Deep Learning and Neural Networks

Demetrio Labate

March 5, 2024

Part 2
Convolutional Neural Networks

Convolutional Neural Networks

Convolutional neural networks (CNNs) form a class of artificial neural networks that has become dominant in various **image processing** and **computer vision** tasks following their success in the object recognition competition known as the ImageNet Large Scale Visual Recognition Competition (ILSVRC) in 2012.

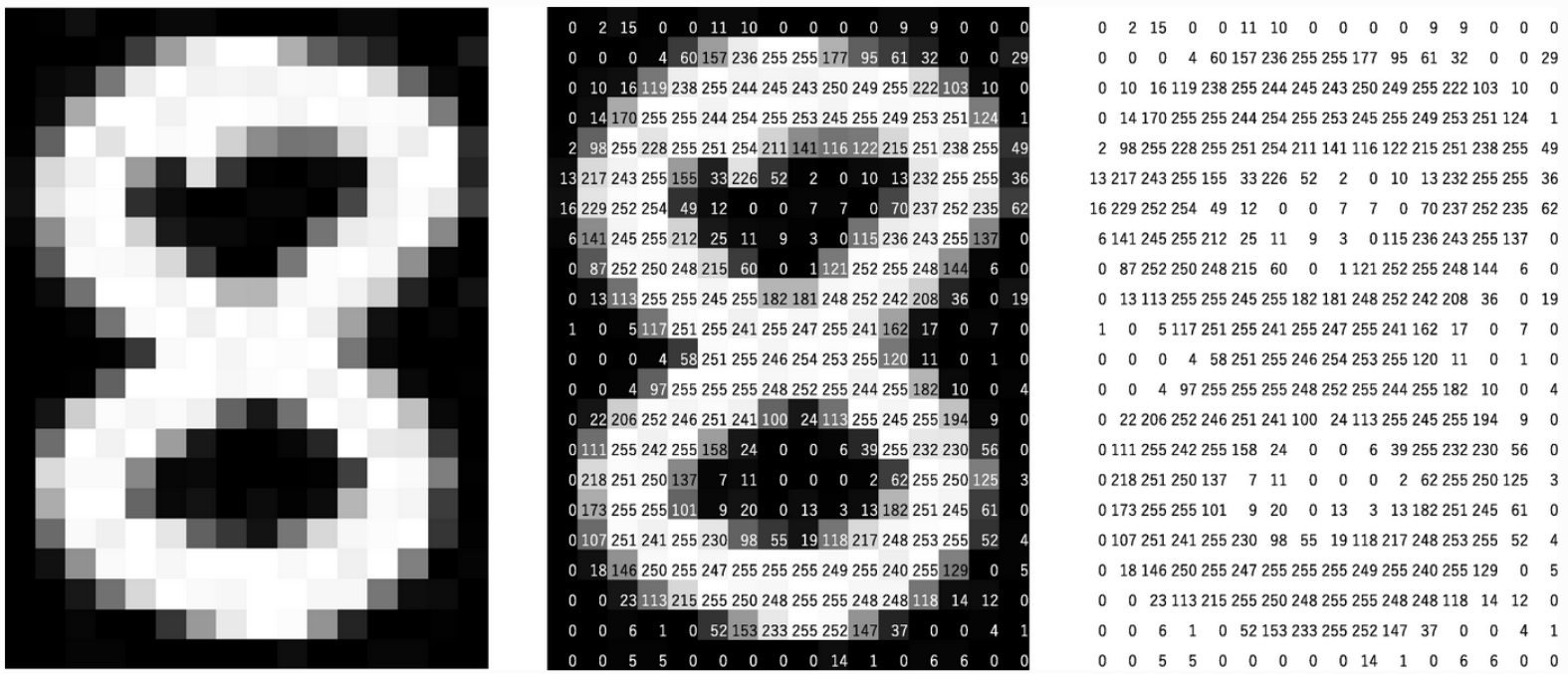
Since then, CNNs have been successfully applied to multiple image analysis and processing tasks including

1. object detection
2. classification
3. segmentation
4. image reconstruction
5. natural language processing

Convolutional Neural Networks

CNNs are motivated by applications of machine learning where the input data has some **spatial structure** that should be leveraged when solving supervised or unsupervised problems.

Images are one predominant example of such data with spatial constraints.



Another example are videos.

Convolutional Neural Networks

Image information is stored over a regular rectangular grid/matrix at small square units/entries called **pixels**.

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

A computer sees an image as an array of numbers. The matrix on the right contains numbers between 0 and 255, each of which corresponds to the pixel brightness in the left image.
The pixel values are, in most cases, of little to no use without knowledge of their spatial location.

Convolutional Neural Networks

CNNs are designed to deal with input data having some spatial structure.

At the heart of such neural network is the **convolution operation** which can be essentially thought of as a weighted linear combination (through a structured set of learnable parameter forming a **kernel**) that *preservers the spatial structure of its input.*

This mathematical operation is modeled on the organization of animal visual cortex and a CNN is designed *to automatically and adaptively learn spatial hierarchies of features, from low- to high-level patterns.*

We will examine the convolution operator in more detail.

Convolutional Neural Networks

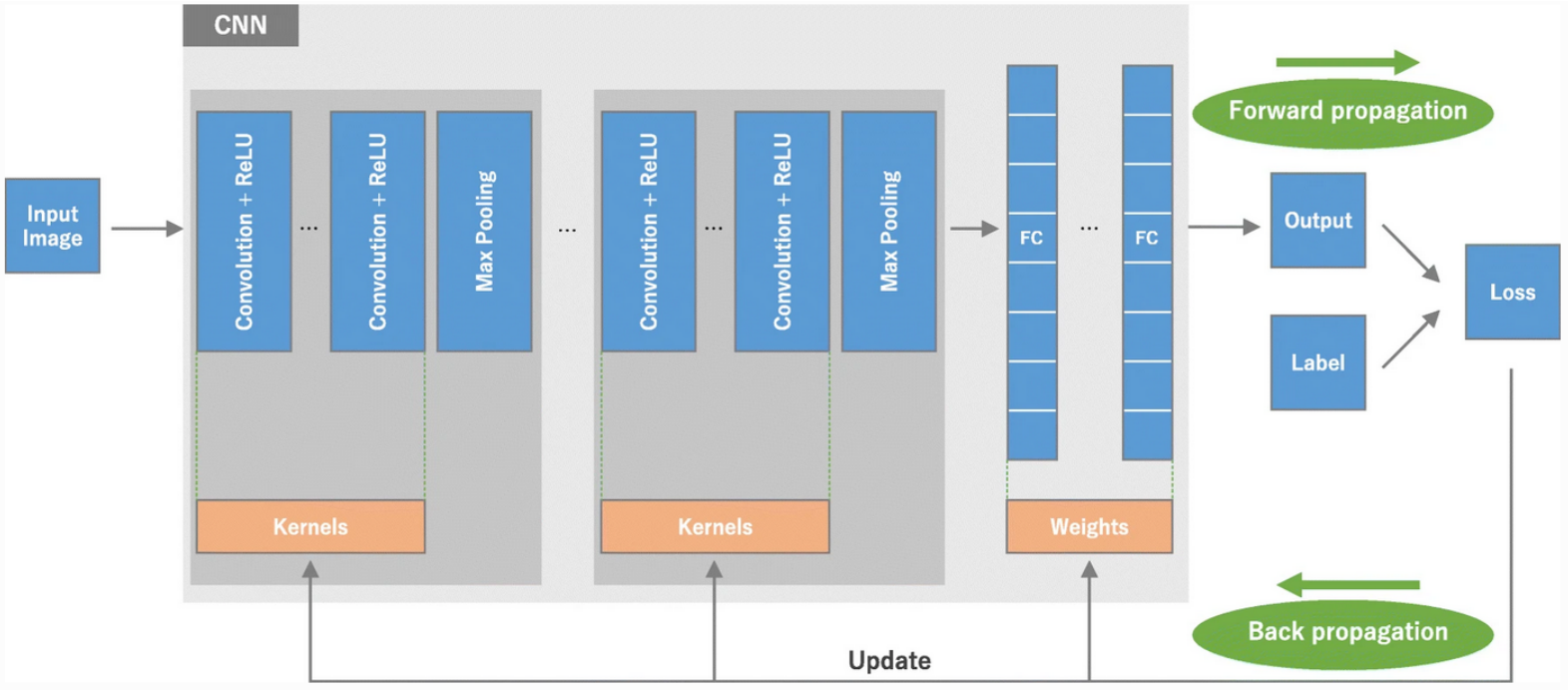
Any CNN is typically composed of three types of layers:

- **convolution layers**
- **pooling layers**
- **fully connected layers**

The first two types of layers, convolution and pooling layers, perform **feature extraction**, whereas the fully connected layer maps the extracted features into final output, such as classification.

A typical CNN architecture consists of multiple blocks of convolution and pooling layers followed by fully connected layers.

Convolutional Neural Networks



A CNN is composed of a stacking of several building blocks: convolution layers, pooling layers (e.g., max pooling), and fully connected (FC) layers.

The first section performs feature extraction and the last section performs an inference task, e.g., classification.

2.1 The convolution operator

The convolution operator

I start by defining the operation of convolution for functions defined on the set of integers $\mathbb{Z}$

Definition.

For functions x, w defined on $\mathbb{Z}$, the **discrete convolution** of x and w is

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

By commutativity

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

Note: we can apply commutativity here because functions have infinite support.

The convolution operator

The operation of convolution extends to functions with **finite support**.

When w has finite support in the set $\{-M, -M+1, \ldots, M\}$, we define the discrete convolution of x and w as

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

Remarks:

- Unlike the infinite-support case, it is **not** true that $x * w = w * x$.
- In electrical engineering, w is typically identified with the *finite input response* associated with a linear time-invariant system.
- In the neural network literature, w is typically referred to as the **convolution filter** or **kernel.**

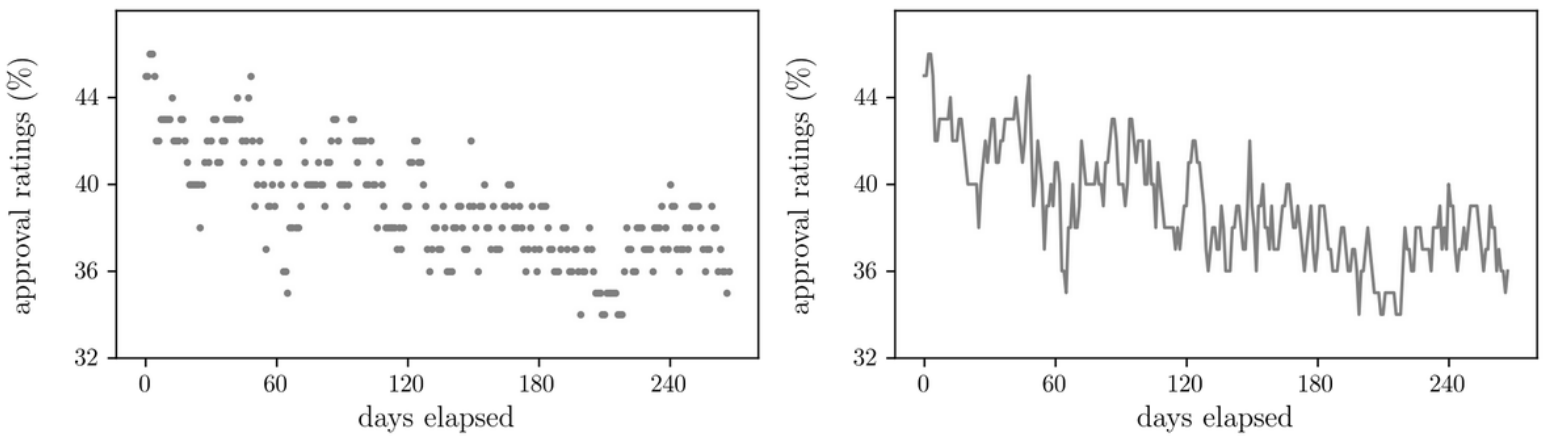
The convolution operator

To illustrate the significance of the operation of convolution, we consider **time-series data**.

A time-series data set usually takes the general form

$$(t_1, x_1), \ldots, (t_N, x_N),$$

where the inputs $t_1, \ldots, t_N$ are time-marks or time-stamps sorted in ascending order (that is, $t_1 < t_2 < \cdots < t_N$) and $x_1, \ldots, x_N$ denote the corresponding output values, respectively.



Example: daily job approval ratings of a political candidate plotted here both without (left) and with (right) linear interpolation.

The convolution operator

Notice how the time-series data set in the Example appears jagged due to the relatively rapid fluctuation in the data.

Many time-series data sets exhibit similar behavior that can be attributed to a high-frequency (i.e., rapidly changing) noise perturbing the smooth signal we are aiming to observe.

We can **denoise** the data by computing the average of each observed data in its M-vicinity as :

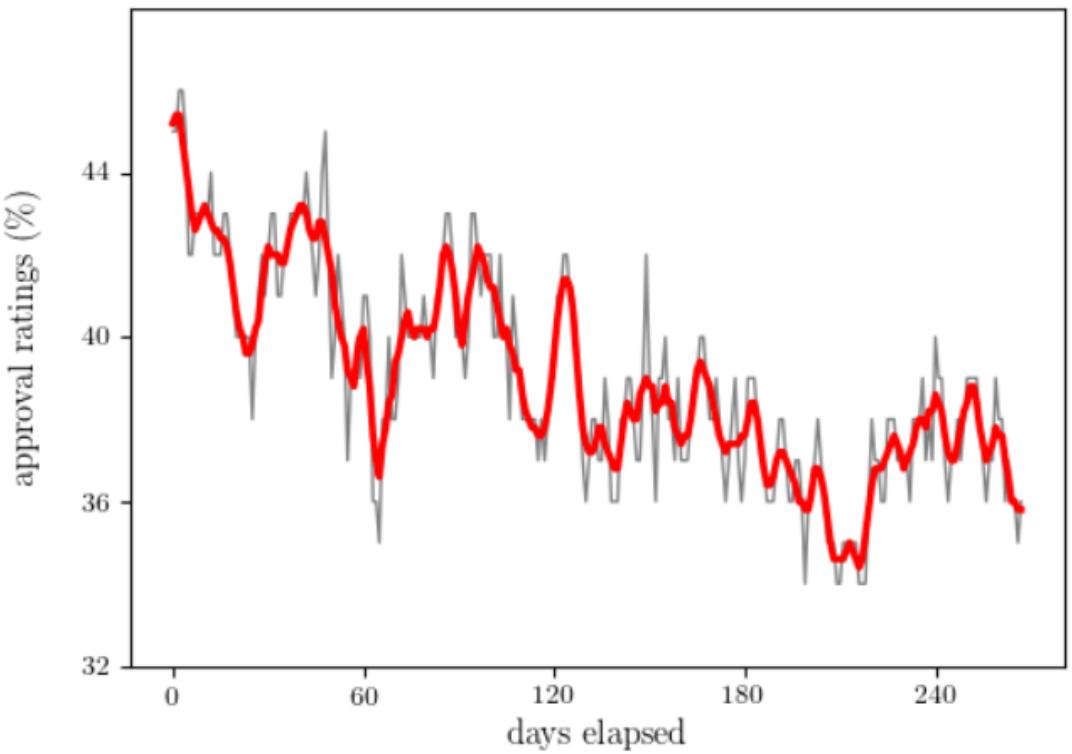
$$y_n = \frac{1}{2M+1}(x_{n-M} + x_{n-M+1} + \cdots + x_{n+M}) = \frac{1}{2M+1} \sum_{m=-M}^{M} x_{n-m}$$

Note that $y_n = (x * w^M)_n$ with

$$w_m^M = \begin{cases} \frac{1}{2M+1} & \text{if } m \in \{-M, \ldots, M\} \\ 0 & \text{otherwise} \end{cases}$$

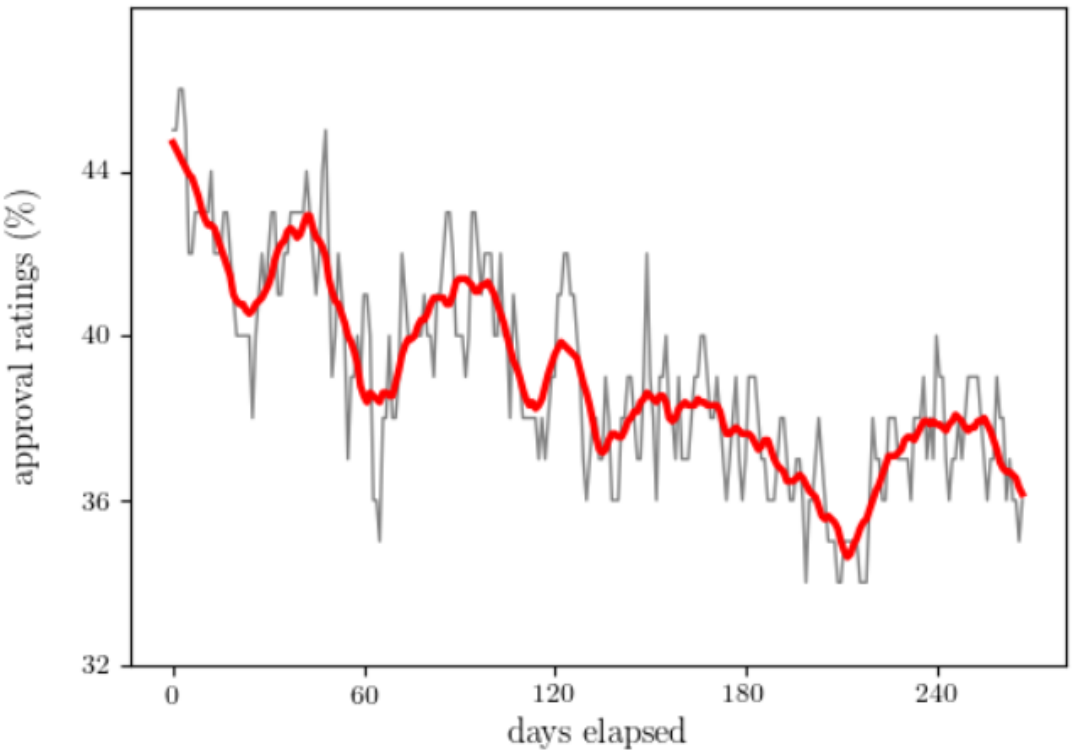
The convolution operator

Here is the result of the denoising experiment where we use the convolution $y = x * w^M$ with $M = 4$



The convolution operator

Here is the result of the denoising experiment where we use the convolution $y = x * w^M$ with $M = 10$



The convolution operator

More generally, we can process the time series x by convolution with a 'bump' function g of finite support.
For example:

$$w_m^M = \begin{cases} \frac{M+1-|m|}{(M+1)^2} & \text{if } m \in \{-M, \ldots, M\} \\ 0 & \text{otherwise} \end{cases}$$

generates a triangle function centered at the origin.

$y = x * w^M$ is now an even smoother (denoised) version of x.

Note: the design of **denoising filters** is part of classical study of Linear Time-Invariant filters in Signal Processing.

The convolution operator

Remark: Boundary conditions

Let $x[n]$ be define for $n = 1, \ldots, N$ and $w[n]$ be defined for $n = -M, \ldots, M$.

Notice that to compute

$$y[n] = (x * w)[n] = \sum_{m=-M}^{M} x[n-m]\, w[m]$$

we need to access elements of x that fall outside its original range.

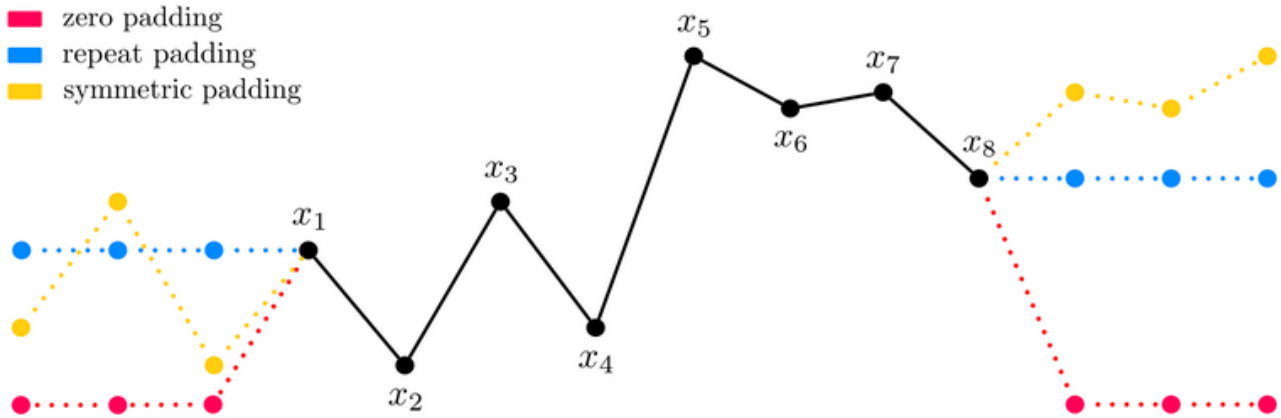
Specifically, the convolution operation requires the evaluation of terms ranging over

$$x[1-M], X[2-M], \ldots, x[N+1], x[N+M]$$

The convolution operator

To fix this issue, we add M entries to both the beginning and the end of x so that for every n, x is defined in an M-vicinity of n.

This insertion operation is called **padding**.



How we pad x is for mostly inconsequential, particularly when M is relatively small compared to N, as padding only affects the first and last M elements of the resulting signal.

In the figure, $N = 8$ and $M = 3$.

The convolution operator

The convolution operation can be extended in a straightforward manner to higher dimensions.
We are mostly interested in the two-dimensional case.

Definition.

For functions x, w defined on $\mathbb{Z}^2$, the **discrete convolution** of x and w is defined entry-wise as

$$(x * w)[n_1, n_2] = \sum_{m_1=-\infty}^{\infty} \sum_{m_2=-\infty}^{\infty} x[n_1 - m_1, n_2 - m_2]\, w[m_1, m_2]$$

The convolution operator

As above, the convolution operation can be defined on two-dimensional functions with compact support.

When g has finite support in the set $n_1 \in \{-L_1, -L_1 + 1, \ldots, L_1\}$, $n_2 \in \{-L_2, -L_2 + 1, \ldots, L_2\}$, the discrete convolution of x and w is given by

$$(x * w)[n_1, n_2] = \sum_{m_1=-L_1}^{L_1} \sum_{m_2=-L_2}^{L_2} x[n_1 - m_1, n_2 - m_2]\, w[m_1, m_2]$$

The convolution operator

In the neural network literature, the convolution operation has been defined historically with a "+" sign. Technically, it should be called *cross-correlation*.

This has no impact of the properties and interpretation of the operation.

In the following, in accord with **neural network convention**, we will refer to the *convolution of x and w* as the following operation

$$(x * w)[n_1, n_2] = \sum_{m_1=-L_1}^{L_1} \sum_{m_2=-L_2}^{L_2} x[n_1 + m_1, n_2 + m_2]\, w[m_1, m_2]$$

In this setting, w is called the **convolution filter** or the **kernel**. It is a matrix of dimension $(2L_1 + 1) \times (2L_2 + 1)$.

The convolution operator

As in the one-dimensional case, to carry out the operation of convolution in the compact support case we need to manage the boundary.

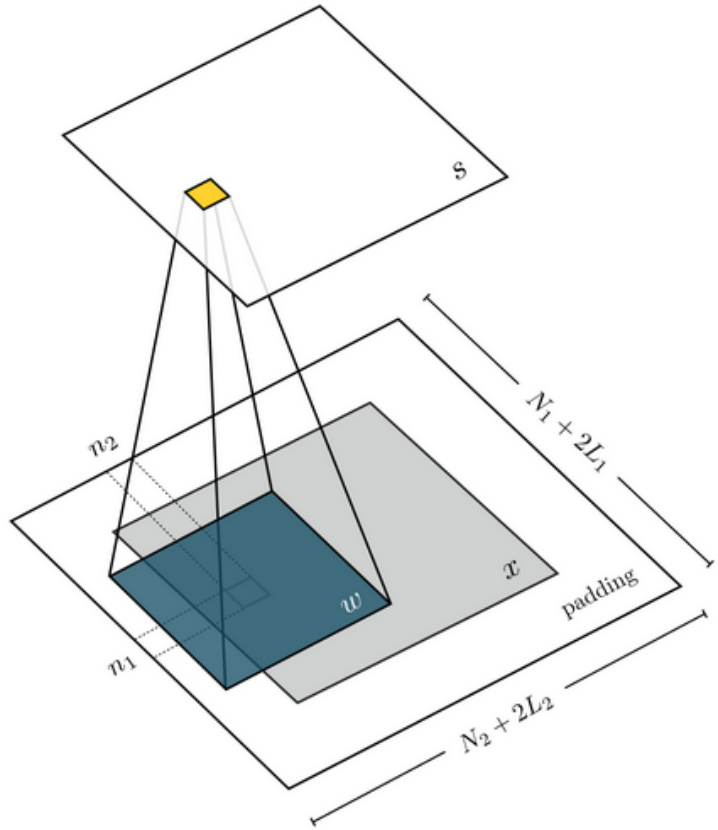
Let $x[n_1, n_2]$ be supported in $[1, N_1] \times [1, N_2]$
and $w[n_1, n_2]$ be supported in $[-M_1, M_1] \times [-M_2, M_2]$

For the operation

$$(x * w)[n_1, n_2] = \sum_{m_1=-L_1}^{L_1} \sum_{m_2=-L_2}^{L_2} x[n_1 + m_1, n_2 + m_2]\, w[m_1, m_2]$$

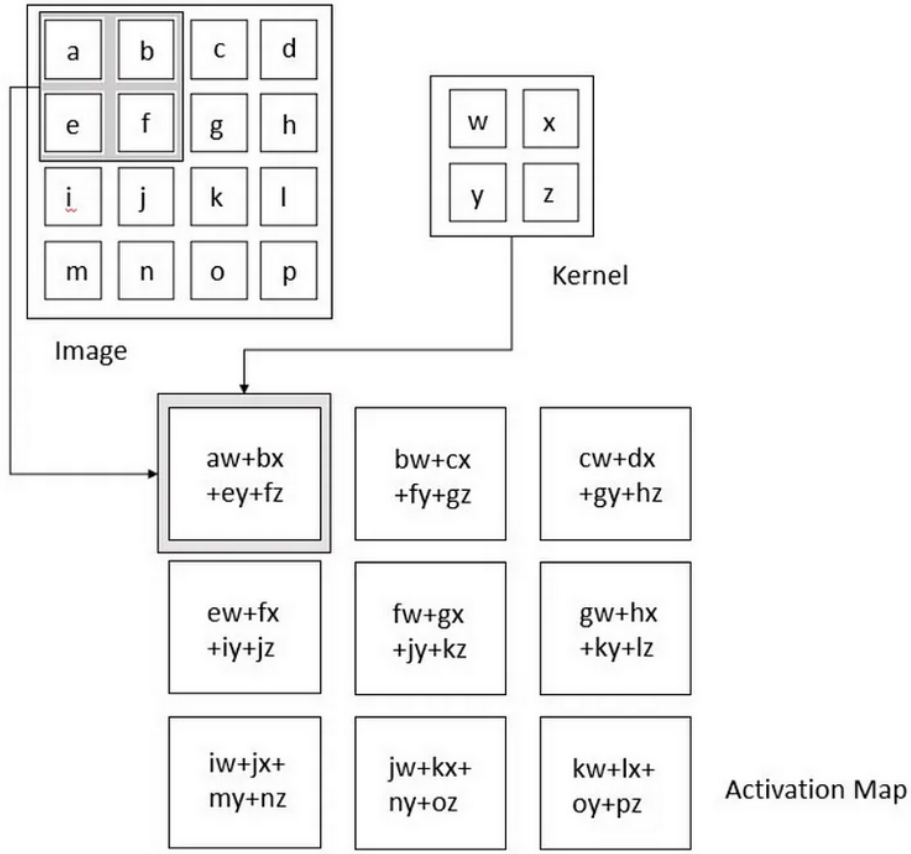
to be defined on the grid $[1, N_1] \times [1, N_2]$ the signal x must be **zero padded**.

The convolution operator



The input matrix x has been zero-padded to become a matrix of size $(N_1 + 2L_1) \times (N_2 + 2L_2)$ so that $(x * w)[n_1, n_2]$ can be evaluated for $1 \leq n_1 \leq N_1$, $1 \leq n_2 \leq N_2$.

The convolution operator



The **kernel** slides across the height and width of the image-producing the image representation of that receptive region. This produces a representation of the image called **feature map** giving the response of the kernel at each spatial position of the image.

The sliding size of the kernel is called a **stride.**

The convolution operator

You can find online animated illustration of the operation of image convolution

link: illustration of convolution

The convolution operator - Implementation note

What is the **output size** of a convolution operation in a CNN?

If we have an input of size $L \times L$ and a kernel of size of $W \times W$ with stride S (by default $S = 1$ in convolution) and amount of padding P, then the size of the output feature map is $L_{out} \times L_{out}$ where L_{out} is determined by the following formula

$$L_{out} = \frac{L - W + 2P}{S} + 1$$

For instance, in the example of slide 24, we have that

$$L_{out} = \frac{4 - 2 + 0}{1} + 1 = 3,$$

hence, the feature map has size 3×3.

Clearly, if we have N kernels of size $W \times W$, the output of the convolution layer will contain N feature maps of size $L_{out} \times L_{out}$.

2.2 Feature Extraction

Feature extraction by convolution

To carry out supervised or unsupervised tasks on image data, like object detection, classification, or image compression, one could always use the raw pixel values directly as features. This naive approach however has been shown experimentally to produce low quality results in virtually all machine learning tasks involving natural images.

An alternative, more efficient approach would be to represent an image by making use of **geometric features** derived from the raw pixel values, e.g., information associated with edges, corners or other landmarks.

This observation is consistent with the way our visual cortex processes information.

Feature extraction by convolution

The distinguishing information in a natural image is largely contained in the relatively small number of edge pixels in an image.

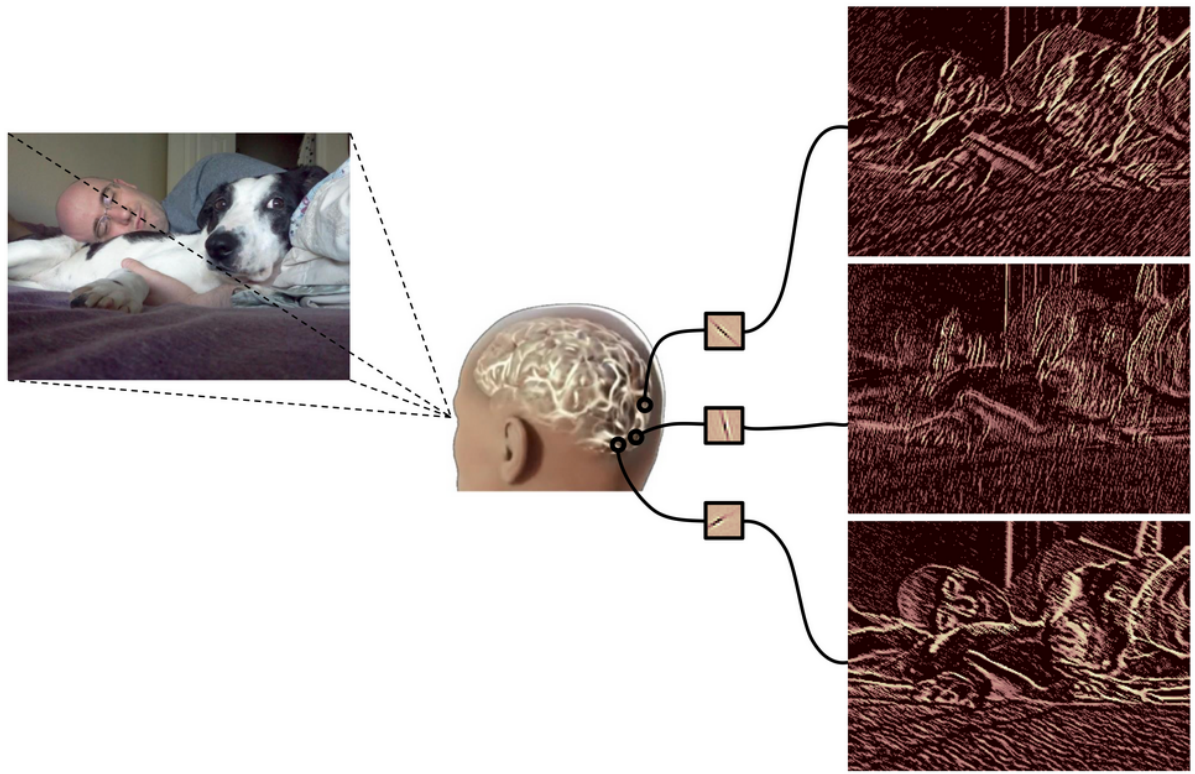
The edge-detected version of the image describes the scene very well using only a fraction of the information contained in the original image

Feature extraction by convolution

Studies performed on frogs, cats, and primates, where a subject is shown visual stimuli while electrical impulses are recorded in a small area in the subject's brain where visual information is processed, show the visual cortex is very **sensitive to edges**.

These studies have determined that individual neurons involved in early stages of visual processing roughly operate by identifying edges of different orientations.

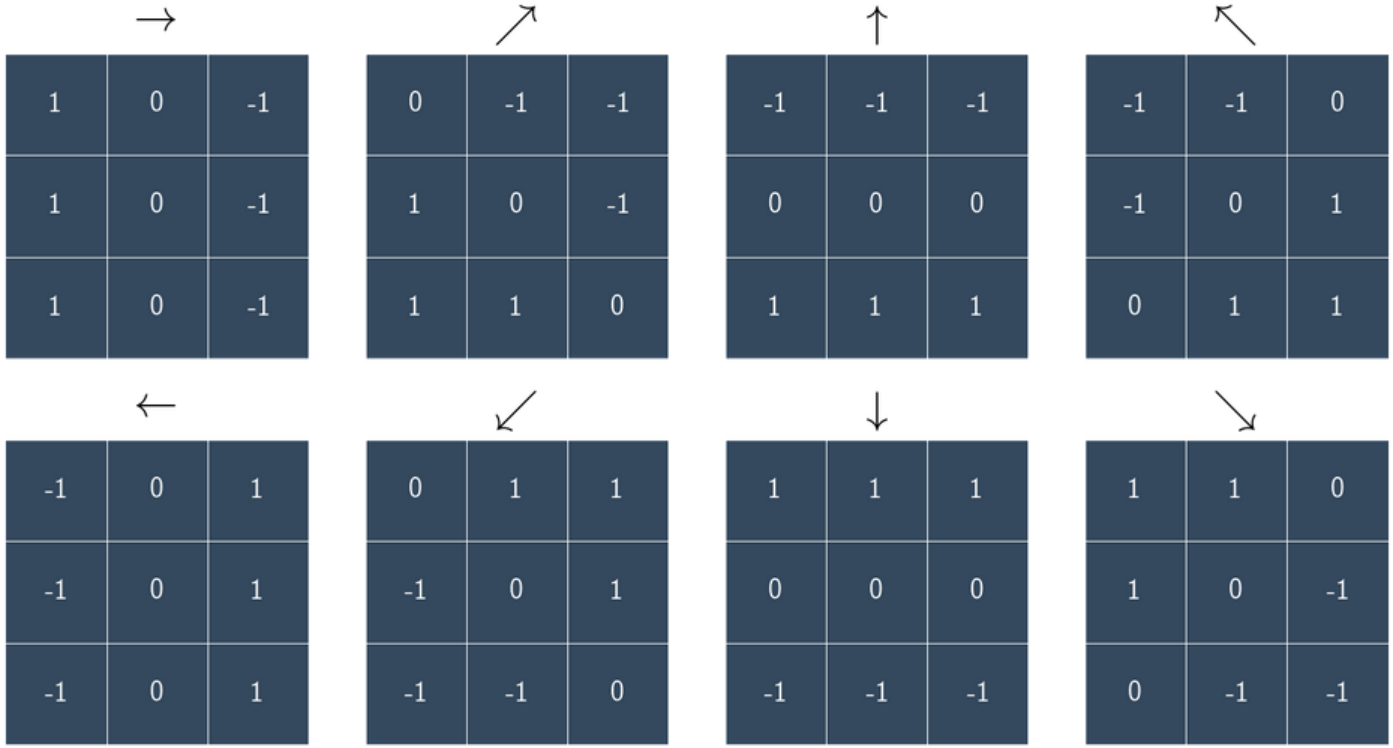
Feature extraction by convolution



Each neuron tacts as a small "edge detector," locating edges in an image of a specific orientation and thickness. It is thought that by combining and processing these edge detected images, humans and other mammals "see."

Feature extraction by convolution

- Edges can be detected within an image using convolution.



Each of the 8 kernels shown above corresponds to one of 8 equally (angularly) spaced edge orientations, starting from 0 degrees with seven additional orientations at increments of 45 degrees.

Feature extraction by convolution

Example. To capture the total 'edge content' of an image x in each direction, we generate image features as follows:

1. we convolve x with the directional derivative kernels $w_1, \ldots, w_8$ given above;
2. we pass the results through a rectified linear unit (ReLU) to remove negative entries;
3. we sum up the remaining positive pixel values.

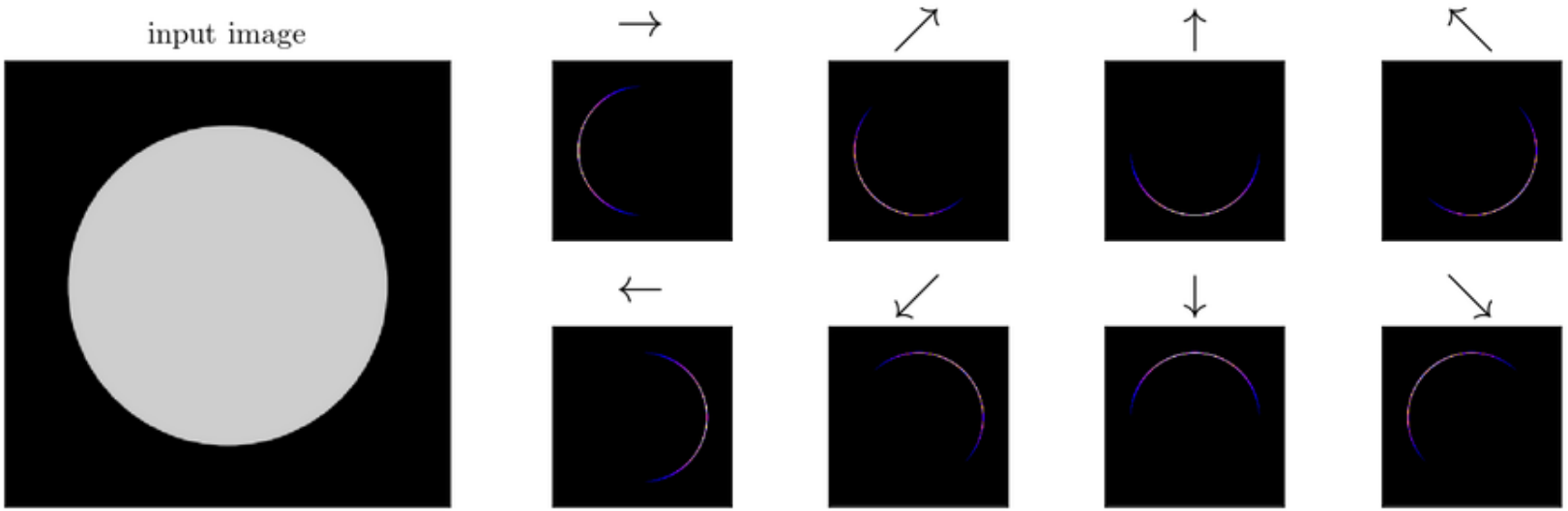
This way we generate the 8 **feature vectors**

$$f_i = \sum_{\text{all pixels}} \max\{0, x * w_i\}, \quad i = 1, \ldots, 8$$

Note: We use ReLU so that large positive values in $w_i * x$ do not get canceled out by possible large negative values in it.

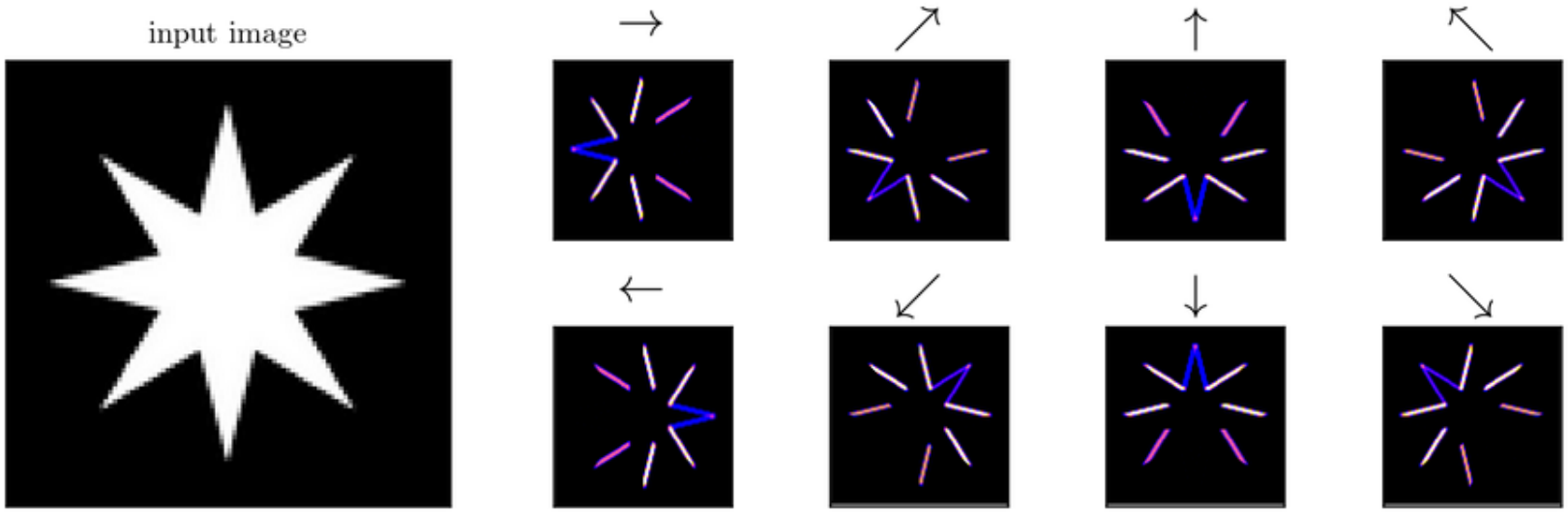
Feature extraction by convolution

Example. Here we plot an input image x representing a circle, along with the convolution plots of the input image with each of the eight kernels w_i displayed above after passing through ReLU.



Feature extraction by convolution

Here we plot an input image x representing a star, along with the convolution plots of the input image with each of the eight kernels w_i displayed above after passing through ReLU.



Feature extraction by convolution

Images used in applications are often significantly more complicated than these simplistic geometrical shapes. Summarizing them using just eight features to solve supervised learning tasks would be extremely ineffective if one wants

In fact, the 8 directional features computed above would not be sufficient to discriminate between the circle and the star.

To fix this issue, instead of computing the edge histogram features over the entire image, one can break the image into relatively small patches (that may be overlapping), compute features for each patch

$$f_{i,j} = \sum_{\text{all pixels in patch } j} \max\{0, x * w_i\}, \quad i = 1, \ldots, 8$$

and then concatenate the results to arrive at the final feature representation.

Pooling

Feature extraction is typically followed a **pooling** layer.

Pooling layers provide an approach for **downsampling feature maps** by summarizing the presence of features in patches of the feature map.
Two common pooling methods are average pooling and max pooling: they summarize the average presence of a feature and the most activated presence of a feature respectively.

Significance: A problem with the output feature maps is that they are sensitive to the location of the features in the input.
Downsampling the feature maps has the effect of making the resulting downsampled feature maps more robust to changes in the position of the feature in the image.
This desirable property is called **local translation invariance**.

Pooling

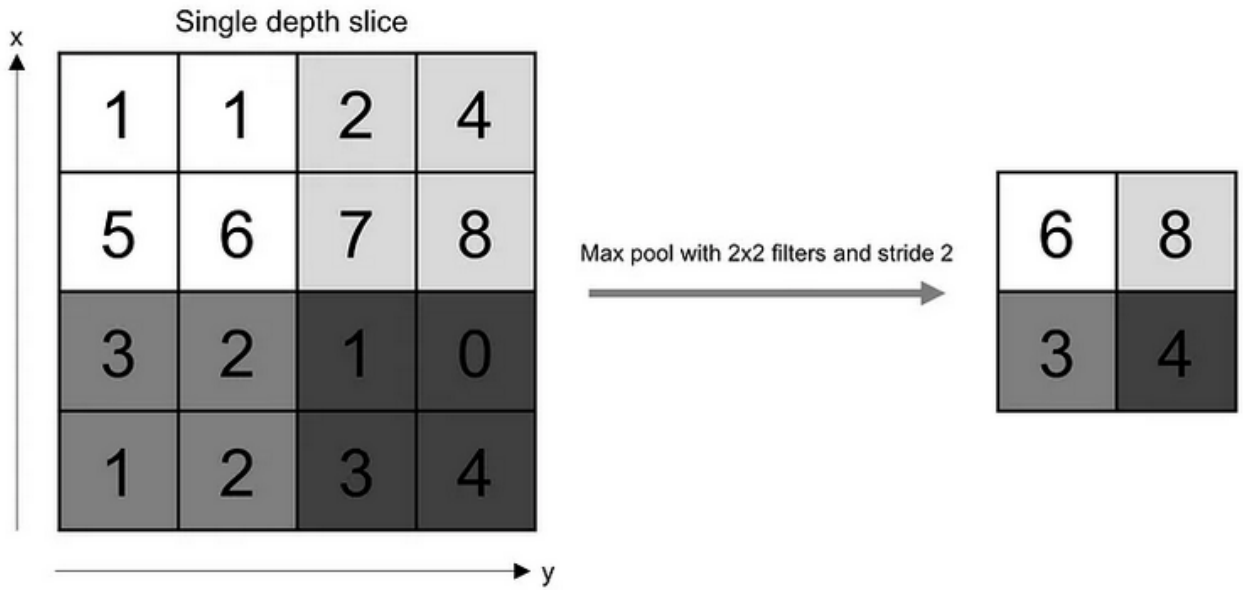
The pooling operation is carried out by applying a sliding window to the input image, similar to convolution but with some differences.

With pooling, the sliding window can jump multiple pixels at a time depending on how much overlap is required between adjacent windows/patches. The number of pixels the sliding window is shifted is referred to as the **stride**.

The second difference is how the content of the sliding window is processed. With convolution, you compute the sum of the entry-wise product between the windowed input and a kernel. With pooling there is no convolution operation but we simply average the entries within the window (**average pooling**) or take the maximum value (**max pooling**).

Pooling

The example below illustrates **max pooling** which reports the maximum output within a window.



In this example, window size is 2 and stride is 2.
With this choice of the stride, it **reduces the size of the feature map in half, here from** 4×4 **to** 2×2

Pooling

In general, if we have an input feature map of size $L \times L$, a pooling kernel of spatial size W and stride S, then the size of output image is $L_{out} \times L_{out}$ where L_{out} is determined by the following formula

$$L_{out} = \frac{L - W}{S} + 1$$

In the example above, $L = 4$, $W = 2$, $S = 2$.

Hence $L_{out} = \frac{4-2}{2} + 1 = 2$

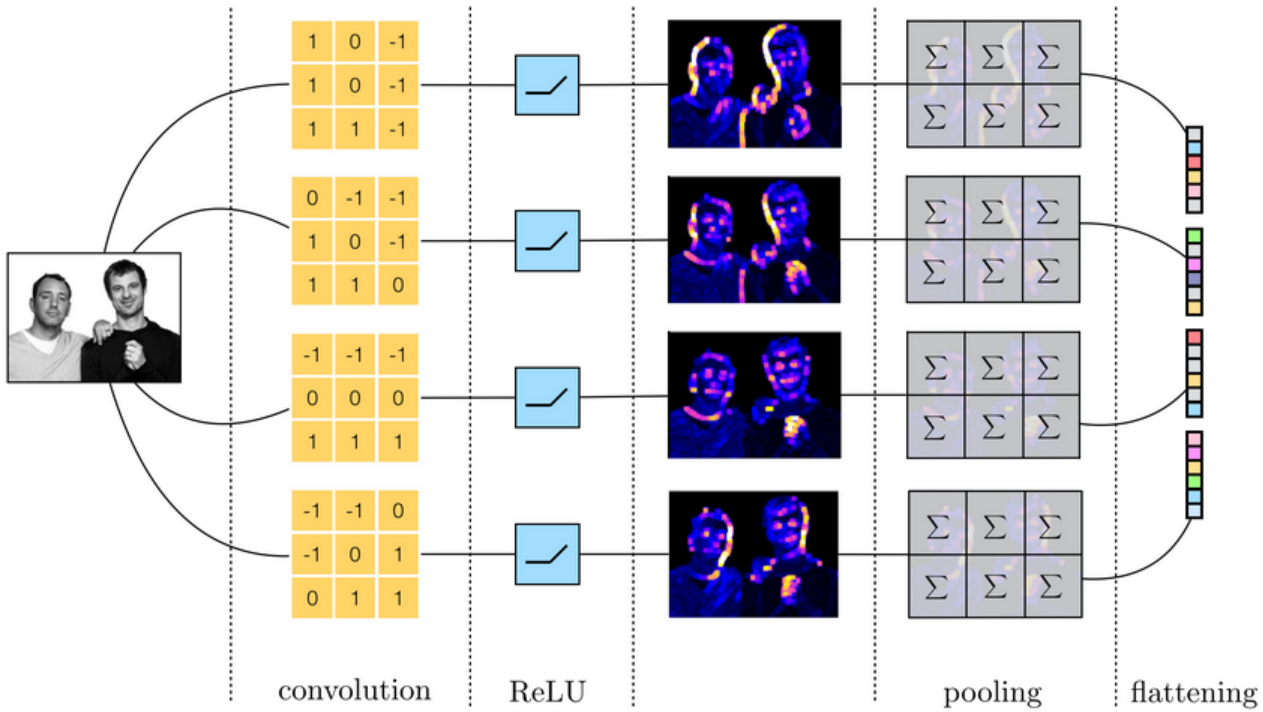
If we choose, $L = 16$, $W = 2$, $S = 2$,

then $L_{out} = \frac{16-2}{2} + 1 = 8$

Note: If $W = S$, then $L_{out} = \frac{L}{S}$

Convolution Neural Networks

Putting together all the components discussed above, i.e., convolution, ReLU, and pooling, we have a complete **end-to-end image feature extraction pipeline**



For compactness only four out of eight kernels are shown.

Convolution Neural Networks - Implementation note

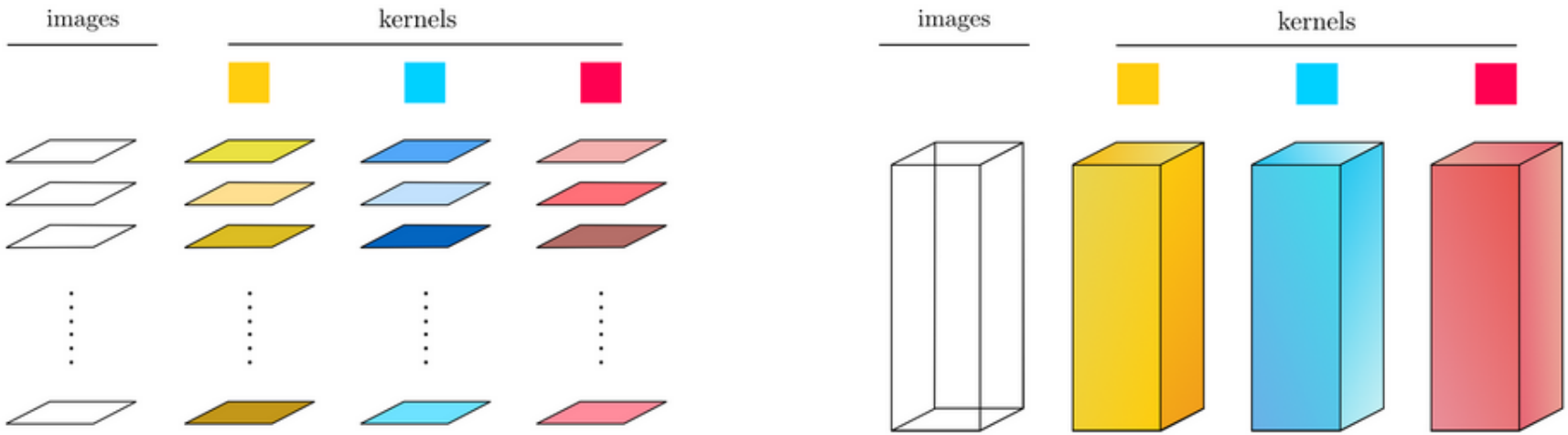
The presentation of the CNN design given above suggests that, given a set of images to analyze, one would process the images one after the other with a loop.

This implementation would be extremely inefficient in Python, particularly for medium- to large-sized datasets.

A careful examination about how convolutional feature maps are constructed on a set of images shows that we can re-write the convolutional layers in a much more efficient manner by employing **tensors**.

Convolution Neural Networks - Implementation note

The figure shows how to process an entire stack (tensor) of images simultaneously, thereby minimizing the number of explicit for-loops required.



Left: An implementation processing images sequentially is computationally slow because feature maps are computed for each individual image and for each individual kernel, using a nested for-loop.
Right: A tensor-based implementation of the convolutional layers speeds up computation considerably.

Convolution Neural Networks

Convolution in neural networks leverages three important ideas that motivated computer vision researchers: sparse interaction, parameter sharing, and locally invariant representation.

- **Sparse interaction.** Fully connected neural network layers use matrix multiplication by a matrix of parameters describing the interaction between the input and output unit. This means that every output unit interacts with every input unit.

  However, CNNs have sparse interaction that is achieved by making kernel smaller than the input e.g., an image can have several thousands of pixels, but while processing it using kernel we can detect meaningful information that is of tens or hundreds of pixels.

  Hence we need to store fewer parameters. This not only reduces the memory requirement of the model but may also improve the statistical efficiency of the model.

Convolution Neural Networks

- **Parameter sharing.** If computing one feature at a spatial point (x_1, y_1) is useful then it should also be useful at some other spatial point say (x_2, y_2). It means that for creating one feature map, neurons are constrained to use the same set of weights.

  In a fully connected neural network, each element of the weight matrix is used once and then never revisited, while a CNN has shared parameters i.e., for getting output, weights applied to one input are the same as the weight applied elsewhere.

- **Locally invariant representation.** Due to the combinatin of convolution and pooling the layers, the features maps generated by a convolution neural network are locally equivariant to translation.
  That is, if we shift the input in a way, the output will also get changed in the same way (locally).

2.3 Transfer learning

Transfer learning

Transfer learning is a machine learning method that reuses a trained model designed for a particular task to accomplish a different yet related task.

In this process, the knowledge acquired from a given task is transferred to the second model that focuses on a new task.

Significance. Transfer learning speeds up the overall process of training a new model and consequently improves its performance. It is primarily used when a model requires large amount of resources and time for training.

Transfer learning

The transfer learning process consists of several steps.

- **1. Access a pre-trained models.** Pre-trained models may be available from open-source repositories.

  Specifically, PyTorch Hub is an open-source repository of pre-trained models (including model definitions and pre-trained weights) for tasks including object detection, image and video classification and more.

  Similarly, TensorFlow Hub is an open repository and reusable ML library with several pre-trained models that can be used for a multiplicity of tasks.

  Example: VGG-19 is a convolutional neural network that is 19 layers deep. A pretrained version of this network trained on more than a million images from the ImageNet database is widely availabe. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

Transfer learning

- **2. Freeze layers.** A typical neural network reveals three blocks: early, middle, and latter layers. In transfer learning, the early and middle layers are retained as they are, and **only the latter layers** are retrained so that the method can use the labeled data of the task that it was previously trained on.

  Intuitively, the earlier layers have learned to recognize "general" object features, so we only need to retrain the latter layers to adapt/tranfer this knowledge to the new task

  By freezing of this group of layers, we avoids the re-initialization of the weights in the model. The re-initialization step can cause the model to lose all its previous learning.

Transfer learning

- **3. Train new layers.** Upon freezing the requisite layers, new layers must be added to the model and trained to make new predictions on the latest dataset.
- **4. Fine-tune the model.** Fine-tuning the base model is not necessary. However, it usually improves the overall model performance. This process includes unfreezing some layers of the model and then retraining it at a low-learning rate to handle a new dataset.

Transfer learning

Example:

Transfer Learning for Computer Vision: A PyTorch Tutorial by Abdullah Meda

blog link

It demonstrates transfer learning using a **PyTorch's ConvNeXT** base model pre-trained on ImageNet to carry out a new classification task using a dataset of **Cats Breed Dataset**.

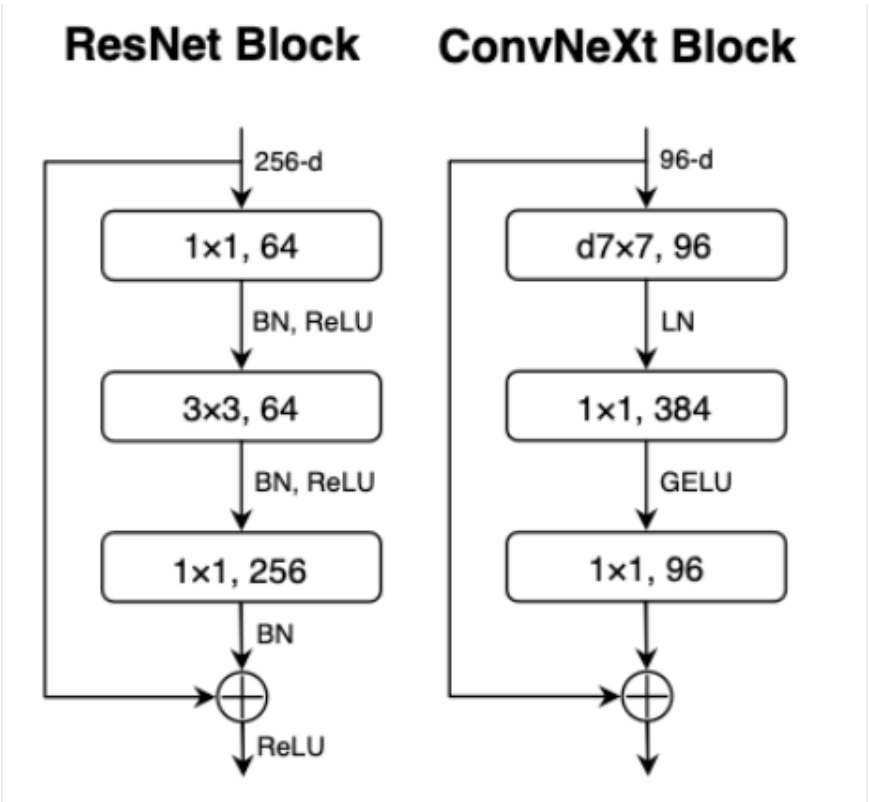
The blog presents two different fine-tuning strategies

1. **Full Fine Tuning.** Network is initialized with a pretrained network, trained on ImageNet (1000 classes). Rest of the training is as usual, that is all layers are trained.
2. **Partial Fine Tuning.** We freeze the weights for all of the network except that of the final fully connected layer. This last fully connected layer is replaced with a new one with random weights and only this layer is trained.

Transfer learning - ConvNeXt model

ConvNeXt was proposed in *A ConvNet for the 2020s* by Liu, Mao, Wu, Feichtenhofer, Darrell and Xie in 2022.

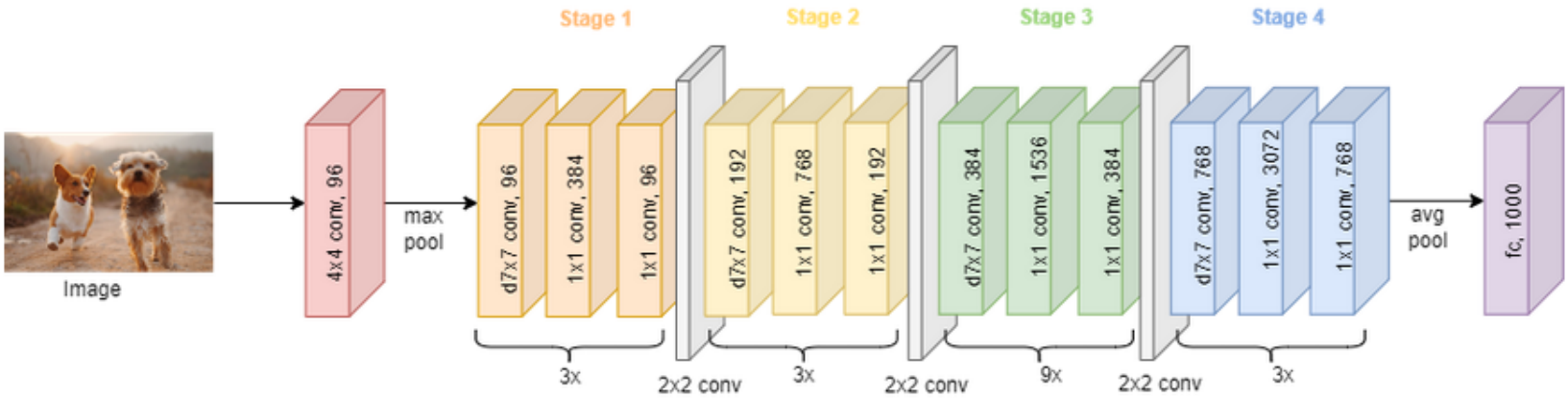
It is a pure convolutional model, that adapts the architecture of ResNeXt and is inspired by the design of **Vision Transformers**.



Note the large kernel size.

Transfer learning - ConvNeXT model

The ConvNeXt architecture consists of several ConvNeXt blocks (skip connections are not portrayed in figure)



Besides the network architecture, the training methodology also has a significant effect on the overall performance.
Vision Transformers introduced a new set of training techniques, like the **AdamW optimizer**. These changes pertain mostly to the optimization strategy and associated hyper-parameter settings.

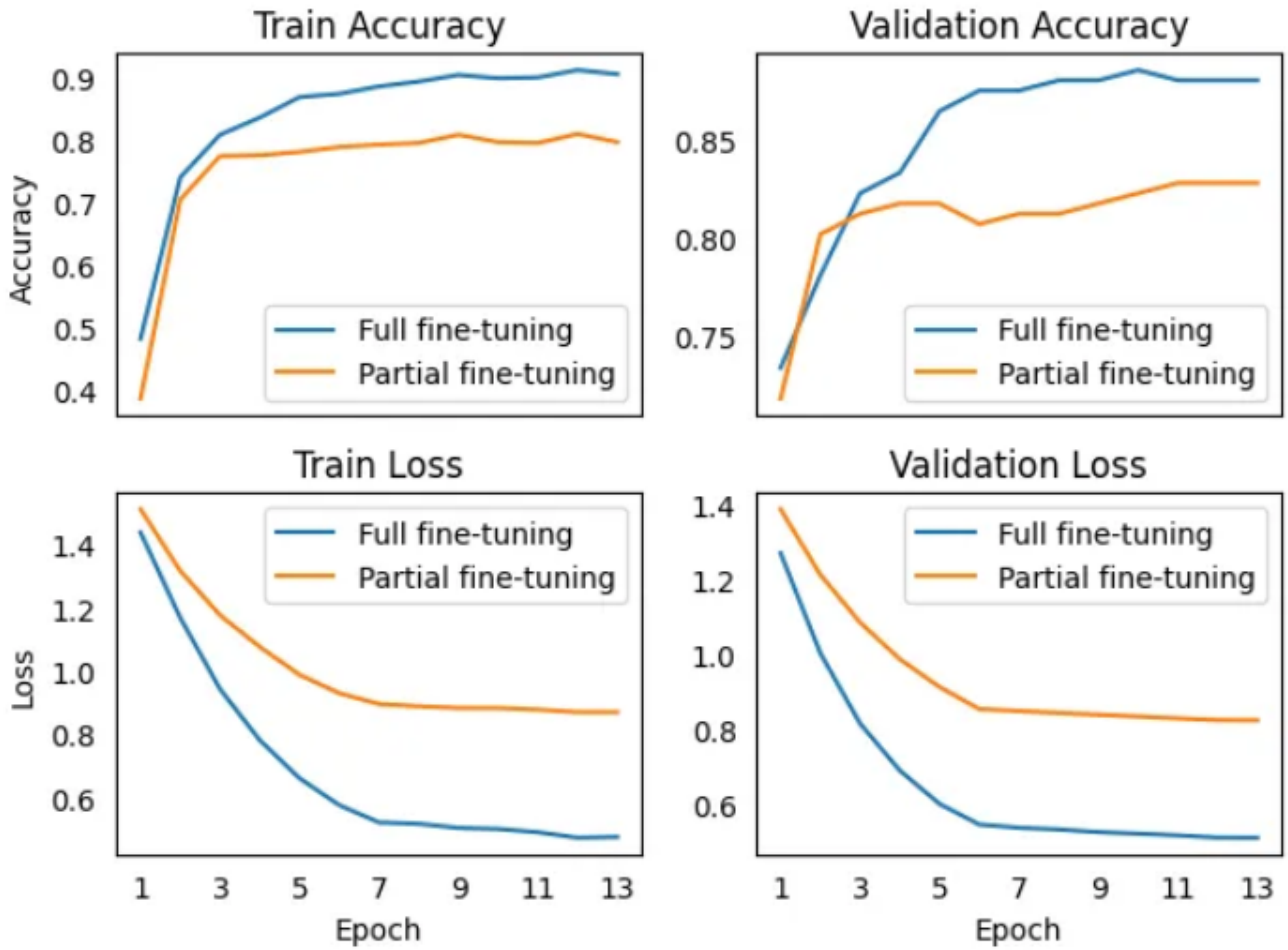
Transfer learning

The **Cats Breed Dataset** on Kaggle is a collection of images that are categorized into different breeds of cats.

Key characteristics:

- Number of Classes: The dataset comprises 5 different classes, each representing a unique breed of cat: Bengal, Domestic Short-hair, Maine Coon, Ragdoll, Siamese.
- Images: The dataset contains a total of 953 images. These images vary in size, shape, and quality. Each image is a colored image of a cat belonging to one of the 5 classes.
- File Format: The images are stored in JPG format.
- Directory Structure: The dataset follows a directory structure where the images of each breed are stored in a separate folder. The folder name corresponds to the breed name.

Transfer learning



Partial fine tuning rapidly (2 epochs) provides a satisfactory accuracy but full fine tuning eventually achieves better accuracy.

Transfer learning

There are a few additional things to keep in mind when performing Transfer Learning:

- **Constraints from pretrained models.** If you wish to use a pretrained network, you may be slightly constrained in terms of the architecture you can use for your new dataset. For example, you cannot arbitrarily take out Conv layers from the pretrained network. However, you can easily run a pretrained network on images of different spatial size.
- **Learning rates.** It is common to use a smaller learning rate for CNN weights that are being fine-tuned, in comparison to the (randomly-initialized) weights for the new linear classifier that computes the class scores of your new dataset. This is because we expect that the pretrained CNN weights are relatively good, so we don't wish to distort them too quickly and too much.

Transfer learning

Here are additional tutorials:

Transfer Learning using ResNet18

Transfer Learning using DenseNet121