Convolutional Neural Networks

Szu-Chi Chung

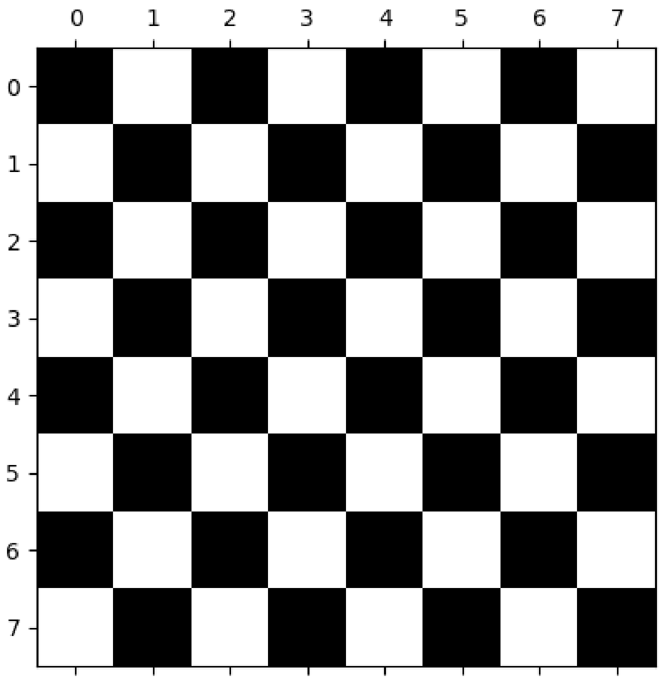
Department of Applied Mathematics, National Sun Yat-sen University

Image representation

- Images are represented as multi-dimensional arrays
  - The origin is located in top-left corner
  - 0 is for black value and 255 (or 1.0) is for white value

Image:	np.ndarray
pixels:	array values: a[2, 3]
channels:	array dimensions
image encoding:	dtype (np.uint8, np.float)
filters:	functions (numpy, skimage, scipy)

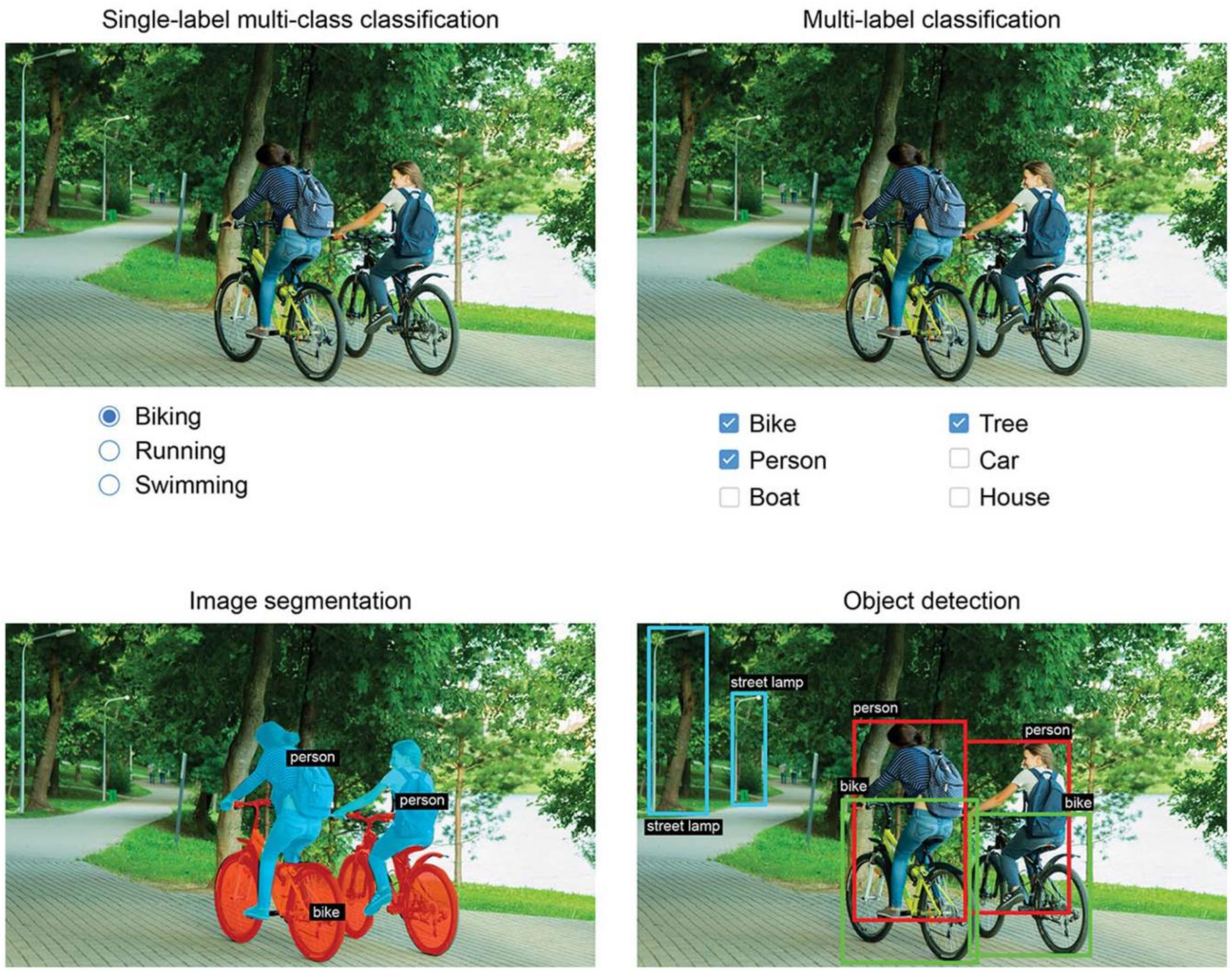
Image type	Coordinates
2D grayscale images	(row, column)
2D multichannel images	(row, column, channel)
batch of 2D grayscale images	(batch, row, column)
2D multichannel images	(batch, row, column, channel)



Three essential computer vision tasks

- **Image classification**—Where the goal is to assign one or more labels to an image. It may be either single-label classification (an image can only be in one category, excluding the others), or multi-label classification (tagging all categories that an image belongs to)
  - For example, when you search for a keyword on the Google Photos app, behind the scenes you're querying a very large multilabel classification
- **Object detection**—Where the goal is to draw rectangles (called bounding boxes) around objects of interest in an image, and associate each rectangle with a class
  - A self-driving car could use an object-detection model to monitor cars, pedestrians, and signs in view of its cameras, for instance
- **Image segmentation**—Where the goal is to "segment" or "partition" an image into different areas, with each area usually representing a category
  - For instance, when Zoom or Google Meet displays a custom background behind you in a video call, it's using an image segmentation model to tell your face apart from what's behind it, at pixel precision

Three essential computer vision tasks



Single-label multi-class classification

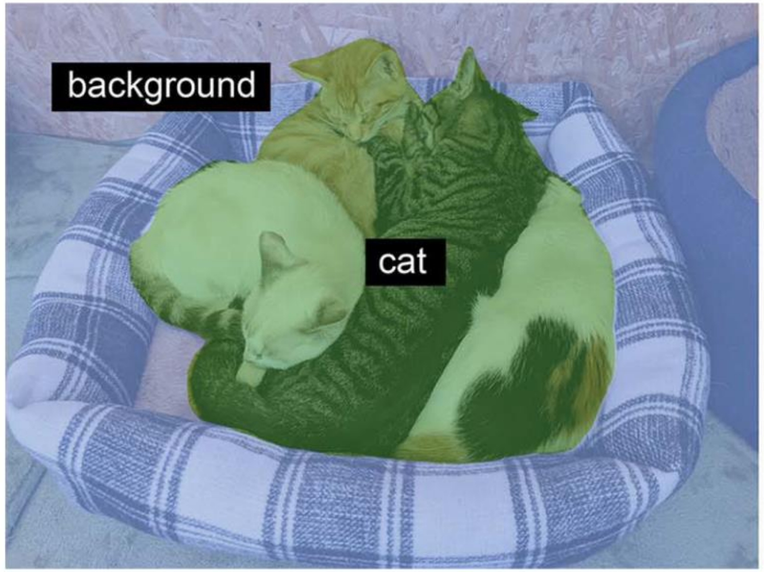
Multi-label classification

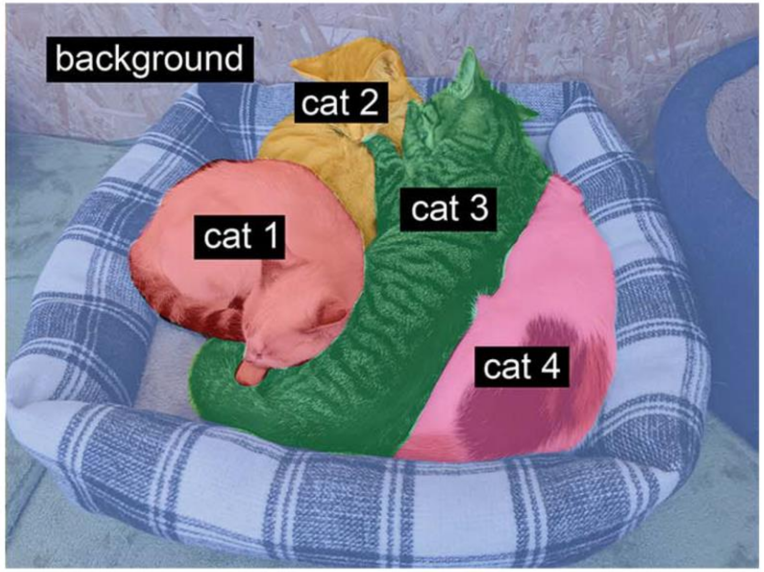
Image segmentation

Object detection

Image Segmentation

- Semantic segmentation, where each pixel is independently classified into a semantic category, like “cat.” If there are two cats in the image, the corresponding pixels are all mapped to the same generic “cat”
- Instance segmentation, which seeks not only to classify image pixels by category, but also to parse out individual object instances. In an image with two cats in it, instance segmentation would treat “cat 1” and “cat 2” as two separate classes of pixels





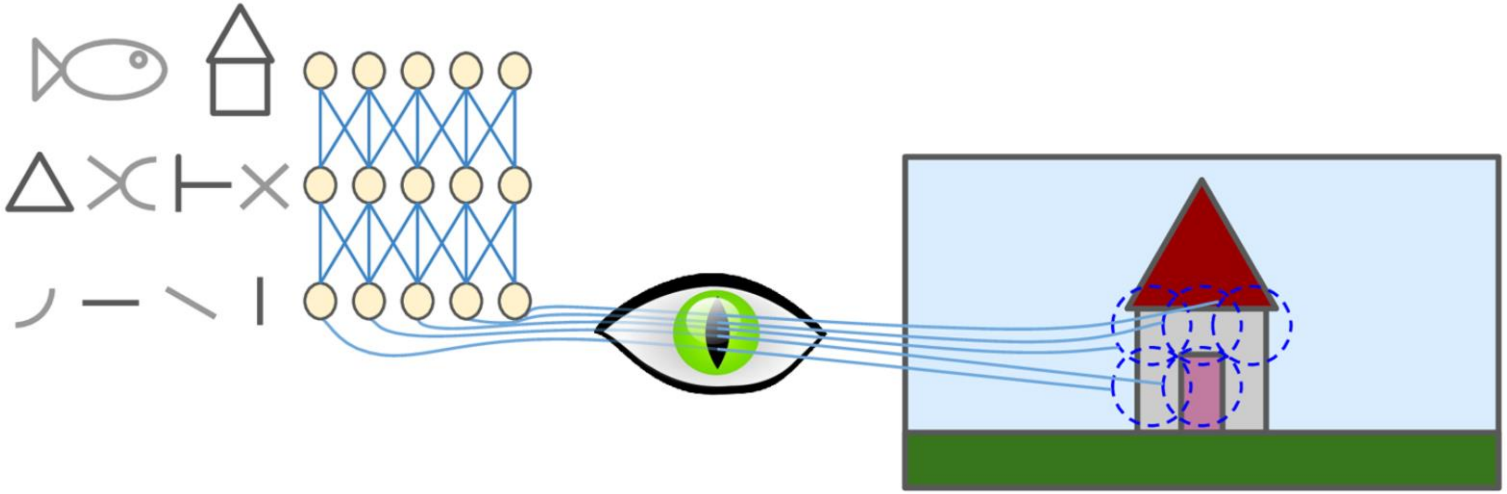
The beginning of the story

- Perception takes place outside consciousness and give us high level features
  - Perception is not trivial at all, and to understand it we must look at how the sensory modules work
  - Recently computers were also able to reliably perform seemingly trivial tasks for human such as detecting a puppy in a picture

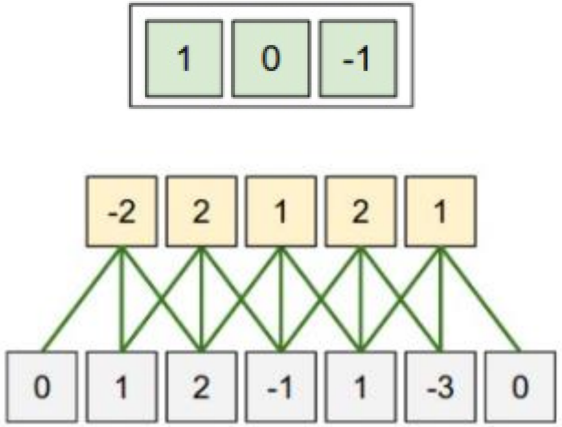
- Convolutional Neural Networks (CNNs) or covnets emerged from the study of the brain's visual cortex, and they have been used in image recognition since the 1980s

The Study of the Visual Cortex

- Studies in 1950s show that neurons in the visual cortex have a small local receptive field
  - Some neurons react only to images of horizontal lines, while others react only to lines with different orientations (the strength may be the same which is *weight sharing*)
  - Some neurons have larger receptive fields, and they react to more complex patterns that are combinations of the lower-level patterns (notice that each neuron is connected only to a few neurons from the previous layer, which is *partially connected*)

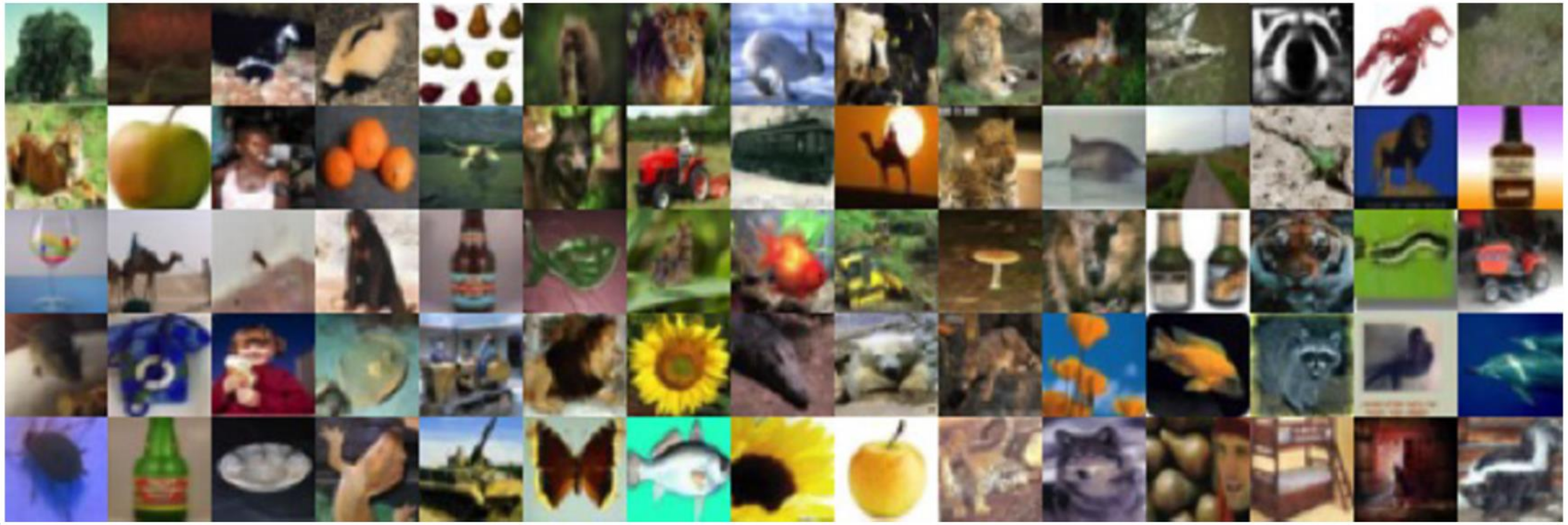


Example of a single filter



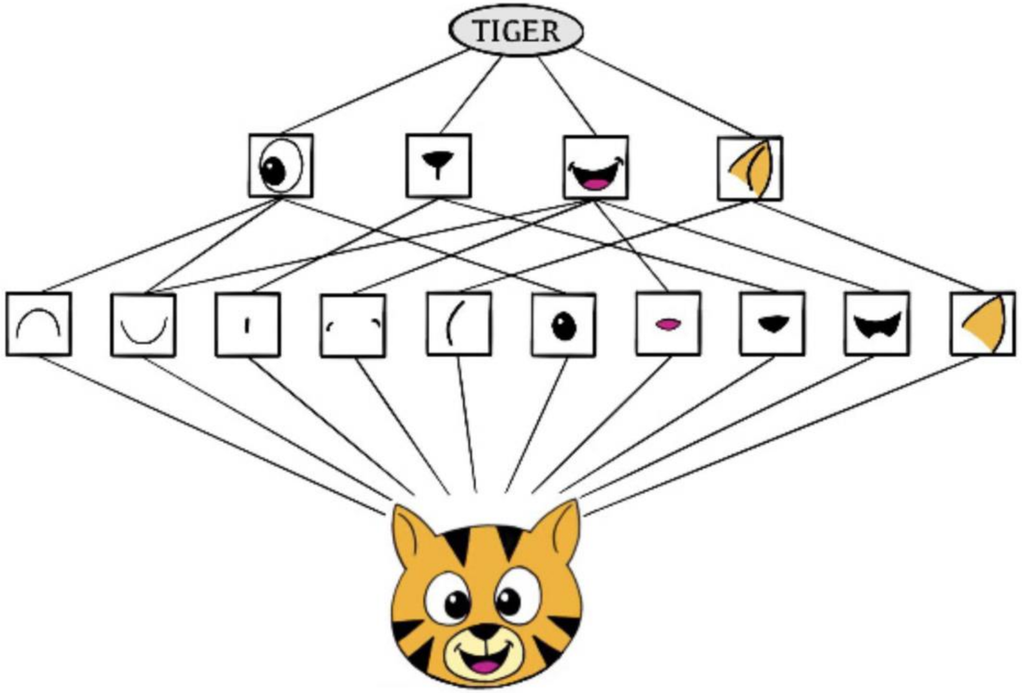
Convolutional Neural Networks

- Neural networks rebounded around 2010 with big successes in image classification
  - Shown are samples from CIFAR100 database. 32×32 color natural images, with 100 classes. 50K training images, 10K test images
  - Each image is a three-dimensional array or *feature map*: $32 \times 32 \times 3$ array of 8-bit numbers. The last dimension represents the three color channels for red, green and blue



How CNNs Work

- The CNN builds up an image in a hierarchical fashion
- Edges and shapes are recognized and pieced together to form more complex shapes, eventually assembling the target image
- This hierarchical construction is achieved using *convolution* and *pooling* layers



1. Convolution operations

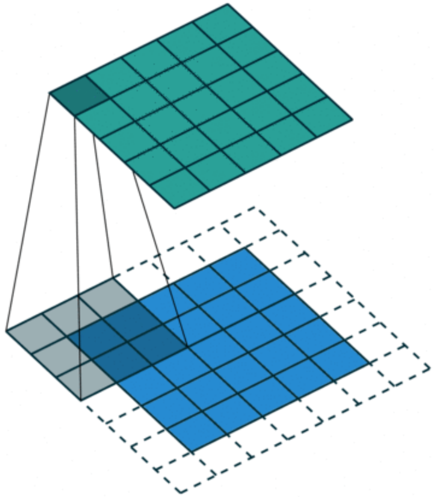
- Input image $= \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix}$ Convolution filter (kernel) $= \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}$

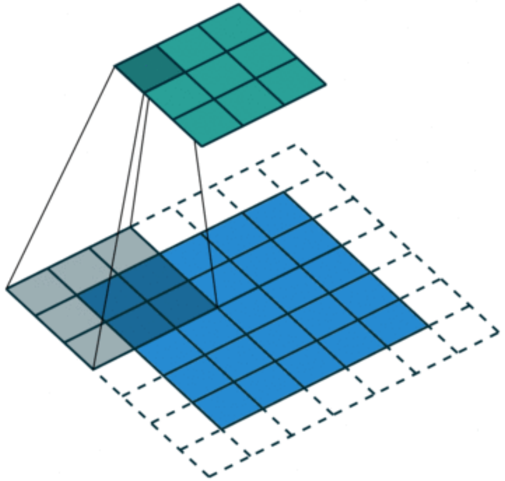
Convoluted image $= \begin{bmatrix} a\alpha + b\beta + d\gamma + e\delta & b\alpha + c\beta + e\gamma + f\delta \\ d\alpha + e\beta + g\gamma + h\delta & e\alpha + f\beta + h\gamma + i\delta \\ g\alpha + h\beta + j\gamma + k\delta & h\alpha + i\beta + k\gamma + l\delta \end{bmatrix}$

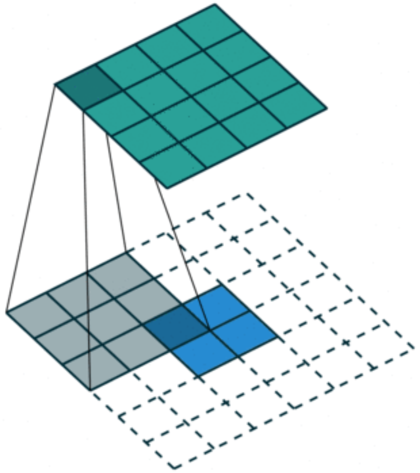
- The filter is itself an image, and represents a small shape, edge etc. We slide it around the input image, scoring for matches
- The scoring is done via *dot-products*, illustrated above. If the subimage of the input image is similar to the filter, the score is high, otherwise low
- The filters are *learned* during training

Convolution Layer

- To have the same height and width as the input image, it is common to add zeros around the inputs which is called *zero padding (Same padding)*
  - If there is no padding applied to the input it is called *valid padding*
- It is also possible to spacing out the receptive fields, the shift from one receptive field to the next is called the *stride*
- *Transposed convolutions (deconvolution)* work by swapping the forward and backward passes of a convolution

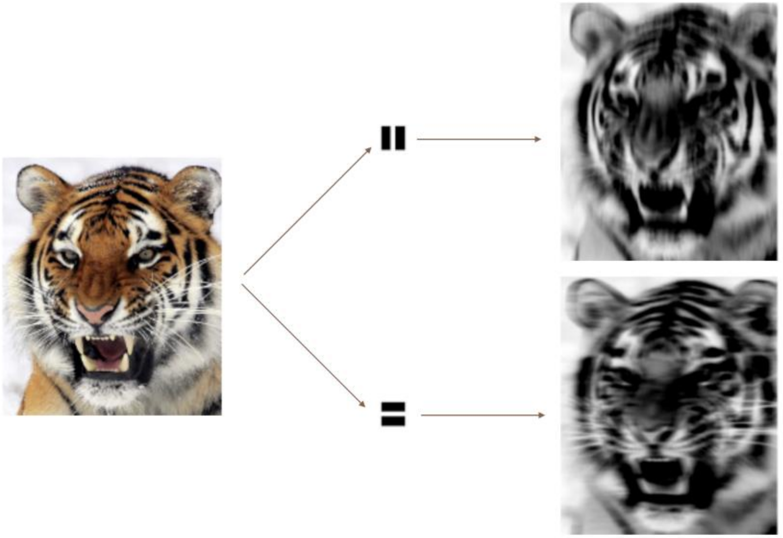






Convolution Example

- The idea of convolution with a filter is to find common patterns that occur in different parts of the image
  - The two filters shown here highlight vertical and horizontal stripes (Note that the neuron's weights can be represented as a small image the size of the receptive field)
  - The result of convolution is a new *feature map or activation maps (units in hidden layer)*
    - Notice the neurons are now locally connected and share the same weight for each feature map
  - Since images have three colors channels, the filter does as well: one filter per channel, and dot-products are summed (integral)

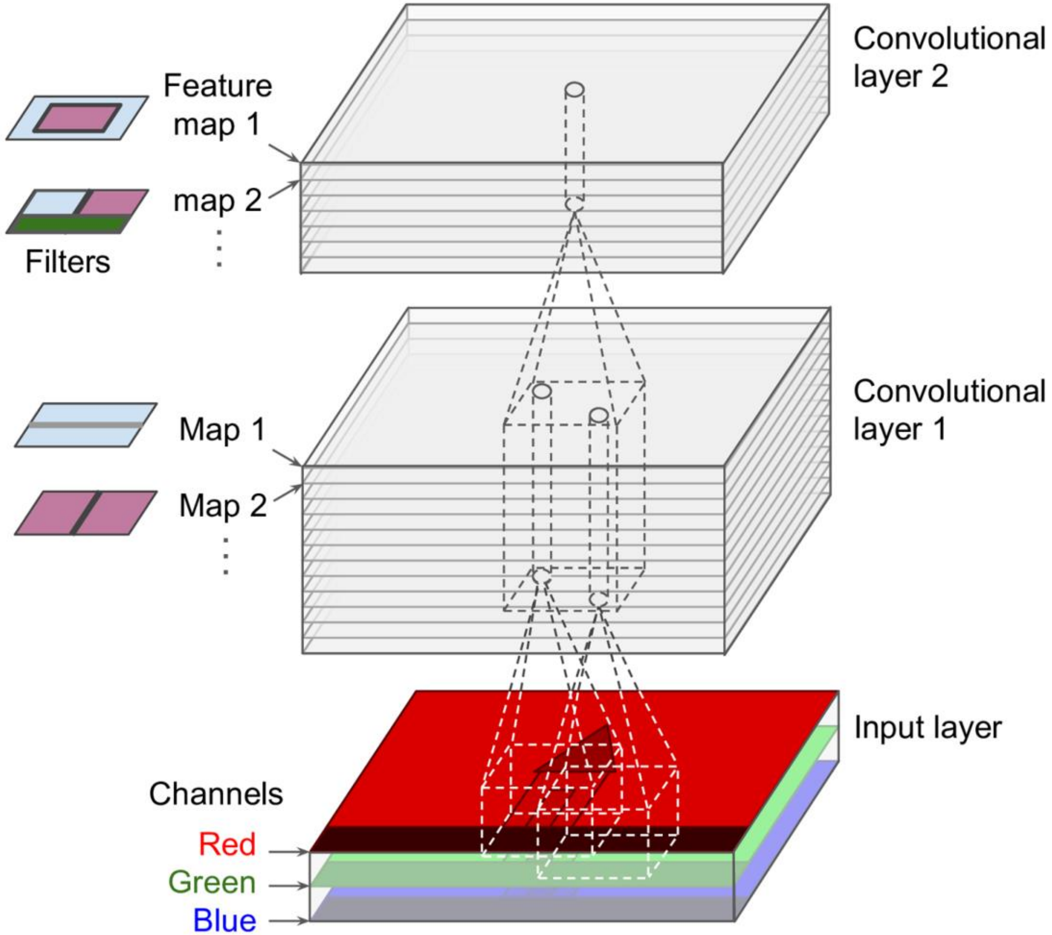


Convolution Example

- In a convolution layer, we use a whole bank of filters to pick out a variety of differently-oriented edges and shapes in the image
- Using predefined filters is standard practice in image processing. By contrast, with CNNs the filters are *learned* for the specific classification task
  - You must defined the size, stride and padding for a given layer
- Filter weights are the parameters going from an input layer to a hidden layer, with one hidden unit for each pixel in the convolved image. The same weights in a given filter are reused for all possible patches in the image
  - All neurons within a given feature map share the same parameters and neurons in different feature maps use different parameters

Stacking Multiple Feature Maps

- If we use K different convolution, we get K two-dimensional output feature maps, which together are treated as a single three-dimensional feature map
  - The third dimension is called the *depth (or channel)*
  - Input images has three channels represented by a three-dimensional feature map. A single convolution filter will also have three channels, one per color, with potentially different filter weights.
  - We typically apply the ReLU activation function to the convolved image

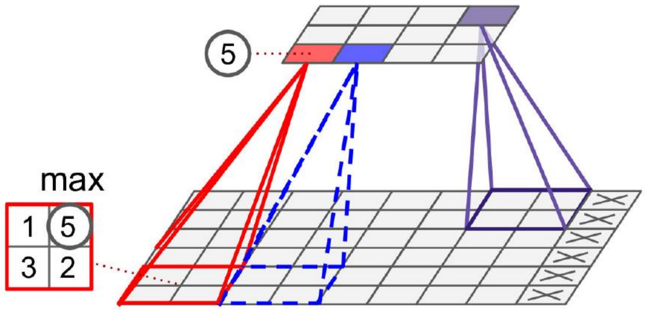


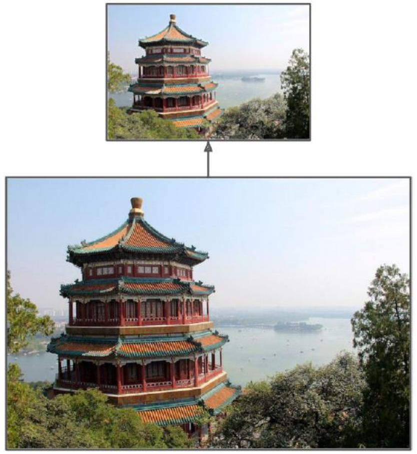
Memory Requirements

- CNNs convolutional layers require a huge amount of RAM. This is especially true during training
  - During inference the RAM occupied by one layer can be released as soon as the next layer has been computed, so you only need as much RAM as required by *two consecutive layers* But during training everything computed during the forward pass needs to be preserved for the reverse pass, so the amount of RAM needed is (at least) the total amount of RAM required by all layers
  - Check appendix for the details about backpropagation
- If training crashes because of an out-of-memory error
  1. Try reducing the mini-batch size
  2. Try reducing dimensionality using a stride, or removing a few layers
  3. Try using 16-bit floats instead of 32-bit floats

2. Pooling Layer

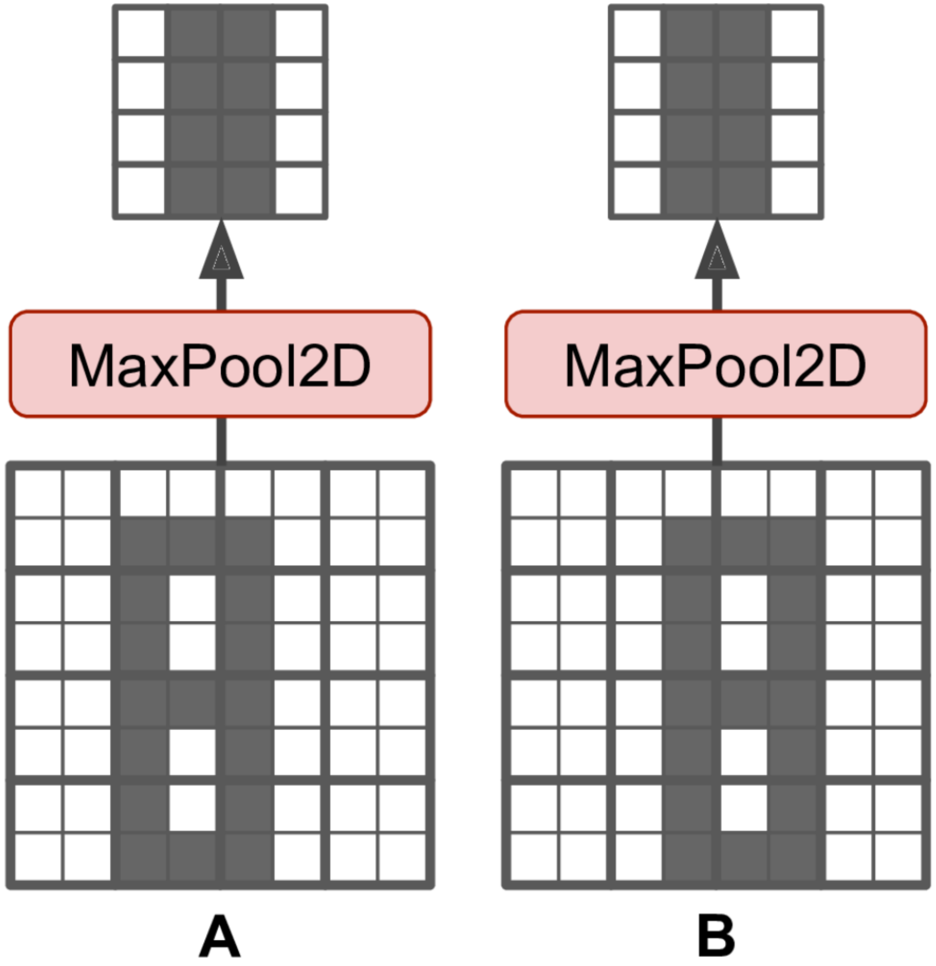
- The goal is to *subsample* the input image in order to reduce the computational load, the memory usage, and the number of parameters
  - Like in convolutional layers, each neuron in a pooling layer is connected to the outputs of a limited number of neurons in the previous layer, located within a small rectangular receptive field. You must define its size, the stride, and the padding type
  - A pooling neuron has no weights
  - A pooling layer typically works on every input channel independently, so the output depth is the same as the input depth





Pooling

- Max pool $\begin{bmatrix} 1 & 2 & 5 & 3 \\ 3 & 0 & 1 & 2 \\ 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 5 \\ 2 & 4 \end{bmatrix}$
- Each non-overlapping 2×2 block is replaced by its maximum
  - This sharpens the feature identification
  - Reduces the dimension by a factor of 4 - i.e. factor of 2 in each dimension
  - Allows for locational invariance
    - Such invariance can be useful in cases where the prediction should not depend on these details, such as in classification tasks

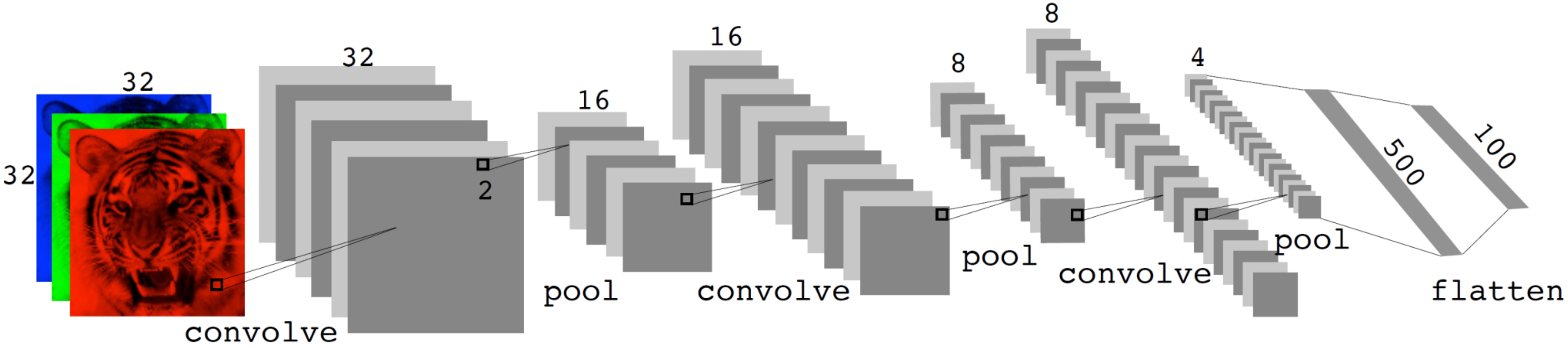


Pooling

- Average pooling is less popular now due to its performance
  - Max pooling preserves only the strongest features, getting rid of all the meaningless ones, so the next layers get a cleaner signal to work with. Moreover, max pooling offers stronger translation invariance than average pooling, and it requires slightly less compute
- Just like dimension reduction, you can also perform it on the depth dimension
  - Allow CNN to learn to be invariant to various features rather than the spatial dimensions
  - Try for example, tensor projection layer instead
- Pooling is very destructive
  - In some applications, invariance is not desirable
  - Take semantic segmentation (the task of classifying each pixel in an image according to the object that pixel belongs to): obviously, if the input image is translated by one pixel to the right, the output should also be translated by one pixel to the right

3. Architecture of a CNN

- Many convolve + pool layers
  - Filters are typically small, e.g. each channel 3 × 3
  - Each filter creates a new channel in convolution layer
  - As pooling reduces size, the number of filters/channels is typically increased
  - At the end, three-dimensional feature maps are *flattened* — the pixels are treated as separate units — and fed into one or more fully-connected layers before reaching the output layer



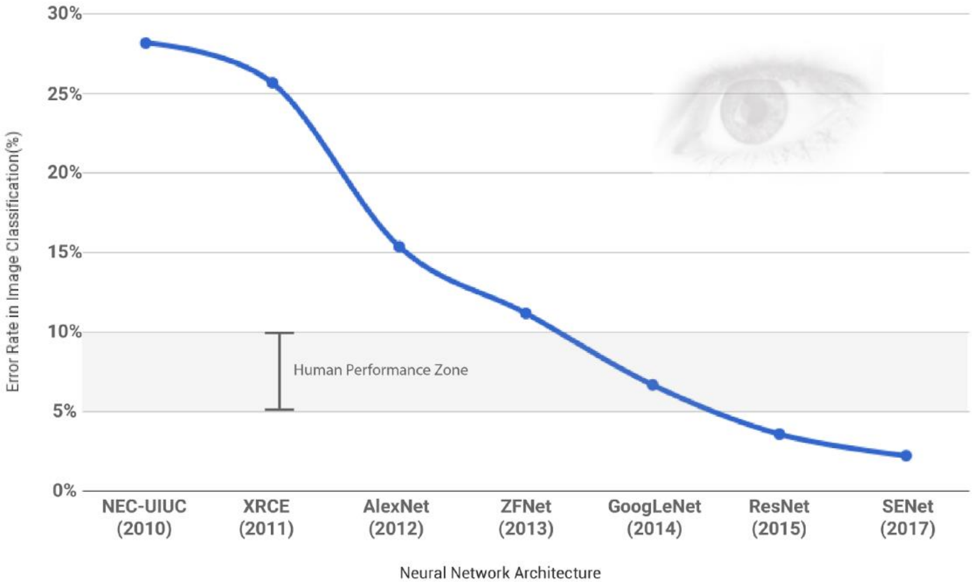
Architecture of a CNN

- The size for the convolution filter
  - A common mistake is to use convolution kernels that are too large
  - For example, instead of using a convolutional layer with a 5 × 5 kernel, stack two layers with 3 × 3 kernels: it will use fewer parameters and require fewer computations, and it will usually perform better
  - One exception is for the first convolutional layer: it can typically have a large kernel, usually with a stride of 2 or more: this will reduce the spatial dimension of the image without losing too much information, and since the input image only has three channels in general, it will not be too costly

Architecture of a CNN

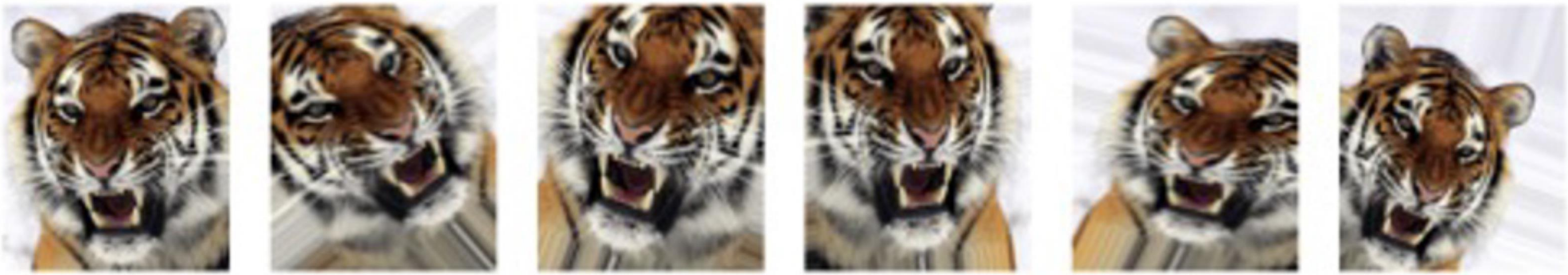
- Over the years, variants of this fundamental architecture have been developed, leading to amazing advances in the field
  - The images in ImageNet are large (256 pixels) and there are 1,000 classes, some of which are really subtle (try distinguishing 120 dog breeds). Looking at the evolution of the winning entries is a good way to understand how CNNs work
  - We can inspect the *covnets* like LeNet-5 architecture (1998), then winners of the ILSVRC challenge: AlexNet (2012), GoogLeNet (2014), ResNet (2015) and SENet (2017)

https://chtseng.wordpress.com/2017/11/20/ilsvrc-%E6%AD%B7%E5%B1%86%E7%9A%84%E6%B7%B1%E5%BA%A6%E5%AD%B8%E7%BF%92%E6%A8%A1%E5%9E%8B/



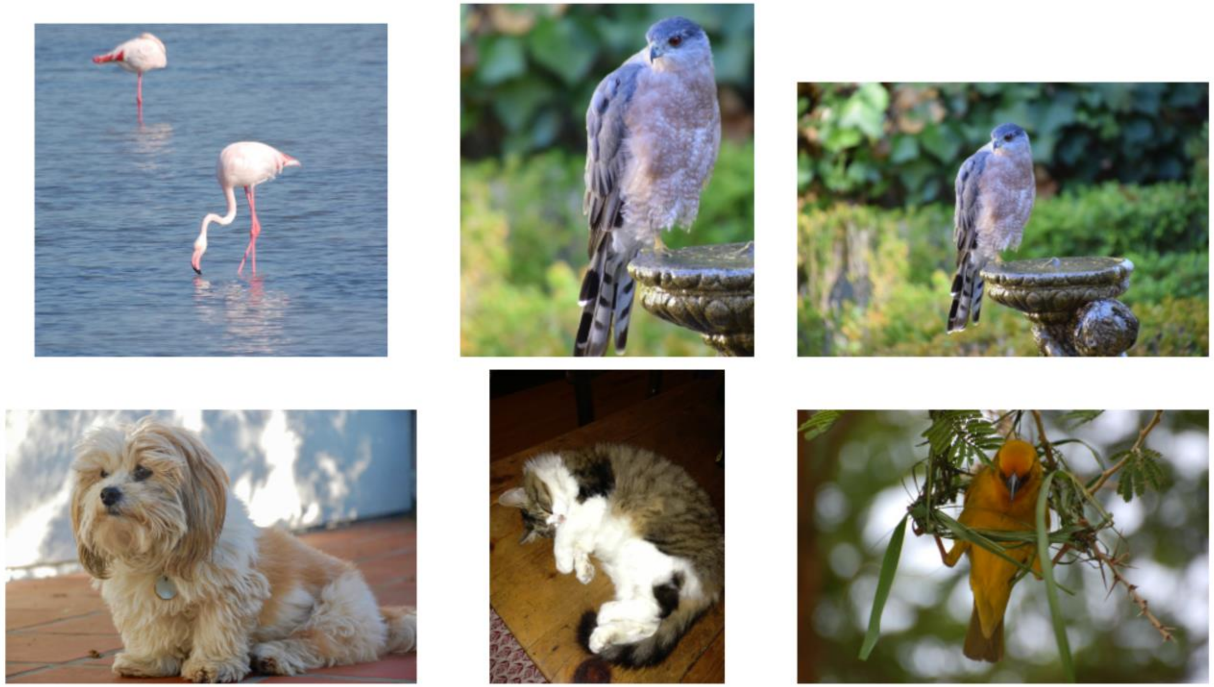
Another regularization method - Data augmentation

- Natural transformations are made of each training image when it is sampled by SGD on-the-fly, thus ultimately making a cloud of images around each original training image
  - The label is left unchanged - in each case still tiger. Improves performance of CNN and is similar to ridge
  - Typical distortions are zoom, horizontal and vertical shift, shear, small rotations, and in this case horizontal flips



Example

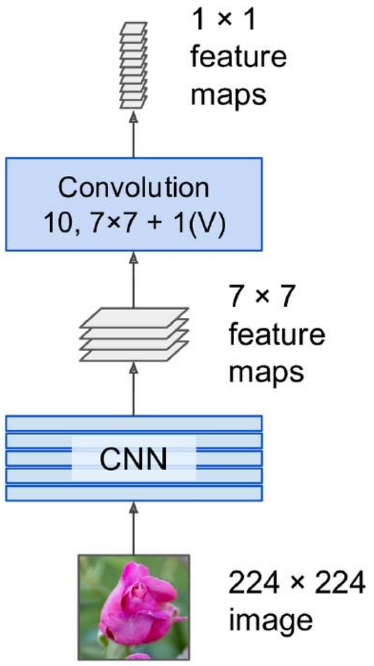
- Here we use the 50-layer resnet50 network trained on the 1000-class imagenet corpus to classify some photographs
- The table below the images displays the true label at the top of each panel, and the top three choices of the classifier. The numbers are the estimated probabilities for each choice. (A kite is a raptor, but not a hawk.)



flamingo		Cooper’s hawk		Cooper’s hawk	
flamingo	0.83	kite	0.60	fountain	0.35
spoonbill	0.17	great grey owl	0.09	nail	0.12
white stork	0.00	robin	0.06	hook	0.07
Lhasa Apso		**cat**		**Cape weaver**	
Tibetan terrier	0.56	Old English sheepdog	0.82	jacamar	0.28
Lhasa	0.32	Shih-Tzu	0.04	macaw	0.12
cocker spaniel	0.03	Persian cat	0.04	robin	0.12

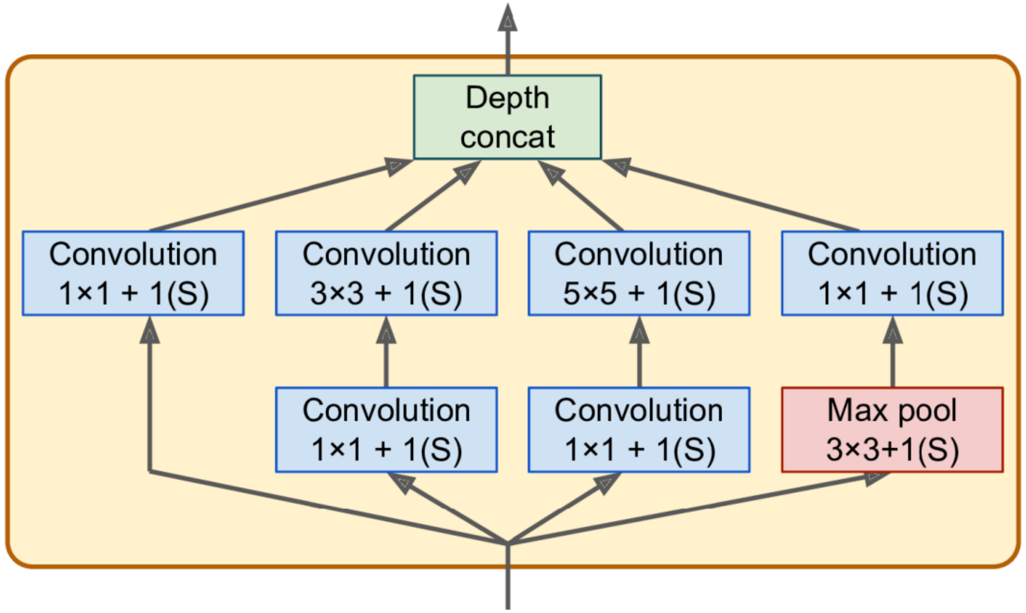
AlexNet - Fully convolution neural network

- AlexNet is the first to stack convolutional layers directly on top of one another, instead of stacking a pooling layer on top of each convolutional layer
  - The idea of Fully Convolutional Networks (FCNs) was then introduced in a 2015 paper by Jonathan Long et al., for semantic segmentation. The authors pointed out that you could replace the dense layers at the top of a CNN by convolutional layers
  - The ideas to replace pooling with convolution also appears:
    - When you do 2 × 2 max pooling, we are completely destroying location information within each pooling window: we return one scalar value per window, with zero knowledge of which of the four locations in the windows the value came from
    - So while max pooling layers perform well for classification tasks, they would hurt us quite a bit for a segmentation task. Meanwhile, strided convolutions do a better job at downsampling feature maps while retaining location information



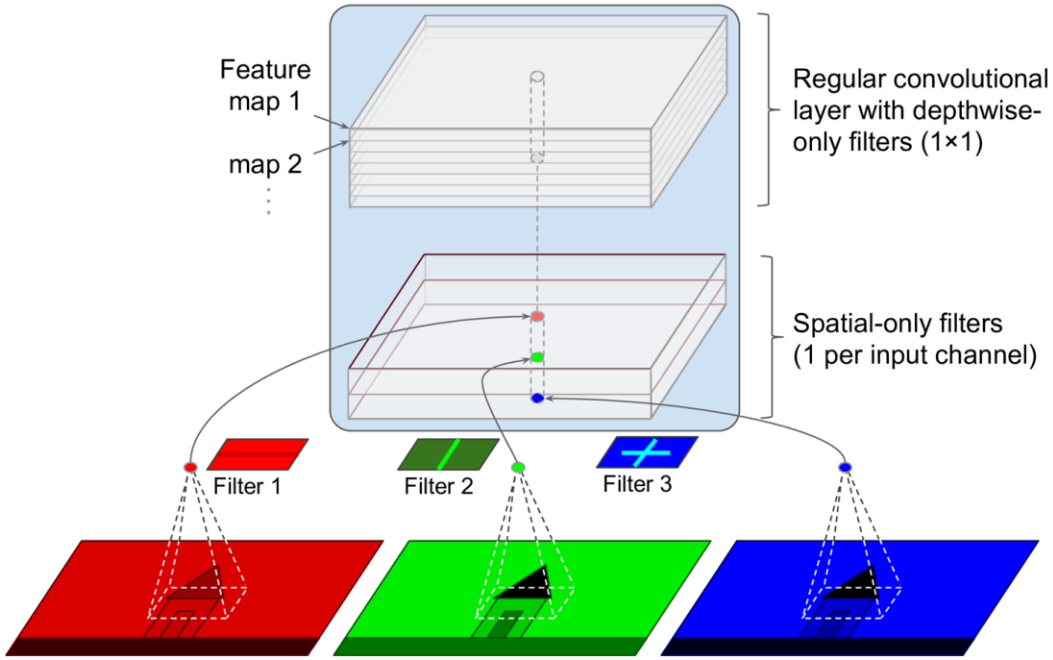
GoogleLeNet - Inception module

- "$3 \times 3 + 1$(S)" means it uses a 3×3 kernel, stride 1, and "same" padding
  - The input signal is first copied and fed to four different layers. The second set of convolutional layers uses different kernel sizes (1×1, 3×3, and 5×5), allowing them to capture patterns at different scales. Although 1×1 kernel cannot capture spatial patterns, they can capture patterns along the depth dimension
  - Concatenate all the outputs along the depth dimension in the *depth concatenation layer*
  - The number of convolutional kernels for each convolutional layer is a hyperparameter



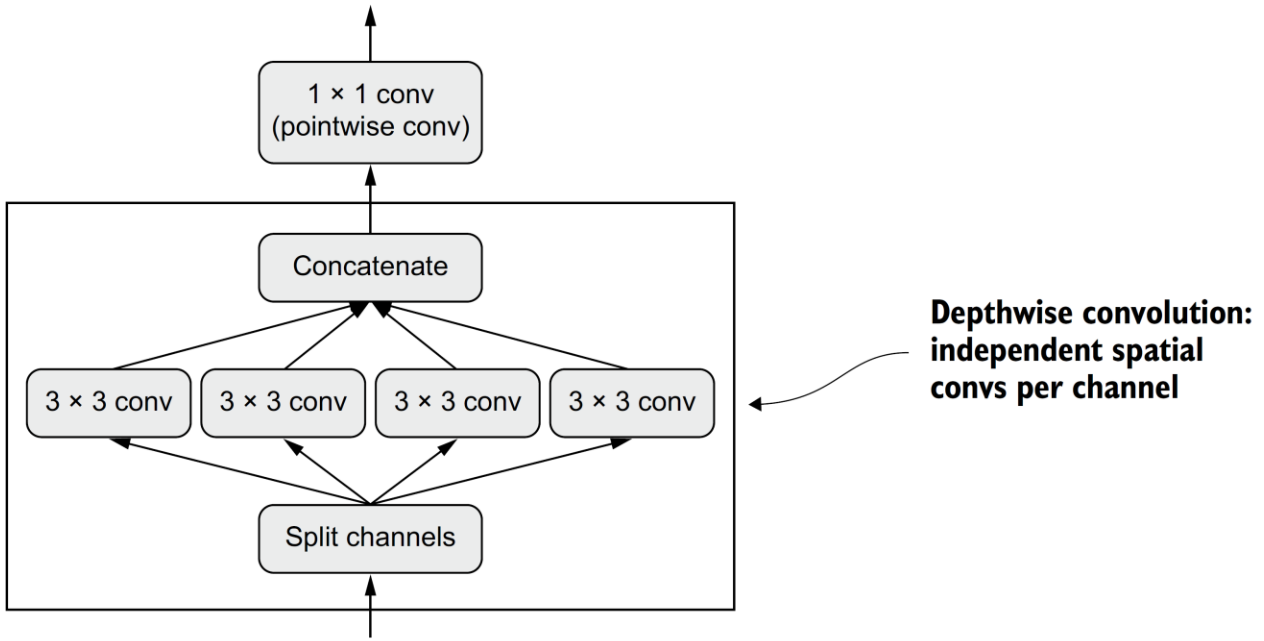
Xception - Depthwise separable convolutions

- Replaces the inception modules with a special type of layer called a *depthwise separable convolution layer* (or *separable convolution layer* for short)
  - While a regular convolutional layer uses filters that try to simultaneously capture spatial patterns (e.g., an oval) and crosschannel patterns (e.g., mouth + nose + eyes = face), a separable convolutional layer makes the strong assumption that spatial patterns and cross-channel patterns can be modeled separately
  - Thus, it is composed of two parts: the first part applies *a single spatial filter for each input feature map*, then the second part looks exclusively for cross-channel patterns—it is just a regular convolutional layer with 1 × 1 filters

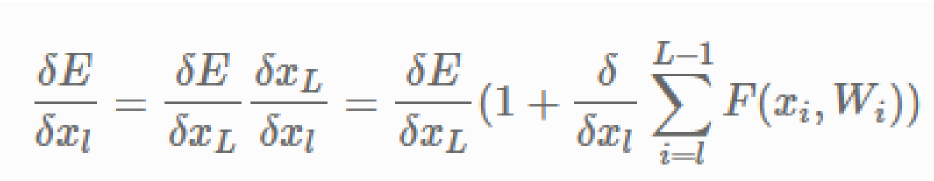

Feature map 1
map 2
Regular convolutional layer with depthwise-only filters (1×1)
Spatial-only filters (1 per input channel)
Filter 1
Filter 2
Filter 3

Xception - Depthwise separable convolutions

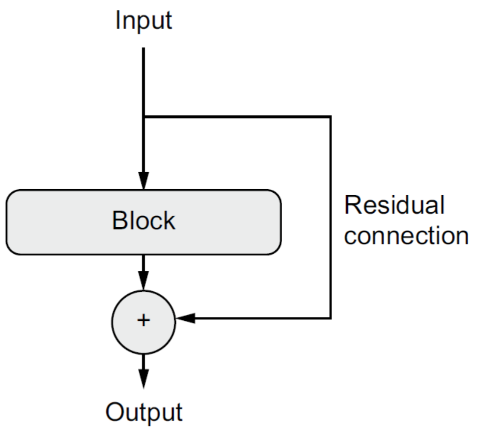
- SeparableConv2D in Keras that can be a drop-in replacement for Conv2D
  - It relies on the assumption that spatial locations in intermediate activations are highly correlated, but different channels are highly independent. Because this assumption is generally true for the image representations learned by deep neural networks, it serves as a useful prior that helps the model make more efficient use of its training data



ResNet - Residual learning

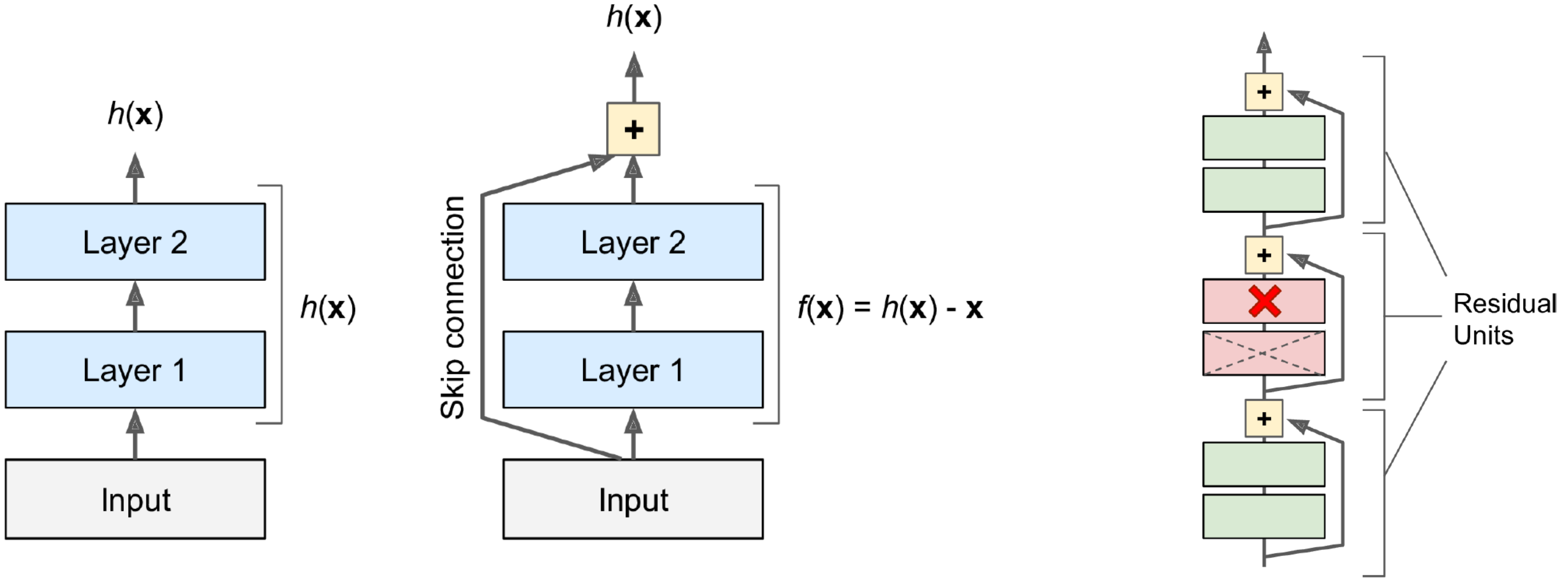
- Solve the vanishing gradient problem using *skip (residual) connection*
  - The connection acts as an information shortcut around destructive or noisy blocks (such as blocks that contain relu activations or dropout layers), enabling error gradient information from higher layers to propagate noiselessly through a deep network
  - Forward (l to $l+1$ layer) (In contrast to $x_L = \prod_{i=l}^{L-1} W_i \, x_l$)
    - $x_{l+1} = x_l + F(x_l, W_l)$, $x_L = x_l + \sum_{i=1}^{L-1} F(x_i, W_i)$
  - Backward (In contrast to $\frac{\partial E}{\partial x_l} = \prod_{i=l}^{L-1} W_i \frac{\partial E}{\partial x_L}$)
    - 

$$\frac{\delta E}{\delta x_l} = \frac{\delta E}{\delta x_L}\frac{\delta x_L}{\delta x_l} = \frac{\delta E}{\delta x_L}(1 + \frac{\delta}{\delta x_l}\sum_{i=l}^{L-1} F(x_i, W_i))$$



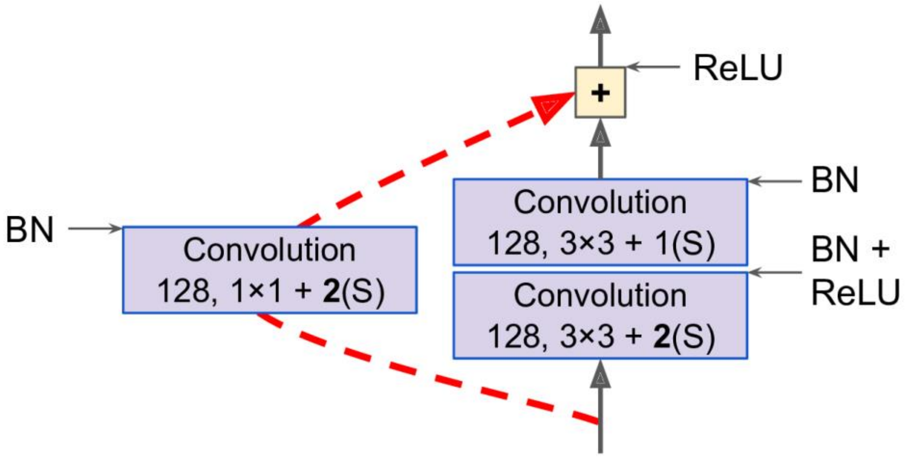
ResNet - Residual learning

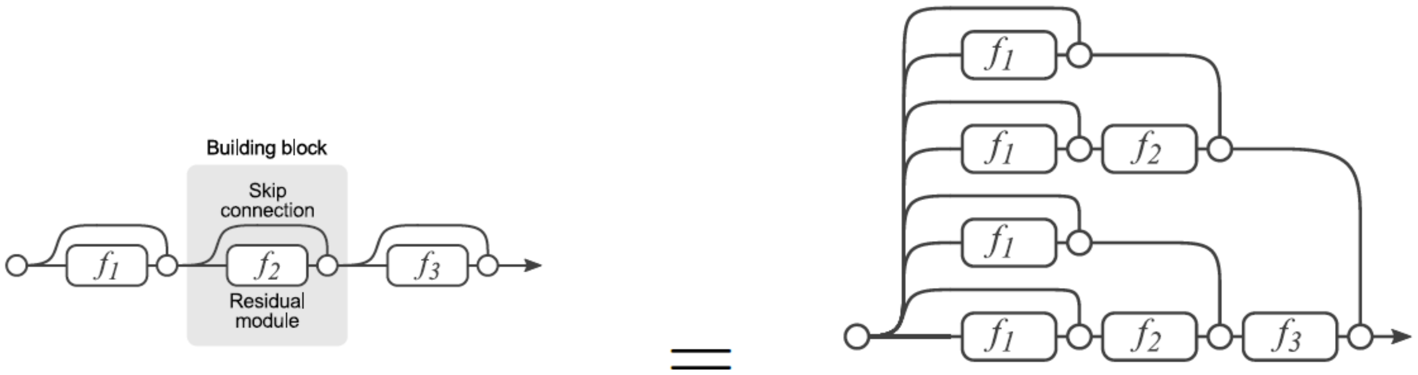
- Residual $f(x)$ is easier to learn using *skip connection*
  - If the target function is fairly close to the identity function (which is often the case), this will speed up training considerably
  - The network can start making progress even if several layers have not started learning yet



ResNet - Residual learning

- When the size is different and can not be added just use 1 × 1 convolution with strides
  - It is noted that some networks use concatenation instead of summation like DenseNet
- Behaves like ensemble





Other popular architectures for image classifications

- https://keras.io/api/applications/
- VGG
- DenseNet
- SENet
- EfficientNet
- MobileNet

Conclusion

- Model should be organized into repeated blocks of layers, usually made of multiple convolution layers and a max pooling layer
  - The number of filters in your layers should increase as the size of the spatial feature maps decreases
  - Deep and narrow is better than broad and shallow
  - Introducing residual connections around blocks of layers helps you train deeper networks
- Deep learning for computer vision also encompasses a number of somewhat more niche tasks besides these three, such as image similarity scoring (estimating how visually similar two images are), keypoint detection (pinpointing attributes of interest in an image, such as facial features), pose estimation, 3D mesh estimation, and so on

References

[1] Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition Chapter 14

[2] An Introduction to Statistical Learning with Applications in R. Second Edition Chapter 10

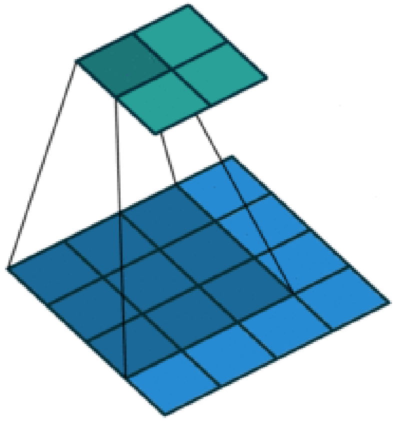
[3] Deep learning with Python, 2nd Edition Chapter 8~9

Appendix

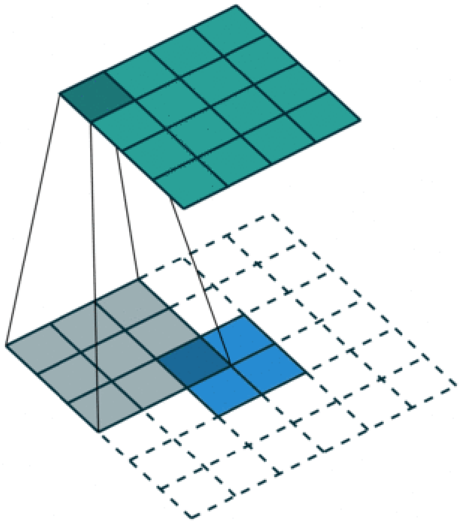
Convolution and transposed convolution

- Convolution:
  - The size of kernel matrix is 3 × 3, stride is 1, no zero padding
  - $C\colon R^{16} \rightarrow R^{4}$

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



- Transposed convolution:
  - $C^{T}\colon R^{4} \rightarrow R^{16}$



Resources

- Understand CNN
  - https://setosa.io/ev/image-kernels/
  - https://github.com/generic-github-user/Image-Convolution-Playground
  - https://poloclub.github.io/cnn-explainer/
- Backpropagation for convolution and pooling layers
  - Convolution layer
  - Max pooling
- Tutorials
  - https://d2l.ai/chapter_convolutional-neural-networks/index.html
- Loading images
  - https://ffcv.io/

AlexNet

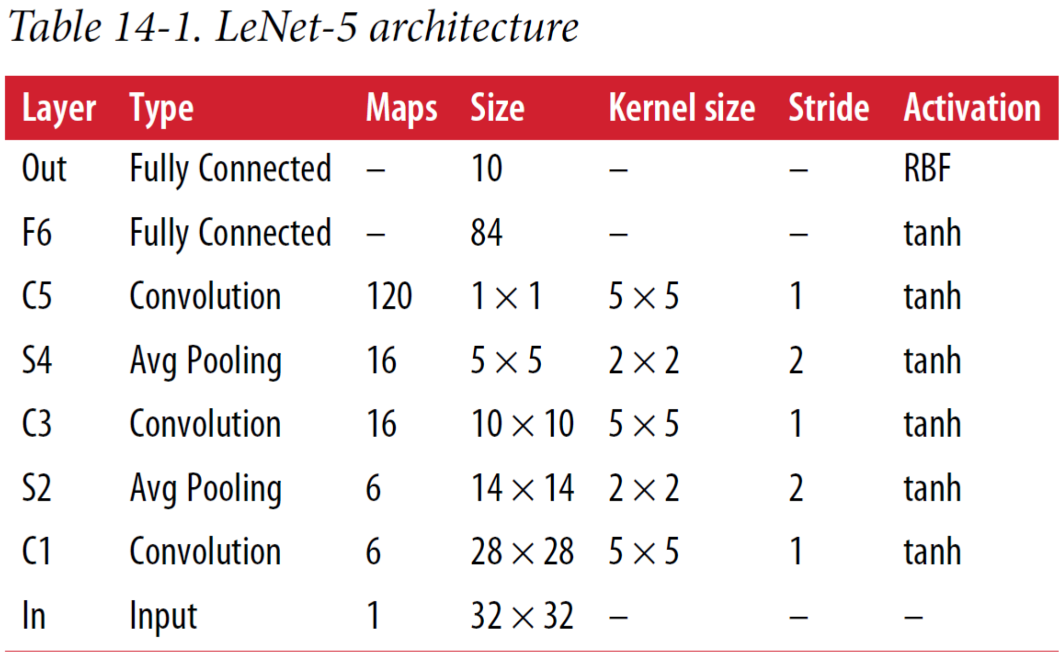


Table 14-1. LeNet-5 architecture

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	–	10	–	–	RBF
F6	Fully Connected	–	84	–	–	tanh
C5	Convolution	120	1 × 1	5 × 5	1	tanh
S4	Avg Pooling	16	5 × 5	2 × 2	2	tanh
C3	Convolution	16	10 × 10	5 × 5	1	tanh
S2	Avg Pooling	6	14 × 14	2 × 2	2	tanh
C1	Convolution	6	28 × 28	5 × 5	1	tanh
In	Input	1	32 × 32	–	–	–

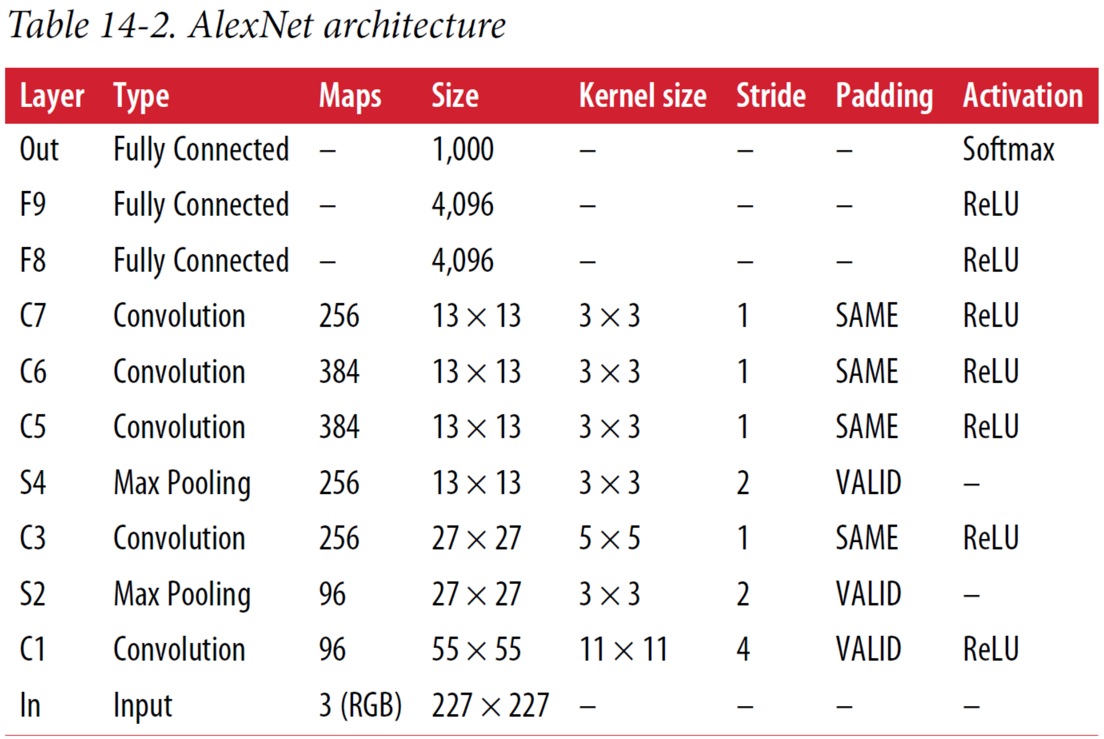
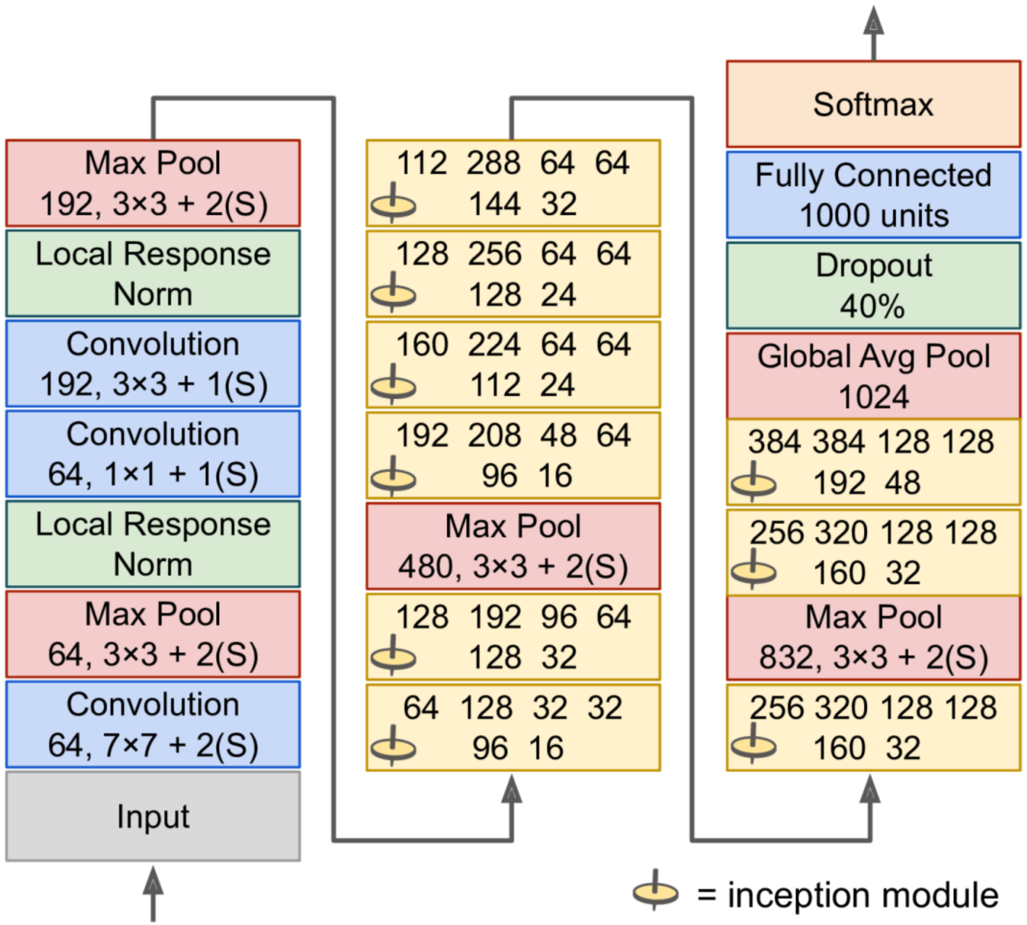
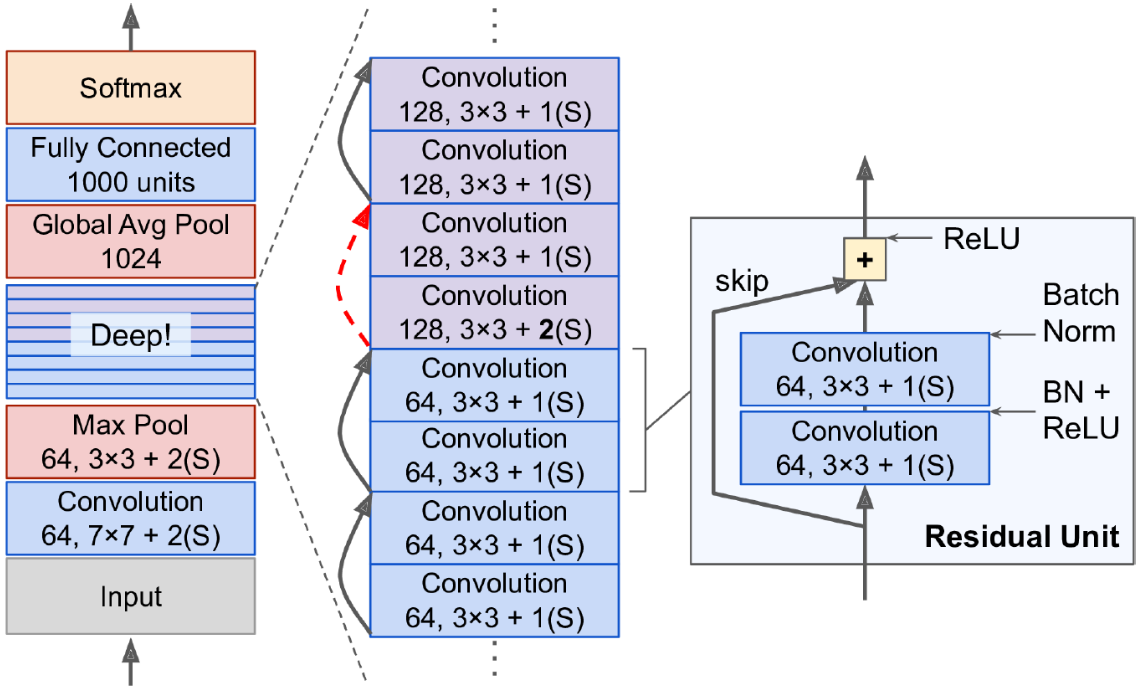


Table 14-2. AlexNet architecture

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	–	1,000	–	–	–	Softmax
F9	Fully Connected	–	4,096	–	–	–	ReLU
F8	Fully Connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	SAME	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
S4	Max Pooling	256	13 × 13	3 × 3	2	VALID	–
C3	Convolution	256	27 × 27	5 × 5	1	SAME	ReLU
S2	Max Pooling	96	27 × 27	3 × 3	2	VALID	–
C1	Convolution	96	55 × 55	11 × 11	4	VALID	ReLU
In	Input	3 (RGB)	227 × 227	–	–	–	–

GoogLeNet and ResNet





Popular architectures for object detection

- YOLO
- YOLOX
- SSD
- Faster-RCNN
- RetinaNet

Popular architectures for image segmentation

- https://github.com/divamgupta/image-segmentation-keras
  - SegNet
  - FCN
  - Unet
  - PSPNet
- https://github.com/google-research/deeplab2
  - DeepLab