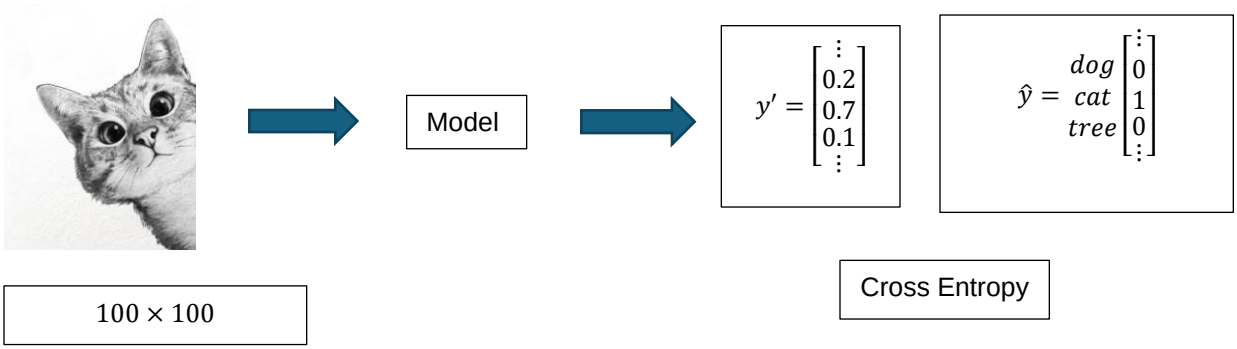
Lecture 6 Convolutional Neural Networks (CNN) Basics

6.2.1 CNN Introduction

Use case: Network Architecture designed for Image.

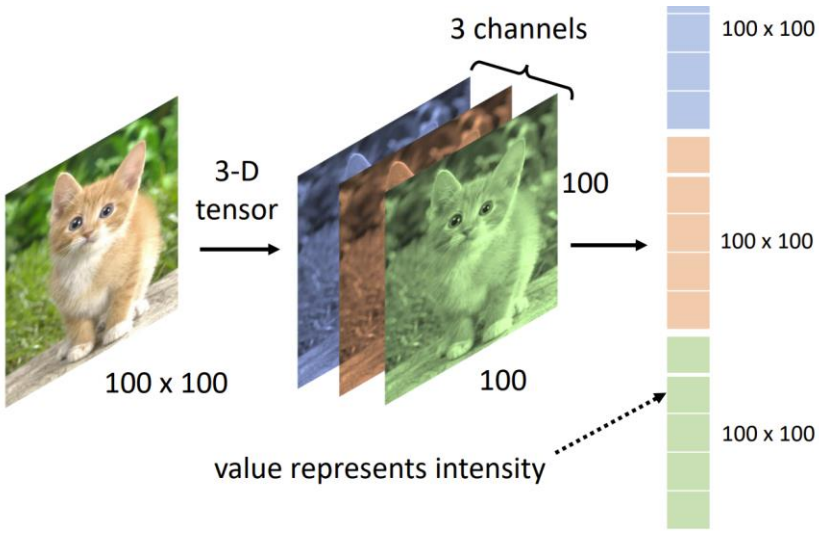
Image Classification



All the images (data input) must be of the same size, e.g. 100 x 100. If images are of different sizes, one must rescale all images to the same size. The goal of the model is classification, which we save in a vector $\hat{y}$. The length of the vector determines how many classifications you receive. If the length is 6, you will receive 6 classifications from CNN. Many advanced CNN applications may support up to higher than $10{,}000 \times 1$ vector length $\hat{y}$, which indicates more than 10,000 classifications. CNN model output is y'. The difference between y' and $\hat{y}$ is "Cross Entropy". The smaller the "Cross Entropy", the better.

6.2.2 Channel

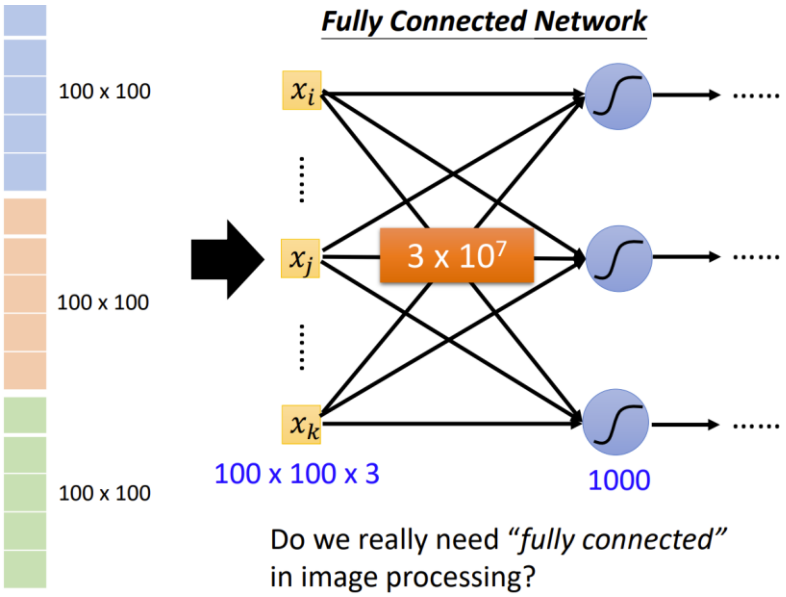
Each picture (100 x 100) is treated as a 3-D tensor (tensor means matrix with dimension > 2). The width 100, the height 100 and 3 channels. For a colored picture, each pixel is composed of R, G and B three colors. The 3 channels represent R, G and B three colors respectively. The height and width of the picture determines the resolution of the picture. A black-and-white picture hence has only 1 channel.



The next step is to stretch the picture into a vector. 100 x 100 x 3 is stretched into three stacked 100x100 vectors. Each element in the vector represents the density of a color (R, G, B) on a particular position of the image.

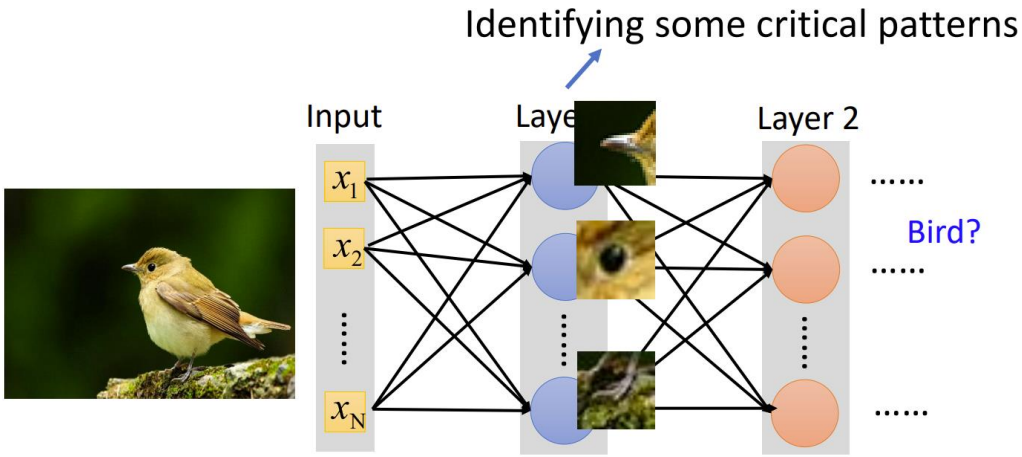
6.2.3 Fully Connected Network

The three stacked 100x100 vectors are now inputs to a Fully Connected Network. The size of the feature vector $x = \begin{bmatrix} \vdots \\ x_i \\ x_j \\ x_k \\ \vdots \end{bmatrix}$ is 100 x 100 x 3. Assuming we have 1000 neurons, each neuron is connected with all elements in the feature vector x by weights. We will then have $3 \times 10^7 = (1000 \times 100 \times 100 \times 3)$ weights (parameters) to be estimated. Increasing the number of parameters may increase the "elasticity" of the model, but too much "elasticity" leads to "Overfitting" of the model. How do we avoid using too many parameters? Considering the special challenges faced by image processing, key patterns only exist in a particular region of an image. We may not need "fully connected" in image processing.

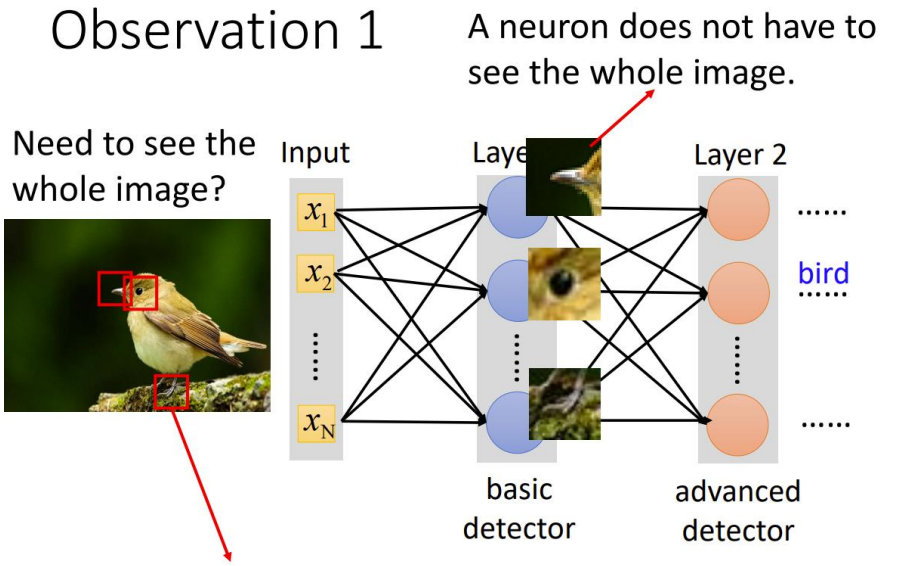


6.2.4 Design of CNN

Observation What to do if we want to identify a bird in an image? Part of the picture contains critical patterns we want to identify, beak, eyes, talons, etc. In Layer 1, one neuron identifies beak, one neuron identifies eyes, and one neuron identifies talons. The network then tells us that it identifies a bird.

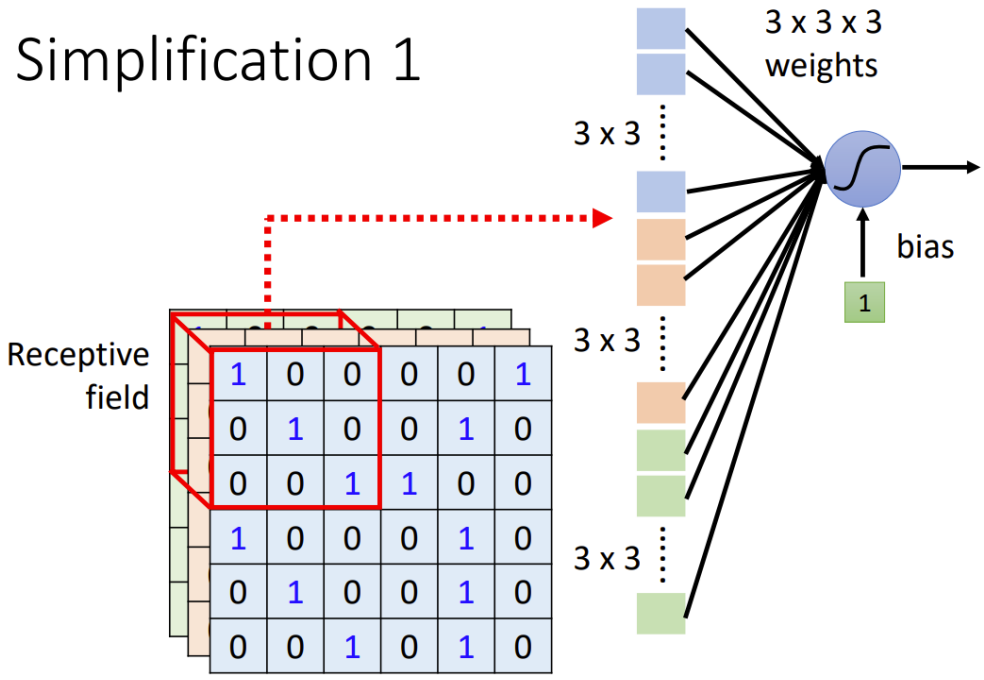


Not all neurons need to see the entire picture, but only a small region that carries the critical patterns to identify a bird. Each neuron may not need the entire picture as input, but only a small region of it.



6.2.5 Receptive Field

CNN specify a **Receptive Field**. Each neuron only cares about what happens to its own Receptive Field. The blue neuron on the right relates to a Receptive Field, which is a cubic with 3x3x3. What it does is to stretch the tensor into a vector, 3x3x3 = 27 dimensions. The 27 dim vector is an input to the neuron, and the neuron assigns weights to each element. There are 27 weights in total. With all 27 weights and bias, the neuron has an output that will be passed to the next layer.



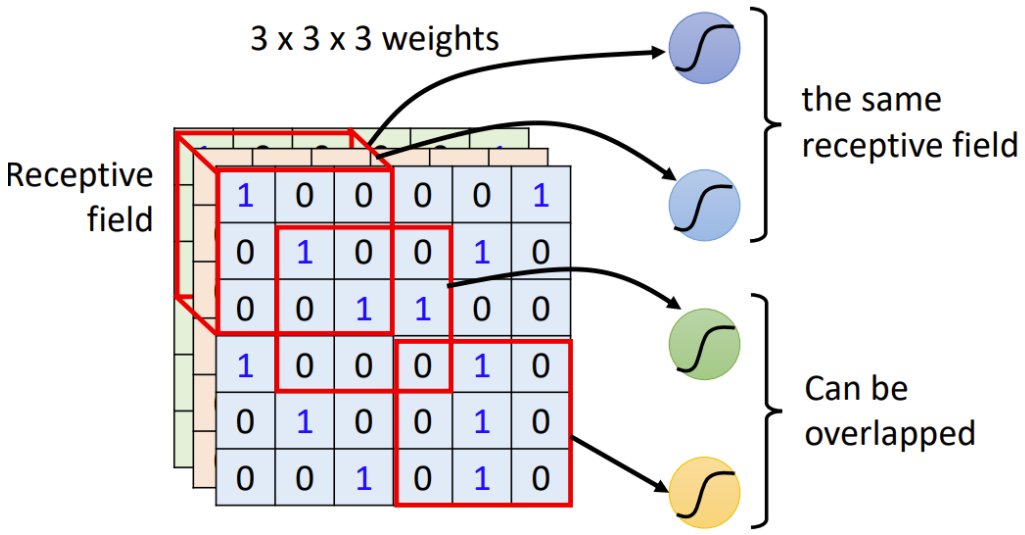
How is the Receptive Field determined? Blue neuron relates to the Receptive Field on upper left. Yellow neuron relates to the Receptive Field on lower right.

Receptive Field of different neurons may overlap with one another. Blue neuron's Receptive Field overlap with Green neuron's Receptive Field. **One Receptive Field can be covered by several neurons** – when one neuron may not be sufficient but need several neurons to identify a pattern.

Different neurons can have different sizes of receptive field. CNN extension.

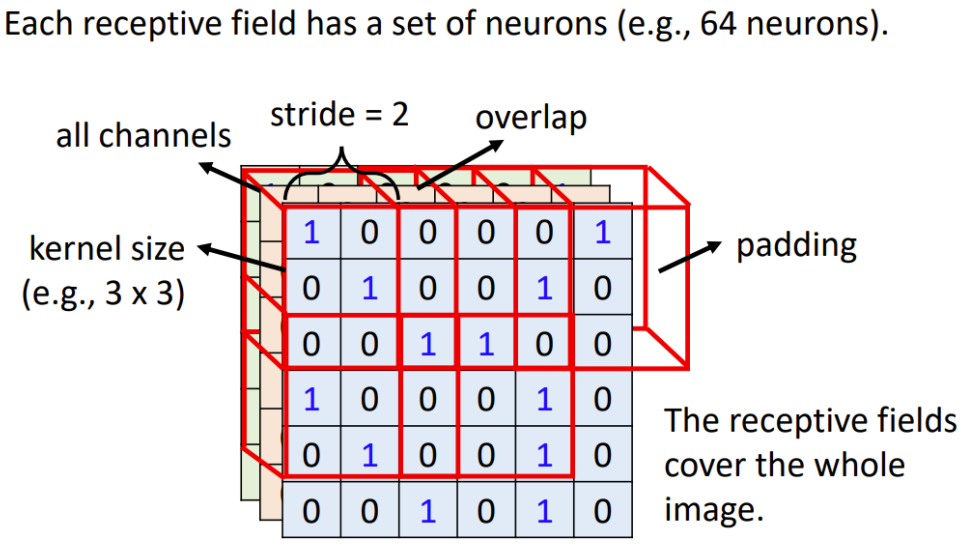
Some patterns may only exist in R, or in B, but not in G. Then neurons can cover only one channel, but not all channels. CNN extension.

Receptive Field doesn't have to be square, can be rectangular. Discontinuous receptive field may be useless.



6.2.6 Stride, Kernel Size and Padding

By default, the neuron often looks at all 3 channels. The height and width of a receptive field is so-called **kernel size** (e.g. 3 x 3), where we don't need to specify the depth (# of channels), as it is always 3. One may use large kernel size, such as 7x7 or 9 x9, but 3x3 is the most frequently used choice. One receptive field is usually covered by several neurons, often 64 neurons or 128 neurons. When we shift the Receptive Field by 2 blocks to the right, we make stride = 2. These are the hyperparameters specify by the developer. One can also make the stride to be 1. The purpose of making stride is to keep the receptive fields overlapping with one another. **Why receptive fields must overlap with one another? If there is one pattern that takes place right at the intersection of two receptive fields, there is no neuron to discover it.** When the receptive field exceeds the boarder of the image, we use "**Padding**", which means we fill with 0 values. There are different ways of padding – may fill with adjacent boarder values, mean values, etc. We could also downward shift the receptive field – until we cover the entire image using receptive fields.

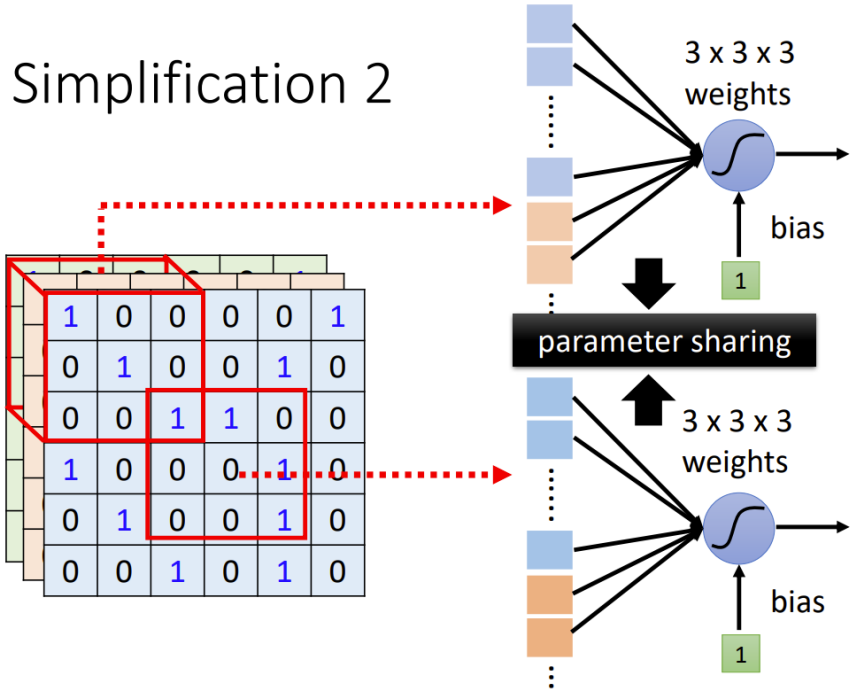


6.2.7 Parameter Sharing

The same pattern may appear in different regions in different images. Blue neurons can detect the beak (in upper left) in the first image. Orange neurons can detect the beak (in the middle) in the second image. The two neurons do the same task, but their receptive fields are different. It may not be necessary to have one "beak-detection" neuron for each receptive field. One simplification solution is to have neurons relating to different regions but of the same function – "parameter sharing." Weights (parameters) used in the upper neuron (relating to upper left receptive field) and lower neuron (relating to the middle receptive field) are exactly the same. Because inputs into each neuron are different (because receptive fields are different), outputs from the two neurons are different.

- Upper Neuron, inputs $x_1, x_2, \dots\ x_{26}, x_{27} (i.e.\ x_{1,1,1}, x_{1,2,1}, \dots x_{1,1,2}, x_{1,2,2}, \dots\ x_{3,2,3}, x_{3,3,3})$ output
$$o(x_i, w_i) = \sigma(w_1 x_1 + w_2 x_2, \dots w_{27} x_{27} + bias)$$
where $\sigma(\cdot)$ is the activation function.
- Lower Neuron, inputs $x_1', x_2', \dots$ output
$$o(x_i', w_i) = \sigma(w_1 x_1' + w_2 x_2', \dots w_{27} x_{27}' + bias)$$
where $\sigma(\cdot)$ is the activation function.

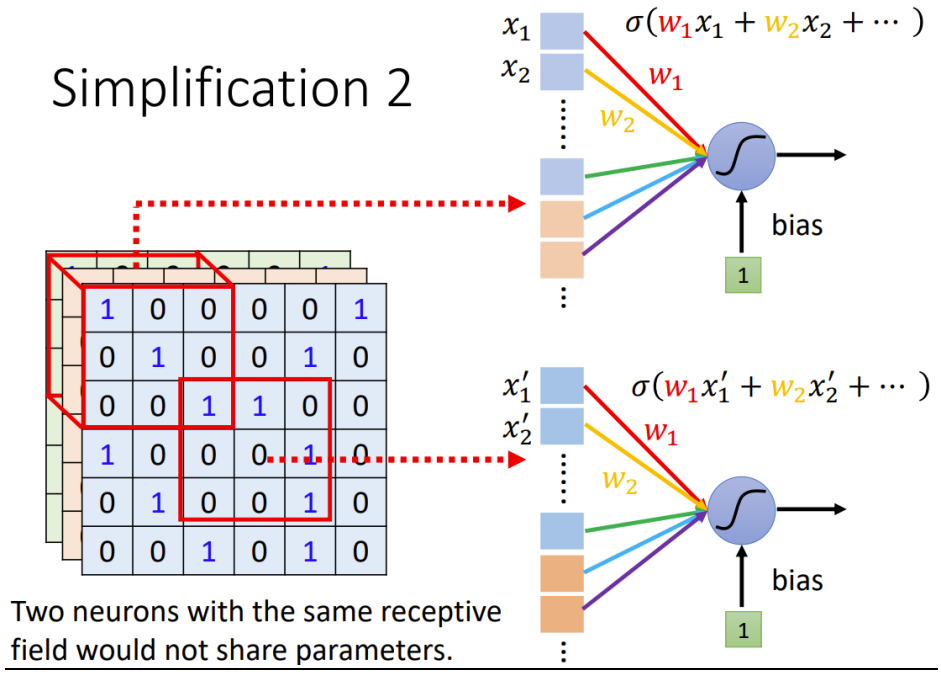
Because inputs are different, even when two neurons are using the exact same weights (parameters), the outputs are different. This is so-called "parameter-sharing." In the meantime, two neurons with the same receptive fields cannot have "parameter sharing."



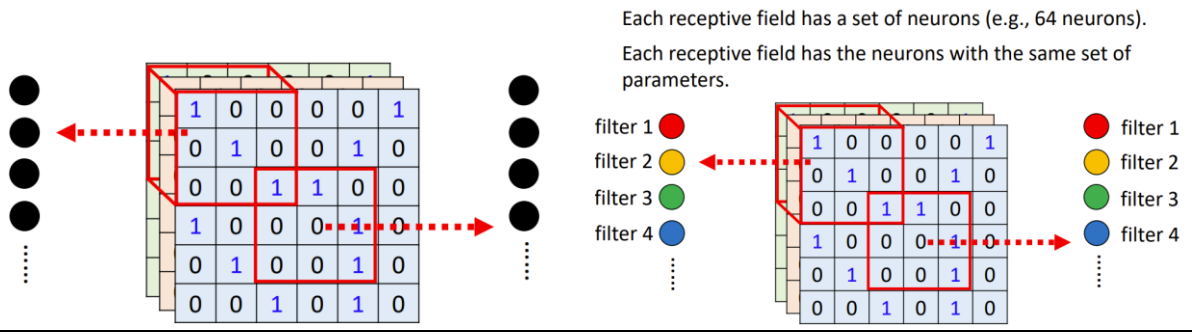
6.2.8 Filters

Each receptive field has a set of neurons (e.g. 64 neurons). How these neurons do parameter-sharing? Red, Yellow, Green and Blue represent neurons. **Same color means the two neurons are sharing the same**

set of parameters (weights) – through they cover different receptive fields. Each set of parameters(weights) are so-called **filters.** Red – filter 1, Yellow – filter 2, Green – filter 3, Blue – filter 4.



Each receptive field has a set of neurons (e.g., 64 neurons).



6.2.9 Parameter Sharing vs Fully Connected Layer

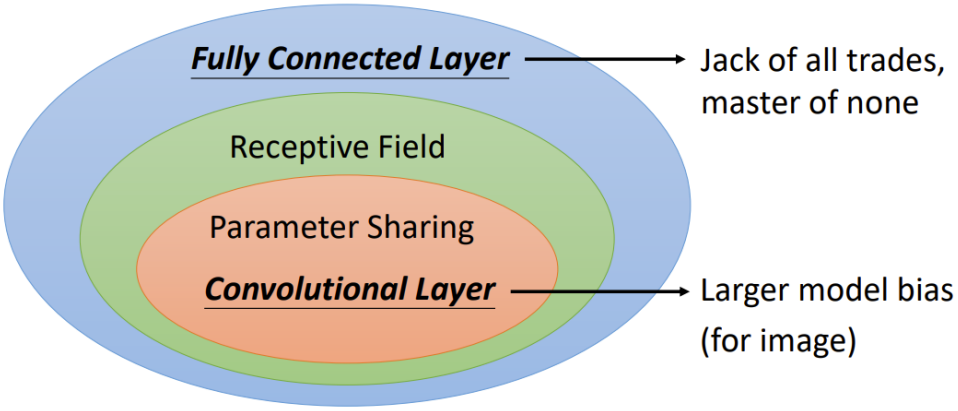
Fully Connected Layer – each neuron looks into the entire image.

Receptive Field – each neuron only looks into a particular region of an image (small, 3x3). It restricted the elasticity of the network to be smaller.

Parameter Sharing – Neurons relating to different receptive fields could use the same set of the weights (filters). The elasticity of the network is further reduced.

Receptive Field + Parameter Sharing = Convolutional Layer. For all networks that use Convolutional Layers, it is called CNN.

Consequently, model bias is large for CNN, but gives more flexibility and avoids over-fitting. One has to be careful when using CNN for tasks that require precision, say stock return prediction or text generation tasks, etc.



- Some patterns are much smaller than the whole image.
- The same patterns appear in different regions.

6.2.10 Filters

Convolutional Layer is thus composed of many "Filters": 3 x 3 x channel tensor. (channel = 3, i.e colored image, R-G-B; channel = 1, black-and-white). Each filter aims to detect a small pattern (3 x 3 channel).

Consider channel = 1, black and white image. Values in the filters are unknown and need to be estimated by Gradient Descent. Once the parameter values are obtained, how filter identify patterns in an image.

Convolutional Layer

Consider channel = 1
(black and white image)

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

(The values in the filters
are unknown parameters.)

Inside Convolutional Layer

Layer 1: the height of the filter is assumed to be 1, raw input is black and white image.

Filter 1 (orange) applies to Receptive Field on upper left corner, calculate inner product $w_1x_1 + w_2x_2, \dots w_Nx_N$.

Then shift the filter to the right by 1 (stride = 1) block, calculate again, -1

Iteratively shift the filter across the entire image, and calculate the inner products for each shift, one receives a matrix with dimension reduction. 6 x 6 is reduced to 4 x 4.

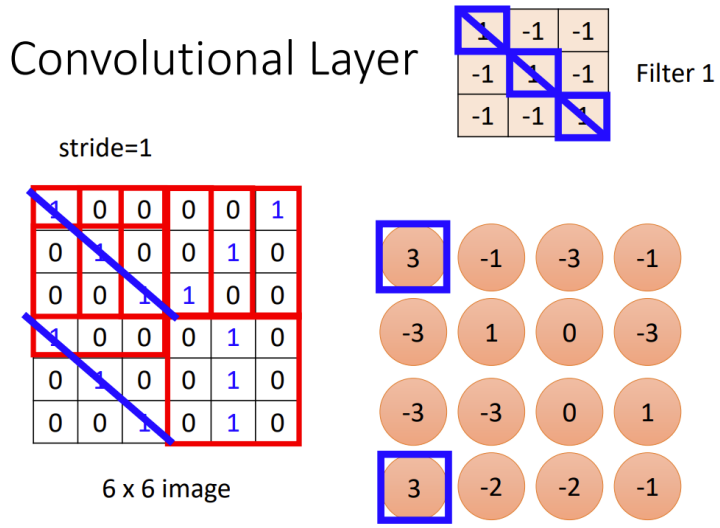
Then Filter 2 (blue) repeats the process that Filter 1 does. 6 x 6 is reduced to 4 x 4.

If we have 64 filters, we will end up with 64 matrices of size 4 x 4. These matrices are called “Feature Map.”

It can be regarded as an image with 64 channels, which says the raw image of 100 x 100 x 3 turns into a 4 x 4 x 64 “image” after one convolutional layer.

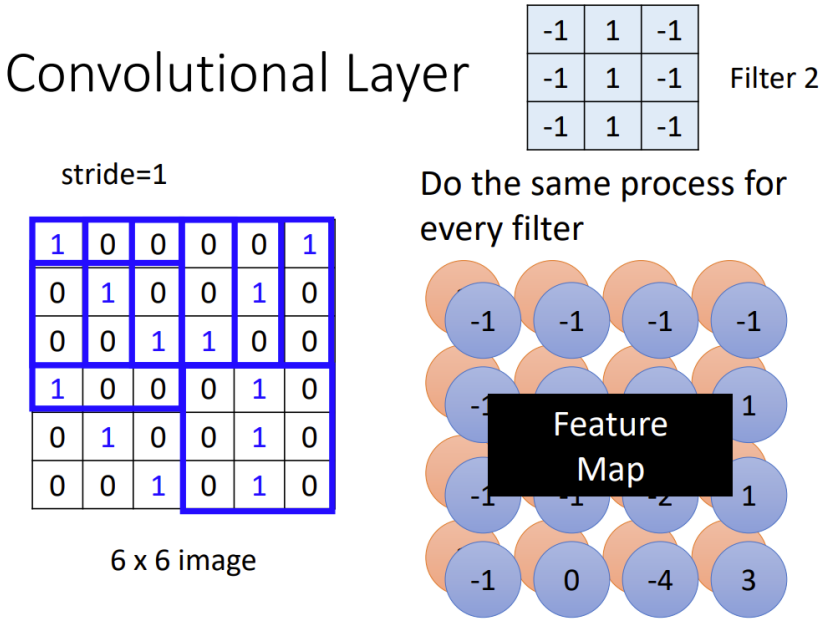
Layer 2 convolution also has a lot of filters. Each filter is specified to be 3 x 3. The height must be set to 64, because the height of the filter is right the channel the filter processes. Filter 3 x 3 x 64.

What if setting the filter 3 x 3 too small that leads to we ignore important patterns? No. Filter looks at 3 x 3, but the 3 x 3 is mapped into 5 x 5 region in the raw image. The more layers you use, the wider the region your network will cover, the more likely you could capture the critical patterns.



Different neurons share weights – means using each filter to scan across the entire image. This is why the method is called "convolution." Neurons that cover different receptive fields share parameters.

6.2.11 Feature Map

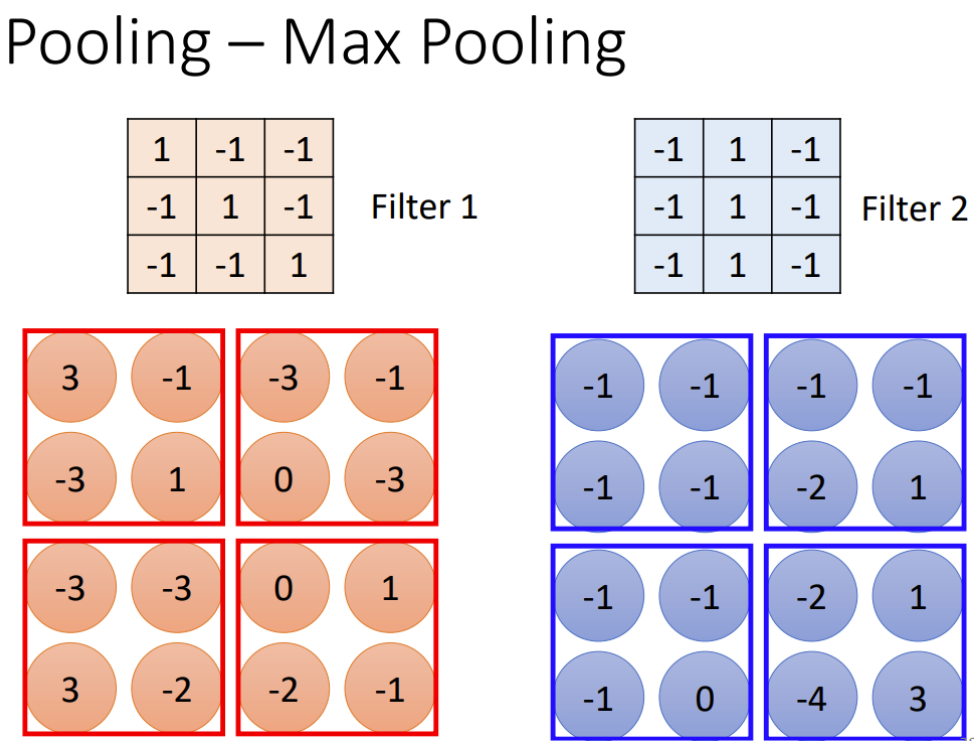


6.2.12 Pooling Layer

Pooling – taking the odds columns and even rows away from the original picture (subsampling), the size of the picture gets smaller. Pooling is like an activation function, sigmoid, ReLu, etc. There is nothing to identify in this procedure. It is simply an operator. It has different versions.

Pooling – Max Pooling

Each filter generates one matrix (one slice in a tensor) in a feature map (3 x 3 filter apply to 6 x 6 image, get 4 x 4 matrix). Segment the matrix into four 2 x 2 matrices. Max Pooling means take the max values from each 2 x 2 matrices. One would also use min, mean, etc. segmentation doesn't have to be 2 x 2, you can specify it yourself.

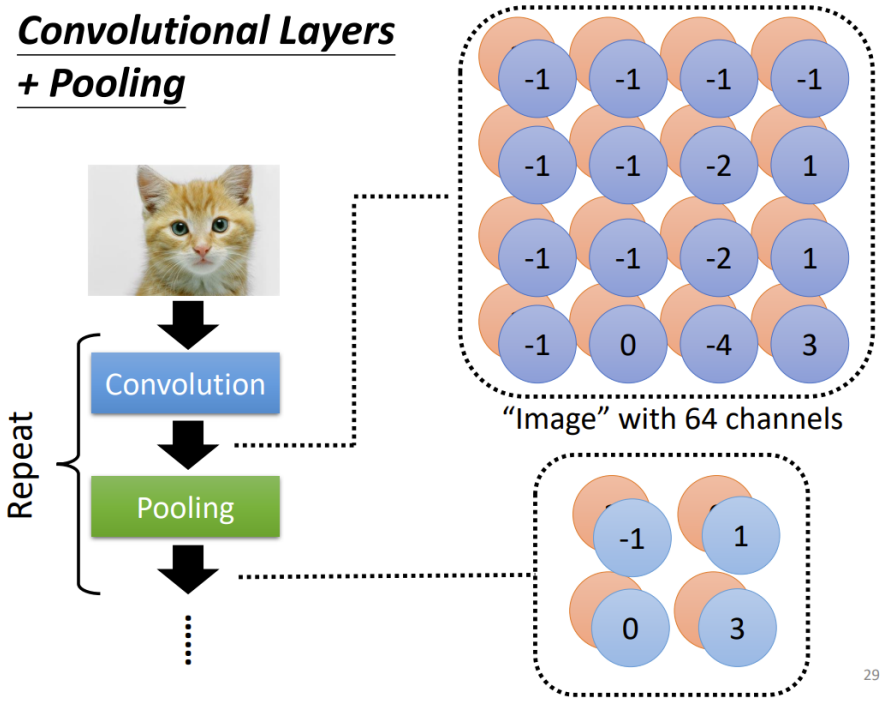


Pooling aims to make the image smaller and reduce the computation burden. Channel do not change, remain to be 64, but each 4x4 matrix is reduced in dimension to 2 x 2. Values kept in each element are the max values for each 2x2 sub-matrix in the 4x4 matrix. In practice, one may take turns using convolution and pooling. The most popular combination is 2 convolution + 1 pooling, another 2 convolution + 1 pooling. Pooling may harm performance if your aim is to capture details. As the computation power advances in recent years, many practitioners choose to use convolutional layers alone – full convolutional networks.

Max Pooling Results for the above Feature Map are:

$$\text{R: } \begin{bmatrix} 3 & 0 \\ 3 & 1 \end{bmatrix}$$

$$\text{B: } \begin{bmatrix} -1 & 1 \\ 0 & 3 \end{bmatrix}$$



6.2.13 CNN Summary

Flatten vectors received from pooling.

Throw the flattened vector into several fully connected layers and then have the output to pass softmax.

Classifications obtained, cat, dog, tree, etc.