Image processing with Convolutional Neural Networks

Szu-Chi Chung

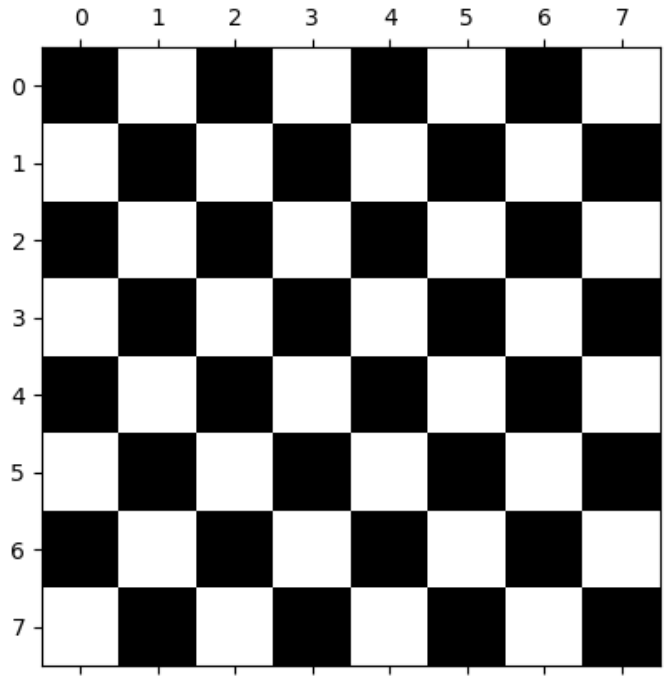
Department of Applied Mathematics, National Sun Yat-sen University

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 (scipy, skimage, opencv)

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

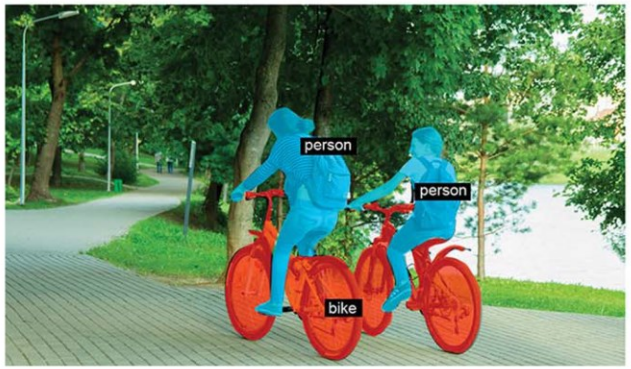
Single-label multi-class classification

- (•) Biking
- ( ) Running
- ( ) Swimming

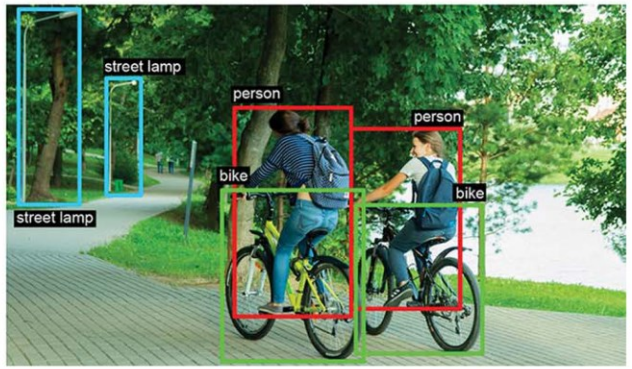
Multi-label classification

- [x] Bike
- [x] Person
- [ ] Boat
- [x] Tree
- [ ] Car
- [ ] House

Image segmentation

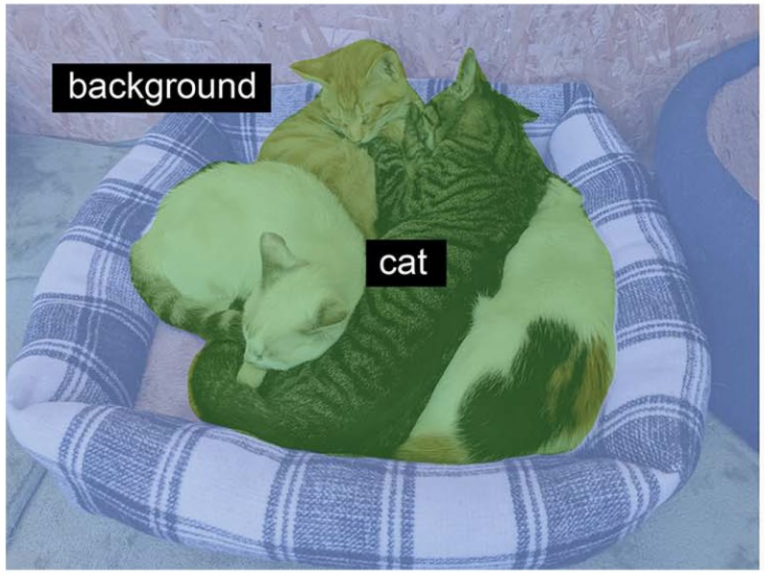


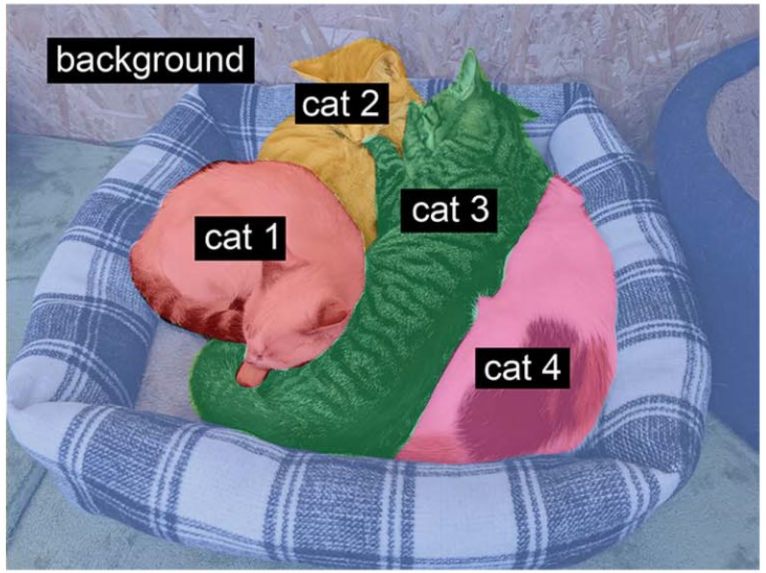
Object detection



Three essential computer vision tasks - 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





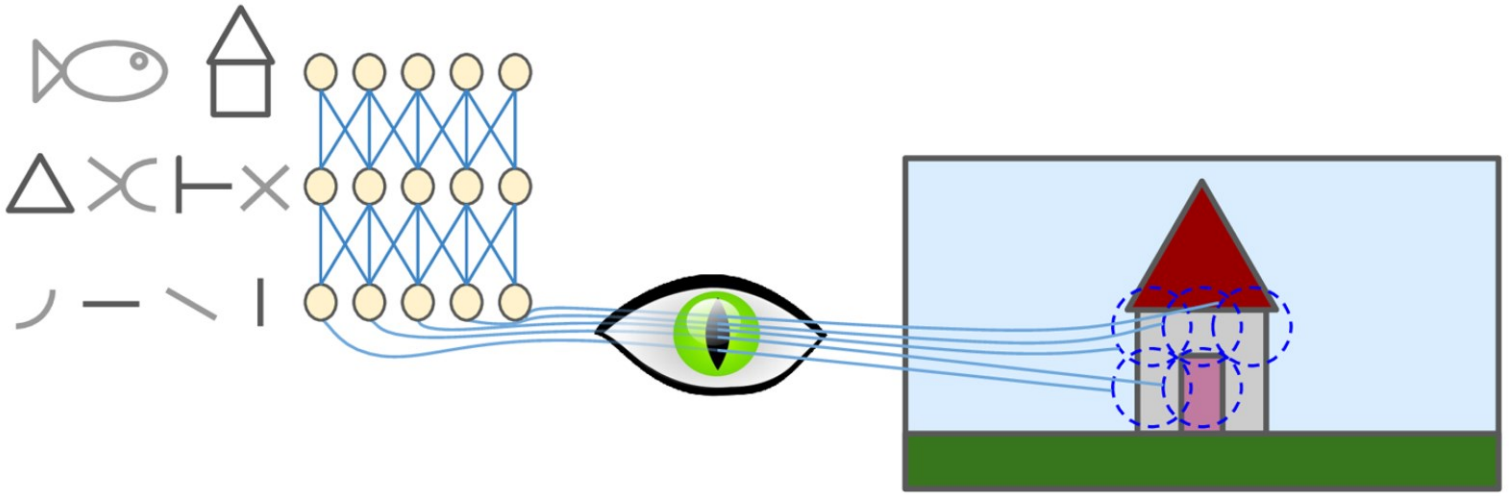
The beginning of the story

- Visual perception takes place outside consciousness and gives us high-level features
  - Visual 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 humans such as detecting a puppy in a picture

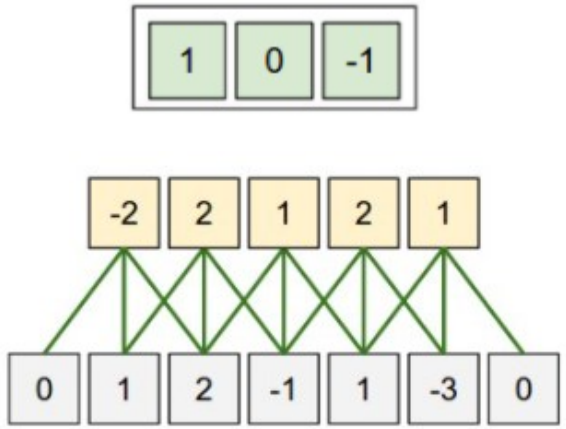
- *Convolutional Neural Networks* (CNNs) or *covnets* emerged from the study of the brain's visual cortex, and they have been used in image recognition since the 1980s

The Study of the Visual Cortex

- Studies in the 1950s show that neurons in the visual cortex have a small local *receptive field*
  - 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 called *partially connected*)
  - 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 called *weight sharing*)

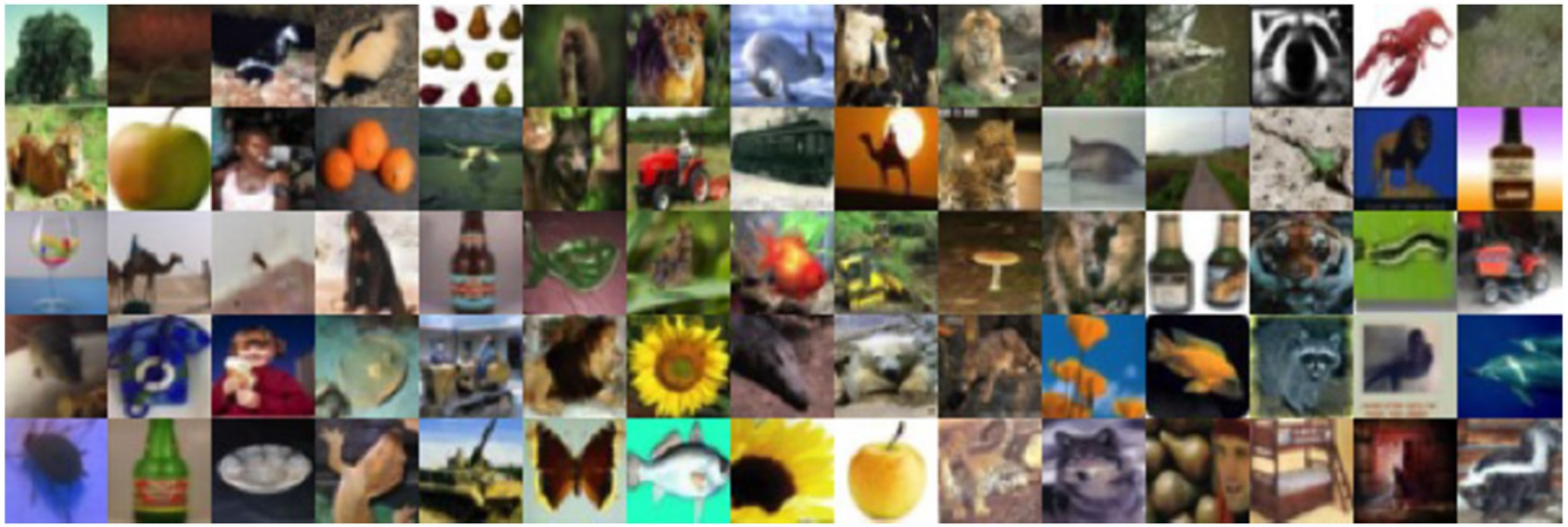


Example of a single filter



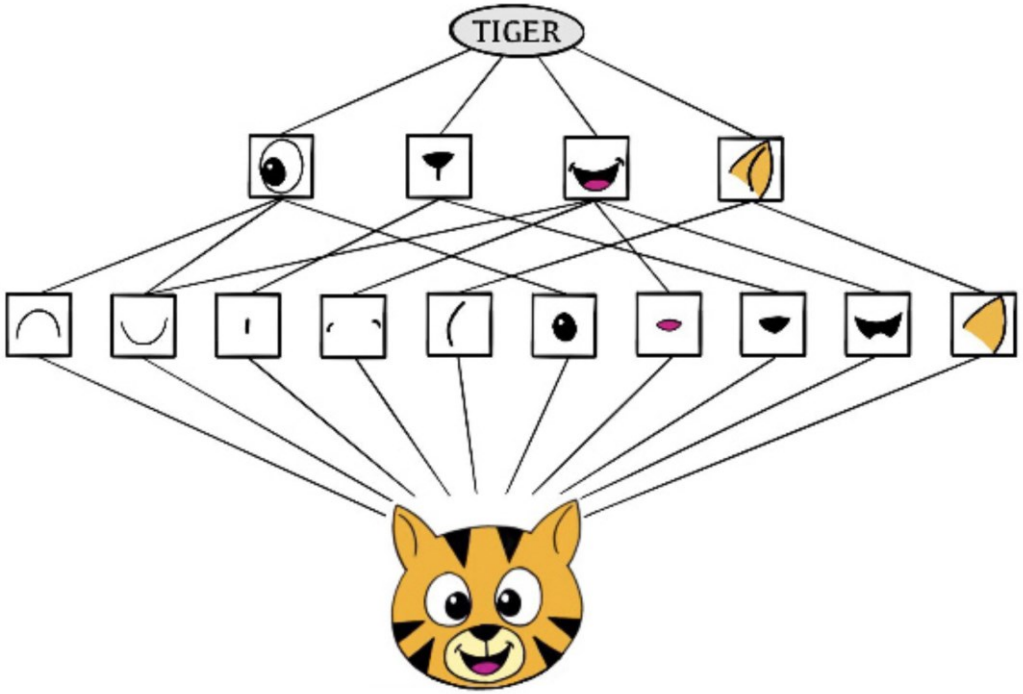
Convolutional Neural Networks in image classification

- 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

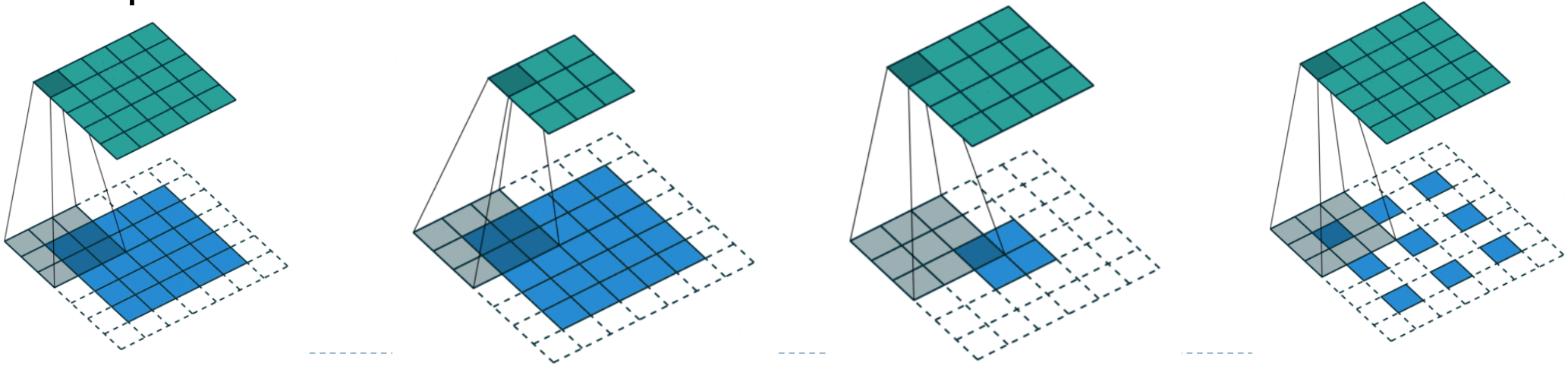
- Input image $= \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix}$ Convolution filter (kernel) $= \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}$

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

1. The filter itself can be represented as an image that represents a small shape, edge etc. We slide it around the input image, scoring for matches
2. 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
3. The filters are *learned* during training

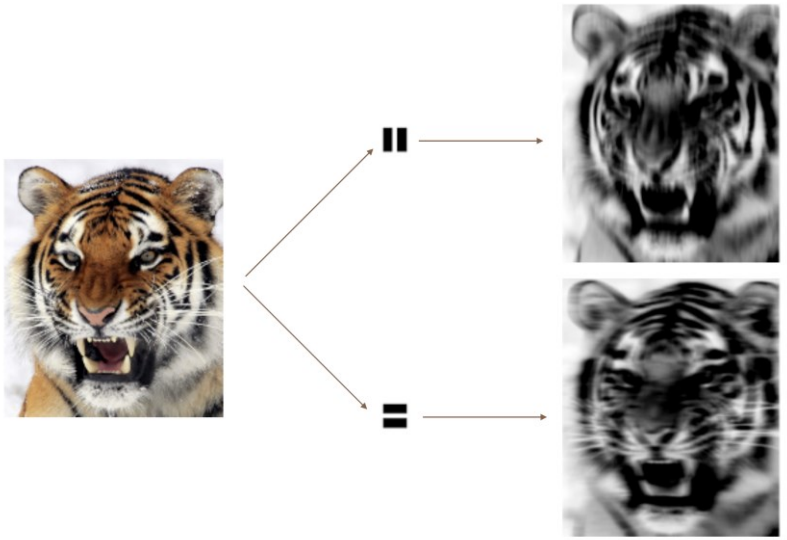
Convolution Layer

- 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 space 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 – locally connected

- 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 that has the size of the receptive field)
  - The result of the convolution is a new *feature map (units in the hidden layer)*
    - Notice the neurons are now locally connected and share the same weight for each feature map

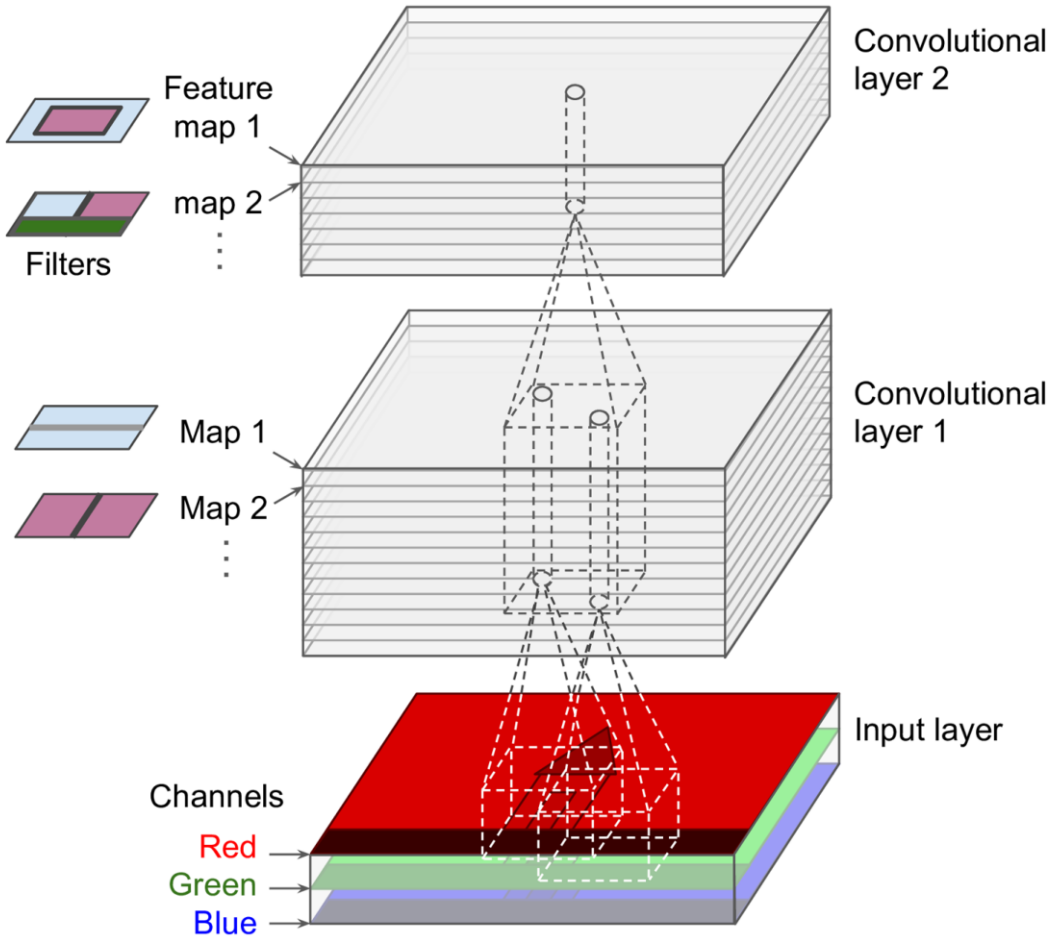


Convolution Example – weight sharing

- 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, and dot-products are summed (integral)
  - We typically apply the ReLU activation function to the convolved image and higher layers will have larger receptive fields

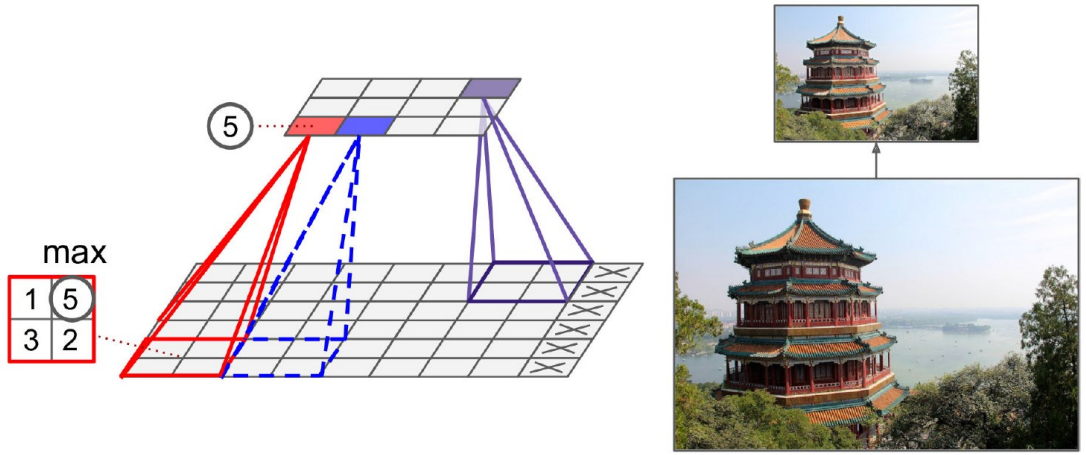


Memory Requirements

- 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

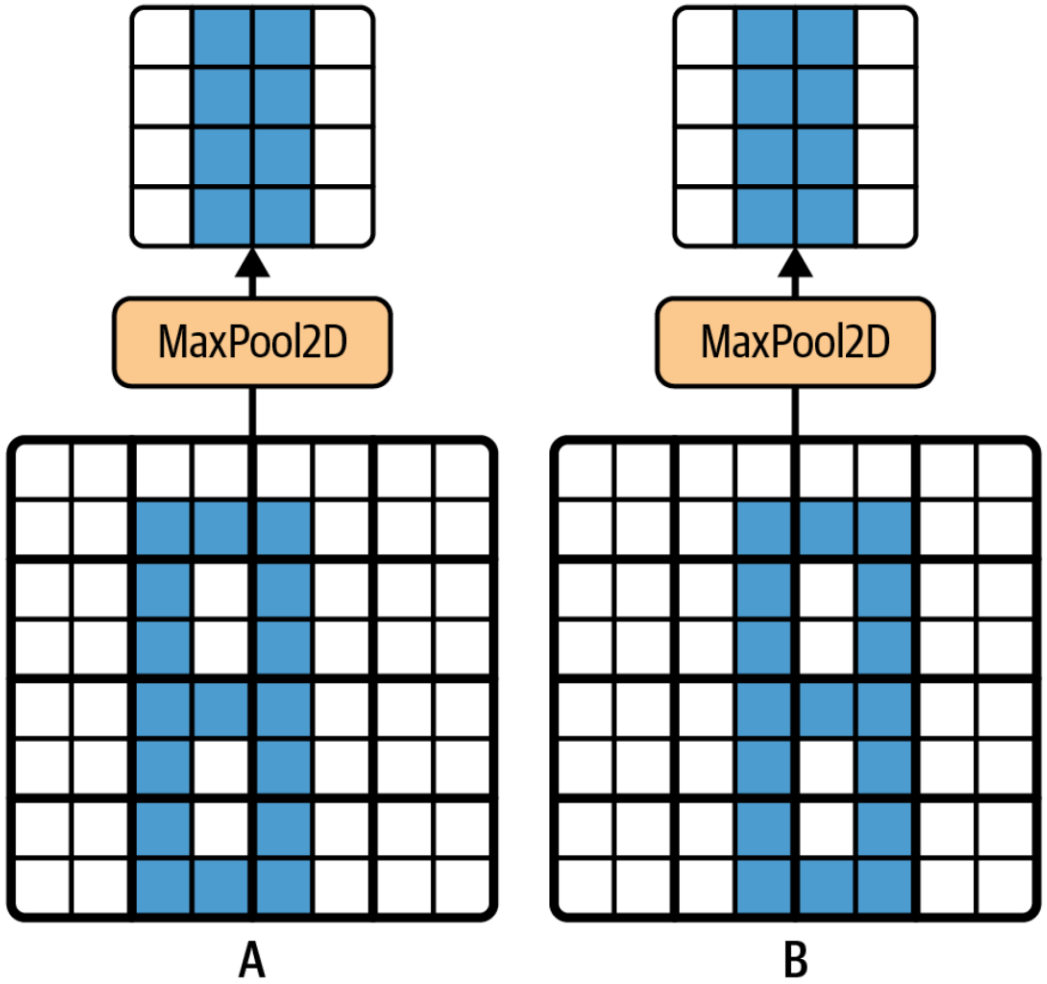
2. Pooling Layer

- The goal is to *subsample* the input image in order to reduce the computational load, the memory usage, and the number of parameters
  1. 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, stride, and the padding type
  2. A pooling neuron has no weights
  3. 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

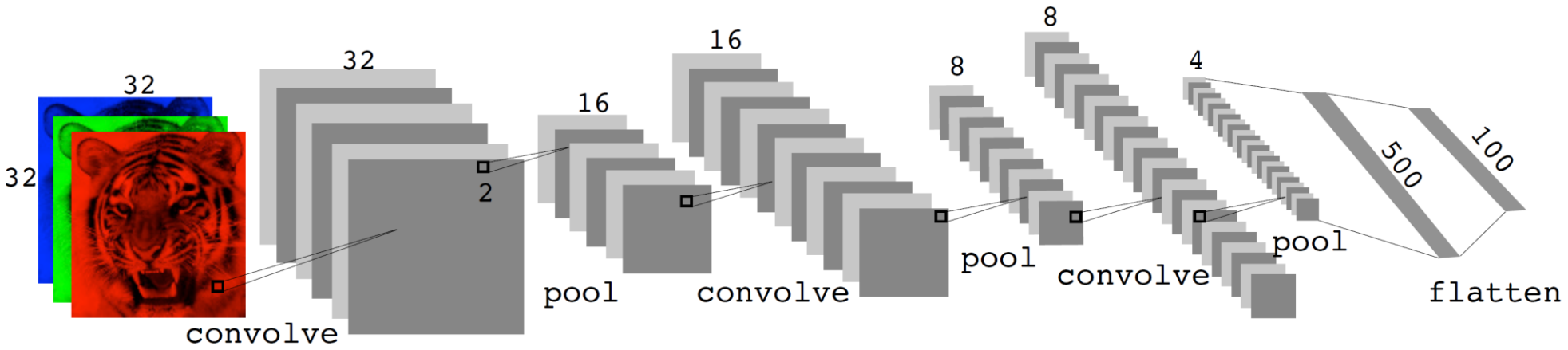


Pooling

- *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

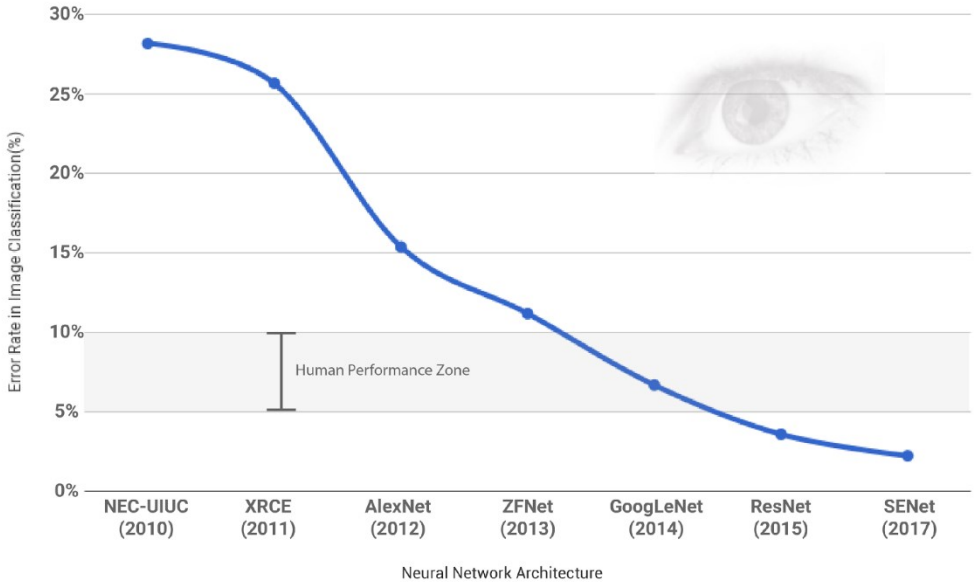
- Many convolve + pool layers
  - Filters are typically small, e.g. each channel 3×3
  - Each filter creates a new channel in the convolution layer
  - As pooling reduces size, the number of filters/channels is typically increased
  - In the end, three-dimensional feature maps are *flattened* - the pixels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



Architecture of a CNN

- 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/



4. Regularization method - Data augmentation (Rule-based)

- 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 a tiger
  - Typical distortions are zoom, horizontal and vertical shift, shear, small rotations, and in this case, horizontal flips

Regularization method - Mixup

- Mixup is a powerful data augmentation technique
  - It's helpful to have techniques that "dial-up/down" the amount of change, to see what works best for you
  1. Select another image from your dataset at random
  2. Pick a weight at random
  3. Take a weighted average (using the weight from step 2) of the selected image with your image; this will be your independent variable
  4. Take a weighted average (with the same weight) of this image's labels (one-hot encoded) with your image's labels; this will be your dependent variable



Another regularization method – Label smoothing

- In the theoretical expression of loss, in classification problems, our targets are one-hot encoded
  - This can become very harmful if your data is not perfectly labeled
  - Instead, we could replace all our 1s with a number a bit less than 1, and our 0s with a number a bit more than 0, and then train. This is called *label smoothing*
  - By encouraging your model to be less confident, label smoothing will make your training more robust, even if there is mislabeled data!
  - For example that has 10 classes, the targets become something like this ($1 - \frac{\epsilon}{N}$ and $\frac{\epsilon}{N}$):

$$[0.01, 0.01, 0.01, 0.91, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01]$$

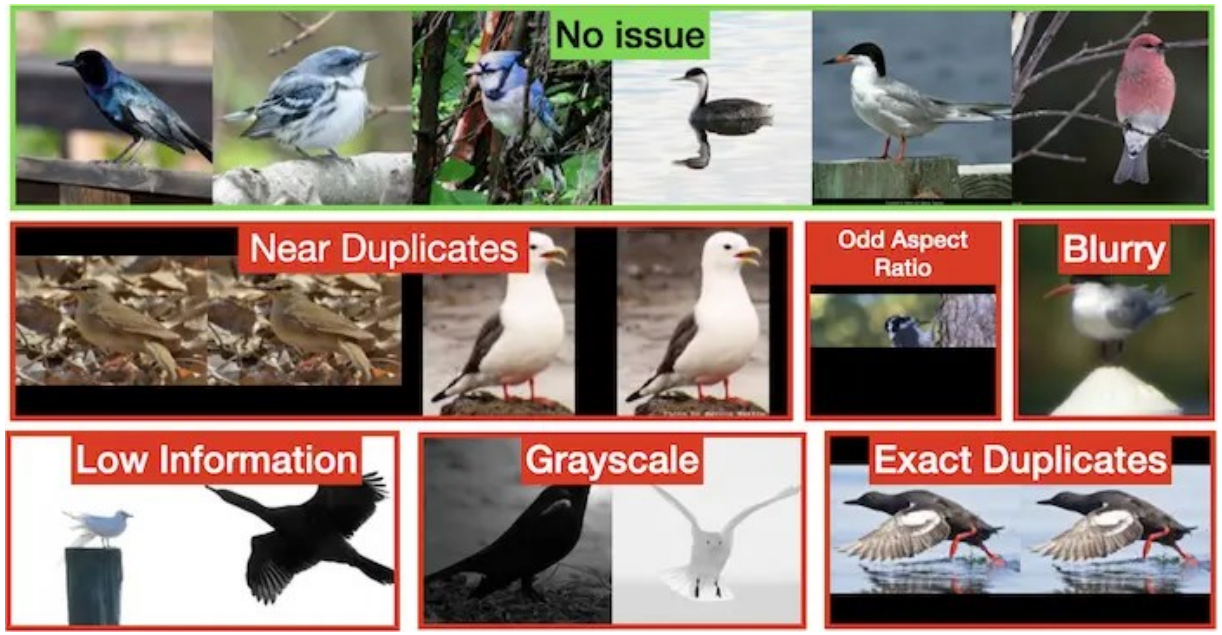
Example

- 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

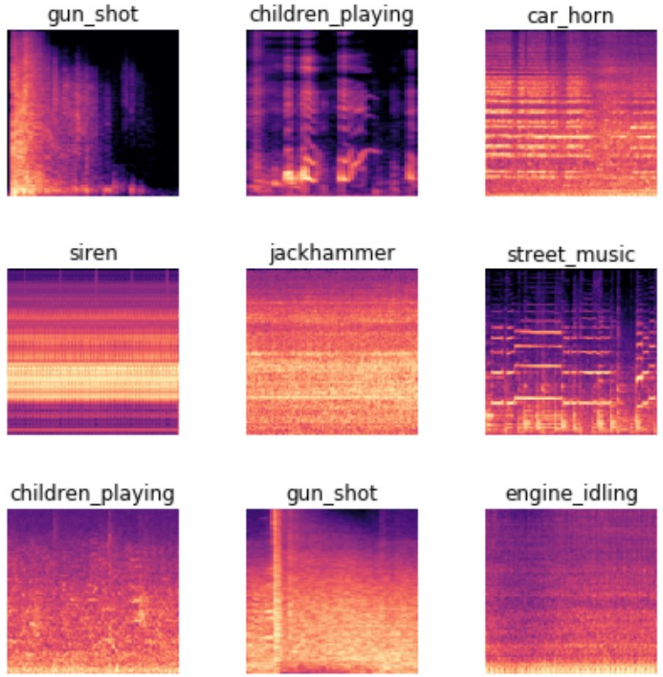
5. Low quality *X* (image)

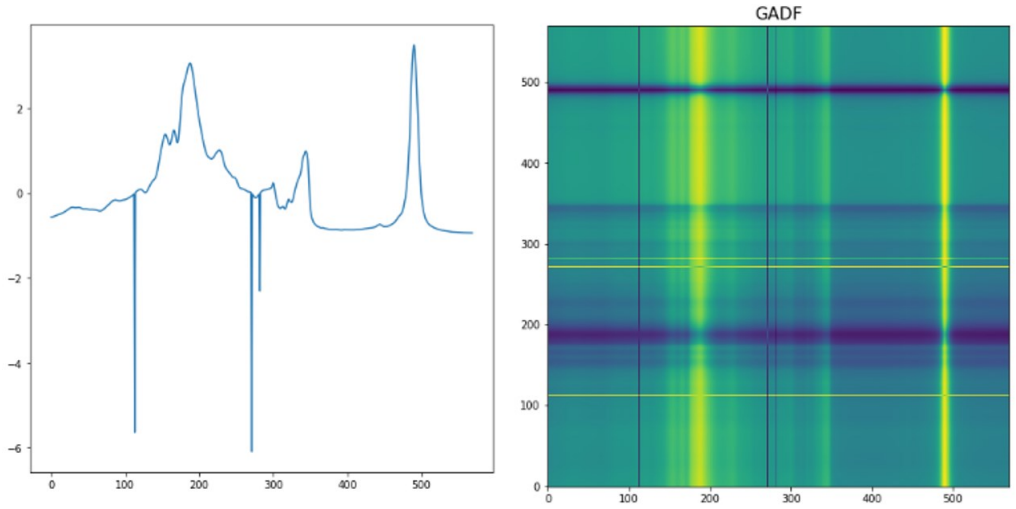
- The quality of machine learning models hinges on the quality of the data used to train them, but it is hard to manually identify all of the low-quality data in a big dataset
  - *CleanVision* helps you automatically identify common types of data issues lurking in image datasets
  - Remember that you can use *cleanlab* to find label issues (y) for images



Which type of problems can be solved using CNN?

- Image recognizer can learn to complete many tasks
  - For instance, a sound can be converted to a *spectrogram*, which is a chart that shows the a mount of each frequency at each time in an audio file and can be tackled using CNN
  - There are various transformations available for time series data. For instance, using a technique called Gramian Angular Difference Field (GADF) and feed into CNN

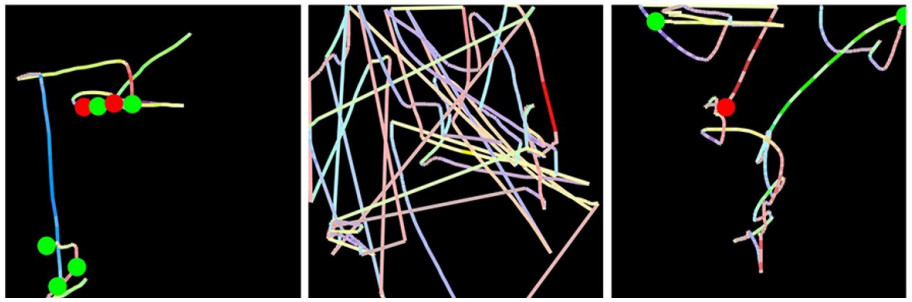


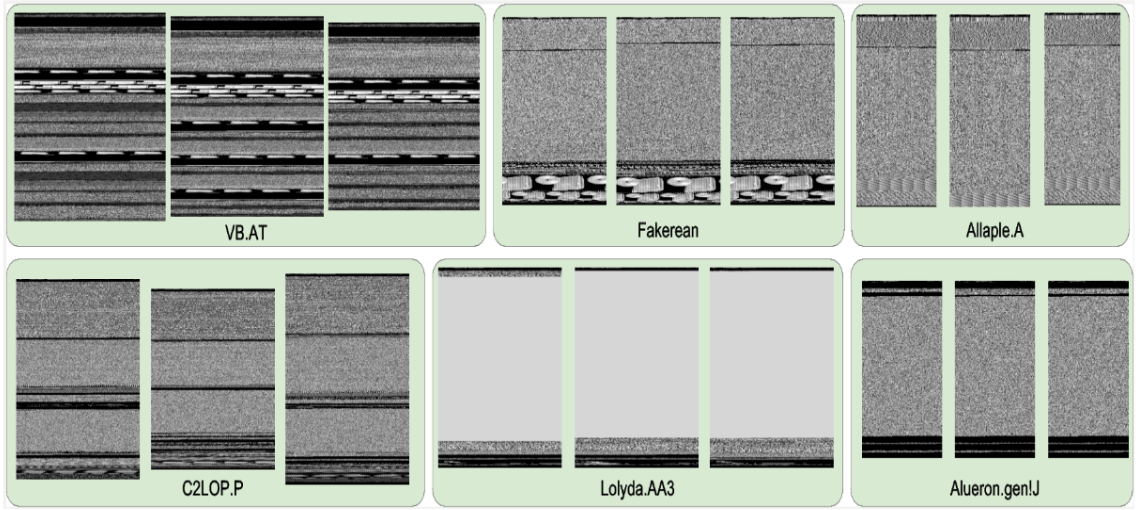


Similar approach can be found in:
https://ndltd.ncl.edu.tw/cgi-bin/gs32/gsweb.cgi/login?o=dncledr&s=id=%22110NSYS5507009%22.&searchmode=basic

Which type of problems can be solved using CNN?

- If the human eye can recognize categories from the images, then a deep learning model should be able to do so too
  - Fraud detection by drawing and image where the position, speed, and acceleration of the mouse pointer by using colored lines and the clicks were displayed using small colored circles. The results are fed into CNN for classification
  - Malware binary file is divided into 8-bit sequences which are then converted to equivalent decimal values. This decimal vector is reshaped and a gray-scale image is generated that represents the malware sample





Conclusion

- The 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
- 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, 3rd 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

[4] Deep Learning for Coders with Fastai and Pytorch: AI Applications Without a PhD Chapter 1 and 7

Appendix

Resources

- Understand CNN
  - https://d2l.ai/chapter_convolutional-neural-networks/conv-layer.html
  - https://setosa.io/ev/image-kernels/
  - https://poloclub.github.io/cnn-explainer/
  - https://cs231n.github.io/convolutional-networks/#conv
  - https://distill.pub/2019/computing-receptive-fields/
  - https://distill.pub/2018/building-blocks/
  - https://distill.pub/2017/feature-visualization/
- Backpropagation for convolution and pooling layers
  - Convolution layer
  - Max pooling

Resources

- Data augmentation
  - https://seunghan96.github.io/cv/vision_15_Data_Augmentation(1)/
  - https://coggle.it/diagram/XttJu5nHqhqCptXn/t/data-augmentation
- Integration to sklearn
  - https://github.com/adriangb/scikeras
  - https://github.com/skorch-dev/skorch
- Data cleaning
  - https://github.com/idealo/imagededup
  - https://github.com/cleanlab/cleanlab
  - https://github.com/cleanlab/cleanvision

Resources

- Image processing library
  - https://github.com/ml-tooling/best-of-ml-python#image-data
  - https://scikit-image.org
  - https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html
  - https://github.com/kornia/kornia
- Dataset
  - https://paperswithcode.com/datasets?mod=images&task=image-classification
    - ImageNet, Imagenette - https://github.com/fastai/imagenette
    - Cifar10, Cifar100 - https://www.cs.toronto.edu/~kriz/cifar.html
    - MNIST, FashionMNIST - https://github.com/zalandoresearch/fashion-mnist
  - https://paperswithcode.com/datasets?mod=images&task=object-detection
  - https://paperswithcode.com/datasets?mod=images&task=semantic-segmentation

Other popular architectures for image classifications

- Tensorflow
  - Keras
  - VGG
  - DenseNet
  - SENet
  - EfficientNet
  - MobileNet
- Pytorch

Popular architectures for object detection

- Tensorflow
  - https://keras.io/guides/keras_cv/object_detection_keras_cv/
  - SSD
  - Faster-RCNN
  - RetinaNet
- Pytorch
  - https://github.com/microsoft/computervision-recipes/tree/staging
  - https://github.com/huggingface/pytorch-image-models
  - https://github.com/facebookresearch/detectron2
- YOLO
  - YOLOX

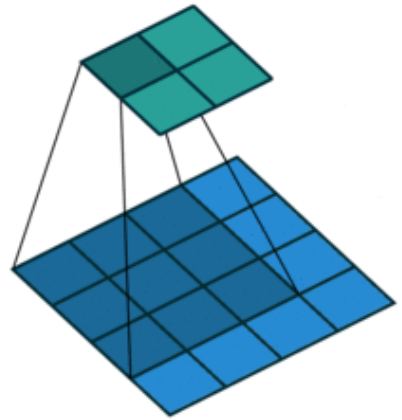
Popular architectures for image segmentation

- Tensorflow
  - https://github.com/divamgupta/image-segmentation-keras
  - https://github.com/google-research/deeplab2
- Pytorch
  - https://github.com/qubvel/segmentation_models.pytorch
- https://github.com/facebookresearch/segment-anything

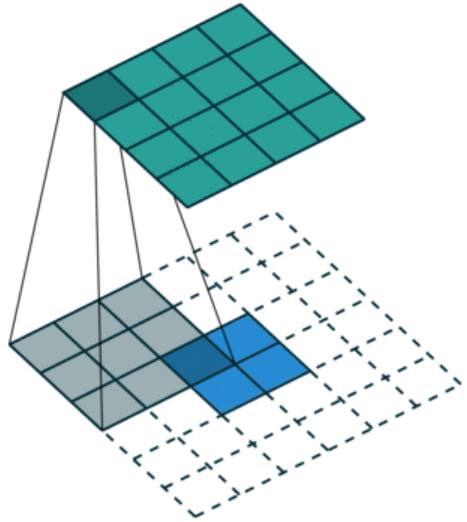
Convolution and transposed convolution

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

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



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

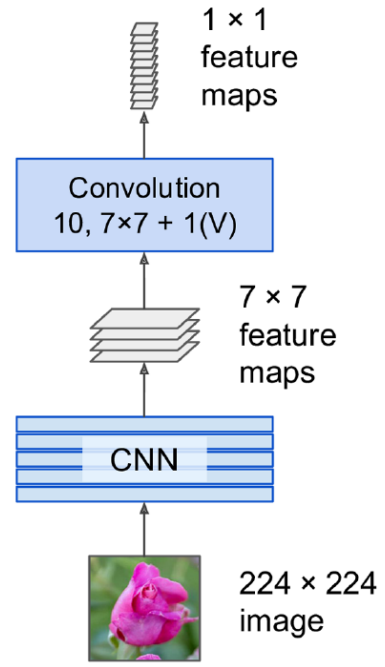


Architecture of a CNN

- The size of the convolution filter
  1. A common mistake is to use convolution kernels that are too large
  2. 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
  3. 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

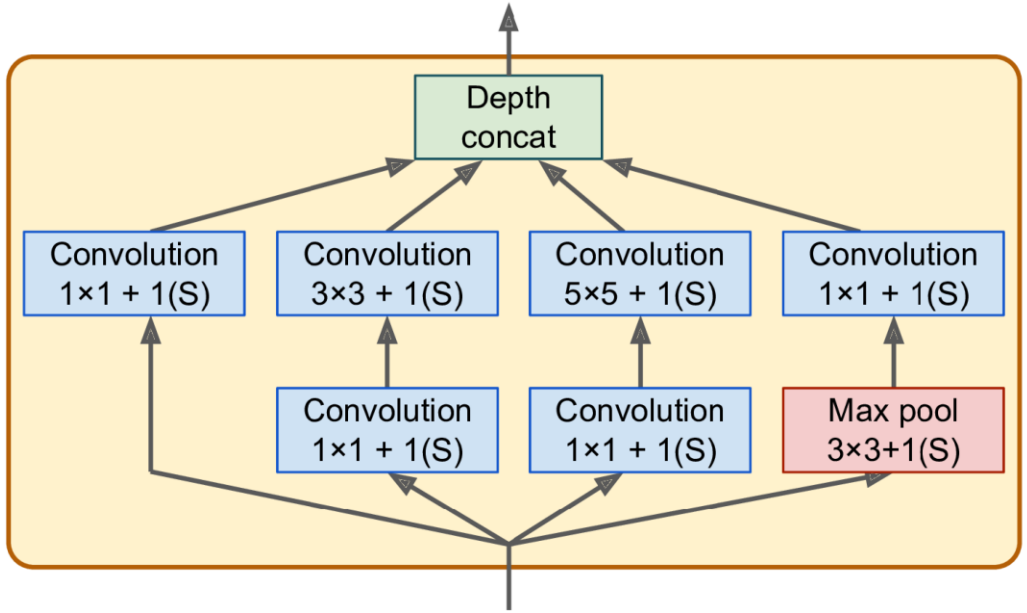
AlexNet - Fully convolution neural network

- 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



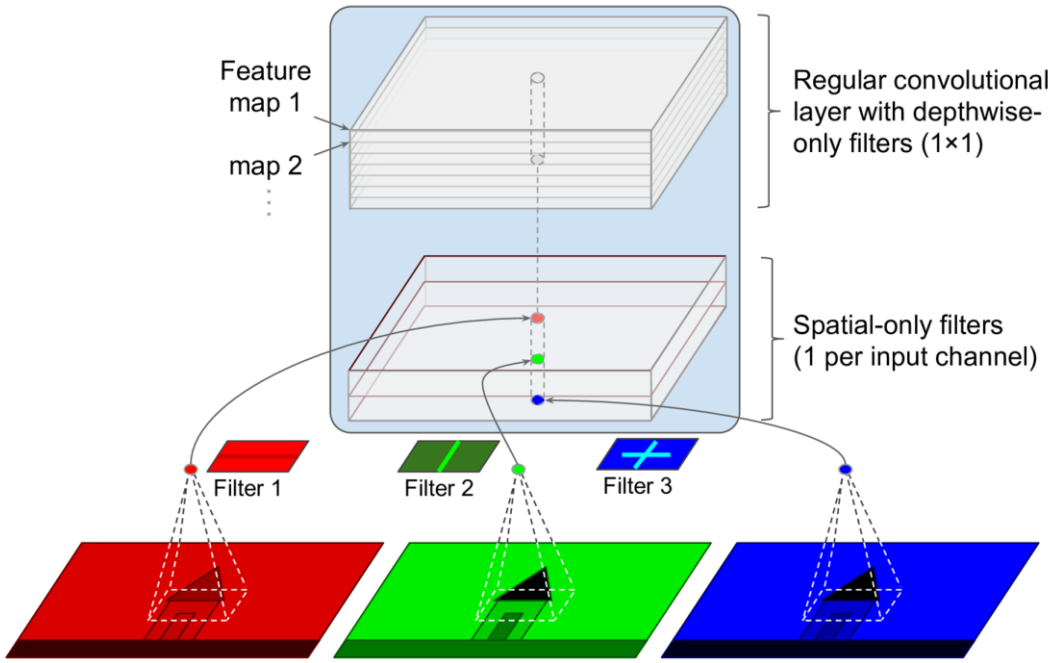
GoogleLeNet - Inception module

- "3 × 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



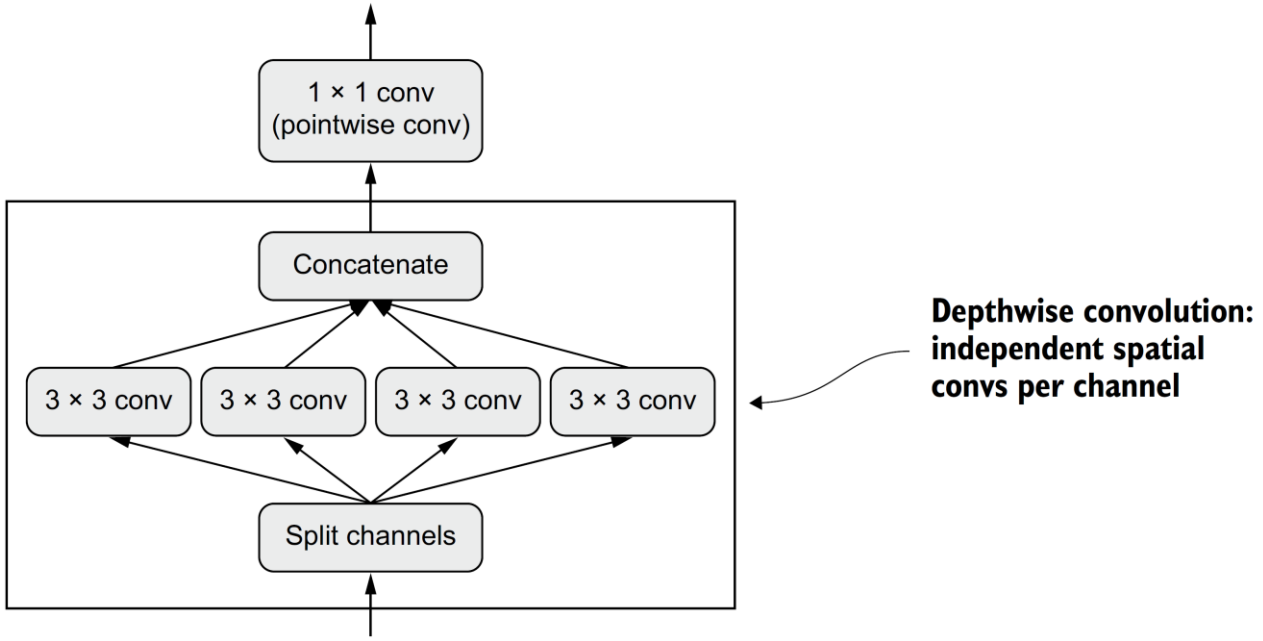
Xception - Depthwise separable convolutions

- 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



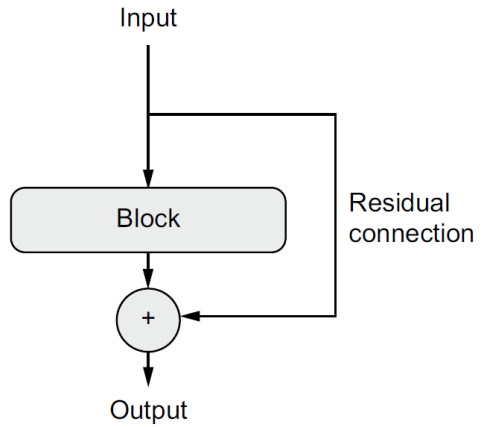
Xception - Depthwise separable convolutions

- 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



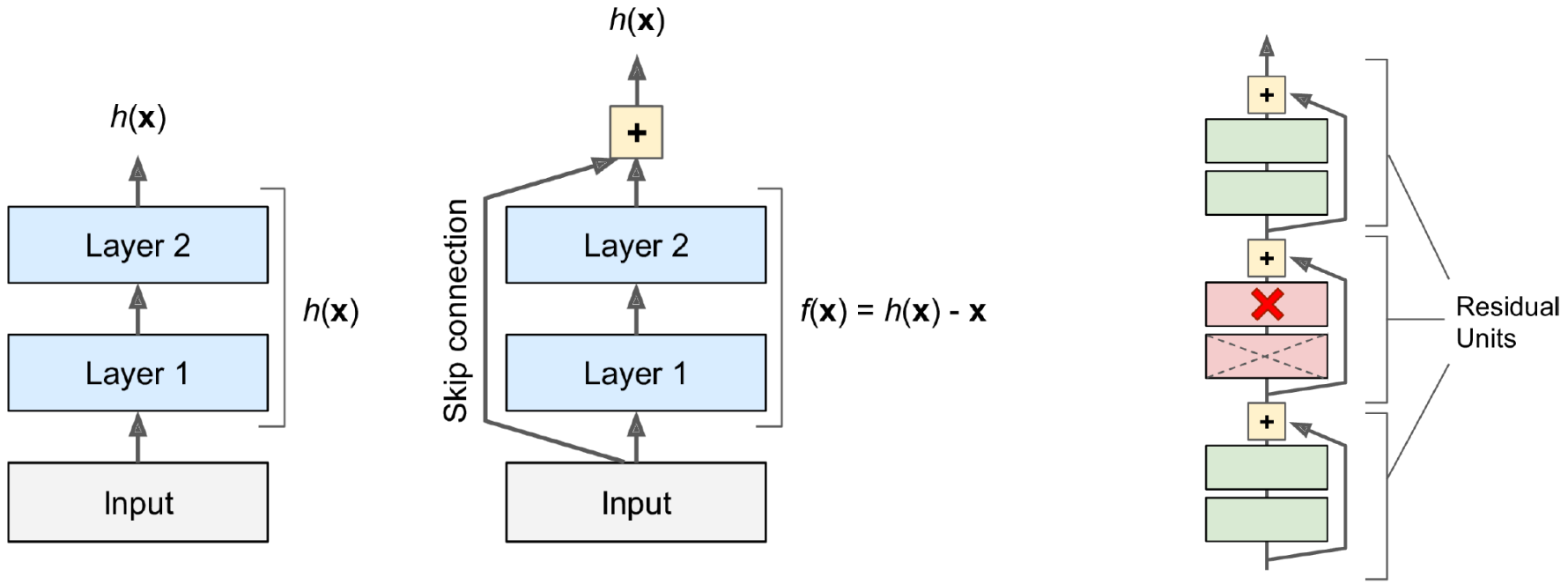
ResNet - Residual learning

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



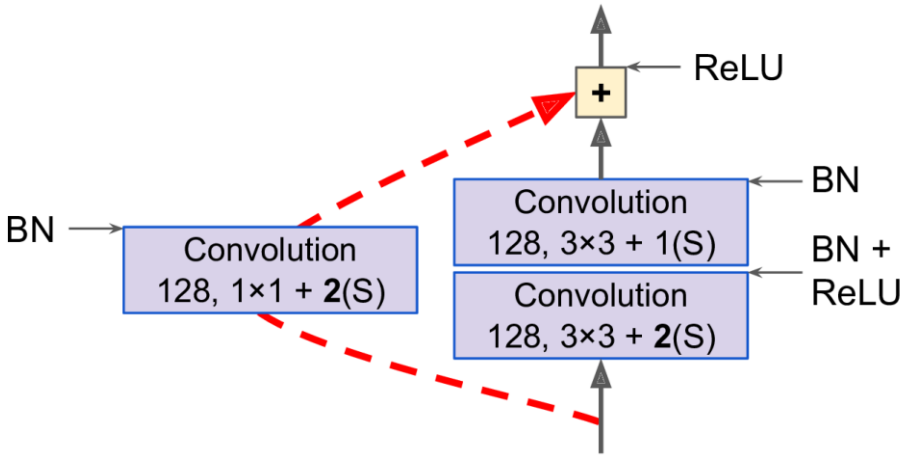
ResNet - Residual learning

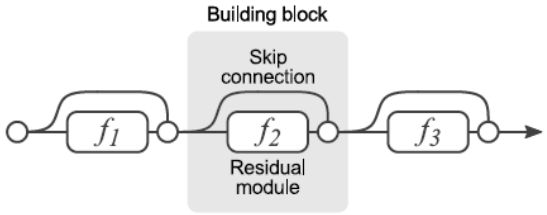
- Residual $f(x)$ is easier to learn using *skip connection*
  - If the target function is fairly close to the identity function (which is often the case), this will speed up training considerably
  - The network can start making progress even if several layers have not started learning yet



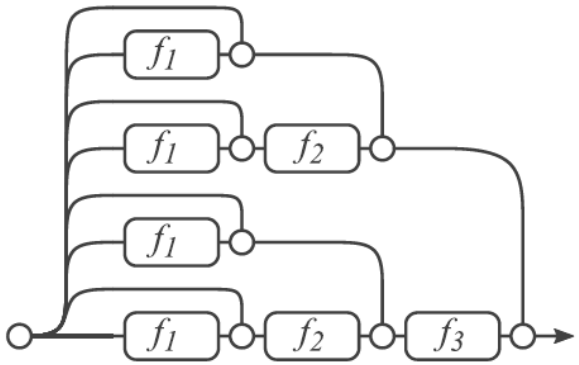
ResNet - Residual learning

- 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





=



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