An intuitive guide to Convolutional Neural Networks

Daphne Cornelisse Follow
Apr 24, 2018 · 11 min read

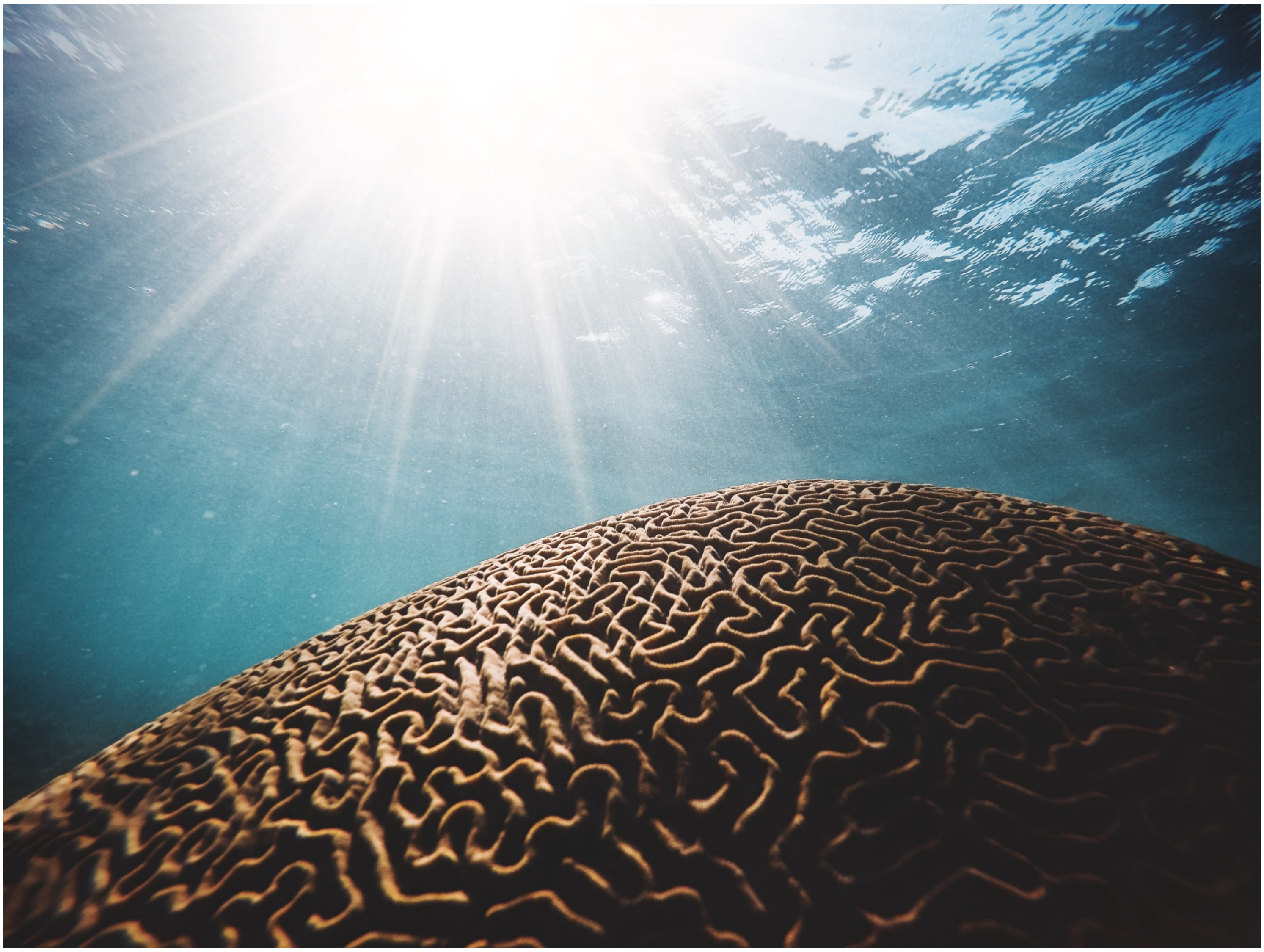
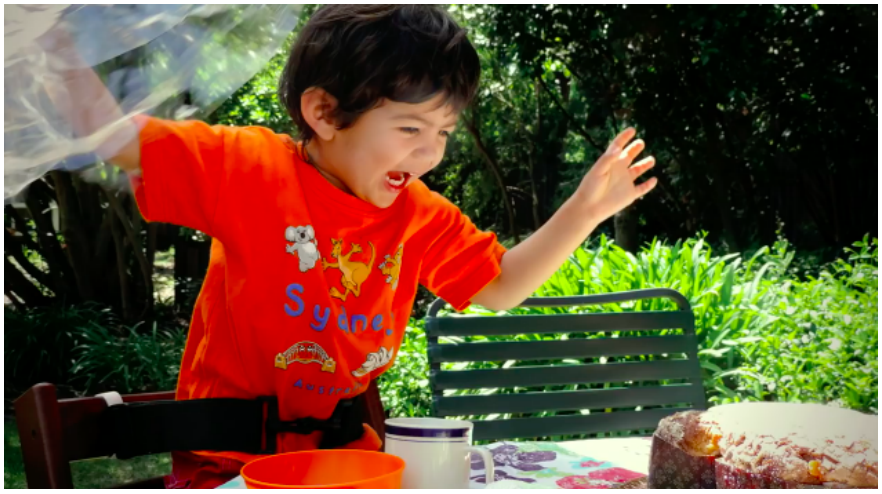


Photo by Daniel Hjalmarsson on Unsplash

In this article, we will explore Convolutional Neural Networks (CNNs) and, on a high level, go through how they are inspired by the structure of the brain. If you want to read more about the brain specifically, there are more resources at the end of the article to help you further.

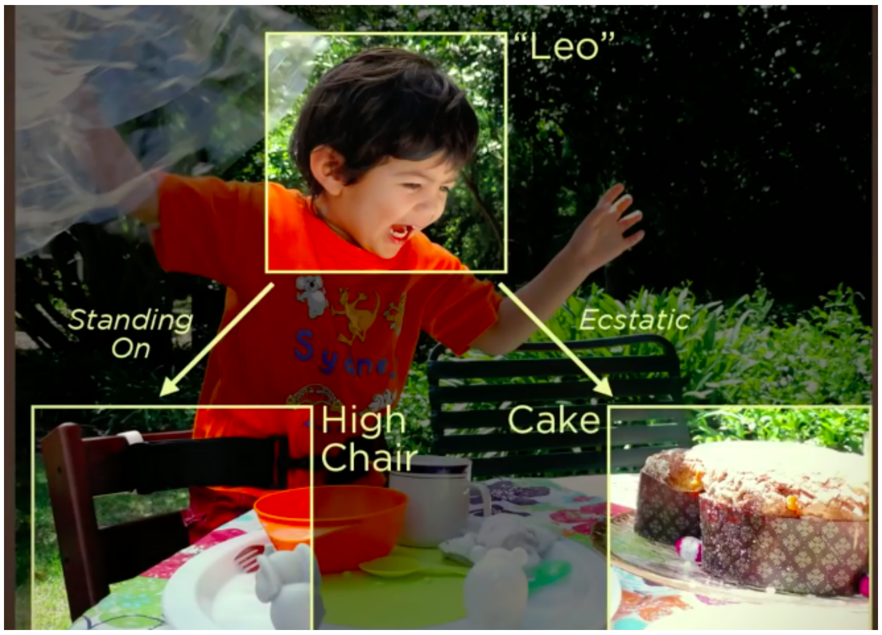
The Brain

We are constantly analysing the world around us. Without conscious effort, we make predictions about everything we see, and act upon them. When we see something, we label every object based on what we have learned in the past. To illustrate this, look at this picture for a moment.



Source: https://www.youtube.com/watch?v=40riCqvRoMs

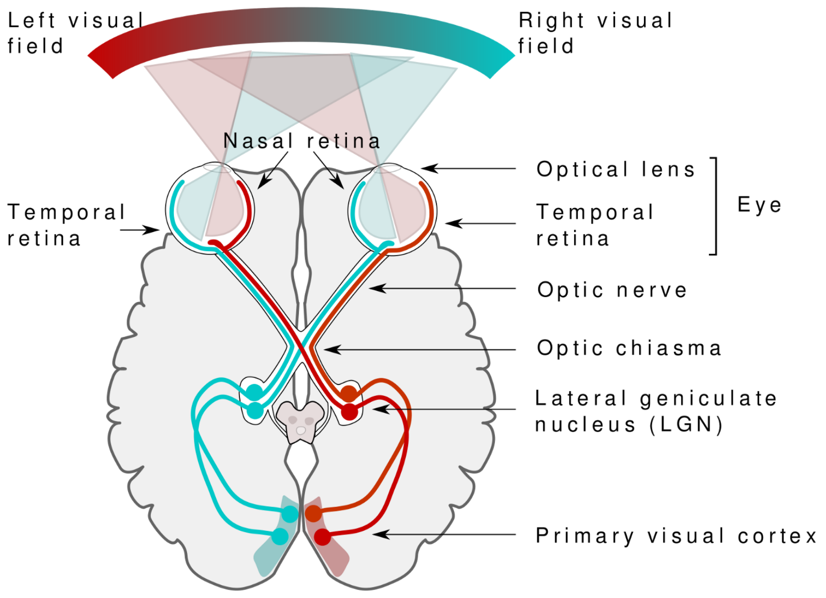
You probably thought something like “that’s a happy little boy standing on a chair”. Or maybe you thought he looks like he is screaming, about to attack this cake in front of him.



Source: https://www.youtube.com/watch?v=40riCqvRoMs

This is what we subconciously do all day. We see, label, make predictions, and recognize patterns. But how do we do that? How is it that we can interpret everything what we see?

It took nature over 500 million years to create a system to do this. The collaboration between the eyes and the brain, called the primary visual pathway, is the reason we can make sense of the world around us.



The Visual Pathway.— Source: https://commons.wikimedia.org/wiki/File:Human_visual_pathway.svg

While vision starts in the eyes, the actual interpretation of what we see happens in the brain, in the **primary visual cortex**.

When you see an object, the light receptors in your eyes send signals via the optic nerve to the primary visual cortex, where the input is being processed. The primary visual cortex makes sense of what the eye sees.

All of this seems very natural to us. We barely even think about how special it is that we are able to recognise all the objects and people we see in our lives. The **deeply complex hierarchical structure** of neurons and connections in the brain play a major role in this process of remembering and labelling objects.

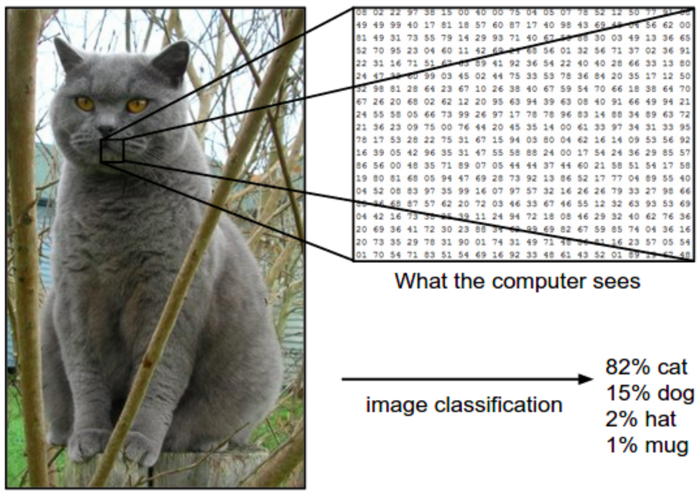
Think about how we learned what, for example, an umbrella is. Or a duck, lamp, candle, or book. In the beginning, our parents or family told us the name of the objects in our direct environment. We learned by examples that were given to us. Slowly but surely we started to recognise certain things more and more often in our environment. They became so common that the next time we saw them, we would instantly know what the name of this object was. They became part of our **model** on the world.

Convolutional Neural Networks

Similar to how a child learns to recognise objects, we need to show an algorithm millions of pictures before it is be able to generalize the input and make predictions for images it has never seen before.

Computers 'see' in a different way than we do. Their world consists of only numbers. Every image can be represented as 2-dimensional arrays of numbers, known as pixels.

But the fact that they perceive images in a different way, doesn't mean we can't train them to recognize patterns, like we do. We just have to think of what an image is in a different way.



How a computer sees an image.—source: http://cs231n.github.io/classification/

To teach an algorithm how to recognise objects in images, we use a specific type of Artificial Neural Network: a Convolutional Neural Network (CNN). Their name stems from one of the most important operations in the network: convolution.

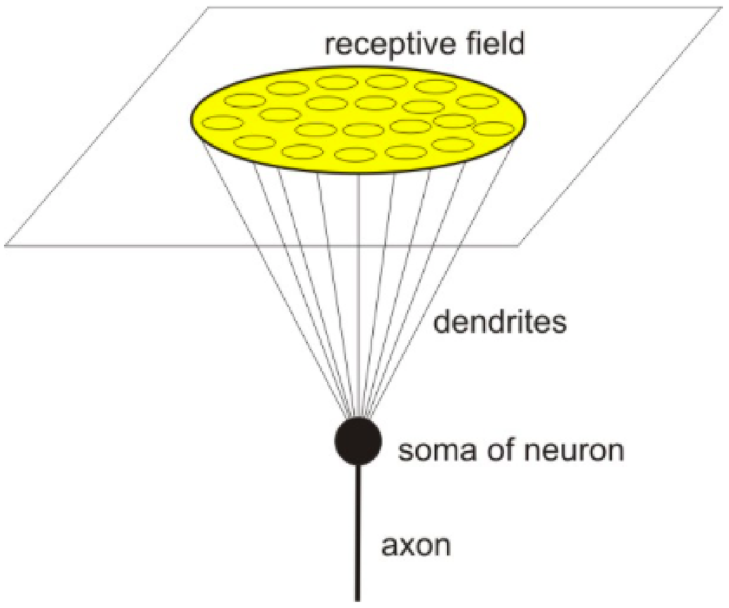
Convolutional Neural Networks are inspired by the brain. Research in the 1950s and 1960s by D.H Hubel and T.N Wiesel on the brain of mammals suggested a new model for how mammals perceive the world visually. They showed that cat and monkey visual cortexes include neurons that exclusively respond to neurons in their direct environment.

In their paper, they described two basic types of visual neuron cells in the brain that each act in a different way: simple cells (**S cells**) and complex cells (**C cells**).

The simple cells activate, for example, when they identify basic shapes as lines in a fixed area and a specific angle. The complex cells have larger receptive fields and their output is not sensitive to the specific position in the field.

The complex cells continue to respond to a certain stimulus, even though its absolute position on the retina changes. Complex refers to more flexible, in this case.

In vision, a **receptive field** of a single sensory neuron is the specific region of the retina in which something will affect the firing of that neuron (that is, will active the neuron). Every sensory neuron cell has similar receptive fields, and their fields are overlying.



A neuron's receptive field.—Source: http://neuroclusterbrain.com/neuron_model.html

Further, the concept of **hierarchy** plays a significant role in the brain. Information is stored in sequences of patterns, in sequential order. The **neocortex**, which is the outermost layer of the brain, stores information hierarchically. It is stored in cortical columns, or uniformly organised groupings of neurons in the neocortex.

In 1980, a researcher called Fukushima proposed a hierarchical neural network model. He called it the **neocognitron**. This model was inspired by the concepts of the Simple and Complex cells. The neocognitron was able to recognise patterns by learning about the shapes of objects.

Later, in 1998, Convolutional Neural Networks were introduced in a paper by Bengio, Le Cun, Bottou and Haffner. Their first Convolutional Neural Network was called **LeNet-5** and was able to classify digits from hand-written numbers.

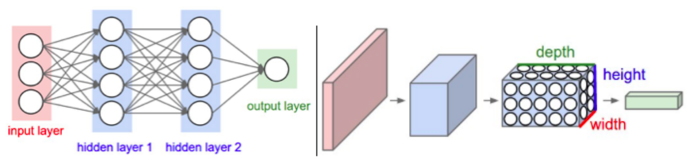
For the entire history on Convolutional Neural Nets, you can go here.

Architecture

In the remainder of this article, I will take you through the architecture of a CNN and show you the Python implementation as well.

Convolutional Neural Networks have a different architecture than regular Neural Networks. Regular Neural Networks transform an input by putting it through a series of hidden layers. Every layer is made up of a **set of neurons**, where each layer is fully connected to all neurons in the layer before. Finally, there is a last fully-connected layer—the output layer—that represent the predictions.

Convolutional Neural Networks are a bit different. First of all, the layers are **organised in 3 dimensions**: width, height and depth. Further, the neurons in one layer do not connect to all the neurons in the next layer but only to a small region of it. Lastly, the final output will be reduced to a single vector of probability scores, organized along the depth dimension.



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Normal NN vs CNN.—Source: http://cs231n.github.io/convolutional-networks/

CNNs have two components:

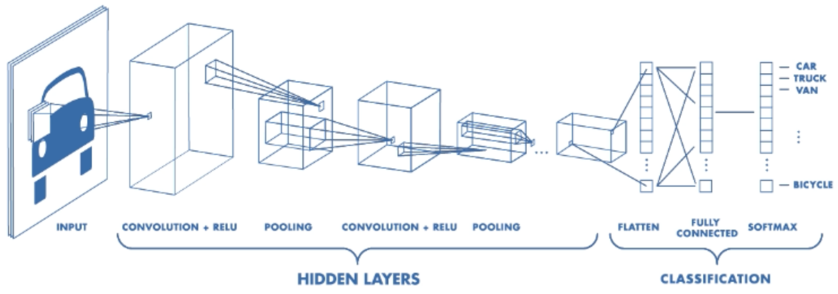
- The Hidden layers/Feature extraction part

In this part, the network will perform a series of **convolutions** and **pooling** operations during which the **features are detected**. If you had a picture of a zebra, this is the part where the network would recognise its stripes, two ears, and four legs.

- The Classification part

Here, the fully connected layers will serve as a **classifier** on top of these extracted features. They will assign a **probability** for the object on the image being what the algorithm predicts it is.

```
# before we start building we import the libraries

import numpy as np

from keras.layers import Conv2D, Activation, MaxPool2D,
Flatten, Dense
from keras.models import Sequential
```



Architecture of a CNN.—Source: https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html

Feature extraction

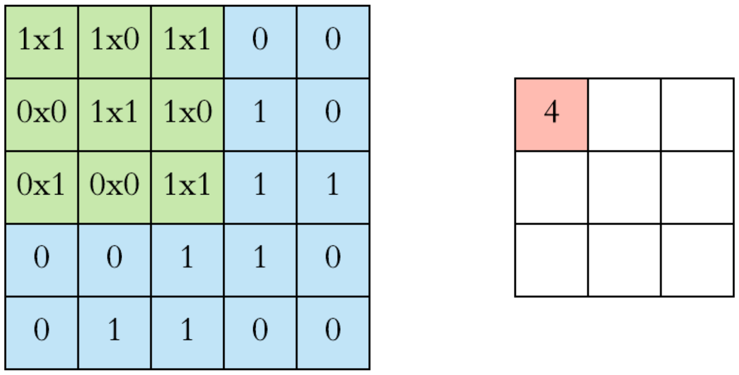
Convolution is one of the main building blocks of a CNN. The term convolution refers to the mathematical combination of two functions to produce a third function. It merges two sets of information.

In the case of a CNN, the convolution is performed on the input data with the use of a **filter** or **kernel** (these terms are used interchangeably) to then produce a **feature map**.

We execute a convolution by sliding the filter over the input. At every location, a matrix multiplication is performed and sums the result onto the feature map.

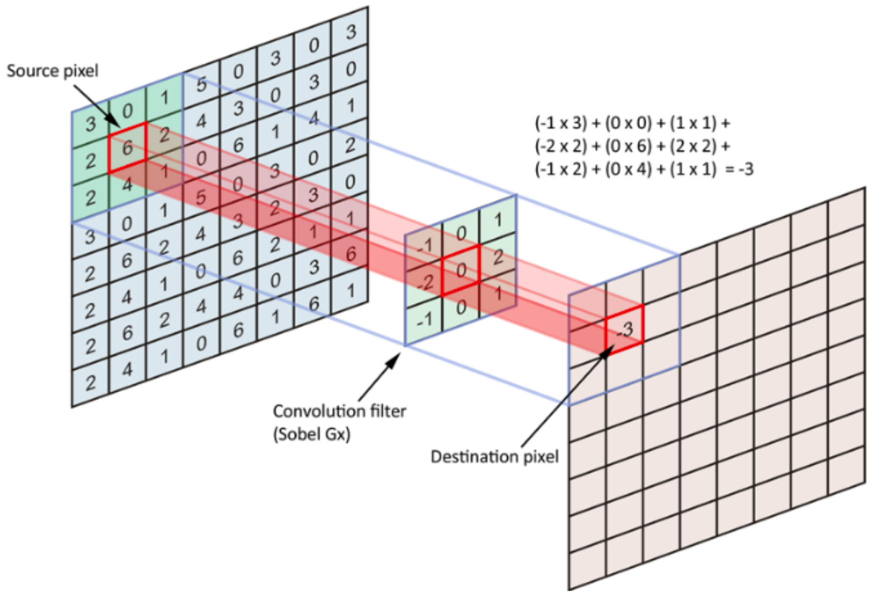
In the animation below, you can see the convolution operation. You can see the **filter** (the green square) is sliding over our **input** (the blue square) and the sum of the convolution goes into the **feature map** (the red square).

The area of our filter is also called the receptive field, named after the neuron cells! The size of this filter is 3x3.



Left: the filter slides over the input. Right: the result is summed and added to the feature map.— Source: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

For the sake of explaining, I have shown you the operation in 2D, but in reality convolutions are performed in 3D. Each image is namely represented as a 3D matrix with a dimension for width, height, and depth. Depth is a dimension because of the colours channels used in an image (RGB).


Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

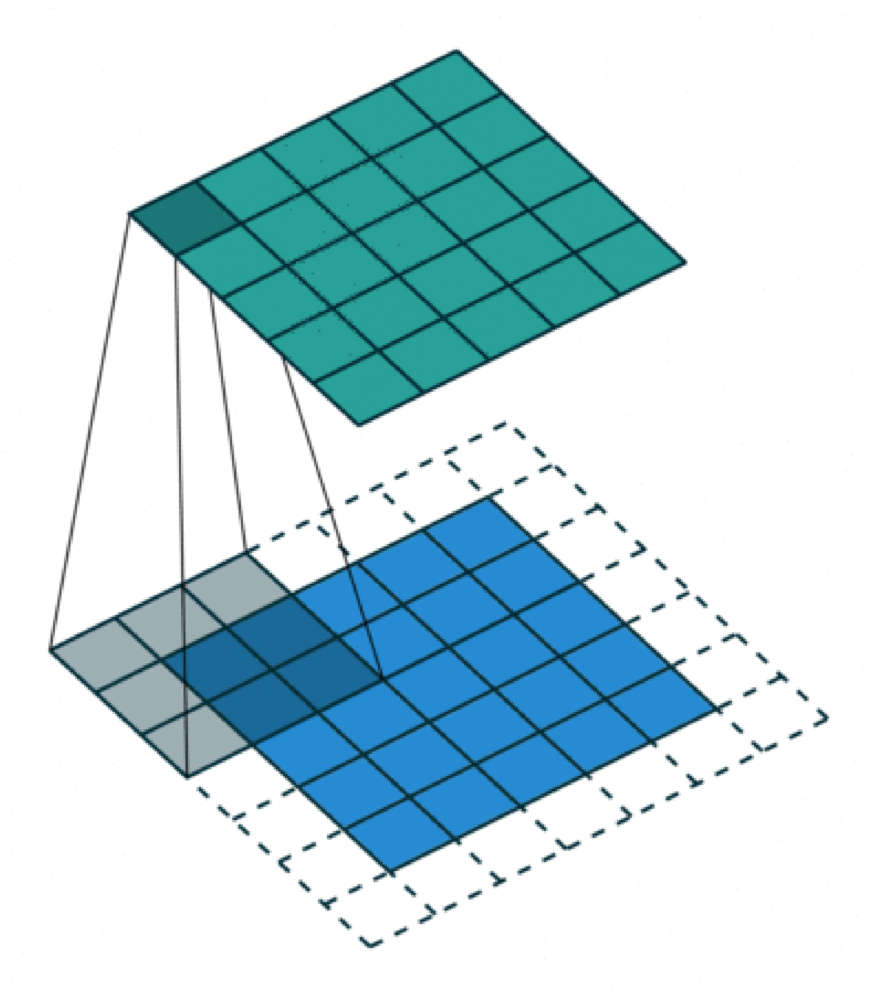
The filter slides over the input and performs its output on the new layer.—Source: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

We perfom numerous convolutions on our input, where each operation uses a different filter. This results in different feature maps. In the end, we take all of these feature maps and put them together as the final output of the convolution layer.

Just like any other Neural Network, we use an **activation function** to make our output non-linear. In the case of a Convolutional Neural Network, the output of the convolution will be passed through the activation function. This could be the ReLU activation function.

Stride is the size of the step the convolution filter moves each time. A stride size is usually 1, meaning the filter slides pixel by pixel. By increasing the stride size, your filter is sliding over the input with a larger interval and thus has less overlap between the cells.

The animation below shows stride size 1 in action.

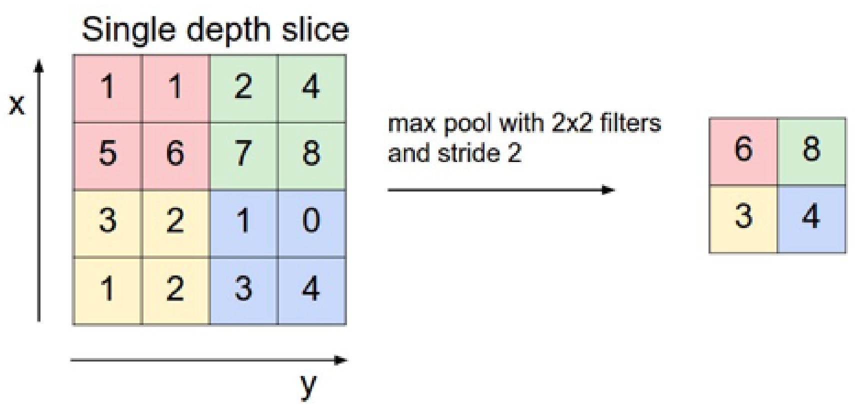


Because the size of the feature map is always smaller than the input, we have to do something to prevent our feature map from shrinking. This is where we use **padding**.

A layer of zero-value pixels is added to surround the input with zeros, so that our feature map will not shrink. In addition to keeping the spatial size constant after performing convolution, padding also improves performance and makes sure the kernel and stride size will fit in the input.

After a convolution layer, it is common to add a **pooling layer** in between CNN layers. The function of pooling is to continuously reduce the dimensionality to reduce the number of parameters and computation in the network. This shortens the training time and controls overfitting.

The most frequent type of pooling is **max pooling**, which takes the maximum value in each window. These window sizes need to be specified beforehand. This decreases the feature map size while at the same time keeping the significant information.



Max pooling takes the largest values.—Source: http://cs231n.github.io/convolutional-networks/

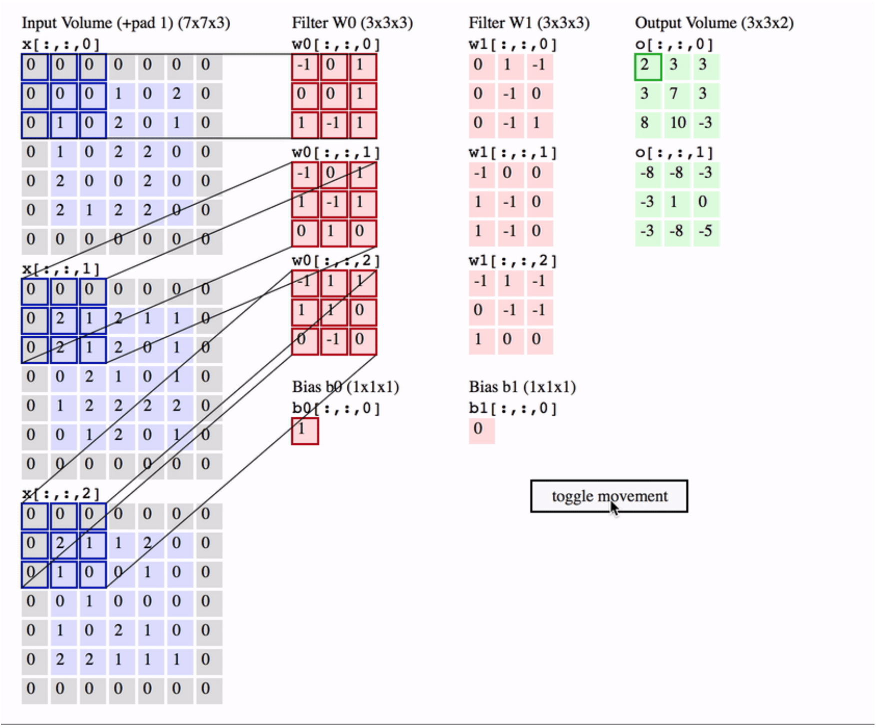
Thus when using a CNN, the four important **hyperparameters** we have to decide on are:

- the kernel size
- the filter count (that is, how many filters do we want to use)
- stride (how big are the steps of the filter)
- padding

```
# Images fed into this model are 512 x 512 pixels with 3
channels

img_shape = (28,28,1)

# Set up the model

model = Sequential()

# Add convolutional layer with 3, 3 by 3 filters and a stride
size of 1
# Set padding so that input size equals output size

model.add(Conv2D(6,2,input_shape=img_shape))
```

```
# Add relu activation to the layer

model.add(Activation('relu'))

#Pooling

model.add(MaxPool2D(2))
```

A nice way of visualizing a convolution layer is shown below. Try to look at it for a bit and really understand what is happening.



How convolution works with K = 2 filters, each with a spatial extent F = 3 , stride, S = 2, and input padding P = 1.—Source: http://cs231n.github.io/convolutional-networks/

Classification

After the convolution and pooling layers, our classification part consists of a few fully connected layers. However, these fully connected layers can only accept 1 Dimensional data. To convert our 3D data to 1D, we use the function `flatten` in Python. This essentially arranges our 3D volume into a 1D vector.

The last layers of a Convolutional NN are fully connected layers. Neurons in a fully connected layer have full connections to all the activations in the previous layer. This part is in principle the same as a regular Neural Network.

```
#Fully connected layers

# Use Flatten to convert 3D data to 1D
model.add(Flatten())

# Add dense layer with 10 neurons
model.add(Dense(10))

# we use the softmax activation function for our last layer
model.add(Activation('softmax'))

# give an overview of our model

model.summary

____________________________________________________________
_____
Layer (type)                 Output Shape              Param
#
============================================================
=====
conv2d_1 (Conv2D)            (None, 27, 27, 6)         30
____________________________________________________________
_____
activation_1 (Activation)    (None, 27, 27, 6)         0
____________________________________________________________
_____
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 6)         0
____________________________________________________________
_____
flatten_1 (Flatten)          (None, 1014)              0
____________________________________________________________
_____
dense_1 (Dense)              (None, 10)                10150
____________________________________________________________
_____
activation_2 (Activation)    (None, 10)                0
============================================================
=====
Total params: 10,180
Trainable params: 10,180
Non-trainable params: 0
____________________________________________________________
______
```

Training

Training a CNN works in the same way as a regular neural network, using backpropagration or gradient descent. However, here this is a bit more mathematically complex because of the convolution operations.

If you would like to read more about how regular neural nets work, you can read my previous article.

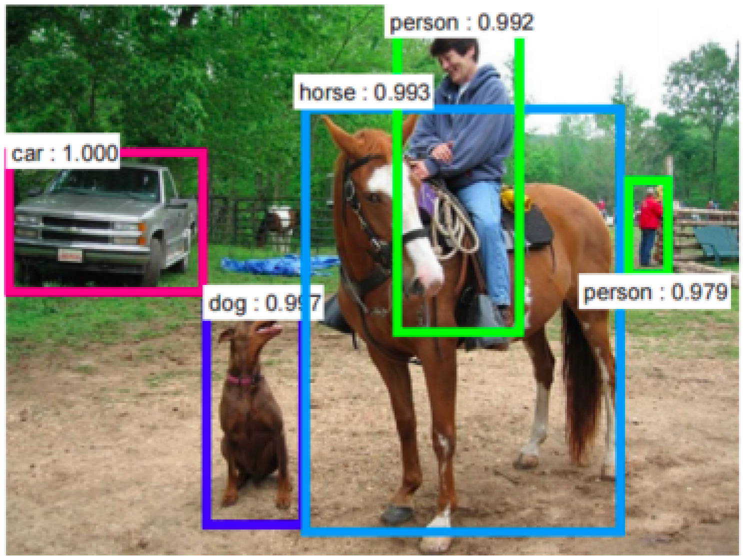
```
"""Before the training process, we have to put together a
learning process in a particular form. It consists of 3
elements: an optimiser, a loss function and a metric."""


model.compile(loss='sparse_categorical_crossentropy',
optimizer = 'adam', metrics=['acc'])


# dataset with handwritten digits to train the model on
from keras.datasets import mnist


(x_train, y_train), (x_test, y_test) = mnist.load_data()



x_train = np.expand_dims(x_train,-1)


x_test = np.expand_dims(x_test,-1)



# Train the model, iterating on the data in batches of 32
samples
# for 10 epochs


model.fit(x_train, y_train, batch_size=32, epochs=10,
validation_data=(x_test,y_test)


# Training...


Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 10s
175us/step - loss: 4.0330 - acc: 0.7424 - val_loss: 3.5352 -
val_acc: 0.7746
Epoch 2/10
60000/60000 [==============================] - 10s
169us/step - loss: 3.5208 - acc: 0.7746 - val_loss: 3.4403 -
val_acc: 0.7794
Epoch 3/10
60000/60000 [==============================] - 11s
176us/step - loss: 2.4443 - acc: 0.8372 - val_loss: 1.9846 -
val_acc: 0.8645
Epoch 4/10
60000/60000 [==============================] - 10s
173us/step - loss: 1.8943 - acc: 0.8691 - val_loss: 1.8478 -
```

```
val_acc: 0.8713
Epoch 5/10
60000/60000 [==============================] - 10s
174us/step - loss: 1.7726 - acc: 0.8735 - val_loss: 1.7595 -
val_acc: 0.8718
Epoch 6/10
60000/60000 [==============================] - 10s
174us/step - loss: 1.6943 - acc: 0.8765 - val_loss: 1.7150 -
val_acc: 0.8745
Epoch 7/10
60000/60000 [==============================] - 10s
173us/step - loss: 1.6765 - acc: 0.8777 - val_loss: 1.7268 -
val_acc: 0.8688
Epoch 8/10
60000/60000 [==============================] - 10s
173us/step - loss: 1.6676 - acc: 0.8799 - val_loss: 1.7110 -
val_acc: 0.8749
Epoch 9/10
60000/60000 [==============================] - 10s
172us/step - loss: 1.4759 - acc: 0.8888 - val_loss: 0.1346 -
val_acc: 0.9597
Epoch 10/10
60000/60000 [==============================] - 11s
177us/step - loss: 0.1026 - acc: 0.9681 - val_loss: 0.1144 -
val_acc: 0.9693
```

Summary

In summary, CNNs are especially useful for image classification and recognition. They have two main parts: a feature extraction part and a classification part.

The main special technique in CNNs is convolution, where a filter slides over the input and merges the input value + the filter value on the feature map. In the end, our goal is to feed new images to our CNN so it can give a probability for the object it thinks it sees or describe an image with text.



— Source: https://arxiv.org/pdf/1506.01497v3.pdf

You can find the entire code here.

More brain related recommendations?

- Read this really cool article on the brain and more.
- I also recommend this book on intelligence and the brain.
- “How to Create a Mind” by Ray Kurzweil.