

3. Convolutional Neural Networks

Introduction

Convolutional networks, also known as convolutional neural networks or CNNs, are a specialized kind of neural network for processing data that has a known, grid-like topology.

The name “convolutional neural network” indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

Convolutional neural network (CNN) is a deep learning neural network designed for processing structured arrays of data such as images. A CNN is a feed-forward neural network, often with up to 20 or 30 layers. The power of a convolutional neural network comes from a special kind of layer called the convolutional layer.

Convolutional neural network is also called ConvNet.

In simple terms, two images that are represented in the form of two matrices, are multiplied to provide an output that is used to extract information from the image

CNN represents the input data in the form of multidimensional arrays. It works well for a large number of labeled data. CNN extract the each and every portion of input image, which is known as receptive field. It assigns weights for each neuron based on the significant role of the receptive field.

Instead of preprocessing the data to derive features like textures and shapes, a CNN takes just the image's raw pixel data as input and "learns" how to extract these features, and ultimately infer what object they constitute.

The goal of CNN is to reduce the images so that it would be easier to process without losing features that are valuable for accurate prediction.

A convolutional neural network is made up of numerous layers, such as convolution layers, pooling layers, and fully connected layers, and it uses a back- propagation algorithm to learn spatial hierarchies of data automatically and adaptively.

To understand the Concept of Convolutional Neural Networks (CNNs), let us take an example of the images our brain can interpret.

As soon as we see an image, our brain starts categorizing it based on the colour, shape and sometimes also the message that image is conveying. Similar thing can be done through machines even after a rigorous training. But the difficulty is there is a huge difference in what humans interpret and what machine does. For a machine, the image is merely an array of pixels. There is a unique pattern included in each object present in the image and the computer tries to find out these patterns to get the information about the image.

Machines can be trained giving tons of images to increase its ability to recognize the objects included in a given input image.

Most of the digital companies have opted for CNNs for image recognition, some of these include Google, Amazon, Instagram, Interest, Facebook, etc.

Hence, we define a convolutional neural network as: "A neural network consisting of multiple convolutional layers which are used mainly for image processing, classification, segmentation and other correlated data".

Advantages and Disadvantages of CNN

Advantages

- CNN automatically detects the important features without any human supervision.
- CNN is also computationally efficient.
- Higher accuracy.
- Weight sharing is another major advantage of CNNs.
- Convolutional neural networks also minimize computation in comparison with a regular neural network.
- CNNs make use of the same knowledge across all image locations.

Disadvantages

- Adversarial attacks are cases of feeding the network 'bad' examples to cause misclassification.
- CNN requires lot of training data.
- CNNs tend to be much slower because of operations like maxpool.

Applications of CNN

CNN is mostly used for image classification, for example to determine the satellite images containing mountains and valleys or recognition of handwriting, etc. image segmentation, signal processing, etc. are the areas where CNN are used.

Object detection: Self-driving cars, AI-powered surveillance systems, and smart homes often use CNN to be able to identify and mark objects. CNN can identify objects on the photos and in real-time, classify, and label them.

Voice synthesis: Google Assistant's voice synthesizer uses Deepmind's WaveNet ConvNet model.

Astrophysics: They are used to make sense of radio telescope data and predict the probable visual image to represent that data.

The Convolution Operation

Convolution operation focuses on extracting/preserving important features from the input. Convolution operation allows the network to detect horizontal and vertical edges of an image and then based on those edges build high-level features in the following layers of neural network.

In general form, convolution is an operation on two functions of a real valued argument. To motivate the definition of convolution, we start with examples of two functions we might use.

Suppose we are tracking the location of a spaceship with a laser sensor. Laser sensor provides a single output $x(t)$, the position of the spaceship at time t . Both " x " and " t " are real-valued, i.e., we can get a different reading from the laser sensor at any instant in time.

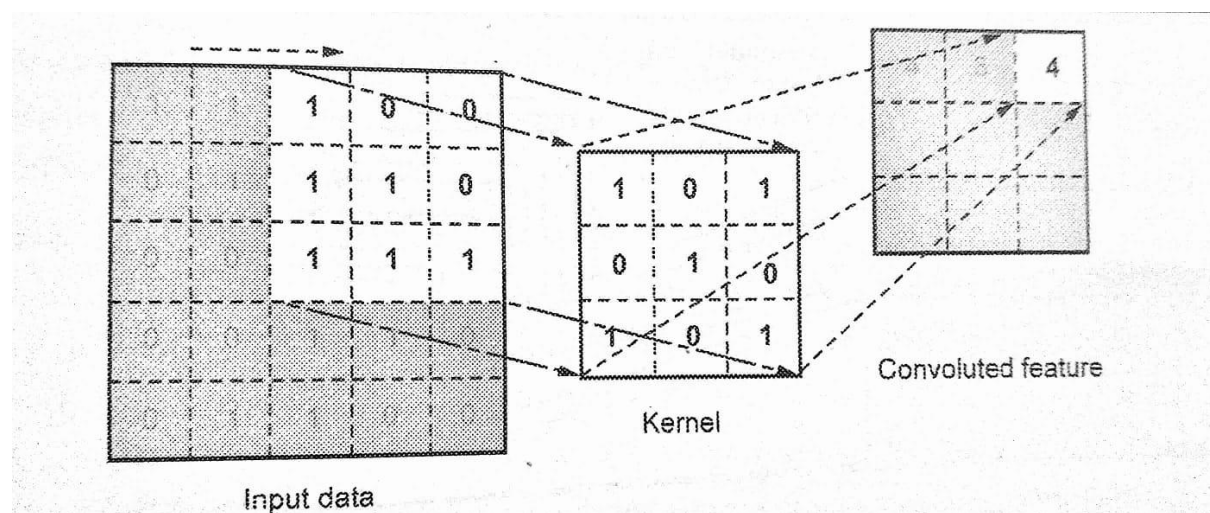
Now suppose that our laser sensor is somewhat noisy. To obtain a less noisy estimate of the spaceship's position, we would like to average together several measurements. Of course, more recent measurements are more relevant, so we will want this to be a weighted average that gives more weight to recent measurements.

We can do this a weighting function $w(a)$, where " a " is the age of a measurement. If we apply such a weighted average operation at every moment, we obtain a new function providing a smoothed estimate of the position " s " of the spaceship.

Convolution operation uses three parameters: Input image, Feature detector and Feature map.

Convolution operation involves an input matrix and a filter, also known as the kernel. Input matrix can be pixel values of a grayscale image whereas a filter is a relatively small matrix that detects edges by darkening areas of input image where there are transitions from brighter to darker areas. There can be different types of filters depending upon what type of features we want to detect, e.g. vertical, horizontal, or diagonal, etc.

Input image is converted into binary 1 and 0. The convolution operation shown in the figure is known as the feature detector of a CNN. The input to a convolution can be raw data or a feature map output from another convolution. It is often interpreted as a filter in which the kernel filters input data for certain kinds of information.



Sometimes a 5x5 or a 7-7 matrix is used as a feature detector. The feature detector is often referred to as a "kernel" or a "filter,". At each step, the kernel is multiplied by the input data values within its bounds, creating a single entry in the output feature map.

Generally, an image can be considered as a matrix whose elements are numbers between 0 and 255. The size of image matrix is: image height * image width * number of image channels.

A grayscale image has 1 channel, where a colour image has 3 channels.

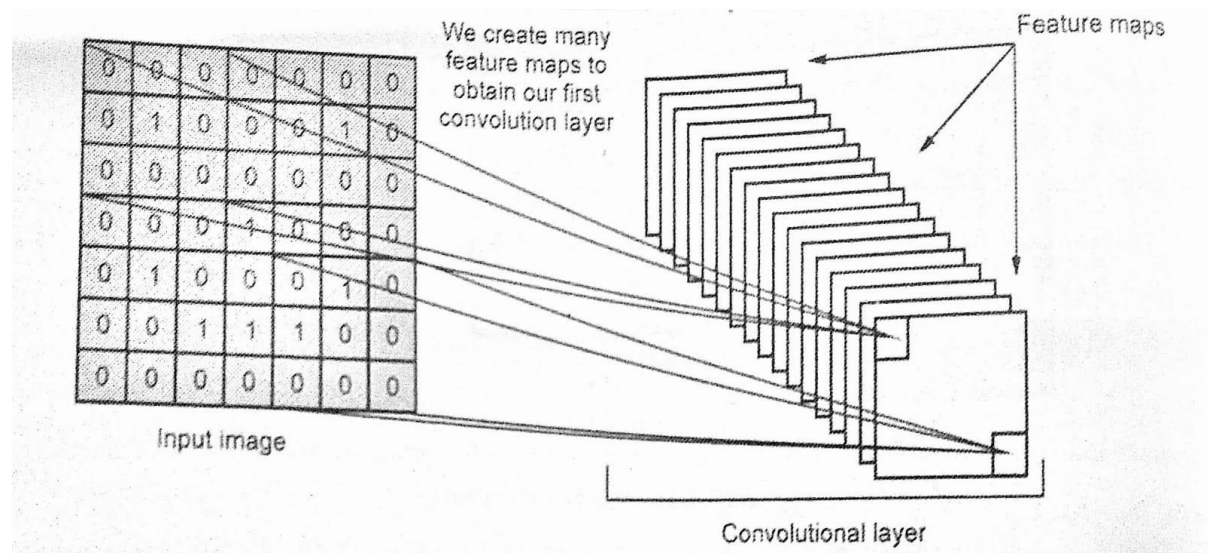
Kernel: A kernel is a small matrix of numbers that is used in image convolutions. Differently sized kernels containing different patterns of numbers produce different results under convolution. The size of a kernel is arbitrary but 3x3 is often used.

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 0 |

Fig. 2.2.2 Example of kernel

Convolutional layers perform transformations on the input data volume that are a function of the activations in the input volume and the parameters.

In reality, convolutional neural networks develop multiple feature detectors and use them to develop several feature maps which are referred to as convolutional layers.



Through training, the network determines what features it finds important in order for it to be able to scan images and categorize them more accurately.

Convolutional layers have parameters for the layer and additional hyper-parameters. Gradient descent is used to train the parameters in this layer such that the class scores are consistent with the labels in the training set.

Components of convolutional layers are as follows:

- Filters
- Activation maps
- Parameter sharing
- Layer-specific hyper-parameters

Filters are a function that has a width and height smaller than the width and height of the input volume. The filters are tensors, and they are used to convolve the input tensor when the tensor is passed to the layer instance. The random values inside the filter tensors are the weights of the convolutional layer.

Sliding each filter across the spatial dimensions (width, height) of the input volume during the forward pass of information through the CNN. This produces a two-dimensional output called an activation map for that specific filter.

The convolution operates on the **input** with a **kernel (weights)** to produce an **output map** given by:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} \underset{\substack{\uparrow \\ \text{input}}}{x(a)} \underset{\substack{\uparrow \\ \text{kernel}}}{w(t-a)}$$

1-D discrete convolution operation can be given by:

$$s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

1-D discrete convolution

2-D discrete convolution operation can be given by:

$$s(i, j) = \sum_m \sum_n x(m, n)w(i-m, j-n)$$

2-D discrete convolution

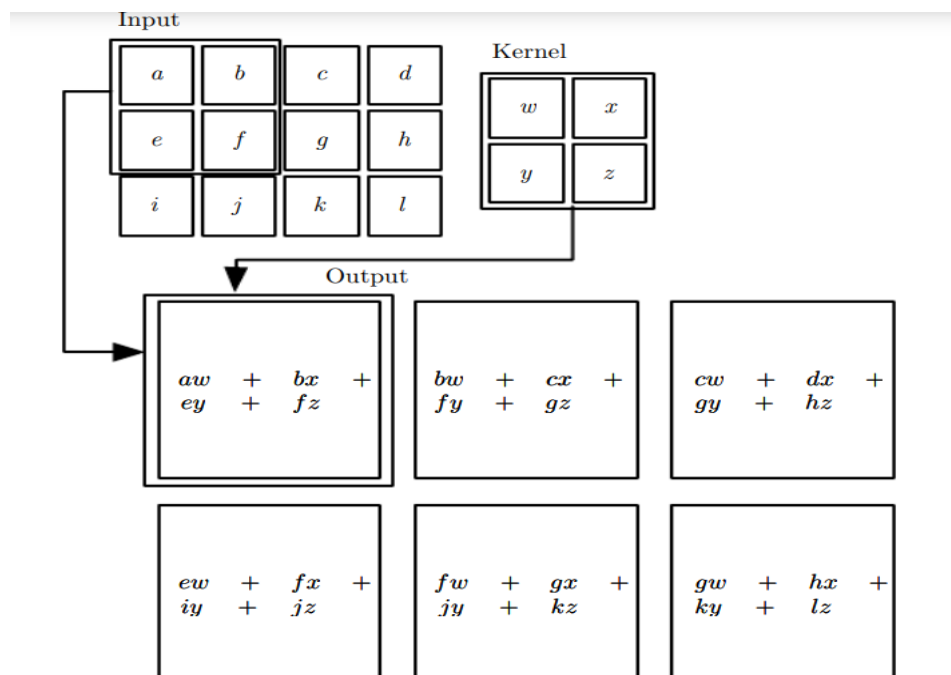


Fig. An example of 2-D convolution without kernel-flipping (Cross Correlation)

Properties of Convolution Operation and Cross-Correlation

Commutative Property

- Convolution operation is **commutative**.
- Commutative property arises because we have **flipped the kernel** relative to the input

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

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

Cross-Correlation

- Function which is analogous to convolution operation without flipping the kernel is called **cross-correlation operation**.
- Cross-correlation is **not commutative**.
- **Convolution operation:**

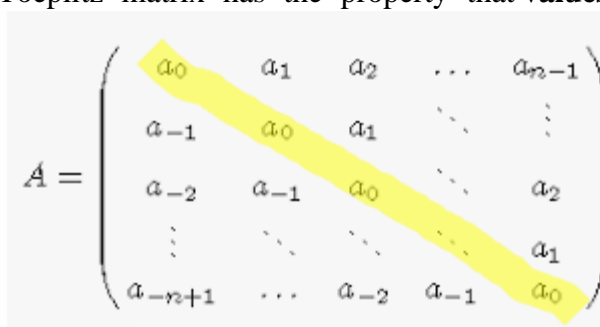
$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n) K(m, n).$$

- **Correlation operation:**

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

Toeplitz Matrix

- **1D** convolution operation can be represented as a **matrix vector product**.
- The kernel matrix is obtained by composing weights into a **Toeplitz matrix**.
- Toeplitz matrix has the property that **values along all diagonals are constant**.

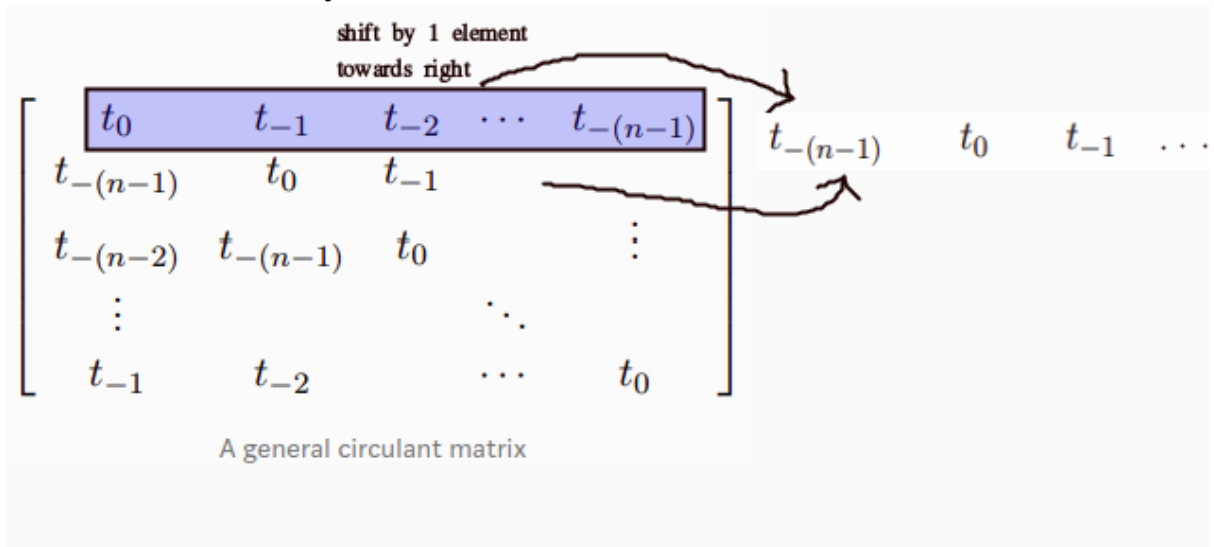


The general structure of a Toeplitz matrix

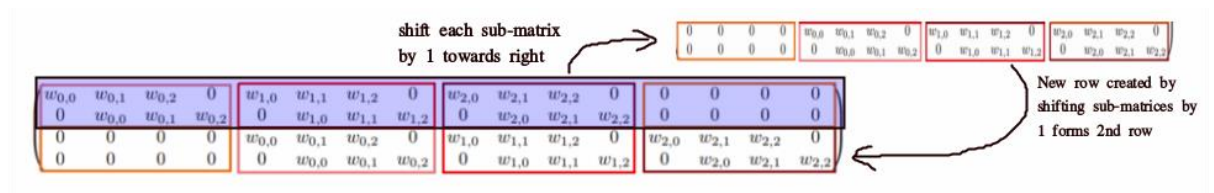
Block-Circulant and Doubly-Block-Circulant Matrix

- To **extend** the concept of Toeplitz matrix towards **2-D input**, we need to **convert 2-D input to 1-D vector**.

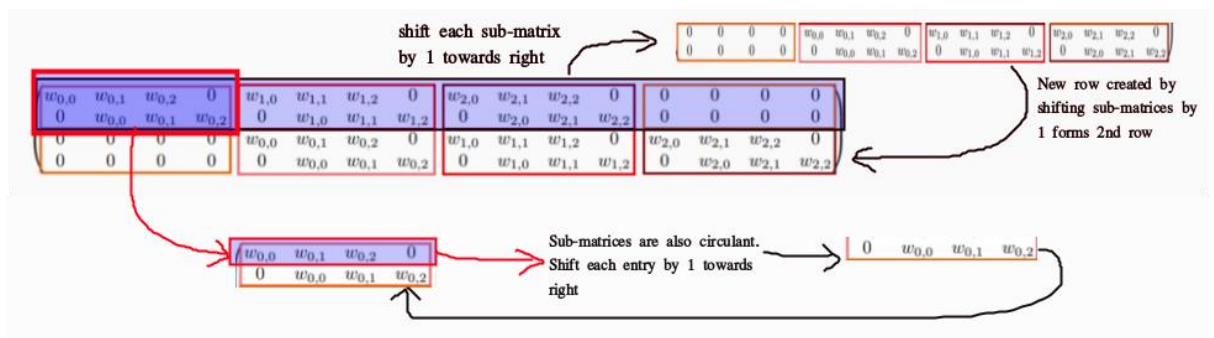
- **Kernel needs to be modified** as before but this time resulting in a **block-circulant matrix**.
- A **circulant matrix** is a special case of a **Toeplitz matrix** where each row is equal to the row above shifted by one element.



- A matrix which is **circulant with respect to its sub-matrices** is called a **block circulant matrix**.



- If each of the **submatrices is itself circulant**, the matrix is called **doubly block-circulant matrix**.



Motivation

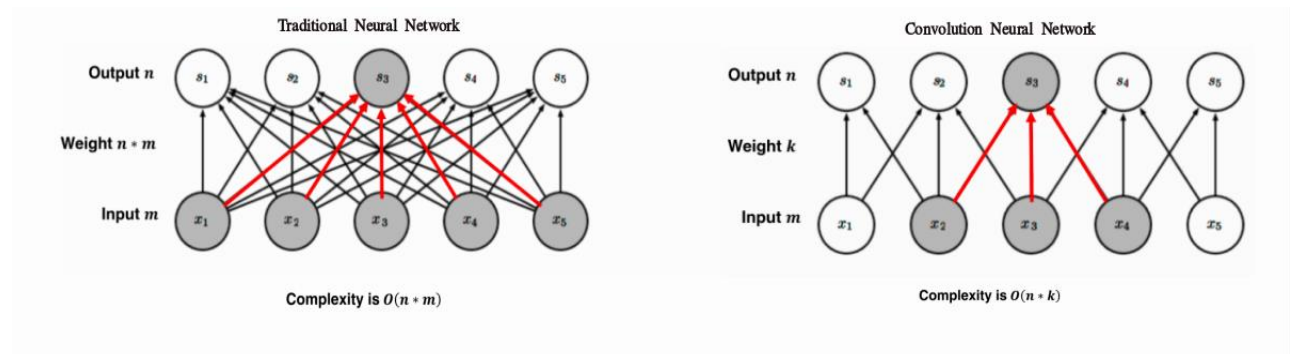
Convolution leverages three important ideas that can help improve a machine learning system: sparse interactions, parameter sharing and equivariant representations.

Sparse Interactions

Traditional neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit.

This means every output unit interacts with every input unit. Convolutional networks, however, typically have sparse interactions (also referred to as sparse connectivity or sparse weights). This is accomplished by making the kernel smaller than the input. For example, when processing an image, the input image might have thousands or millions of pixels, but we can detect small, meaningful features such as edges with kernels that occupy only tens or hundreds of pixels.

These improvements in efficiency are usually quite large. If there are m inputs and n outputs, then matrix multiplication requires $m \times n$ parameters and the algorithms used in practice have $O(m \times n)$ runtime (per example). If we limit the number of connections each output may have to k , then the sparsely connected approach requires only $k \times n$ parameters and $O(k \times n)$ runtime.



Parameter Sharing

Parameter sharing refers to using same parameter for more than one function in a model.

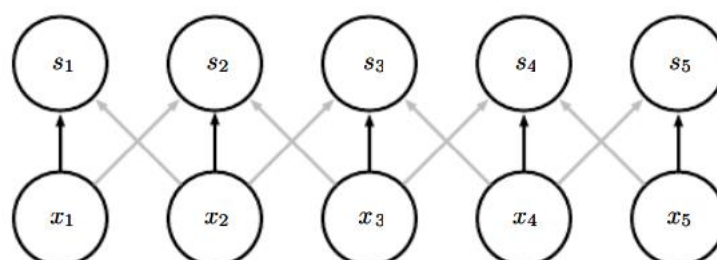
Parameter sharing is used in CNN to control the total parameter count. Convolutional layers reduce the parameter count further by using a technique called parameter sharing.

The user can reduce the number of parameters by making an assumption that if one feature can compute at some spatial position (x_1), then it is useful to compute a different place (x_2, y_2).

In other words, denoting a single 2D slice of depth as a depth slice. For example, during back-propagation, every neuron in the network will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.

If all neurons in a single depth slice are using the same weight vector, then the forward pass of the convolutional layer can be computed in each depth slice as a convolution of the neuron's weights with the input volume. This is the reason why it is common to refer to the sets of weights as a filter (or a kernel), that is convolved with the input.

Fig. shows convolution shares the same parameters across all spatial locations.



Equivariant Representation

Convolution function is equivariant to translation. This means that shifting the input and applying convolution is equivalent to applying convolution to the input and shifting it.

If we move the object in the input, its representation will move the same amount in the output.

A function f is said to be equivariant to a function g if

$$f(g(x)) = g(f(x))$$

i.e. if input changes, the output changes in the same way.

Convolution is equivariant to translation. This is a direct consequence of parameter sharing.

It is useful when detecting structures that are common in the input. For example, edges in an image.

Equivariance in early layers is good. We are able to achieve translation-invariance (via max-pooling) due to this property.

- Example of equivariance: With 2D images convolution creates a map where certain features appear in the input. If we move the object in the input, the representation will move the same amount in the output. It is useful to detect edges in first layer of convolutional network. Same edges appear everywhere in image, so it is practical to share parameters across entire image.

Pooling

A typical layer of a convolutional network consists of three stages. In the first stage, the layer performs several convolutions in parallel to produce a set of linear activations. In the second stage, each linear activation is run through a nonlinear activation function, such as the rectified linear activation function. This stage is sometimes called the detector stage. In the third stage, we use a pooling function to modify the output of the layer further.

A pooling function replaces the output of net at a certain location with summary statistic of nearby outputs.

Common summary statistics are: mean, median, weighted average.

The pooling layer operates upon each feature map separately to create a new set of the same number of pooled feature maps. Pooling involves selecting a pooling operation, much like a filter to be applied to feature maps.

The size of the pooling operation or filter is smaller than the size of the feature map. This means that the pooling layer will always reduce the size of each feature map by a factor of 2, e.g. each dimension is halved, reducing the number of pixels or values in each feature map to one quarter the size.

For example, a pooling layer applied to a feature map of 6×6 (36 pixels) will result in an output pooled feature map of 3×3 (9 pixels). The pooling operation is specified, rather than learned.

The pooling operation, also called subsampling is used to reduce the dimensionality of feature maps from the convolution operation. Max pooling and average pooling are the most common pooling operations used in the CNN.

Pooling units are obtained using functions like max-pooling, average pooling and even L2-norm pooling. At the pooling layer, forward propagation results in an N-N pooling block being reduced to a single value- value of the "winning unit". Back- propagation of the pooling layer then computes the error which is acquired by this single value "winning unit",

Pooling layers, also known as down sampling, conducts dimensionality reduction, reducing the number of parameters in the input. Similar to the convolutional layer, the pooling operation sweeps a filter across the entire input, but the difference is that this filter does not have any weights. Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array. There are two main types of pooling:

Max pooling: As the filter moves across the input, it selects the pixel with the maximum value to send to the output array. As an aside, this approach tends to be used more often compared to average pooling.

Average pooling: As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.

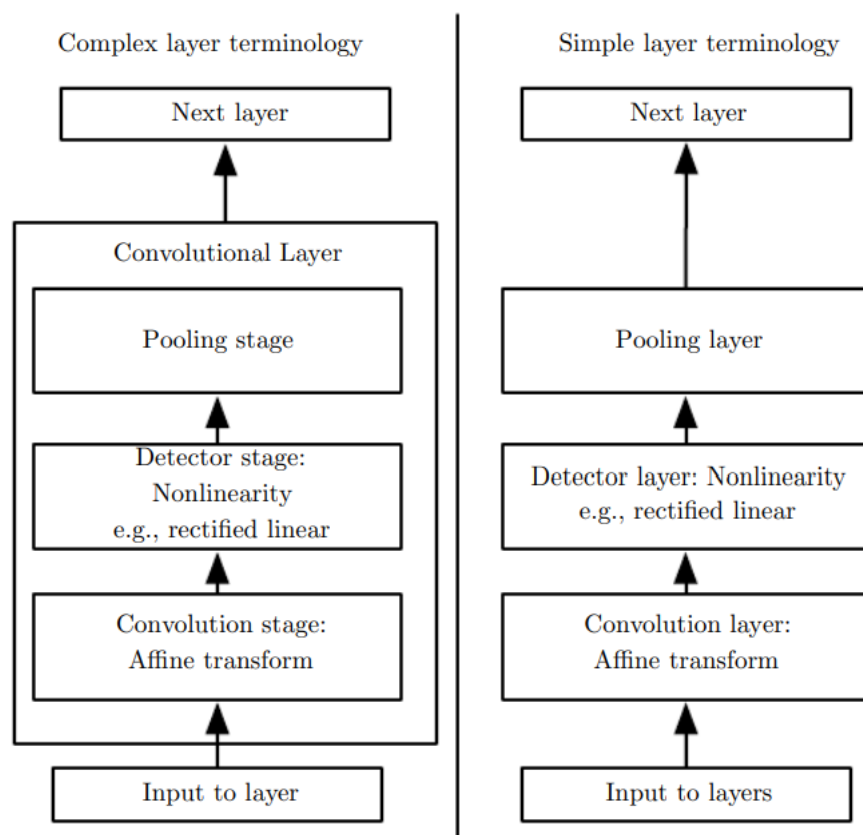
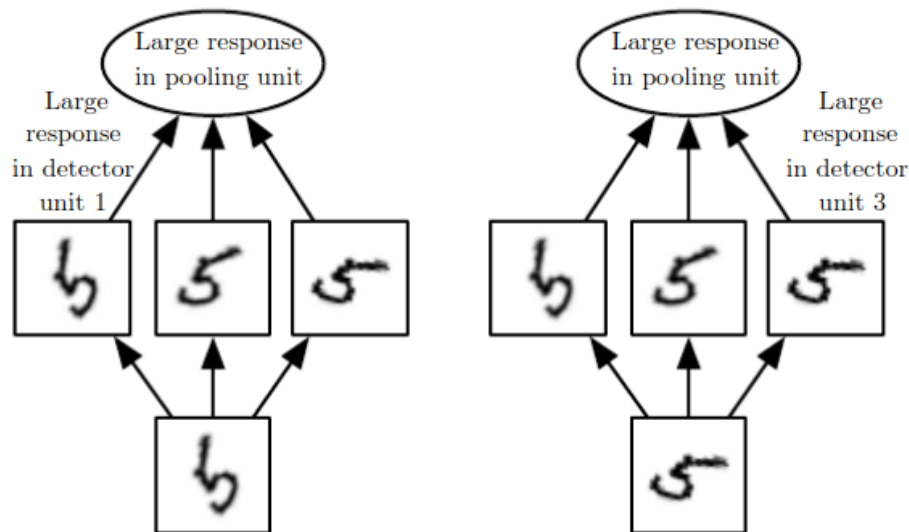


Fig. The components of a typical convolutional neural network layer.

Learned Invariances

A pooling unit that pools over multiple features that are learned with separate parameters can learn to be invariant to transformations of the input. Here we show how a set of three learned filters and a max pooling unit can learn to become invariant to rotation. All three filters are intended to detect a hand-written 5. Each filter attempts to match a slightly different orientation of the 5. When a 5 appears in the input, the corresponding filter will match it and cause a large activation in a detector unit. The max pooling unit then has a large activation regardless of which detector unit was activated. We show here how the network processes two different inputs, resulting in two different detector units being activated.



Convolution and Pooling as an Infinitely Strong Prior

Priors can be considered weak or strong depending on how concentrated the probability density in the prior is. A weak prior is a prior distribution with high entropy, such as a Gaussian distribution with high variance. Such a prior allows the data to move the parameters more or less freely. A strong prior has very low entropy, such as a Gaussian distribution with low variance. Such a prior plays a more active role in determining where the parameters end up.

An infinitely strong prior places zero probability on some parameters and says that these parameter values are completely forbidden, regardless of how much support the data gives to those values.

We can imagine a convolutional net as being similar to a fully connected net, but with an infinitely strong prior over its weights. This infinitely strong prior says that the weights for one hidden unit must be identical to the weights of its neighbor, but shifted in space. The prior also says that the weights must be zero, except for in the small, spatially contiguous receptive field assigned to that hidden unit.

Convolution imposes an infinitely strong prior by making the following restrictions on weights:

- Adjacent units must have the same weight but shifted in space.
- Except for a small spatially connected region, all other weights must be zero.

Variants of the Basic Convolution Function

In practical implementations of the convolution operation, certain modifications are made which deviate from the discrete convolution operation:

- In general a convolution layer consists of application of several different kernels to the input. This allows the extraction of several different features at all locations in the input. This means that in each layer, a single kernel (filter) isn't applied. Multiple kernels (filters), usually a power of 2, are used as different feature detectors.
- The input is generally not real-valued but instead vector valued (e.g. RGB values at each pixel or the feature values computed by the previous layer at each pixel position). Multi-channel convolutions are commutative only if number of output and input channels is the same.
- In order to allow for calculation of features at a **coarser level** *strided convolutions* can be used. The effect of strided convolution is the same as that of a convolution followed by a downsampling stage. This can be used to reduce the representation size.

Effect of Strides

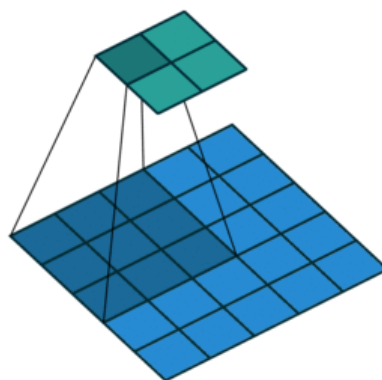
Stride is the number of pixels shifts over the input matrix.

In order to allow for calculation of features at a coarser level strided convolutions can be used.

The effect of strided convolution is the same as that of a convolution followed by a downsampling stage.

Strides can be used to reduce the representation size.

Below is an example representing 2-D Convolution, with $(3 * 3)$ Kernel and Stride of 2 units.



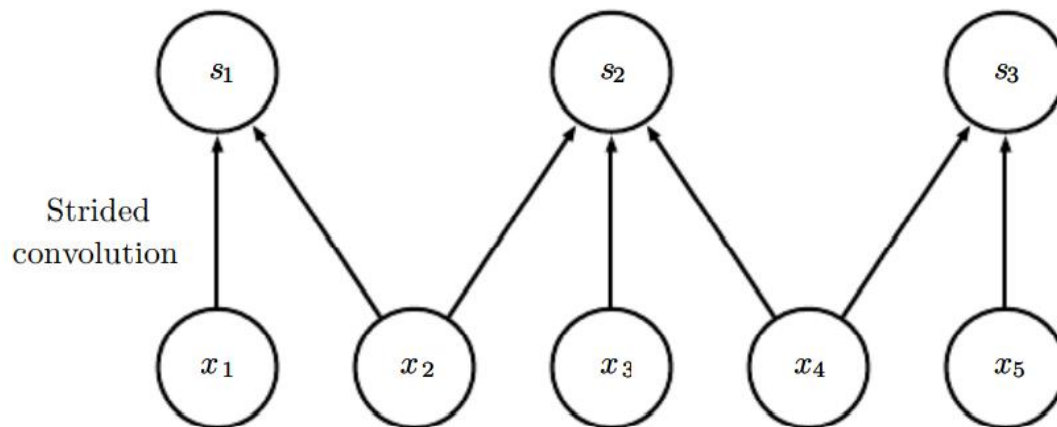


Fig. Convolution with a stride length of two implemented in a single operation.

Effect of Zero Padding

Convolution networks can implicitly zero pad the input V , to make it wider.

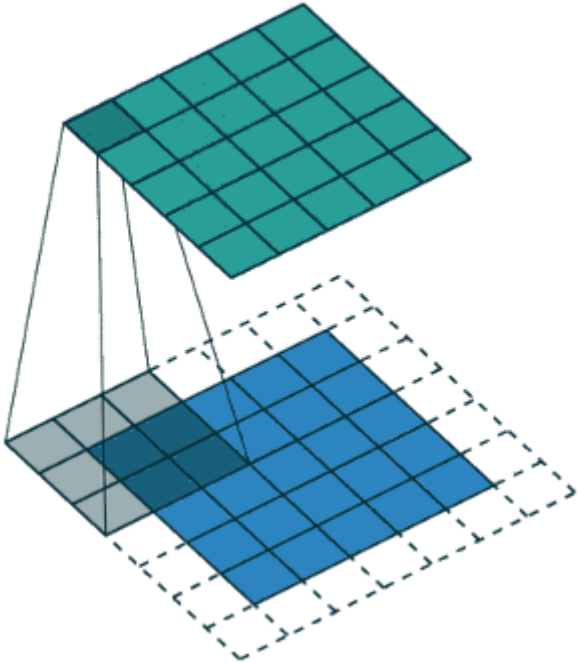
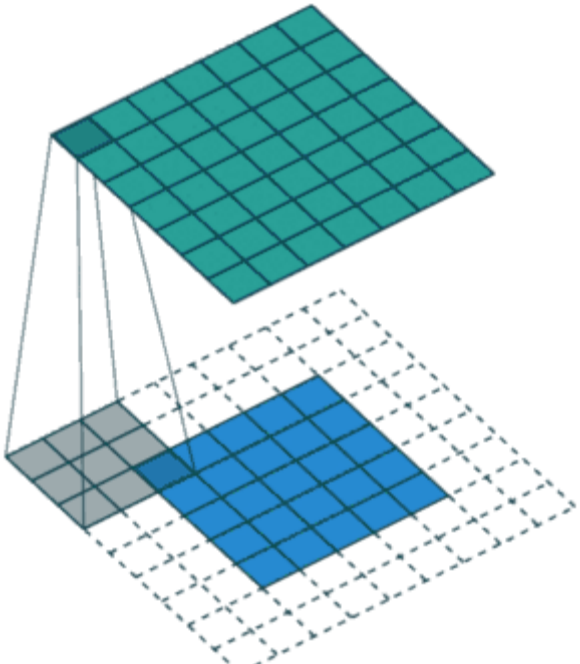
Without zero padding, the width of representation shrinks by one pixel less than the kernel width at each layer.

Zero padding the input allows to control kernel width and size of output independently.

Zero Padding Strategies

3 common zero padding strategies are:

| Zero Padding Type | Properties | Example |
|---------------------------|--|---------|
| Valid Zero-Padding | <ol style="list-style-type: none"> 1. No zero padding is used. 2. Output is computed only at places where entire kernel lies inside the input. 3. Shrinkage > 0 4. Limits #convolution layers to be used in network 5. Input's width = m, Kernel's width = k, Width of Output = $m-k+1$ | |

| | | |
|----------------------------|--|---|
| Same Zero-Padding | <ol style="list-style-type: none"> 1. Just enough zero padding is added to keep: 1.a. $\text{Size(Outpu)} = \text{Size(Input)}$ 2. Input is padded by (k-1) zeros 3. Since the #output units connected to border pixels is less than that for centre pixels, it may under-represent border pixels. 4. Can add as many convolution layers as hardware can support 5. Input's width = m, Kernel's width = k, Width of Output = m |  |
| Strong Zero-Padding | <ol style="list-style-type: none"> 1. The input is padded by enough zeros such that each input pixel is connected to same #output units. 2. Allows us to make an arbitrarily deep NN. 3. Can add as many convolution layers as hardware can support 4. Input's width = m, Kernel's width = k, Width of Output = m+k-1 |  |

Types of Convolution

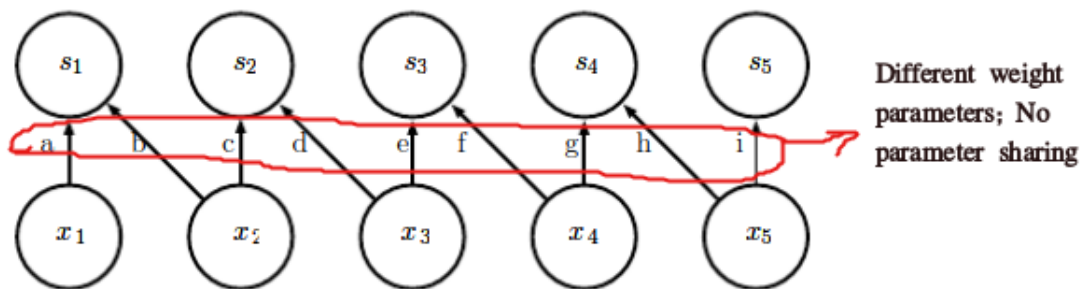
Comparing Unshared, Tiled and Traditional Convolutions

| Convolution Type | Properties | Advantages and Disadvantages |
|------------------|------------|------------------------------|
|------------------|------------|------------------------------|

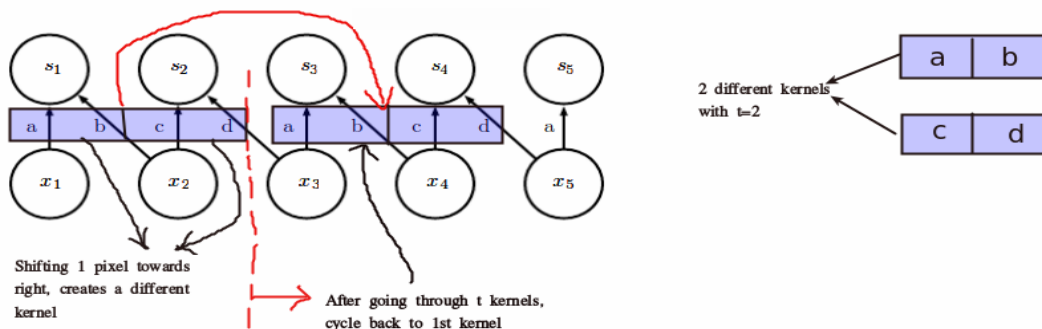
| | | |
|-------------------------|--|---|
| Unshared Convolution | <ol style="list-style-type: none"> 1. No Parameter sharing. 2. Each output unit performs a linear operation on its neighbourhood but parameters are not shared across output units. 3. Captures local connectivity while allowing different features to be computed at different spatial locations. | <p>Advantages</p> <ol style="list-style-type: none"> 1. Reducing memory consumption 2. Increasing statistical efficiency 3. Reducing the amount of computation needed to perform forward and back-propagation. <p>Disadvantages</p> <ol style="list-style-type: none"> 1. requires much more parameters than the convolution operation. |
| Tiled Convolution | <ol style="list-style-type: none"> 1. Offers a compromise b/w unshared and traditional convolution. 2. Learn a set of kernels and cycle/rotate them through space. 3. Makes use of parameter sharing. | <p>Advantages</p> <ol style="list-style-type: none"> 1. Reduces the parameters in model. |
| Traditional Convolution | <ol style="list-style-type: none"> 1. Equivalent to tiled convolution with $t=1$. 2. Has the same connectivity as unshared convolution | |

Examples of Unshared, Tiled and Traditional Convolutions

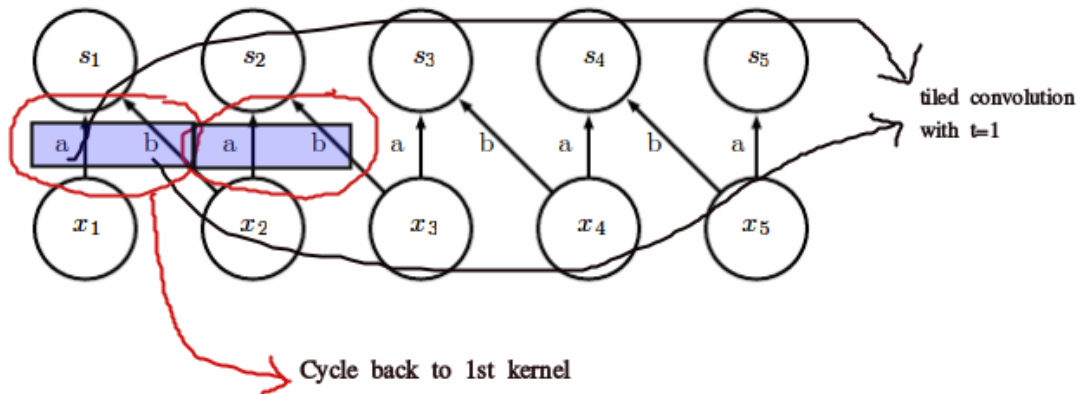
Unshared Convolution



Tiled Convolution



Traditional Convolution



Comparing Computation Times

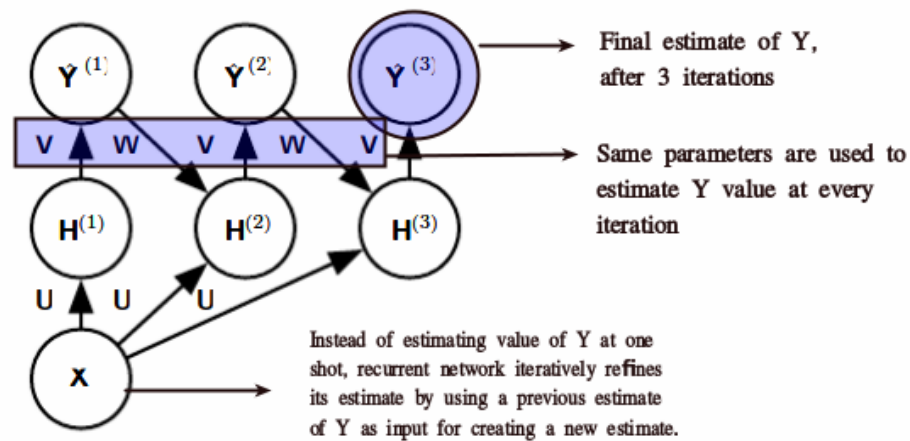
| Type | Computations | Parameters | |
|-------------------|--------------|------------|--|
| Fully connected | $O(mn)$ | $O(mn)$ | • m = number of input units |
| Locally connected | $O(kn)$ | $O(kn)$ | • n = number of output units |
| Tiled | $O(kn)$ | $O(kl)$ | • k = kernel size |
| Traditional | $O(kn)$ | $O(k)$ | • l = number of kernels in the set (for tiled convolution) |

Structured Outputs

Convolutional networks can be trained to output high-dimensional structured output rather than just a classification score. A good example is the task of image segmentation where each pixel needs to be associated with an object class. Here the output is the same size (spatially) as the input. The model outputs a tensor S where $S[i,j,k]$ is the probability that pixel (j,k) belongs to class i .

To produce an output map as the same size as the input map, only **same-padded** convolutions can be stacked. Alternatively, a coarser segmentation map can be obtained by allowing the output map to shrink spatially.

The output of the first labelling stage can be refined successively by another convolutional model. If the models use tied parameters, this gives rise to a type of **recursive model** as shown below. (H^1, H^2, H^3 share parameters)



| Variable | Description |
|----------|--|
| X | Input image tensor |
| Y | Probability distribution over tensor for each pixel |
| H | Hidden representation |
| U | Tensor of convolution kernels |
| V | Tensor of kernels to produce estimation of lables |
| W | Kernel tensor to convolve over Y to provide input to H |

Data Types

The data used with a convolutional network usually consist of several channels, each channel being the observation of a different quantity at some point in space or time.

One advantage to convolutional networks is that they can also process inputs with varying spatial extents.

When the output is accordingly variable sized, no extra design change needs to be made. If however the output is fixed sized, as in the classification task, a pooling stage with kernel size proportional to the input size needs to be used.

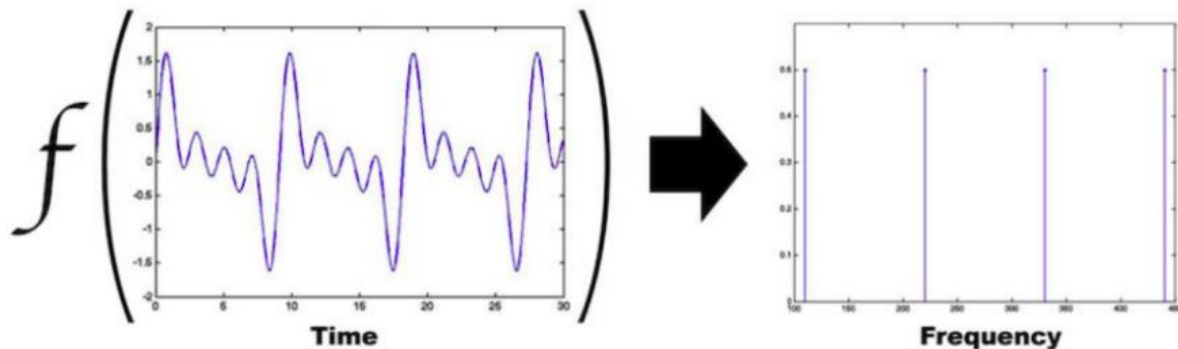
| Dimensions | Single channel | Multichannel |
|------------|---|---|
| 1D | Raw audio (single amplitude value per time point) | Skeleton animatin data (orientation of each joint) |
| 2D | Audio spectrogram (one FFT coefficient per time point per frequency) | Color image (RGB triplet per (x,y) tuple) |
| 3D | CT scan (one value per (x,y,z) tuple) | Color video (one RGB triplet per (x,y) tuple per time instant) |

Different data types based on the number of spatial dimensions and channels

Efficient Convolution Algorithms

Fourier Transform

The Fourier Transform is a tool that breaks a waveform (a function or signal) into an alternate representation, characterized by sine and cosines.



Separable Kernels

- Convolution is equivalent to converting both **input and kernel** to frequency domain using a **Fourier transform**, performing **point-wise multiplication of two signals**:

$$F\{f \star g\} = F\{f\} \times F\{g\}$$

- Converting back to **time domain** using an **inverse Fourier transform**.

$$f \star g = F^{-1}\{F\{f \star g\}\} = F^{-1}\{F\{f\} \times F\{g\}\}$$

- When a **d-dimensional kernel** can be expressed as **outer product of d vectors**, one vector per dimension, the kernel is called **separable**.
- The kernel also takes **fewer parameters** to represent as vectors.

| Kernel Type | Runtime complexity for <i>d</i> -dimensional kernel with <i>w</i> elements wide |
|--------------------|---|
| Traditional Kernel | $O(w^d)$ |
| Separable Kernel | $O(w \times d)$ |

Random or Unsupervised Features

To reduce the computational cost of training the CNN, we can use features not learned by supervised training.

- Random initialization** has been shown to create filters that are *frequency selective and translation invariant*. This can be used to inexpensively select the model architecture.

Randomly initialize several CNN architectures and just train the last classification layer. Once a winner is determined, that model can be fully trained in a supervised manner.

2. **Hand designed kernels** may be used; e.g. to detect edges at different orientations and intensities.
3. **Unsupervised training** of kernels may be performed; e.g. *applying k-means clustering to image patches* and using the centroids as convolutional kernels. *Unsupervised pre-training may offer regularization effect (not well established)*. It may also allow for training of larger CNNs because of reduced computation cost.

Another approach for CNN training is **greedy layer-wise pretraining** most notably used in *convolutional deep belief network*. For example, in the case of multi-layer perceptrons, starting with the first layer, each layer is trained in isolation. Once the first layer is trained, its output is stored and used as input for training the next layer, and so on.

The Neuroscientific Basis for Convolutional Networks

Hubel and Wiesel studied the activity of neurons in a cat's brain in response to visual stimuli. Their work characterized many aspects of brain function.

In a simplified view, we have:

1. The light entering the eye stimulates the **retina**. The image then passes through the optic nerve and a region of the brain called the **LGN (lateral geniculate nucleus)**
2. **V1 (primary visual cortex)**: The image produced on the retina is transported to the V1 with minimal processing. The properties of V1 that have been replicated in CNNs are:
 - The V1 response is localized spatially, i.e. the upper image stimulates the cells in the upper region of V1 [**localized kernel**].
 - V1 has simple cells whose activity is a linear function of the input in a small neighbourhood [**convolution**].
 - V1 has complex cells whose activity is invariant to shifts in the position of the feature [**pooling**] as well as some changes in lighting which cannot be captured by spatial pooling [**cross-channel pooling**].
3. There are several stages of V1 like operations [**stacking convolutional layers**].
4. In the **medial temporal lobe**, we find **grandmother cells**. These cells respond to specific concepts and are invariant to several transforms of the input. In the **medial temporal lobe**, researchers also found neurons spiking on a particular concept, e.g. the **Halle Berry neuron** fires when looking at a photo/drawing of Halle Berry or even reading the text Halle Berry. Of course, there are neurons which spike at other concepts like Bill Clinton, Jennifer Aniston, etc.

Some of the major differences between the human visual system (HVS) and the CNN model are:

- The human eye is low resolution except in a region called **fovea**. Essentially, the eye does not receive the whole image at high resolution but stitches several patches through eye movements called **saccades**. This attention based gazing of the input image is an active research problem. *Note:* attention mechanisms have been shown to work on natural language tasks.
- Integration of several senses in the HVS while CNNs are only visual.
- The HVS processes rich 3D information, and can also determine relations between objects. CNNs for such tasks are in their early stages.
- The feedback from higher levels to V1 has not been incorporated into CNNs with substantial improvement.
- While the CNN can capture firing rates in the IT, the similarity between intermediate computations is not established. The brain probably uses different activation and pooling functions. Even the linearity of filter response is doubtful as recent models for V1 involve quadratic filters.

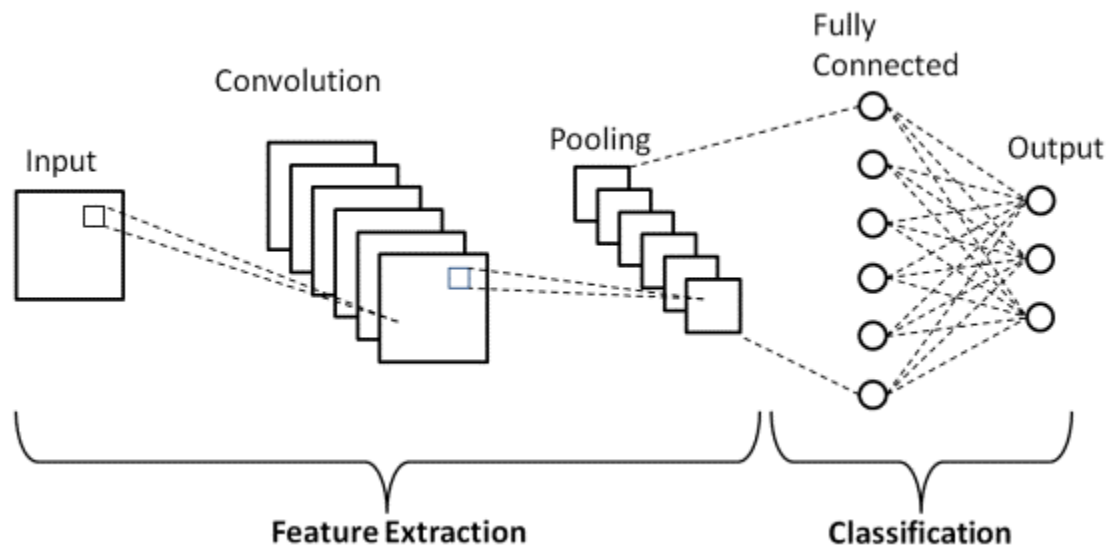
Neuroscience tells us very little about the training procedure. Backpropagation which is a standard training mechanism today is not inspired by neuroscience and sometimes considered biologically implausible.

In order to determine the filter parameters used by neurons, a process called **reverse correlation** is used. The neuron activations are measured by an electrode when viewing several white noise images and a linear model is used to approximate this behaviour. It has been shown experimentally that the weights of the fitted model of V1 neurons are described by **Gabor functions**. If we go by the simplified version of the HVS, if the simple cells detect Gabor-like features, then complex cells learn a function of simple cell outputs which is invariant to certain translations and magnitude changes.

Explain CNN Architecture.

There are two main parts to a CNN architecture

- A convolution tool that separates and identifies the various features of the image for analysis in a process called as Feature Extraction.
- The network of feature extraction consists of many pairs of convolutional or pooling layers.
- A fully connected layer that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.
- This CNN model of feature extraction aims to reduce the number of features present in a dataset. It creates new features which summarises the existing features contained in an original set of features. There are many CNN layers as shown in the CNN architecture diagram.



Convolution Layers

There are three types of layers that make up the CNN which are the convolutional layers, pooling layers, and fully-connected (FC) layers. When these layers are stacked, a CNN architecture will be formed. In addition to these three layers, there are two more important parameters which are the dropout layer and the activation function which are defined below.

1. Convolutional Layer

This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size $M \times M$. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter ($M \times M$).

The output is termed as the Feature map which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image.

The convolution layer in CNN passes the result to the next layer once applying the convolution operation in the input. Convolutional layers in CNN benefit a lot as they ensure the spatial relationship between the pixels is intact.

2. Pooling Layer

In most cases, a Convolutional Layer is followed by a Pooling Layer. The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending upon method used, there are several types of Pooling operations. It basically summarises the features generated by a convolution layer.

In Max Pooling, the largest element is taken from feature map. Average Pooling calculates the average of the elements in a predefined sized Image section. The total sum of the elements in the predefined section is computed in Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional Layer and the FC Layer.

This CNN model generalises the features extracted by the convolution layer, and helps the networks to recognise the features independently. With the help of this, the computations are also reduced in a network.

3. Fully Connected Layer

The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture.

In this, the input image from the previous layers are flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place. The reason two layers are connected is that two fully connected layers will perform better than a single connected layer. These layers in CNN reduce the human supervision

4. Dropout

Usually, when all the features are connected to the FC layer, it can cause overfitting in the training dataset. Overfitting occurs when a particular model works so well on the training data causing a negative impact in the model's performance when used on a new data.

To overcome this problem, a dropout layer is utilised wherein a few neurons are dropped from the neural network during training process resulting in reduced size of the model. On passing a dropout of 0.3, 30% of the nodes are dropped out randomly from the neural network.

Dropout results in improving the performance of a machine learning model as it prevents overfitting by making the network simpler. It drops neurons from the neural networks during training.

5. Activation Functions

Finally, one of the most important parameters of the CNN model is the activation function. They are used to learn and approximate any kind of continuous and complex relationship between variables of the network. In simple words, it decides which information of the model should fire in the forward direction and which ones should not at the end of the network.

It adds non-linearity to the network. There are several commonly used activation functions such as the ReLU, Softmax, tanH and the Sigmoid functions. Each of these functions have a specific usage. For a binary classification CNN model, sigmoid and softmax functions are preferred for a multi-class classification, generally softmax is used. In simple terms, activation functions in a CNN model determine whether a neuron should be activated or not. It decides whether the input to the work is important or not to predict using mathematical operations.