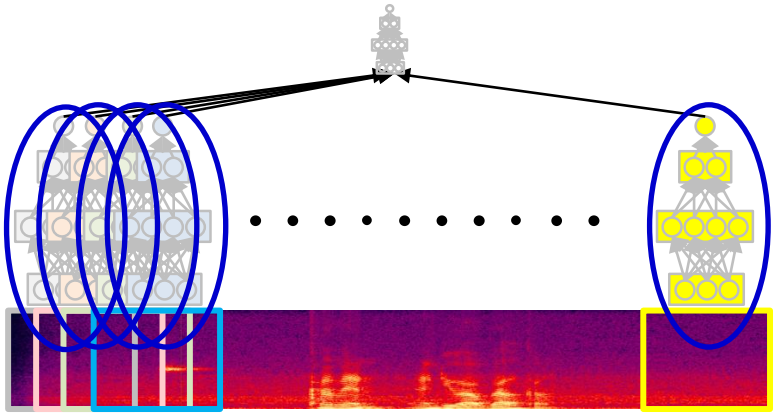
1. What are the properties of this network?



2. The sub-networks are shared parameter networks.
3. All lower-level subnets are identical (Are all searching for the same pattern?)
4. Any update of the parameters of one copy of the subnet must equally update all copies

The second solution: scanning signal

1. What are the properties of this network?



2. The network is sparse.
3. All sub-networks are used for local regions.

Training the network with shared parameter

Training the network with shared parameter

1. We use conventional back-propagation to learn the parameters.
2. By providing many training examples, we use gradient descent to minimize the cost function.
3. There are shared parameter networks and any update of the parameters of one copy of the subnet must equally update all copies.
4. Consider a simple network with the following parameter sharing $w_{ij} = w_{kl} = w_s$
5. We can calculate the gradient with respect to w_s.

$$\begin{aligned}\frac{\partial J(w)}{\partial w_s} &= \frac{\partial J(w)}{\partial w_{ij}}\frac{\partial w_{ij}}{\partial w_s} + \frac{\partial J(w)}{\partial w_{kl}}\frac{\partial w_{kl}}{\partial w_s} \\ &= \frac{\partial J(w)}{\partial w_{ij}} + \frac{\partial J(w)}{\partial w_{kl}}\end{aligned}$$

Convolution

1. The following operation is called convolution.

$$s(t) = \int_{-\infty}^{+\infty} x(\tau) w(t-\tau) d\tau$$

2. The convolution operation is typically denoted with an asterisk(*).

$$s(t) = (x * w)(t) = \int_{-\infty}^{+\infty} x(\tau) w(t-\tau) d\tau$$

3. This operation has two steps: flipping and shifting and then multiplication and integration.
4. If we now assume that x and w are defined only on integer t, we can define the discrete convolution:

$$s[n] = (x * w)[n] = \sum_{m=-\infty}^{\infty} x[m] w[n-m]$$

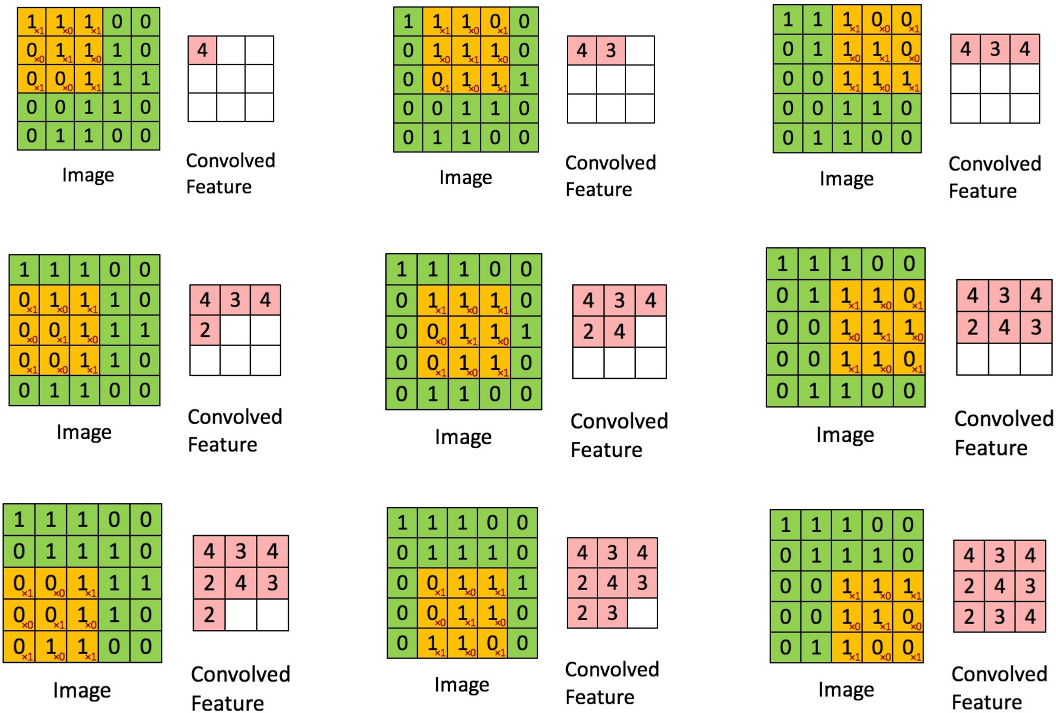
Discrete convolution

1. In practice, we often use convolutions over more than one axis at a time.

$$s[i,j] = (I * K)[i,j] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I[m,n]K[i-m, j-n]$$

2. The input is usually a multidimensional array of data.
3. The kernel is usually a multidimensional array of parameters that should be learned.
4. We assume that these functions are zero everywhere but the finite set of points for which we store the values.

Convolution (example)

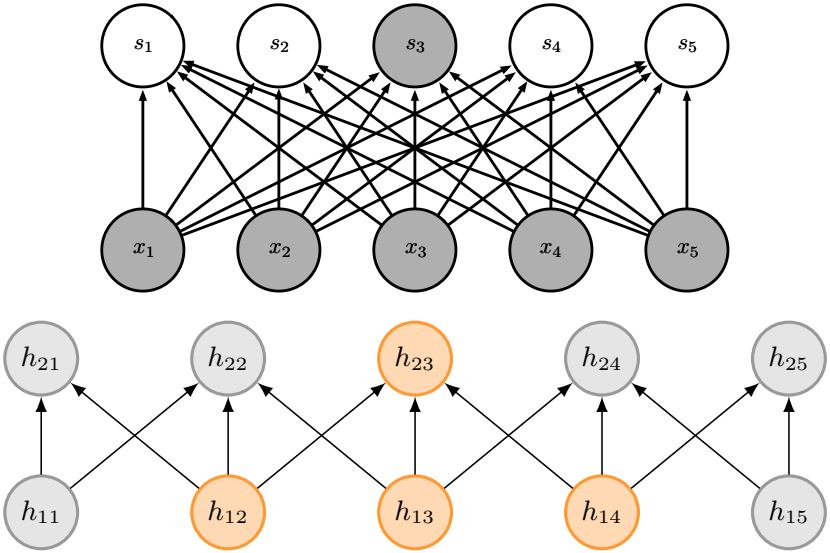


Convolutional Neural networks

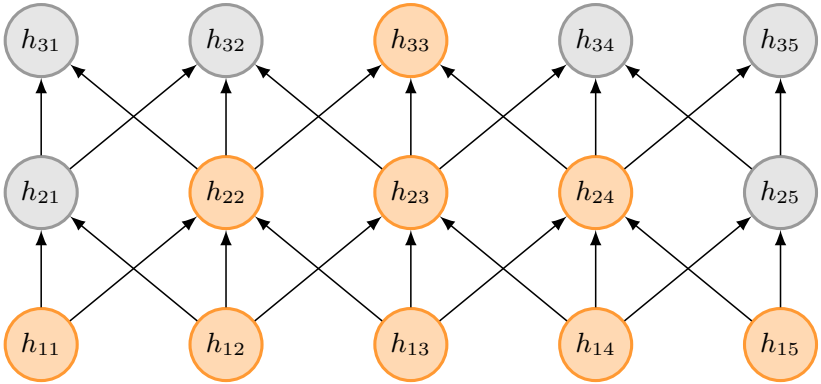
Convolutional Neural networks

1. Convolutional neural network is a kind of feed-forward neural network that is able to extract features from data with convolution structures (Li et al. 2020).
2. The architecture of CNN is inspired by visual perception.
3. A convolutional layer is a network layer that implements convolution.
4. A convolutional neural network is a feed–forward network in which at least one layer is convolution layer.
5. Convolutional neural networks have the following characteristics
   - Sparse & local interactions
   - Parameter sharing
   - Down-sampling dimensionality reduction
   - Equi-variant representations

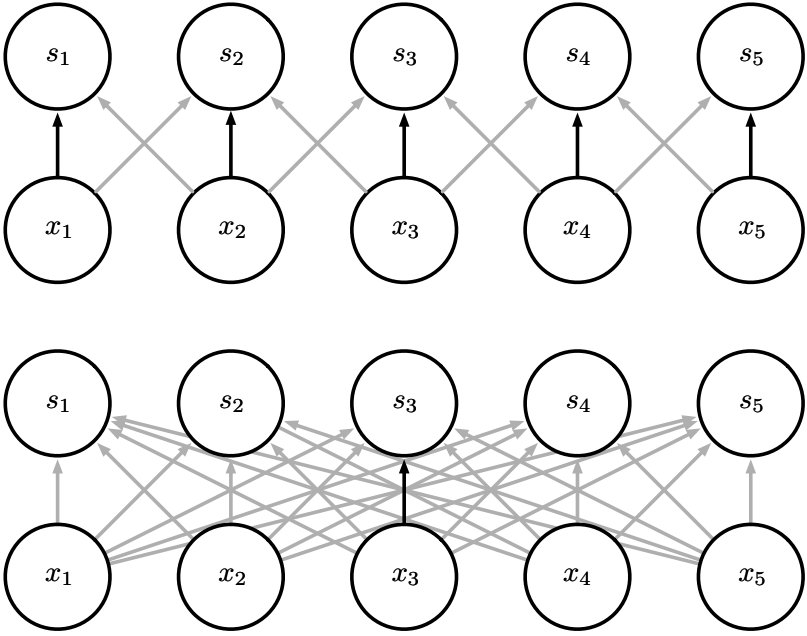
1. Sparse vs dense connectivity



1. Sparse connectivity in multi-layers



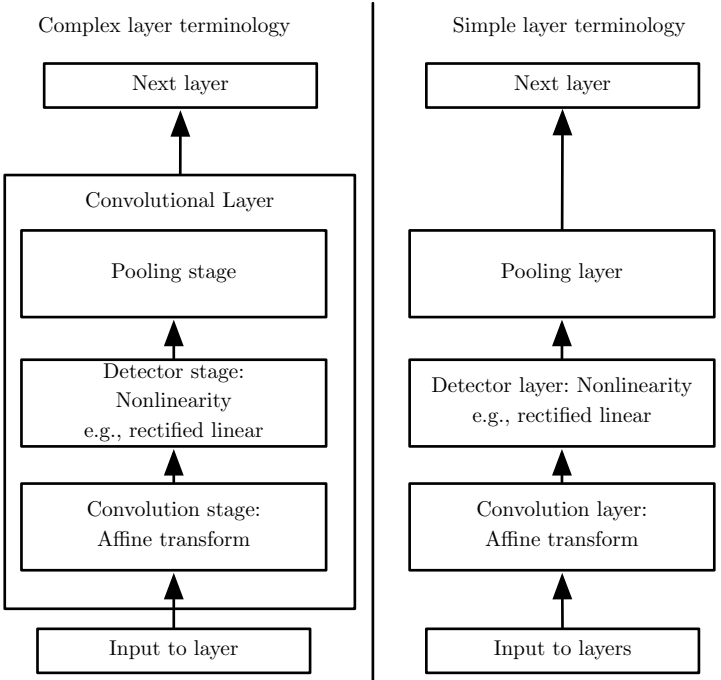
1. In CNNs each member of the kernel is used at every position of the input



Equivariance

1. A function $f(.)$ is equivariant to a function $g(.)$ if $f(g(x)) = g(f(x))$.
2. A convolutional layer is equivariance to translation.
3. For example
$$g(x)[i] = g(x)[i-1]$$
4. If we apply this transformation to x, then apply convolution, the result will be the same as if we applied convolution to x, then applied the transformation to the output. (Check it.)
5. If we move an object in the image, its representation will move the same amount in the output
6. Note that convolution is not equivariant to some other transformations, such as scaling or rotating an image.

1. The components of a typical convolutional neural network layer.



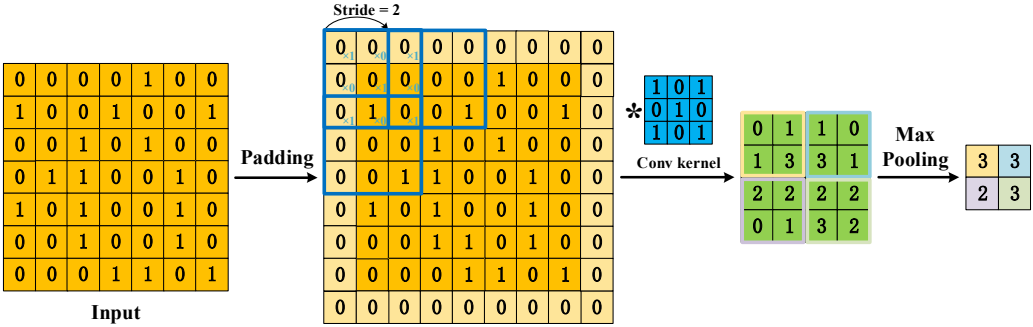
1. To build a CNN model, four components are typically needed (Li et al. 2020).

   Convolution The outputs of convolution can be called feature maps.

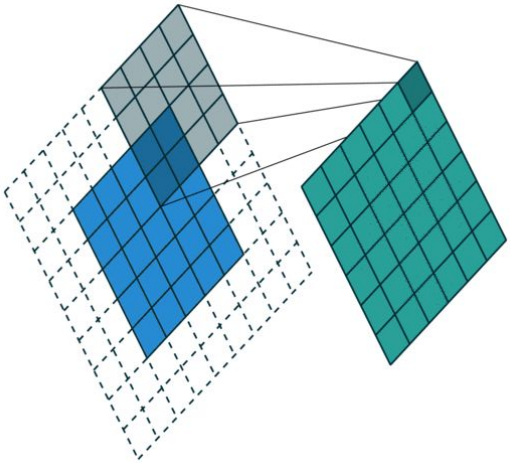
   Padding Padding enlarges the input with zero value.

   Stride For controlling the density of convolving, stride used.

   Pooling As a result, Pooling (down-sampling) such as max-pooling and average-pooling obviates large number of features in feature map.



The convolutional stage is

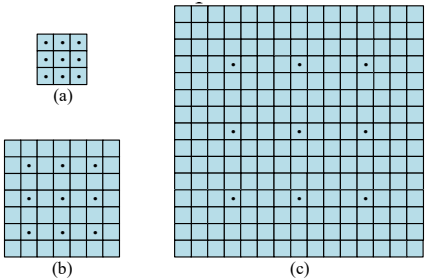


For convolution kernels to perceive larger area, dilated convolution used (Li et al. 2020).

(a) A general 3×3 convolution kernel.

(b) A 2-dilated 3×3 convolution kernel.

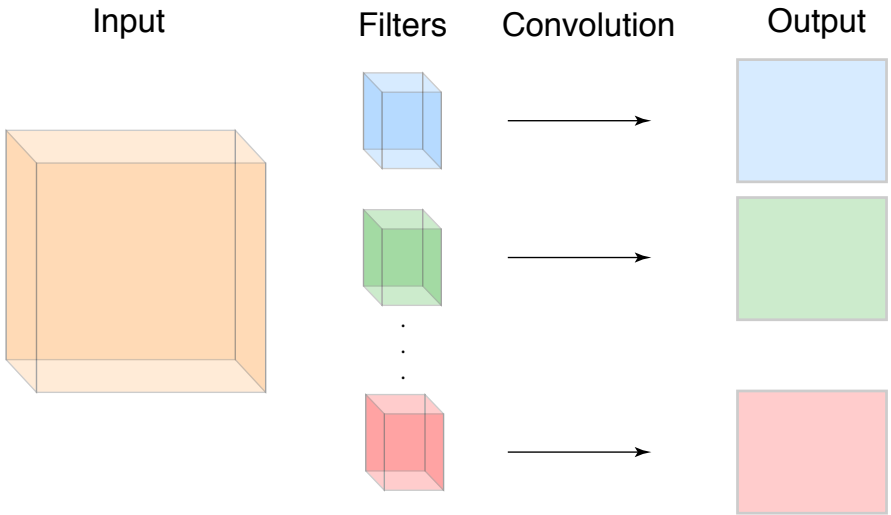
(c) A 4-dilated 3×3 convolution kernel.



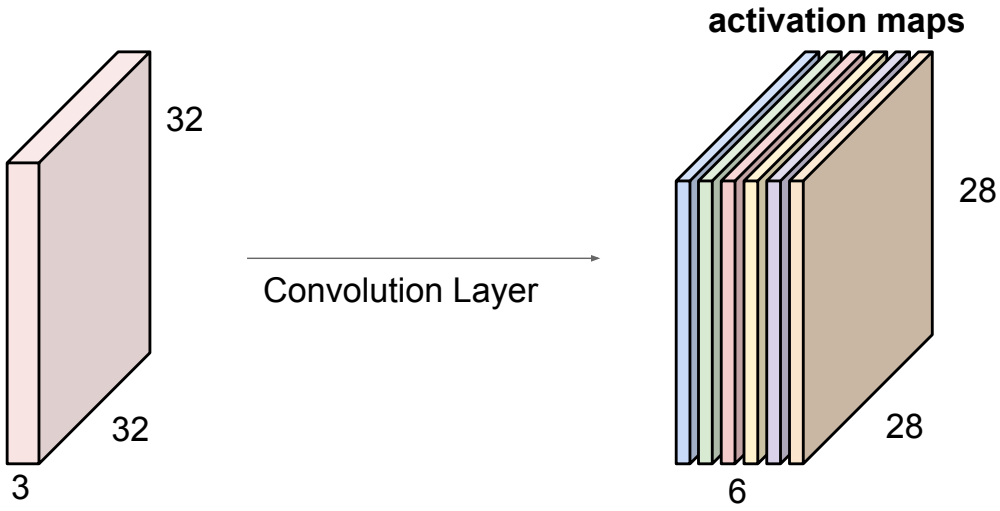
Color Image Representation

Convolutional stage

1. Now, we don't have just one filter in a convolutional layer, but multiple filters.
2. We call each output (matrix) a slice.



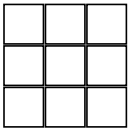
1. Multiple filters.

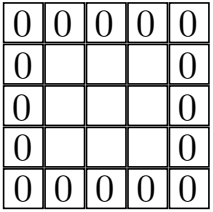


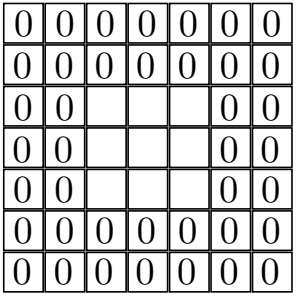
Padding

1. Usually, we specify a variable pad which is the amount of zeros to pad on each border.

$\texttt{pad} = 0$ $\texttt{pad} = 1$ $\texttt{pad} = 2$

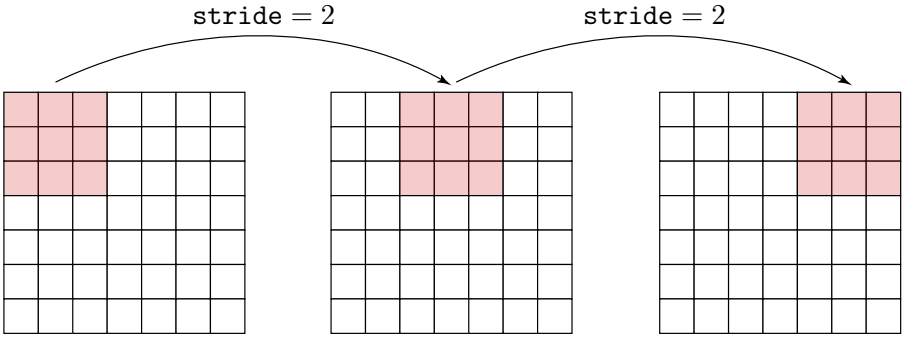






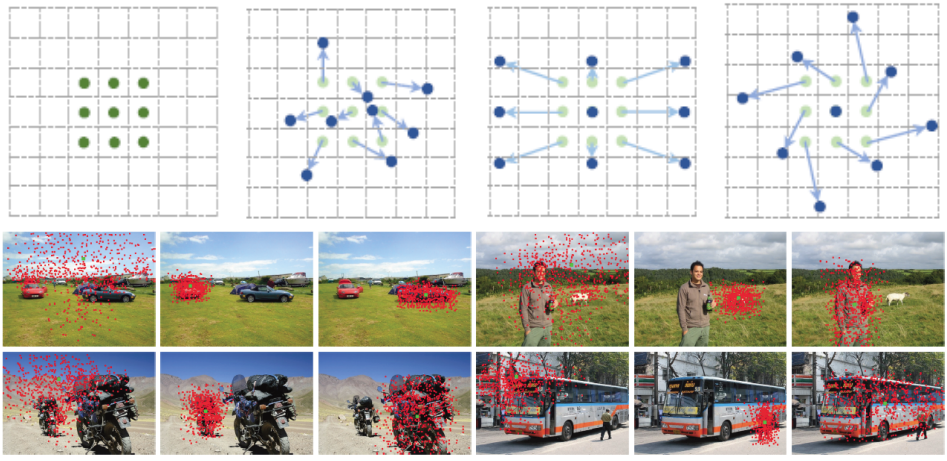
Convolution stride

1. Another variable to control is the stride, which defines how much the filter moves in the convolution.



Deformable convolution

1. To handle irregular shapes, deformable convolution is used (Dai et al. 2017)



Detector stage

1. In the second stage, each linear activation is run through a nonlinear activation function, such as the rectified linear activation function. This stage is sometimes called the detector stage.

The size of feature map

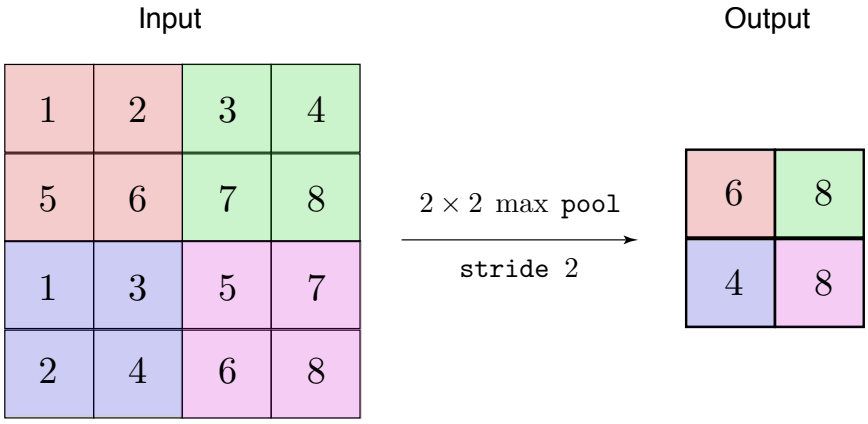
1. Suppose that we have
   - image size: $H \times W$
   - kernel size: $K \times K$
   - pad size: P
   - stride size: S
2. If we use the same stride horizontally and vertically, then the size of feature map will be

$$\lfloor \frac{H + 2P - K}{S} - 1 \rfloor \times \lfloor \frac{W + 2P - K}{S} - 1 \rfloor$$

3. Extend this equation for multi-dimensional kernel and multi-channel input.
4. Calculate the number of parameters for this layer.

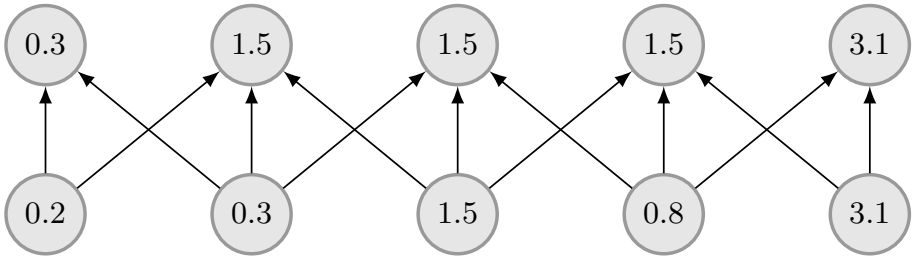
Pooling stage

1. CNNs also incorporate pooling layers, where an operation is applied to all elements within the filtering extent. This corresponds, effectively, to downs–sampling.
2. The pooling filter has width and height and is applied with a given stride. It is most common to use the *max*() operation as the pooling operation.

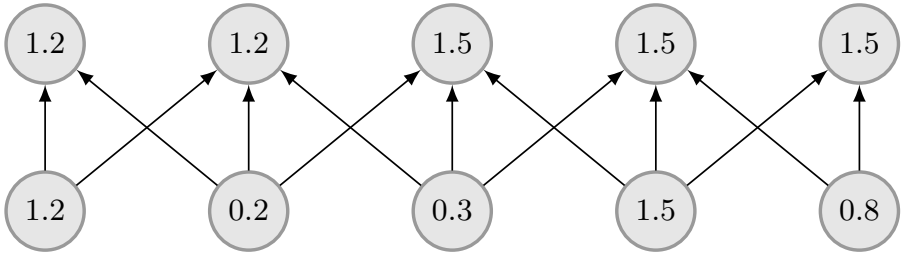


Pooling stage

1. Invariance example:



2. Inputs shifted by 1. Five inputs changed but only three outputs changed.



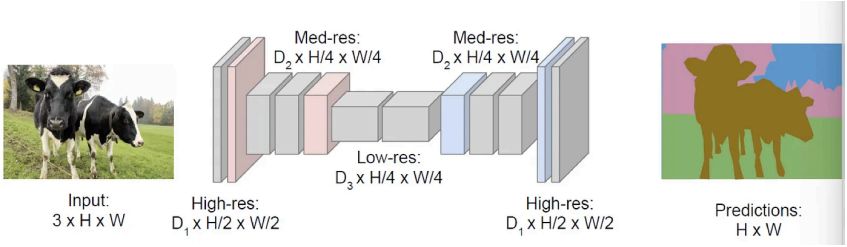
Convolution and pooling as an infinitely strong prior

1. A weak prior is a prior distribution with high entropy, such a Gaussian distribution with high variance.
2. A strong prior has very low entropy, such as a Gaussian distribution with low variance.
3. An infinitely strong prior places zero probability on some parameters.
4. A convolutional net is similar to a fully connected net with an infinitely strong prior over its weights
5. The weights for one hidden unit must be identical to the weights of its neighbor, but shifted in space.
6. The use of pooling is in infinitely strong prior that each unit should be invariant to small translations.

Up Sampling

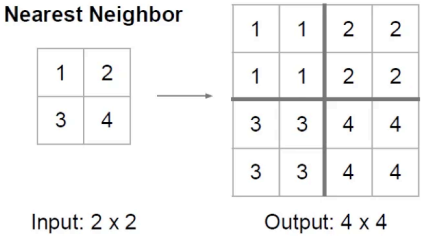
Up Sampling

1. Consider the following network.

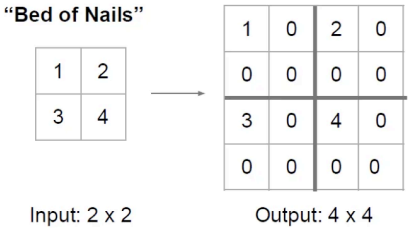


2. We have two problems
   - Unpooling
   - Deconvolution

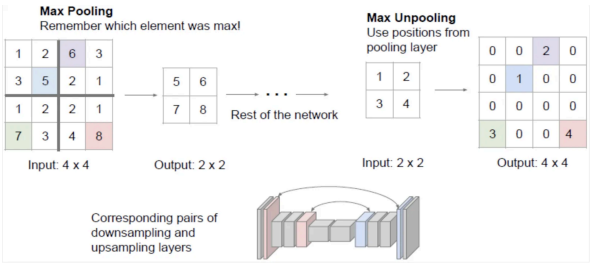
Nearest-Neighbor is simple and output is blocky.



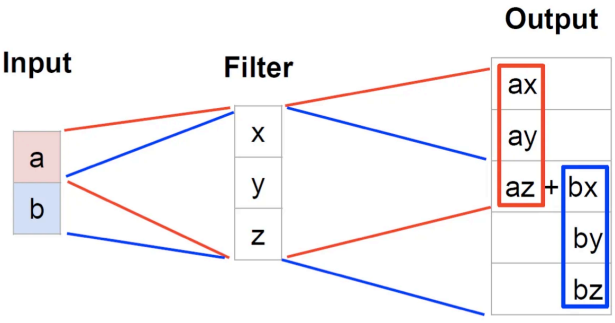
In Bed of Nails elements always have a fixed location.



Max Unpooling remembers indices of where the largest elements come from before max pooling.

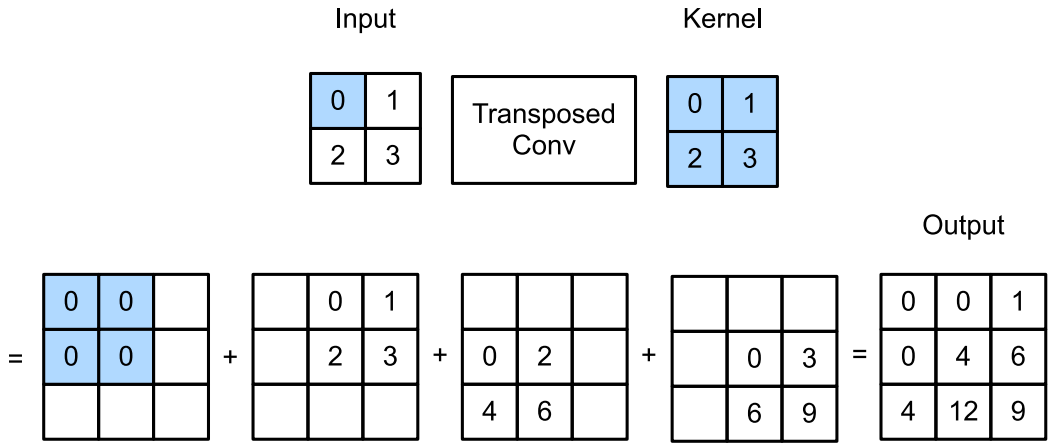


1. Consider the following setting.
   - A 2×1 input
   - A 3×1 filter
   - Transpose convolution with stride of 2
2. Output of the operation has the size 5×1.



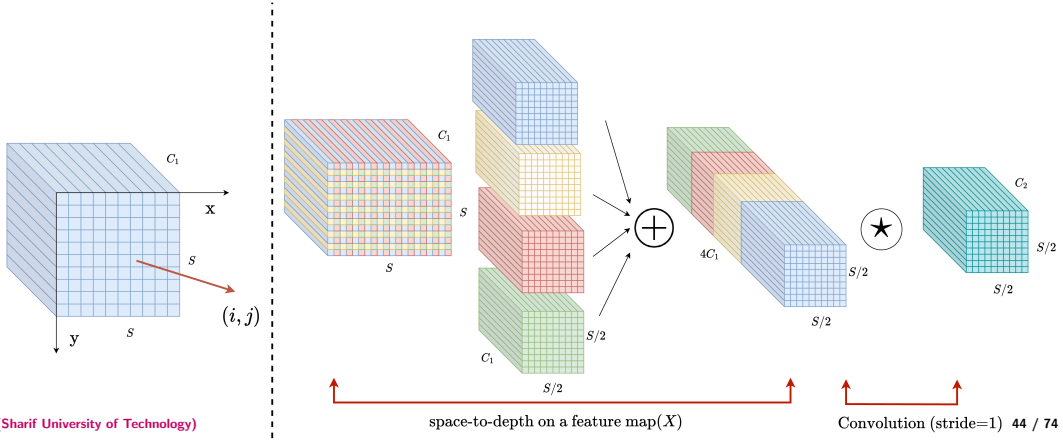
3. Compute the size of feature map for transpose convolution.

Transpose Convolution



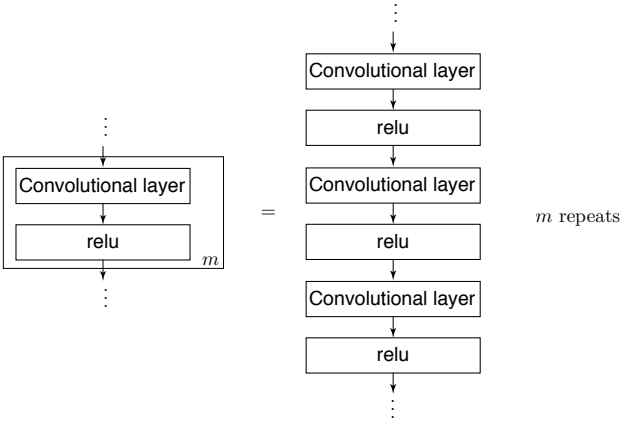
Space to dept layer

1. Performance of CNNs degrade rapidly on tasks where images are of low resolution or objects are small.
2. Sunkara and Luo designed a CNN in which strided convolution and pooling **are not used** (Sunkara and Luo 2022).
3. A space to depth layer permutes the spatial blocks of the input into the depth dimension.
4. Use this layer when you need to combine feature maps of different size without discarding any feature data.

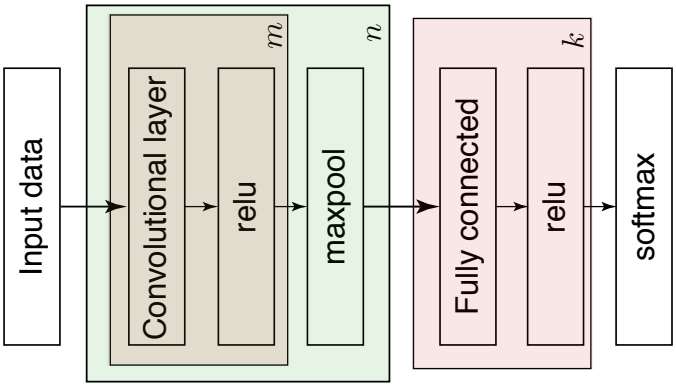


Convolutional neural network architectures

1. Typically, convolutional neural networks are comprised of units that are composed of:
   - Several paired convolutional-relu layers.
   - Potentially one max pooling layer.
2. These units are cascaded. Finally, a CNN may end with a fully connected-relu layer, followed by a softmax classifier.



1. The following describes a fairly typical CNN architecture.



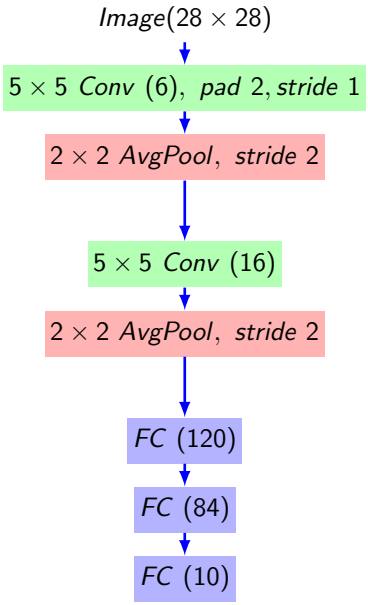
Case studies

Convolutional neural network architectures

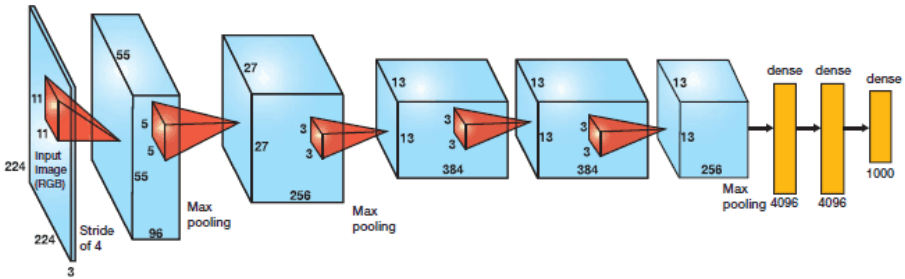
1. Beyond this, there are many architectural considerations (as measured by ImageNet performance).
    - LeNet-5 (LeCun et al. 1998)
    - AlexNet (Krizhevsky, Sutskever, and Hinton 2012) and ZFNet (Zeiler and Fergus 2014)
    - VGGNet and small filters (Simonyan and Zisserman 2015)
    - GoogLeNet and inception layers (Szegedy et al. 2015)
    - ResNet and residual layers (He et al. 2016)
    - FractalNet and the importance of reducing effective depth (Larsson, Maire, and Shakhnarovich 2017)

LeNet-5

- LeNet-5 is a convolutional network designed for handwritten and machine-printed character recognition (LeCun et al. 1998).
- It has 2 convolutional layers (with sigmoid activation function) and 3 fully-connected layers (with sigmoid activation function) (hence "5").
- The average-pooling layer was used as a sub-sampling layer.
- ReLUs and max-pooling work better.
- This architecture has about $60,000$ parameters.
- Demo : `http://yann.lecun.com/exdb/lenet/`



1. AlexNet has been proposed for image classification (Krizhevsky, Sutskever, and Hinton 2012).

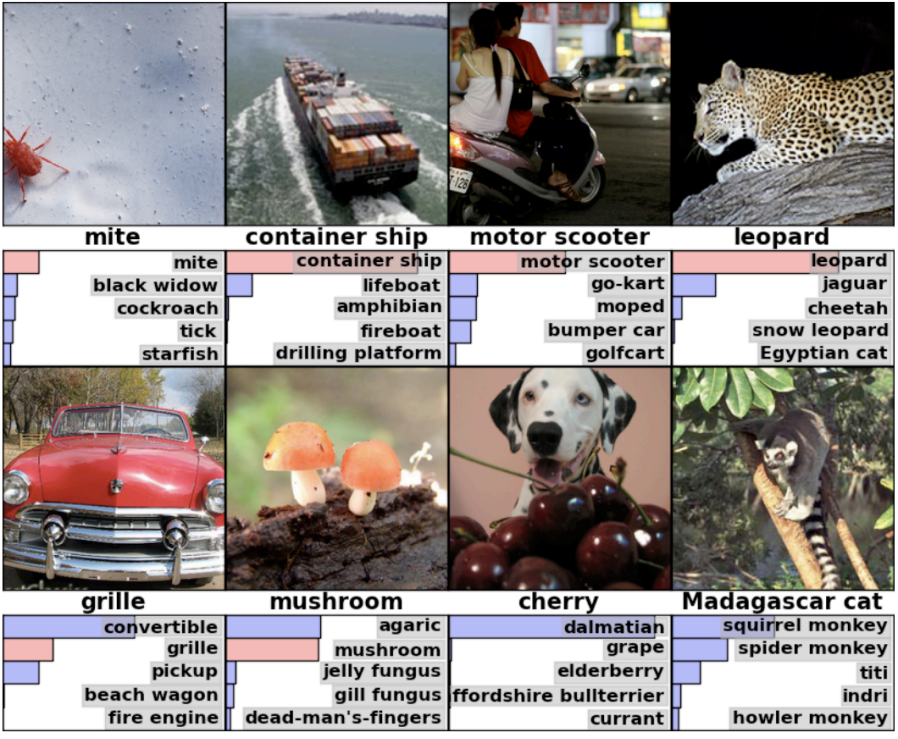


2. Dropout is used in the last few fully-connected layers to randomly ignore some neurons during training to avoid overfitting.
3. First layer (CONV1): 96 11×11 filters applied at stride 4
4. MaxPool (MaxPool1) 3×3 filters applied at stride 2. Max pooling can avoid the blurred result of average pooling.
5. Local response normalization is used.
6. Two GPUs are used to train AlexNet.
7. AlexNet has 60M parameters.

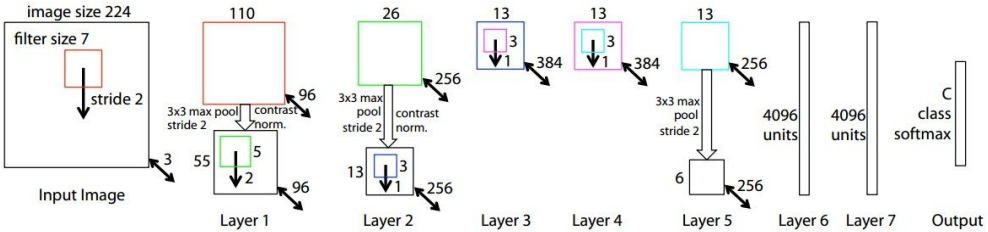
AlexNet

227x227x3	INPUT
55x55x96	CONV1: 96 11x11 filters at stride 4, pad 0
27x27x96	MAX POOL1: 3x3 filters at stride 2
27x27x96	NORM1: Normalization layer
27x27x256	CONV2: 256 5x5 filters at stride 1, pad 2
13x13x256	MAX POOL2: 3x3 filters at stride 2
13x13x256	NORM2: Normalization layer
13x13x384	CONV3: 384 3x3 filters at stride 1, pad 1
13x13x384	CONV4: 384 3x3 filters at stride 1, pad 1
13x13x256	CONV5: 256 3x3 filters at stride 1, pad 1
6x6x256	MAX POOL3: 3x3 filters at stride 2
4096	FC6: 4096 neurons
4096	FC7: 4096 neurons
1000	FC8: 1000 neurons (class scores)

AlexNet Results



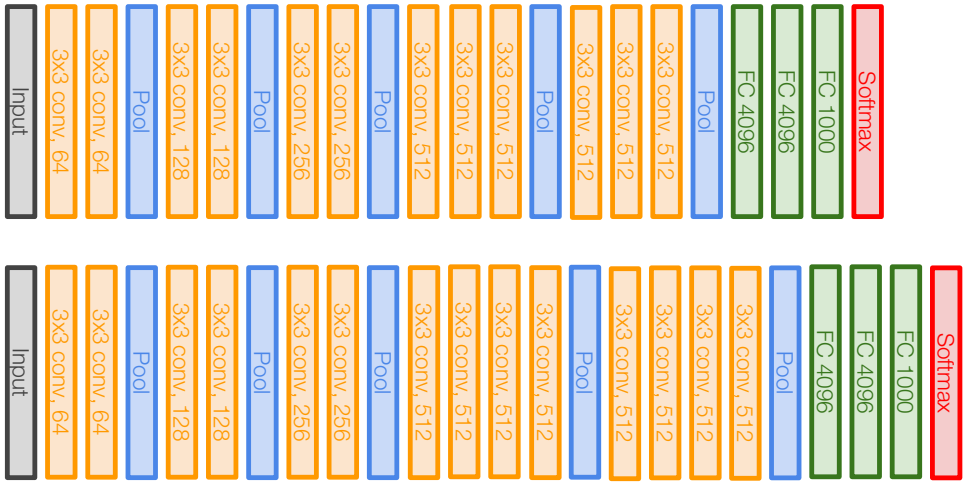
1. ZFNet has the following architecture (Zeiler and Fergus 2014).



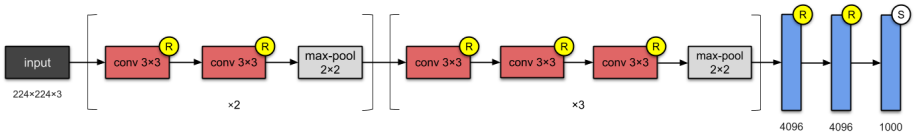
2. ZFNet is AlexNet with the following modifications
3. CONV1: change from (11x11 stride 4) to (7x7 stride 2)
4. CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

VGGNet

1. VGGNets are a series of convolutional neural network algorithms proposed by the Visual Geometry Group (VGG) of Oxford University, including VGG-11, VGG-11-LRN, VGG-13, VGG-16, and VGG-19.
2. VGGNets secured the first place in the localization track of ImageNet Challenge 2014.
3. The LRN (Local Response Normalization) layer was removed since the author of VGGNets found the effect of LRN in deep CNNs was not obvious.
4. VGGNets use 3×3 convolution kernels.
5. VGGNet uses small filters, deeper networks
6. VGGNet has the following architecture (Simonyan and Zisserman 2015).
   - Only 3×3 CONV with stride 1,
   - Pad 1
   - 2×2 MAX POOL with stride 2


VGG16
Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 64
3x3 conv, 64
Input
VGG19
Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 64
3x3 conv, 64
Input

1. The VGG-16 which has 13 convolutional and 3 fully-connected layers with ReLU activation function.



2. This network stacks more layers onto AlexNet.
3. It uses smaller size filters (2×2 and 3×3). It consists of 138M parameters.
4. The contribution from this paper is the designing of deeper networks (roughly twice as deep as AlexNet).

GoogLeNet

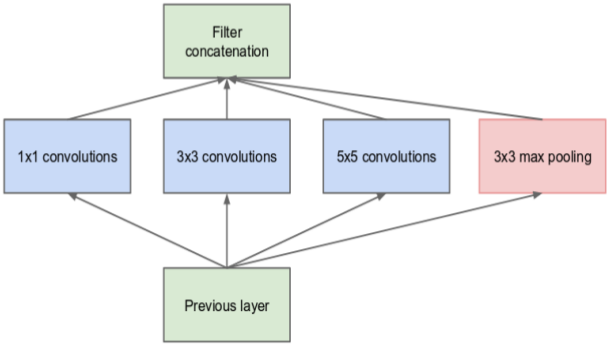
1. GoogLeNet is the winner of the ILSVRC 2014 image classification algorithms (Szegedy et al. 2015).
2. It is the first large-scale CNN formed by stacking with Inception modules.
3. GoogLeNet didn't use fully connected layers and global average pooling is used at the end of the network and eliminating a large amount of parameters that do not seem to matter much.
4. This layer takes a feature map of 7×7 and averages it to 1×1.
5. This 22-layer architecture with 5M parameters is called the Inception-v1.
6. Its main contribution was the development of an Inception Module that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M).
7. It achieved a top-5 error rate of 6.67%!

Inception Modules

1. Objects in images have different distances to cameras and then have large variation in size.
2. An object with a large (small) proportion of an image usually prefers a large (small) convolution kernel or a few small (many large) ones.
3. Hence, choosing the right kernel size becomes a challenging problem.
4. A larger/smaller kernel is useful for globally/locally distributed information.
5. Very deep networks may overfit and stacking large convolution layers is computationally expensive.
6. GoogleNet use filters with multiple sizes operating on the same level. Hence the network becomes wider.
7. Inception modules are used to implement filters with multiple sizes operating on the same level.
8. Inception modules have four versions, namely Inception v1, Inception v2, Inception v3, and Inception v4.

Inception V1

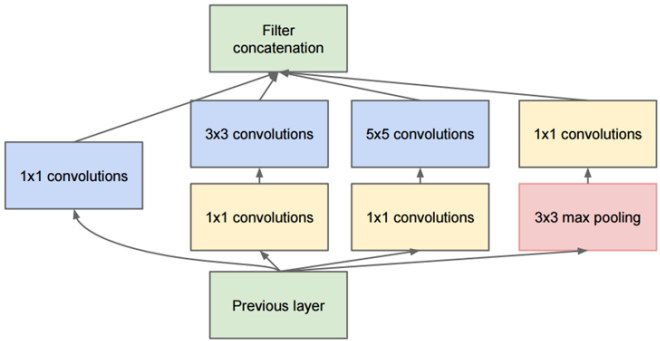
1. Inception V1 module has the following architecture.



2. Inception V1 performs convolution on an input, with 3 different sizes of filters: 1×1, 3×3, and 5×5.
3. It also performs max pooling.
4. The outputs of these filters are concatenated and sent to the next inception module.

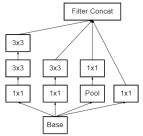
Inception V1 with dimensionality reduction

1. To reduce computational cost, inception module uses 1×1 convolution kernel to reduce the number of channels.
2. Consider a $28 \times 28 \times 190$ input and use 48 5×5 kernels resulting $28 \times 28 \times 48$ feature map.
3. Number of parameters: $(5 \times 5 \times 190) \times (28 \times 28 \times 48) = 178,752,000$.
4. Now first use 16 1×1 kernels and then use 48 5×5 resulting $28 \times 28 \times 48$.
   - Number of parameters in first layer: $(5 \times 5 \times 190) \times (28 \times 28 \times 16) = 118,750$.
   - Number of parameters in second layer: $(5 \times 5 \times 16) \times (28 \times 28 \times 48) = 15,052,800$.
   - The total number of parameters: $118,750 + 15,052,800 = 15,171,550$.

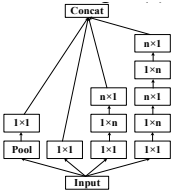


Inception V2

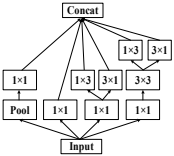
1. Inception V2 uses batch normalization layer.
2. Inception V1 5×5 filter aggressively decreases the feature map.
3. Inception V2 replaces 5×5 filter by two 3×3 filters.



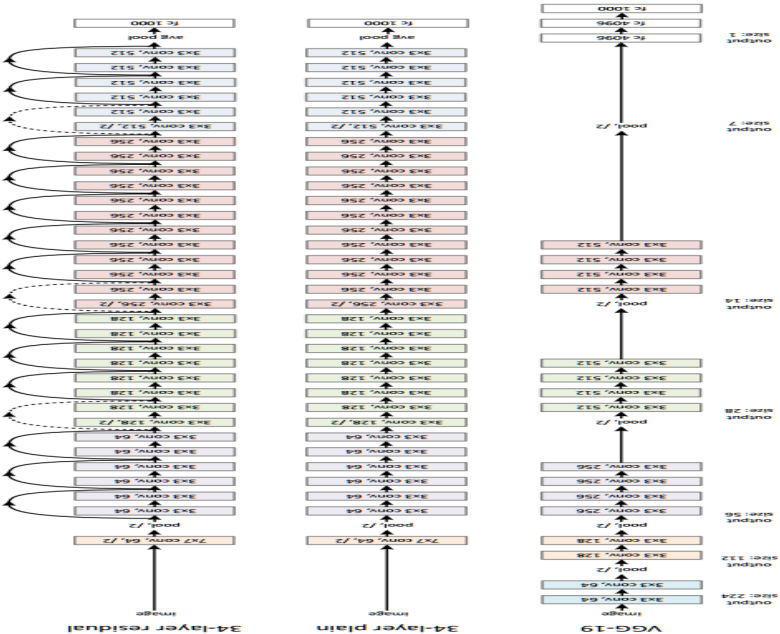
1. Inception V2 also uses asymmetric convolution.
2. Each 3×3 filter factorized by 1×3 and 3×1 filters.



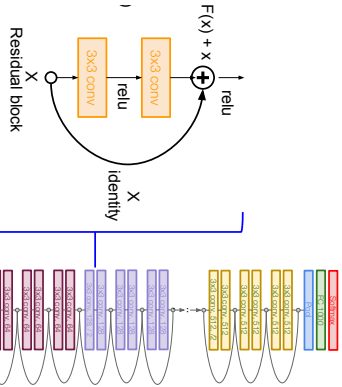
1. The use of factorization is not effective in the early layers.
2. It is better to use it on medium-sized feature maps.
3. Filter banks should be expanded (wider but not deeper) to improve high dimensional representations.



1. ResNet is a new 152 layer network architecture (He et al. 2016).
2. ResNet has the following architecture.



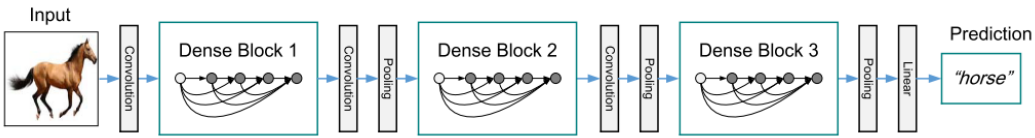
1. ResNet has stacked residual blocks.
2. Every residual block has two 3x3 conv layers



3. The authors believe that "it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping".
4. The ResidualNet creators proved empirically that it's easier to train a 34-layer residual compared to a 34-layer cascaded (Like VGG).

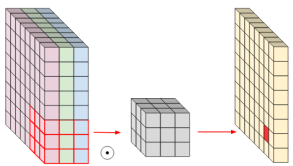
DenseNet

1. DenseNet has stacked dense blocks (Huang et al. n.d.).
2. Each layer is connected to every other layer (name Densely Connected).
3. DenseNets layers are very narrow (e.g. 12 filters).
4. They add a small set of new feature-maps.

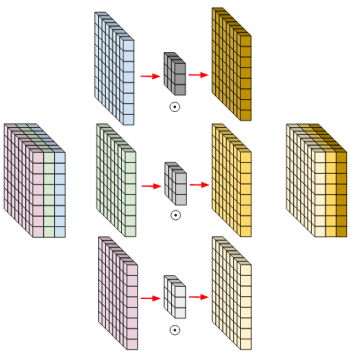


1. Google proposed MobileNet V1 to reduce the model size and complexity(Howard et al. 2017).
2. Depthwise Separable Convolution is used.
3. It is useful for mobile and embedded vision applications

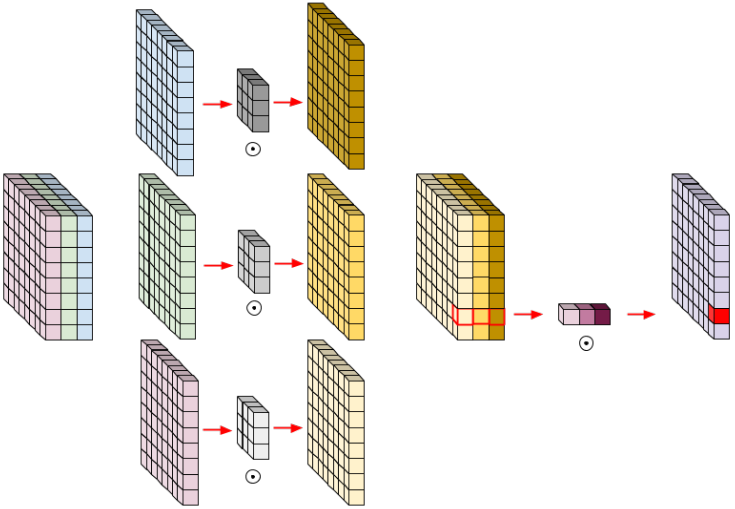
In a standard convolution, we would directly convolve in depth dimension.



In depthwise convolution, each filter channel only used at one input channel.



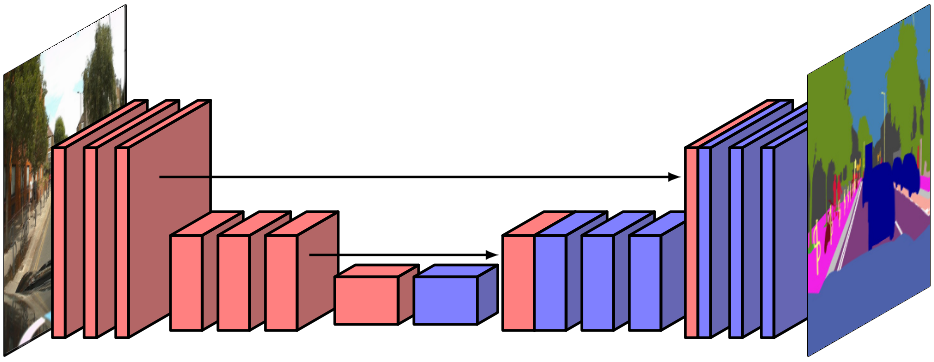
Depthwise separable convolution uses 1×1 convolution after Depthwise convolution[2].



[2]Figure taken from Atul Pandey page

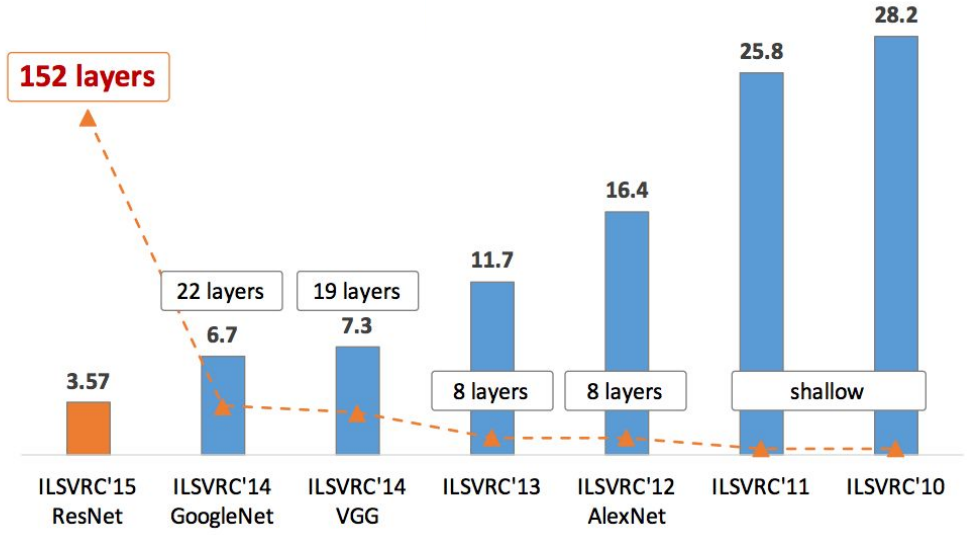
The U-net architecture

1. U-Net consists of a contracting path and an expansive path (Ronneberger, Fischer, and Brox 2015).
2. The contracting path captures contextual information and reduce the spatial resolution of the input.
3. The expansive path decodes the encoded data and use the information from the contracting path via skip connections to generate a segmentation map.

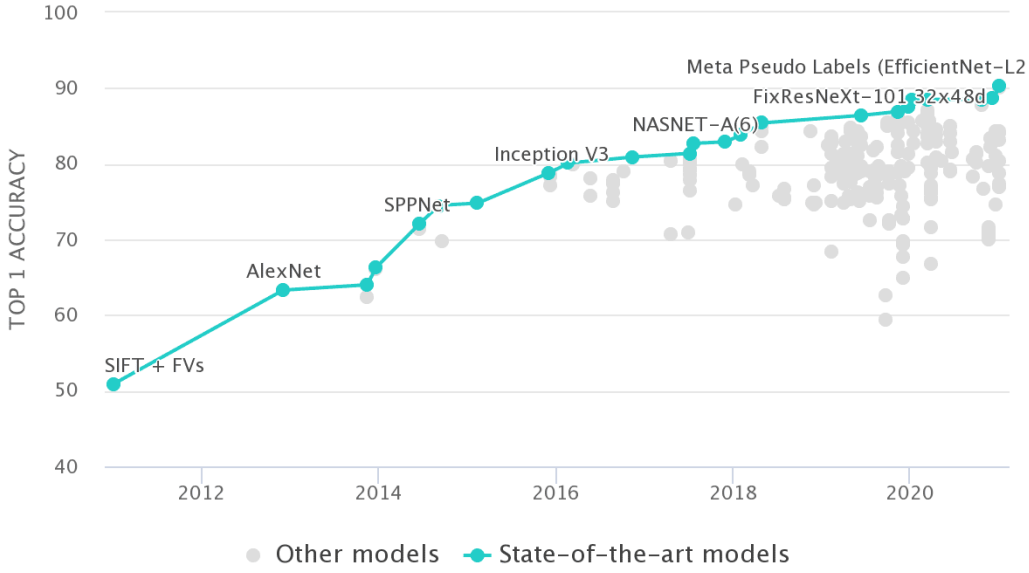


4. Please read this paper.

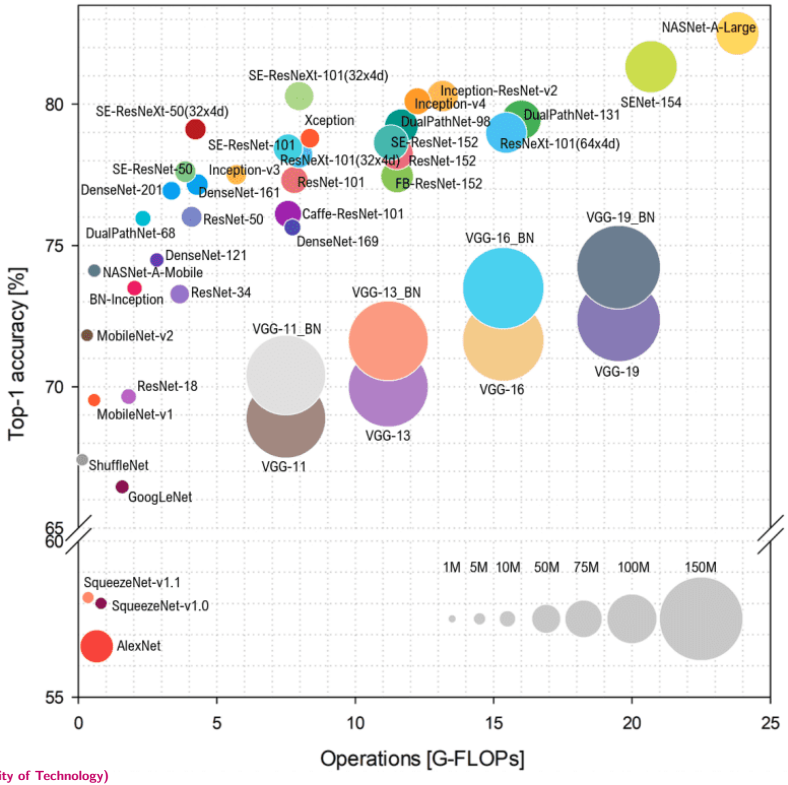
Comparing different CNN architectures on ImageNet



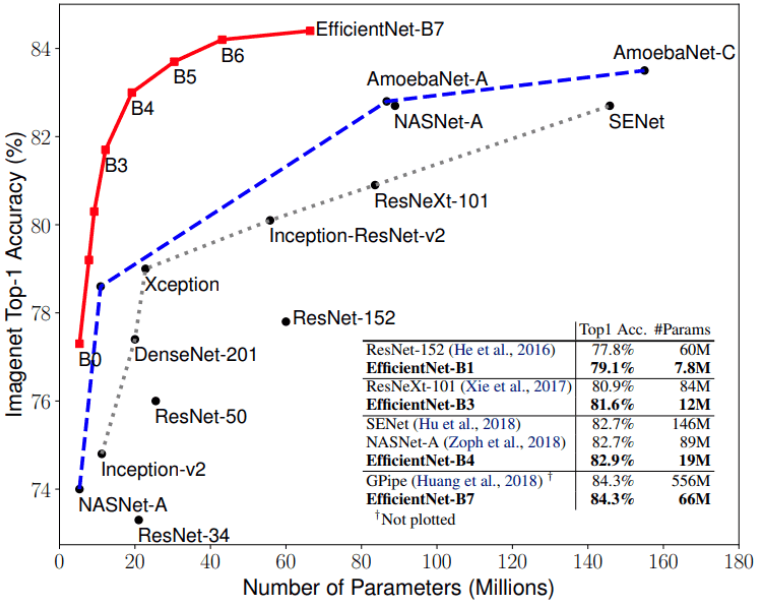
Comparing different CNN architectures on ImageNet



Comparing different CNN architectures



Comparing different CNN architectures



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

Some other CNN networks

1. Inception-v3 (2015)
2. Xception (2016)
3. Inception-v4 (2016)
4. Inception-ResNet-V2 (2016)
5. ResNeXt-50 (2017)
6. DenseNet (CVPR 2017)
7. MobileNets (2017 & 2018)
8. EfficientNets (2020)
9. For more networks, please study the networks given in
   https://paperswithcode.com/sota/image-classification-on-imagenet

Reading

Readings

1. Chapter 9 of Deep Learning Book[3]

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. The MIT Press.

References i

- Dai, Jifeng et al. (2017). "Deformable Convolutional Networks". In: *IEEE International Conference on Computer Vision*, pp. 764–773.
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. The MIT Press.
- He, K. et al. (2016). "Deep Residual Learning for Image Recognition". In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.
- Howard, Andrew G. et al. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. eprint: 1704.04861.
- Huang, Gao et al. (n.d.). "Densely Connected Convolutional Networks". In: *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2261–2269.
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*, pp. 1097–1105.
- Larsson, Gustav, Michael Maire, and Gregory Shakhnarovich (2017). "FractalNet: Ultra-Deep Neural Networks without Residuals". In: *International Conference on Learning Representations*.
- LeCun, Yann et al. (1998). "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE*. Vol. 86. 11, pp. 2278–2324.
- Li, Zewen et al. (2020). "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects". In: *CoRR* abs/2004.02806.

References ii

- Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention*. Vol. 9351, pp. 234–241.
- Simonyan, Karen and Andrew Zisserman (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition.". In: *International Conference on Learning Representations*.
- Sunkara, Raja and Tie Luo (2022). "No More Strided Convolutions or Pooling: A New CNN Building Block for Low-Resolution Images and Small Objectso". In: *Proc. of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*.
- Szegedy, Christian et al. (2015). "Going Deeper with Convolutions". In: *Computer Vision and Pattern Recognition (CVPR)*.
- Zeiler, Matthew D. and Rob Fergus (2014). "Visualizing and Understanding Convolutional Networks". In: *Proc. of European Conference on Computer Vision*, pp. 818–833.

Questions?