



# Convolutional Neural Networks

---

Statinfer.com

# Note

- This presentation is just class notes. This course material is prepared by statinfer team, as an aid for training sessions.
- The best way to treat this is as a high-level summary; the actual session went more in depth and contained detailed information and examples
- Most of this material was written as informal notes, not intended for publication
- Please send questions/comments/corrections to [info@statinfer.com](mailto:info@statinfer.com)
- Please check our website [statinfer.com](https://statinfer.com) for latest version of this document

*- Team Statinfer*



# Contents

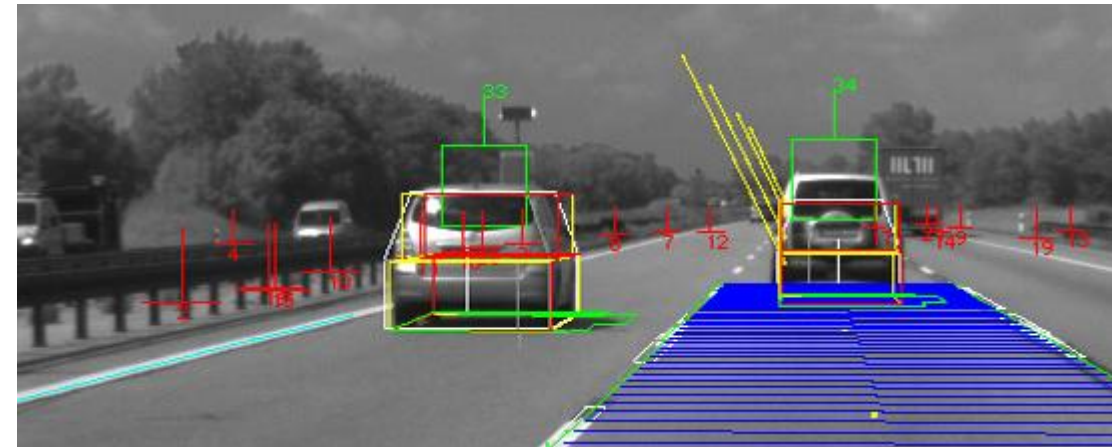
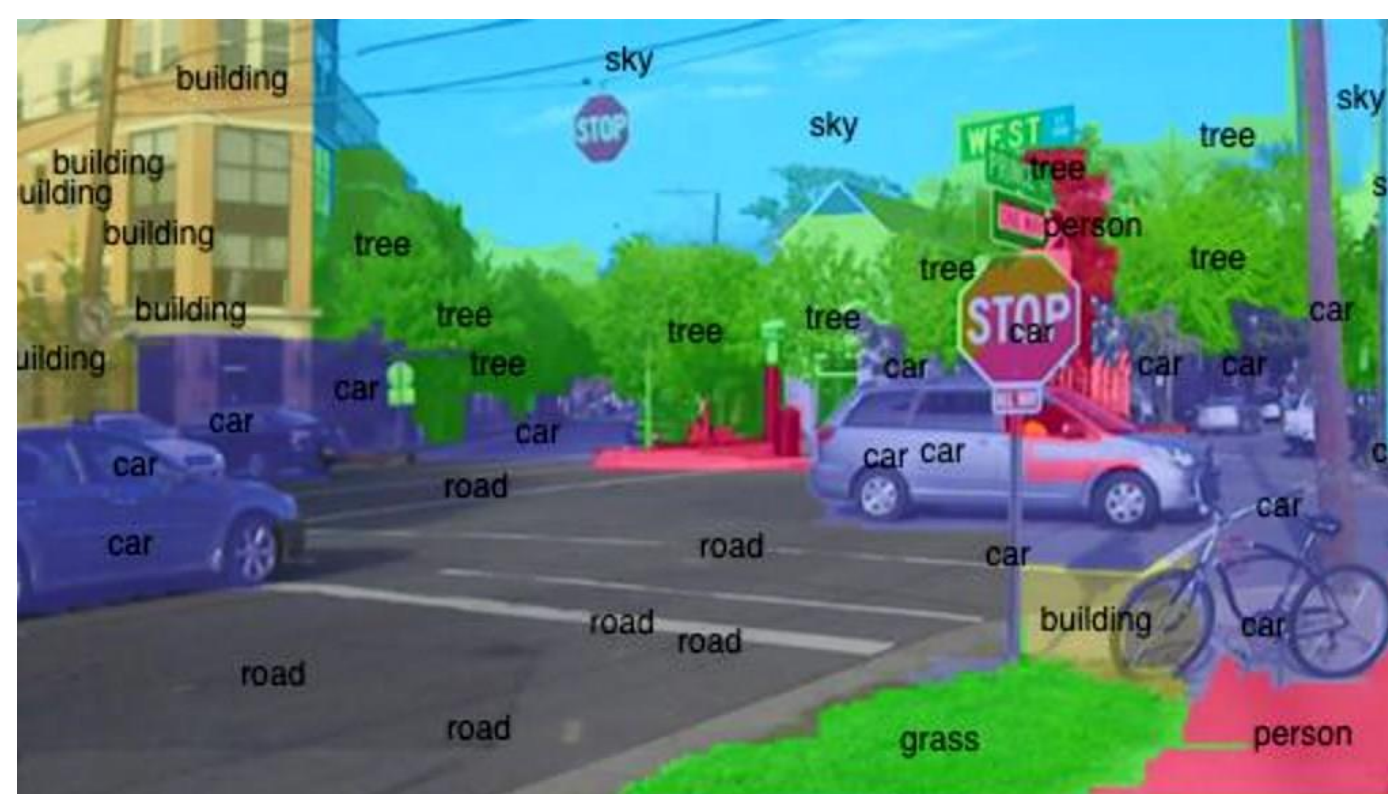
---

# Contents

- ANN for Large datasets
- Building ANN models deep networks
- CNN Introduction
- Convolution Layer
- Pooling
- CNN Theory
- CNN Implementation
- Building CNN

# CNN for Image recognition systems

CNNs are most widely used in image recognition systems



# Interesting applications of CNN

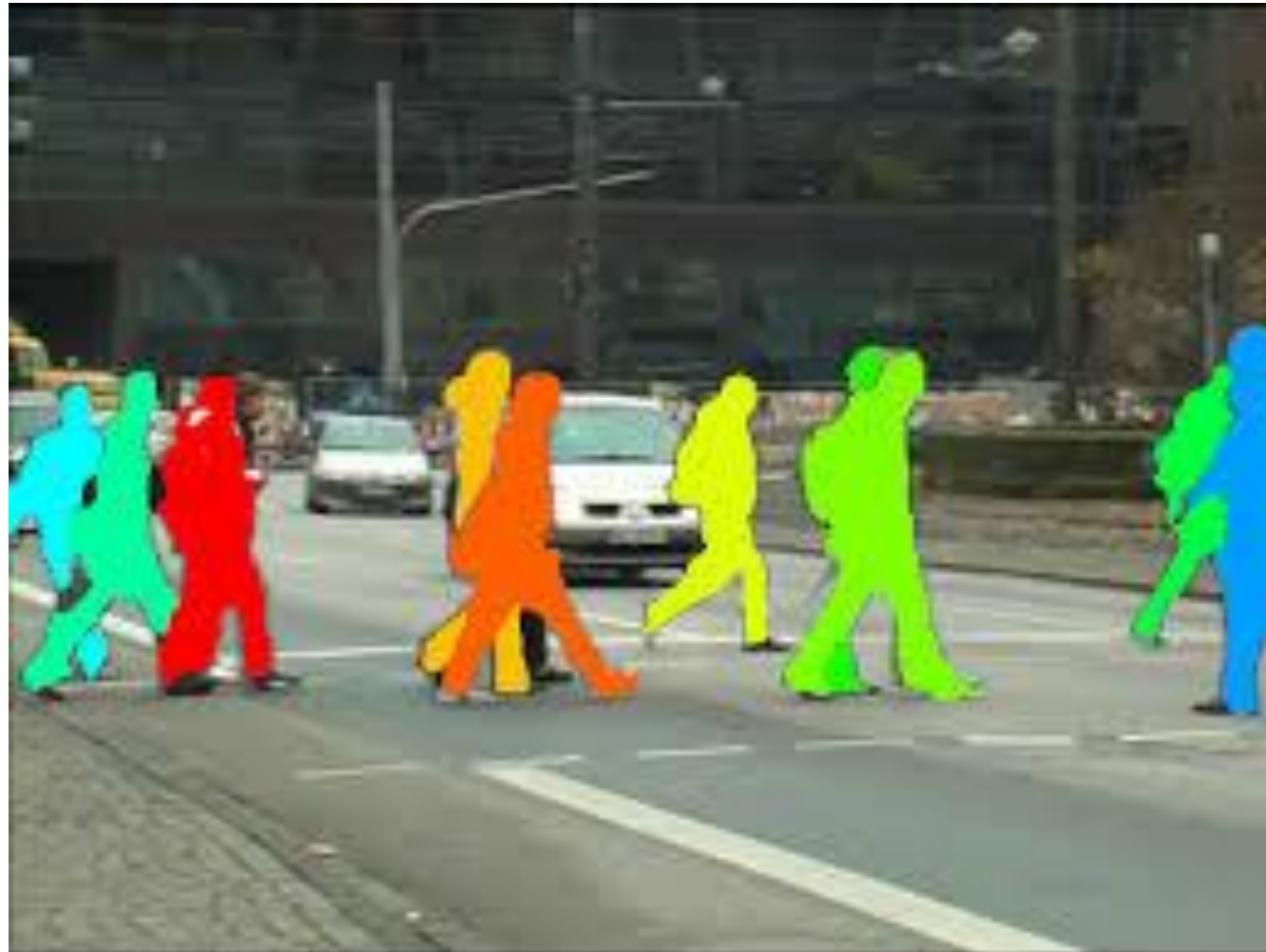
How a cucumber farm in Japan uses AI to sort its crops





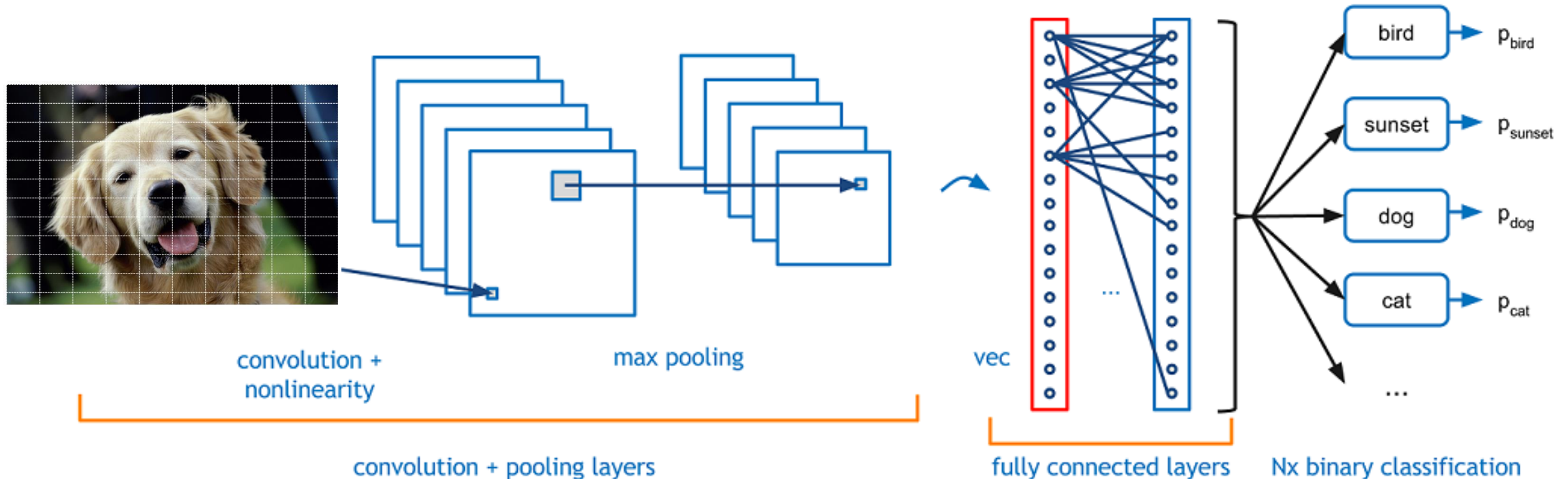
# CNN for Image recognition systems

- Some of the major applications of CNN are related to computer vision
  - Image classification
  - Face detection
  - Video Analysis



# CNN for Image recognition systems

- To start with CNN, one has to start with image recognition and **computer vision**.
- In fact, most the CNN theory and architecture is often explained by taking images underlined data/examples.





# How computer sees an image



Humans see this

-1	-1	-1	-1	-1	-1	-1	-1	-1	0.9	-0	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	0.3	1	0.3	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-0	1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	0.8	1	0.6	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	0.5	1	0.8	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	0.1	1	0.9	-0	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-0	1	1	-0	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	0.9	1	0.3	-1	-1	-1	-1	0.5	1	0.9	0.1	-1	-1
-1	-1	0.3	1	0.9	-1	-1	-1	0.1	1	1	1	1	1	-1	-1
-1	-1	0.8	1	0.3	-1	-1	0.4	1	0.7	-0	-0	1	1	-1	-1
-1	-1	1	1	0.1	-1	0.1	1	0.3	-1	-1	-0	1	0.6	-1	-1
-1	-1	1	1	0.8	0.3	1	0.7	-1	-1	-1	0.5	1	0	-1	-1
-1	-1	0.8	1	1	1	1	0.5	0.2	0.8	0.8	1	0.9	-1	-1	-1
-1	-1	-0	0.8	1	1	1	1	1	1	1	1	0.1	-1	-1	-1
-1	-1	-1	-0	0.8	1	1	1	1	1	1	1	0.2	-1	-1	-1
-1	-1	-1	-1	-1	-0	0.3	0.8	1	0.5	-0	-1	-1	-1	-1	-1

Computer sees this

# How computer sees an image

-1	-1	-1	-1	-1	-1	-1	-1	0.9	-0	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	0.3	1	0.3	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-0	1	1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	0.8	1	0.6	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	0.5	1	0.8	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	0.1	1	0.9	-0	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-0	1	1	-0	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	0.9	1	0.3	-1	-1	-1	-1	0.5	1	0.9	0.1	-1	-1
-1	-1	0.3	1	0.9	-1	-1	-1	0.1	1	1	1	1	1	-1	-1
-1	-1	0.8	1	0.3	-1	-1	0.4	1	0.7	-0	-0	1	1	-1	-1
-1	-1	1	1	0.1	-1	0.1	1	0.3	-1	-1	-0	1	0.6	-1	-1
-1	-1	1	1	0.8	0.3	1	0.7	-1	-1	-1	0.5	1	0	-1	-1
-1	-1	0.8	1	1	1	1	0.5	0.2	0.8	0.8	1	0.9	-1	-1	-1
-1	-1	-0	0.8	1	1	1	1	1	1	1	1	0.1	-1	-1	-1
-1	-1	-1	-0	0.8	1	1	1	1	1	1	0.2	-1	-1	-1	-1
-1	-1	-1	-1	-1	-0	0.3	0.8	1	0.5	-0	-1	-1	-1	-1	-1

Computer sees this

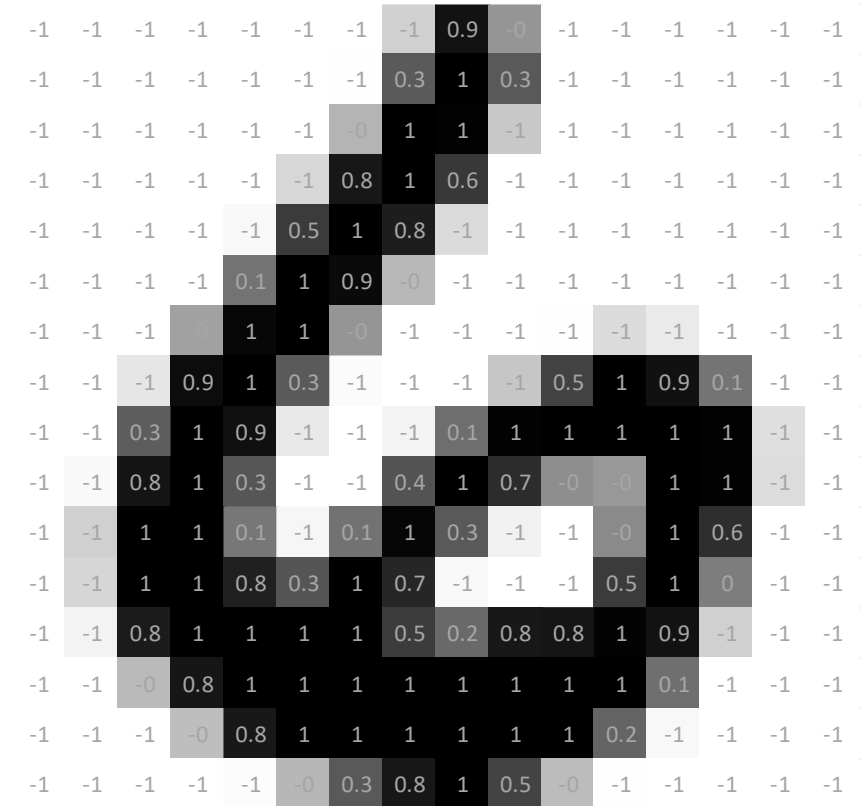
-1	-1	-1	-1	-1	-1	-1	-1	0.9	-0	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	0.3		0.3	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-0			-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	0.8		0.6	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	0.5		0.8	-1		-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	0.1			0.9	-0	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-0				-0	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	0.9			0.3	-1	-1	-1	-1	0.5		0.9	0.1	-1
-1	-1	0.3		0.9	-1	-1	-1	0.1						-1	-1
-1	-1	0.8		0.3	-1	-1	0.4		0.7	-0	-0			-1	-1
-1	-1		0.1	-1	0.1		0.3	-1	-1	-0		0.6		-1	-1
-1	-1		0.8	0.3		0.7	-1	-1	-1	0.5		0	-1	-1	-1
-1	-1	0.8			0.5	0.2	0.8	0.8		0.9	-1	-1	-1	-1	-1
-1	-1	-0	0.8							0.1	-1	-1	-1	-1	-1
-1	-1	-1	-0	0.8						0.2	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-0	0.3	0.8		0.5	-0	-1	-1	-1	-1	-1

Same matrix, highlight the cells based on cell value

# How computer sees an image



Human Vision



Computer Vision

# How computer sees an image

- Computers treat images as structured arrangement of numbers
- The information in an image is finally represented as numbers
- Handling images is nothing but handling numerical data but in a slightly different methodology

# Computer Vision – Image classification

- Based on the pixel values a computer identifies images.
- If we have images of numbers and their labels, then we can build a neural network model to the classify the new image
- There are intermediate steps
  - Take image as input. Convert them to computer readable images / pixel intensities
  - Convert each image into a single row by re-shaping it.
  - Collect as many images as possible to increase the accuracy of the model

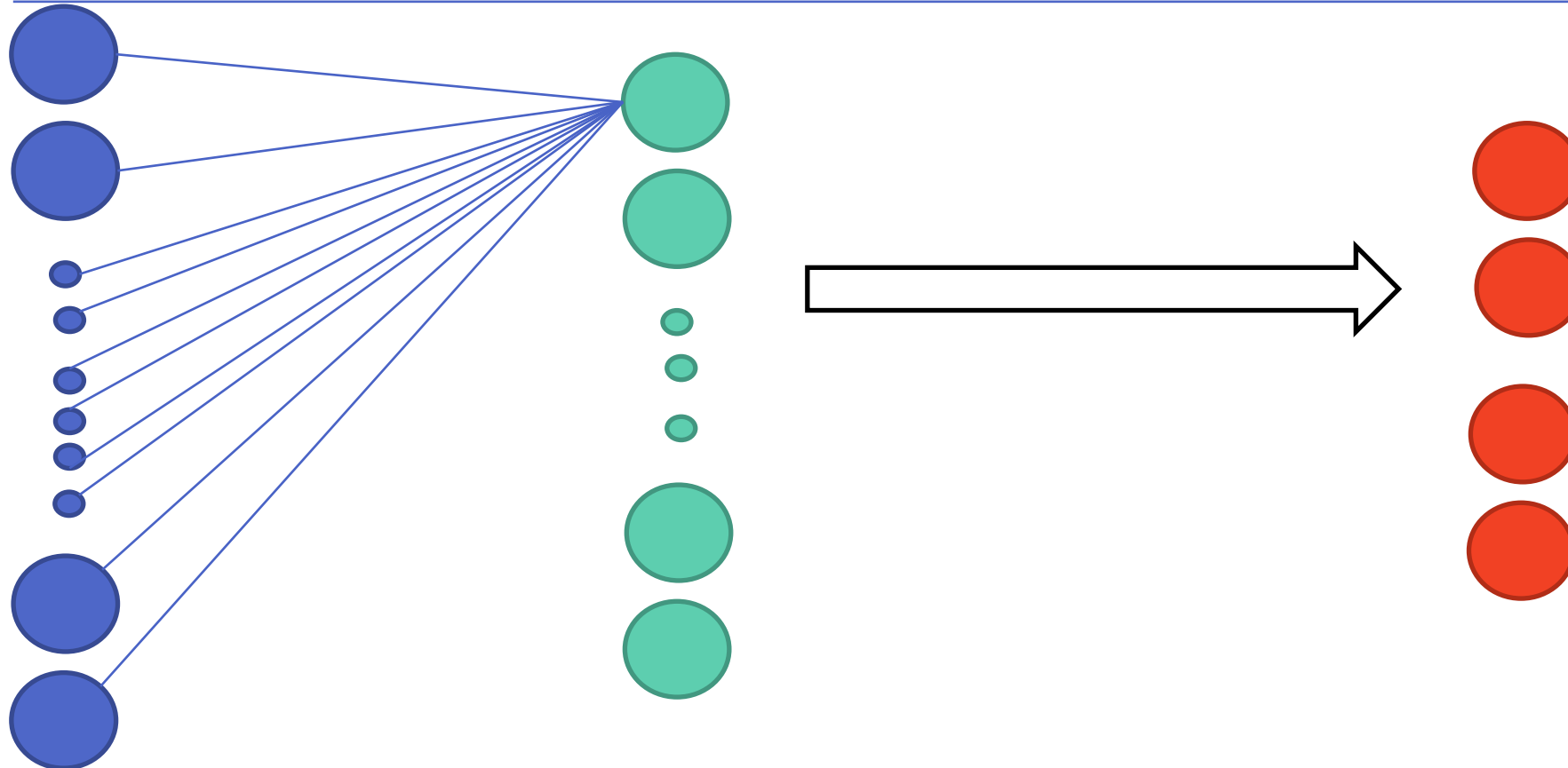
# The count of free parameters

- What are free parameters ?
- What are we training while training the model? - Weights
- Count the number of free parameters in our network.
  - Number of weights to train : for 784 pixel images
  - Input layer - 784 pixels (28X28)
  - Hidden layer - 20 nodes
  - Output layer - 10 nodes



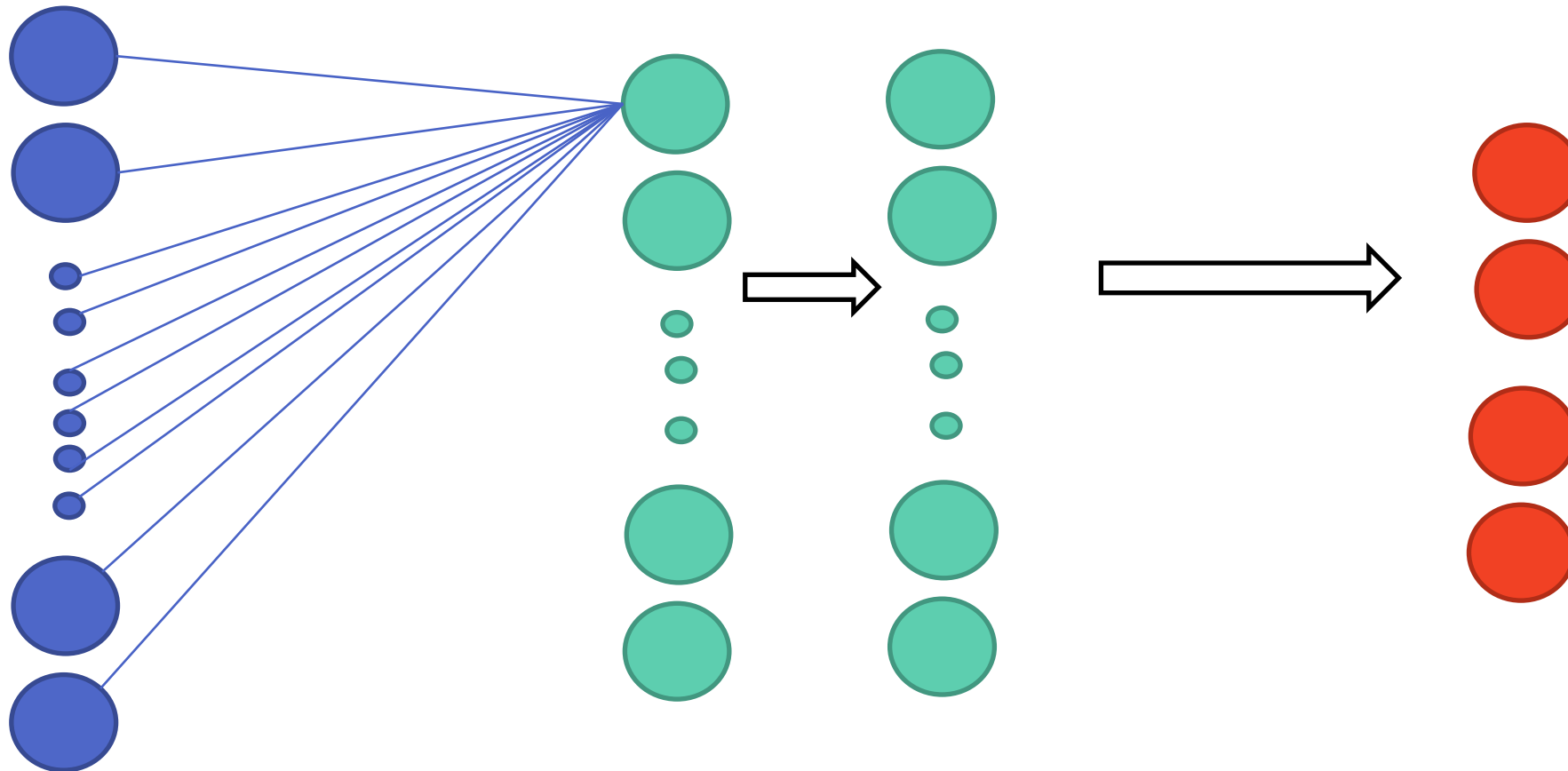
# The count of free parameters - ANN

784 + 1	20	10	Weights in an iteration
	15700		
		210	<u>15,910</u>



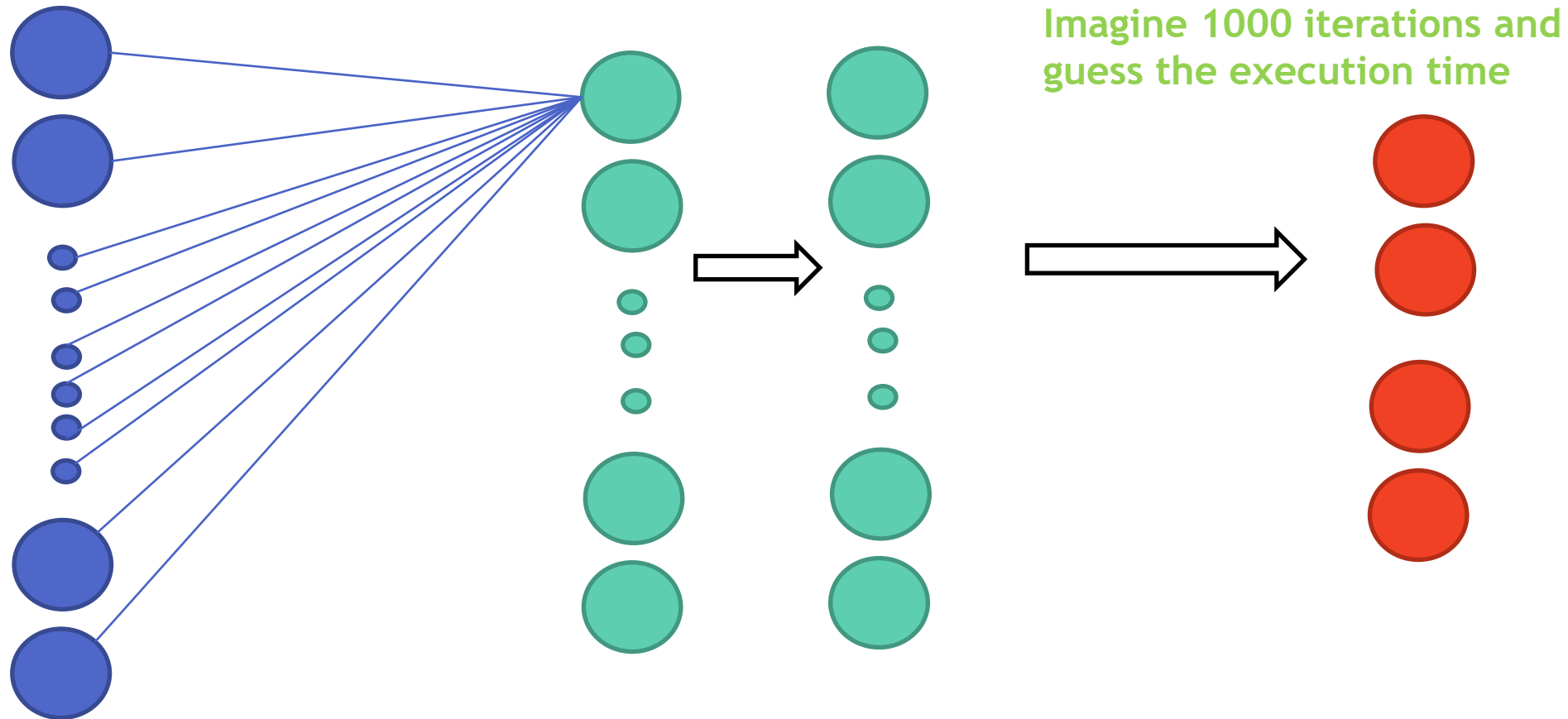
# For a mega pixel image

- Imagine simple ANN for 1 mega pixel image. Imagine with 5000 hidden nodes
- Calculate the number of weights(parameters to update)



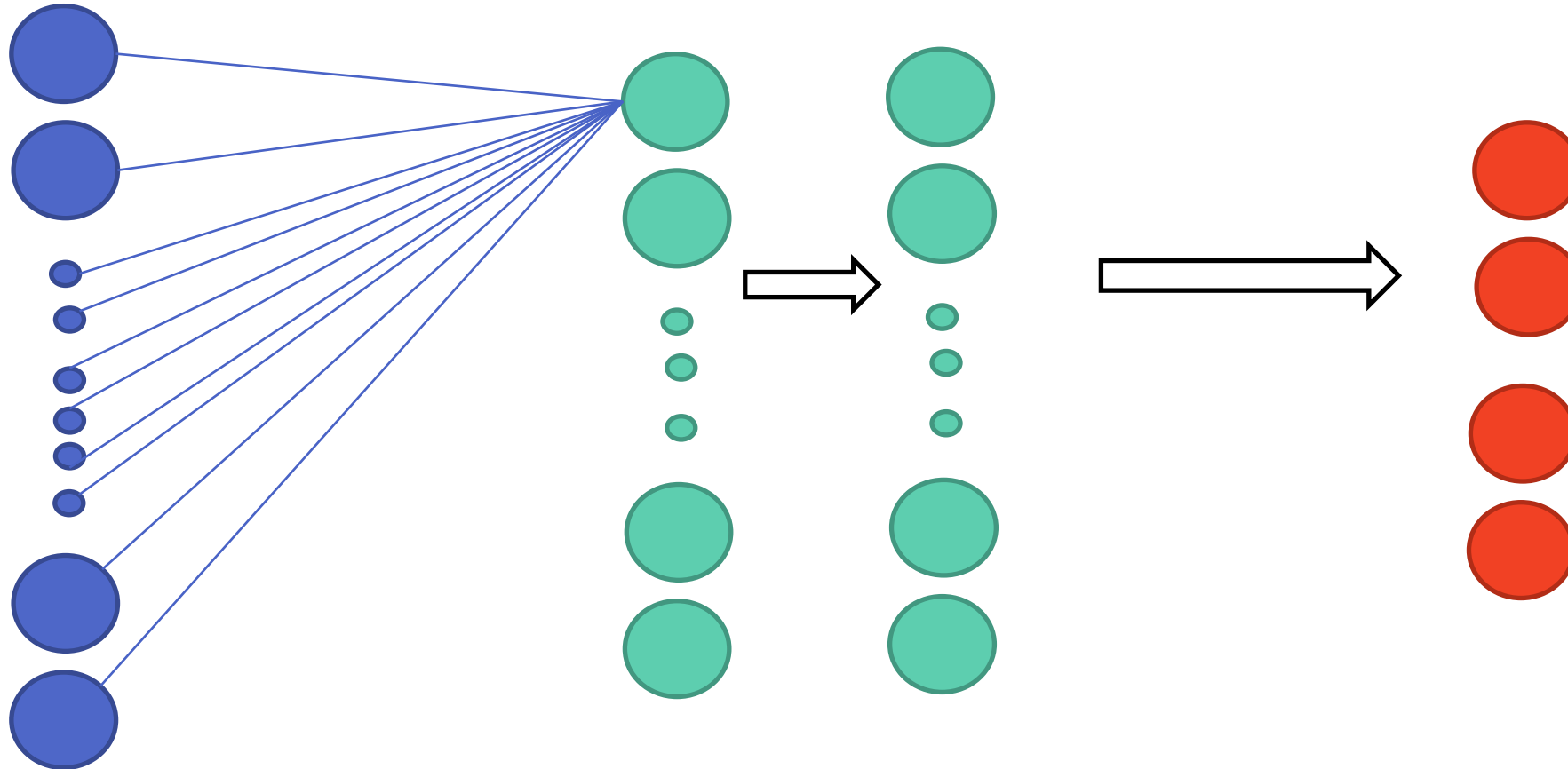
# For a mega pixel image

- Imagine simple ANN for 1 mega pixel image. Imagine with 5000 hidden nodes
- $1,000,000 * 5,000 = 5,000,000,000$  (5 Billion parameters to update)



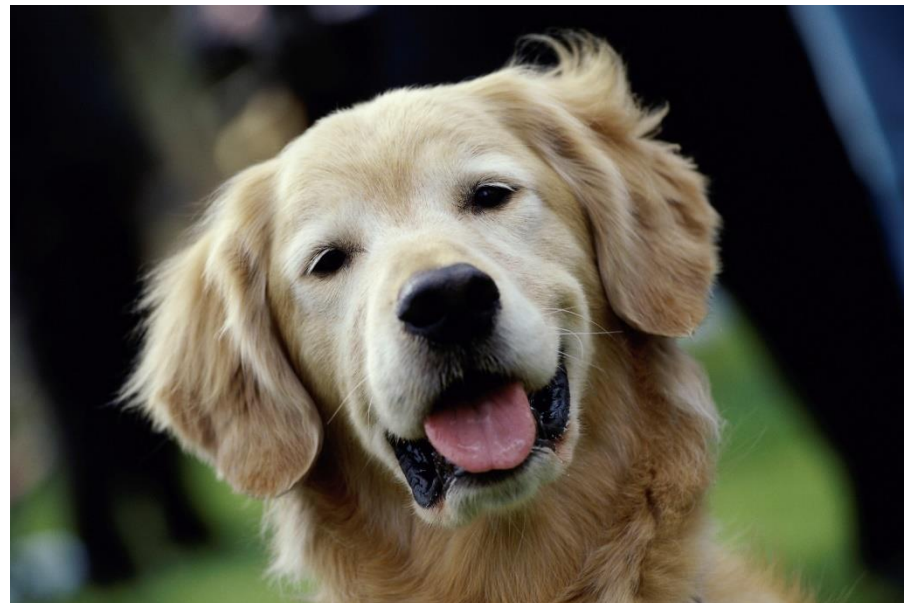
# For a mega pixel image

- Imagine simple ANN for 1 mega pixel image. Imagine with 5000 hidden nodes
- $1,000,000 * 5,000 = 5,000,000,000$  (5 Billion parameters to update)
- $5,000,000,000 * 1000$  updates =  $5,000,000,000,000$  (5 Trillion updates overall)



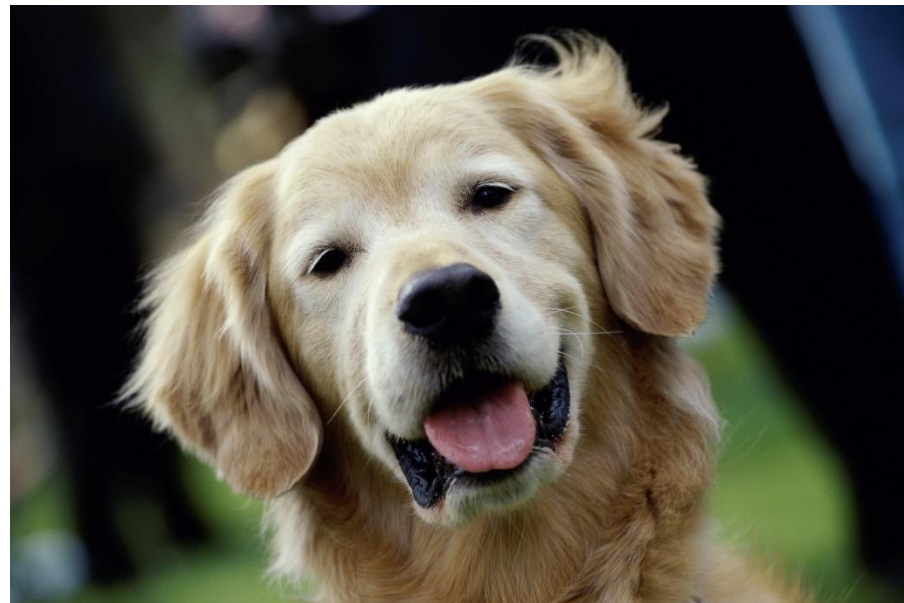
# Issues with large image data

- 5 Trillion is the basic estimate of the calculations, it takes a lot of time
- It is not very smart to take each pixel as input when there is a **spatial dependence** between pixels - Details later



# Issues with large image data

- Taking pixels as it is and building ANN model is almost impossible
- 5 Trillion is the basic estimate of the calculations
- It is almost impossible for most of the machines
- We need an alternative to pixel level calculations - **What is it?**

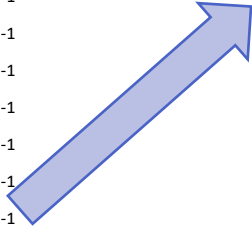




# Spatial dependence

- The issue is not with computations. We can somehow manage computations.
- “taking each pixels as it is” - Is the issue;
- Images have spatial dependence between pixels
- What is **spatial dependence** ?
  - Can you identify a pixel depending on its surrounding pixels?
  - Do you think pixels in image are dependent on their surrounding area(space)?
- Each pixel value is correlated to its surrounding pixels.
  - Eg:
    - Pixels around eye
    - Pixels around nose



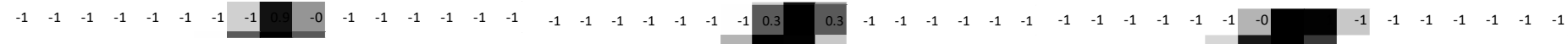
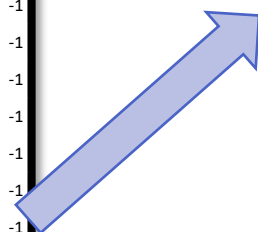


# Special dependence

- Imagine a 28X28 pixel image.
- Can you take it as input and convert to a vector of 754 pixels?
- What happens to special dependency ?
  - Lost
  - Intact?
- The number of calculations inflated majorly due to the loss of Special dependence(While we are preparing data for ANN)
- Is there a way to preserve the special dependency ?

# The Story So Far...

- ANN is a powerful algorithm
- But there are too many parameters to train in mega pixel images
- The training time is overlong
- The spatial integrity is compromised while preparing the data for the ANN the calculations
- We need an alternative to ANN's fully connected network architecture



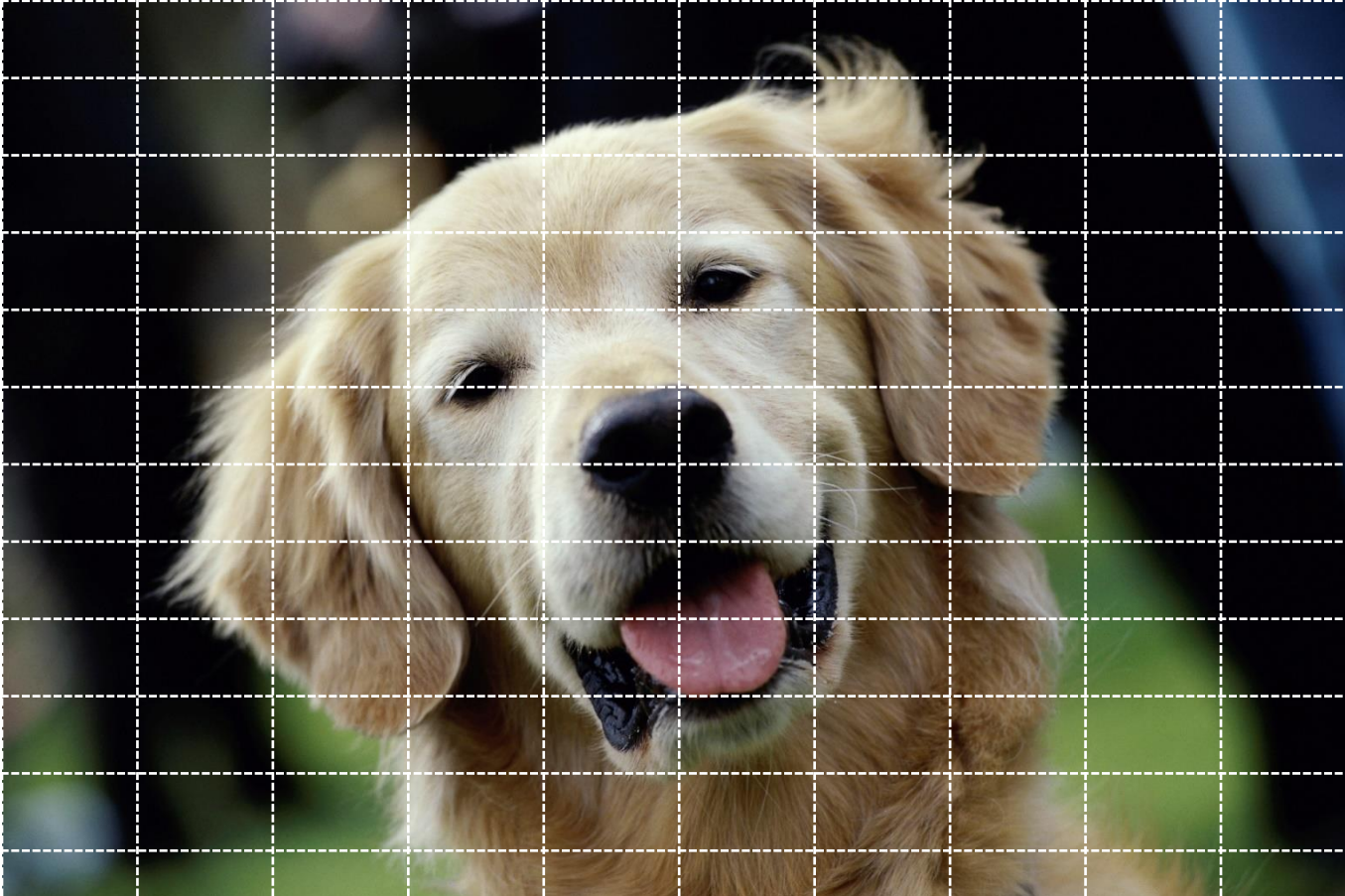
# CNN Uses the Local regions property



- Images have Spatial importance
- Apart from a single pixel, its surrounding pixels also matter while doing the image classification
- Its better to take sub regions in an image instead of single pixel.
- In fact every pixel is a primitive sub region on a picture



# Spatial Local Correlation

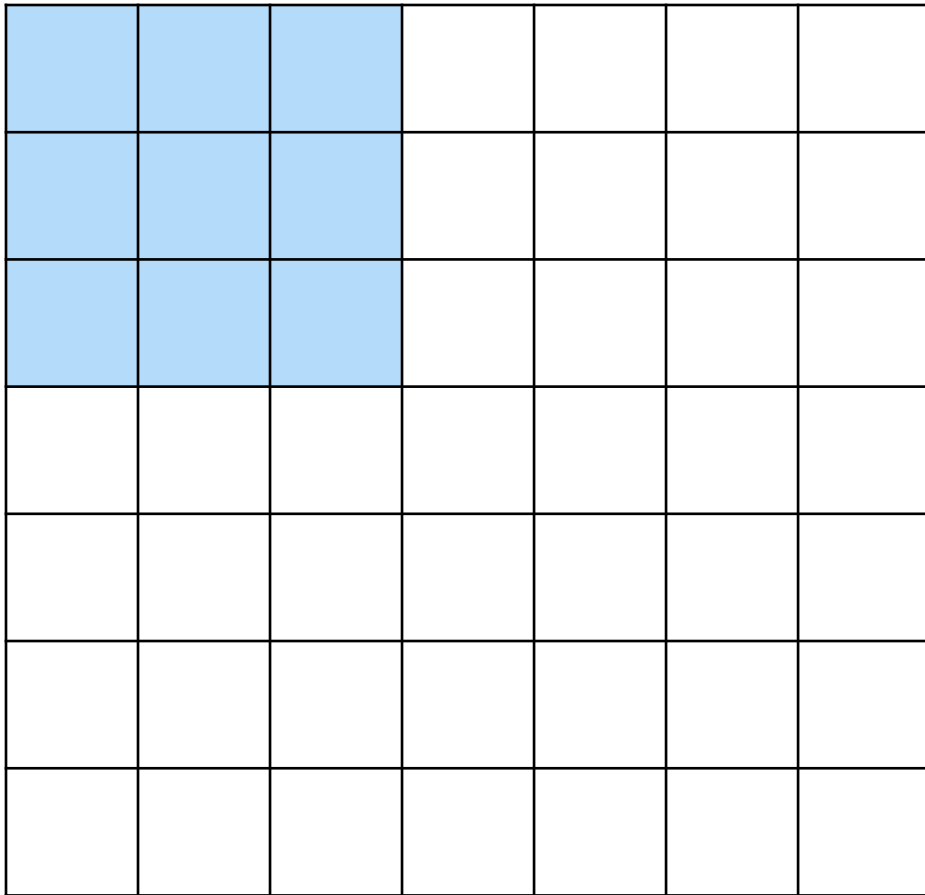


- Images have Spatial importance
- Apart from a single pixel, its surrounding pixels also matter while doing the image classification
- Each pixel value is correlated to its surrounding pixels.
  - Eg: Pixels around eye
- Its better to take sub regions in an image instead of single pixel.
- In fact every pixel is a small sub region on picture

# Filter

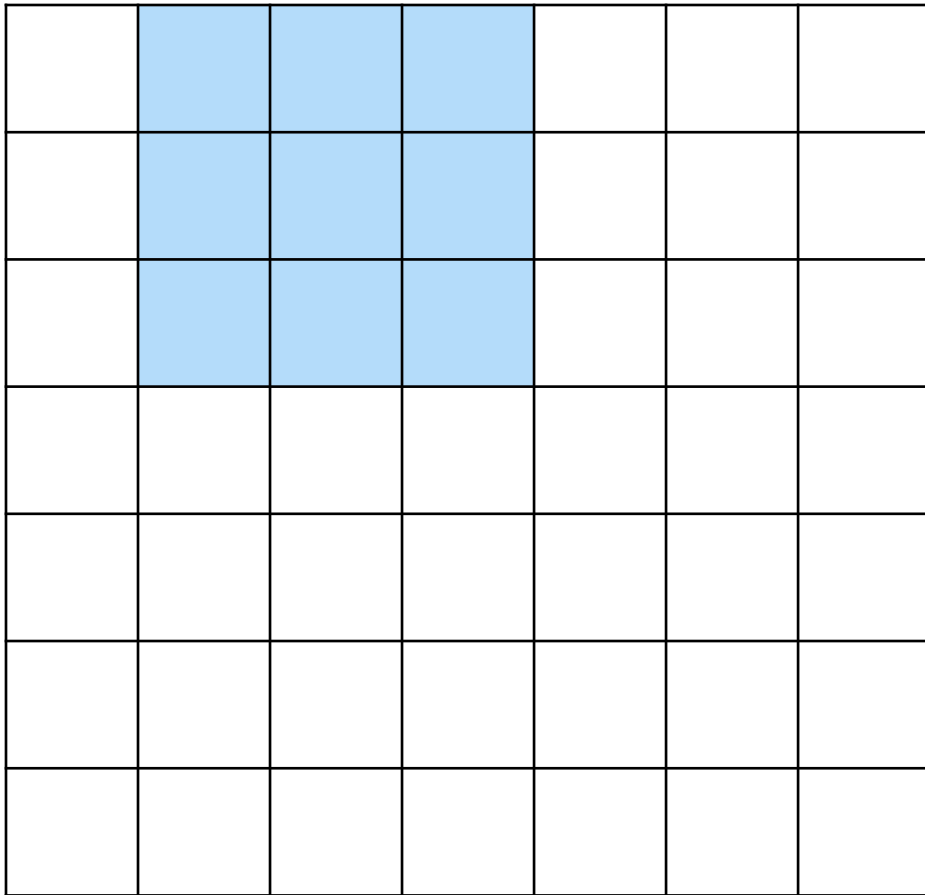
- Sub regions on the image are picked by using filters
- Applying filter is the first step to reduce the computational complexity and to preserve the spatial integrity
- A filter is a sub-region or a portion of surrounding pixels on the image
- Filter is also known as Kernel (Details later)
- How filter works?

# How Filter Works



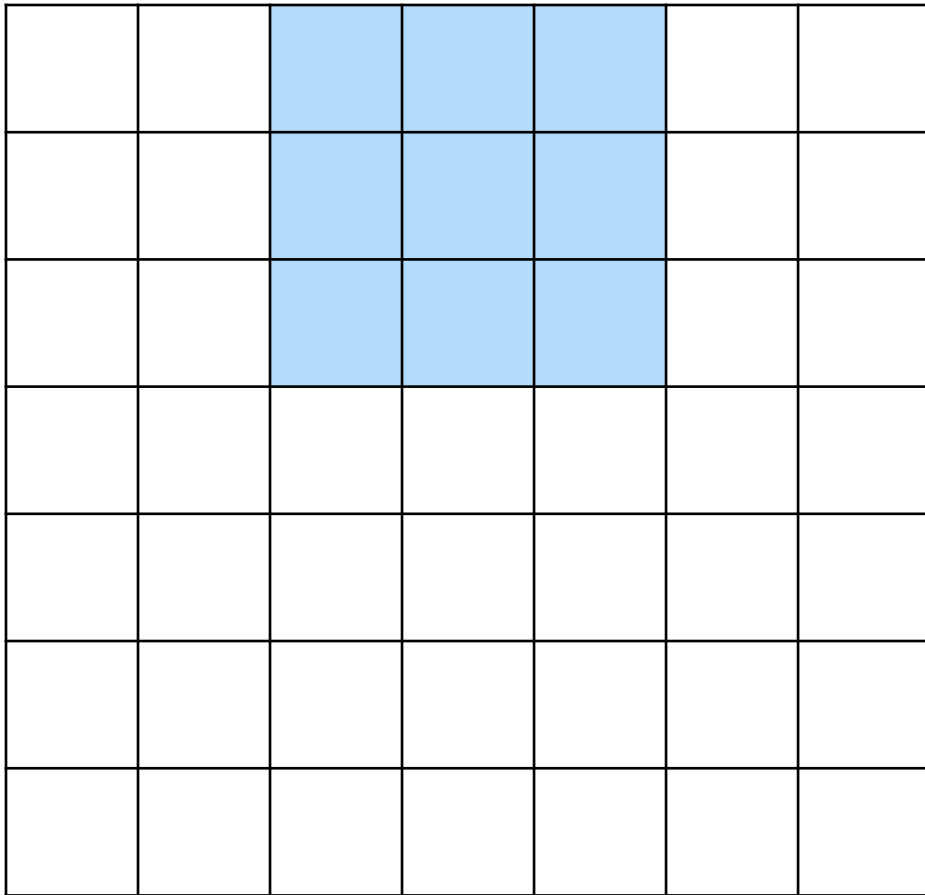
- Imagine a 7X7 pixel image and assume a 3X3 filter

# How Filter Works



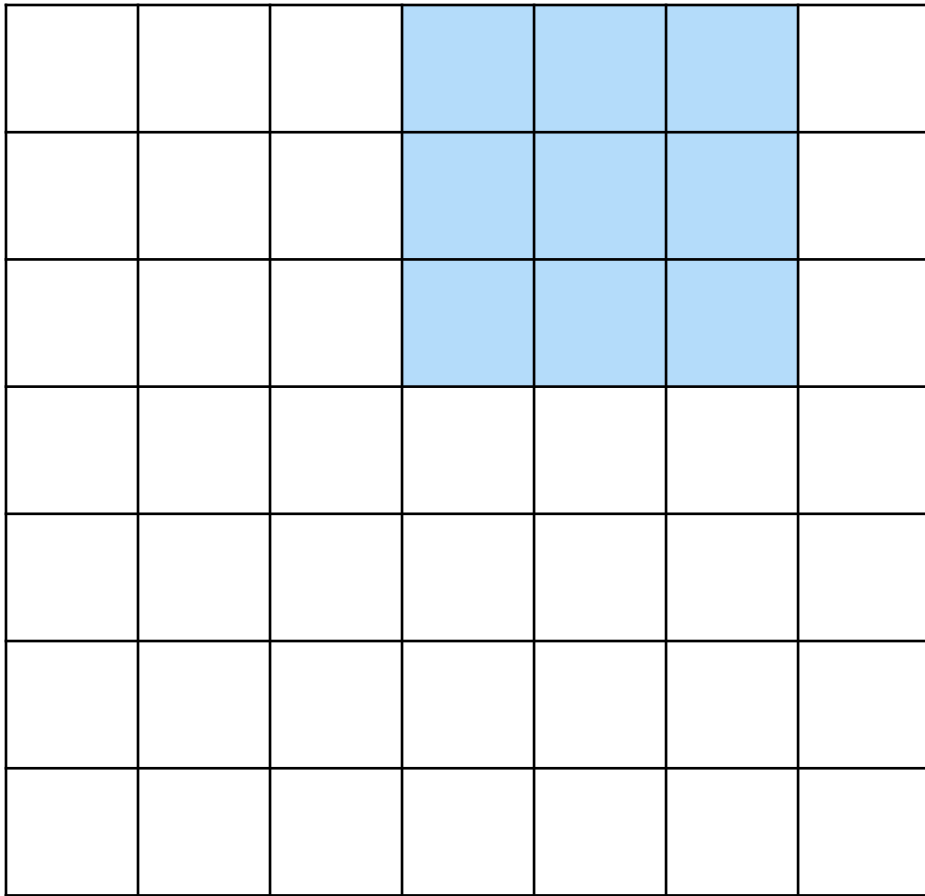
- Imagine a 7X7 pixel image and assume a 3X3 filter

# How Filter Works



- Imagine a 7X7 pixel image and assume a 3X3 filter

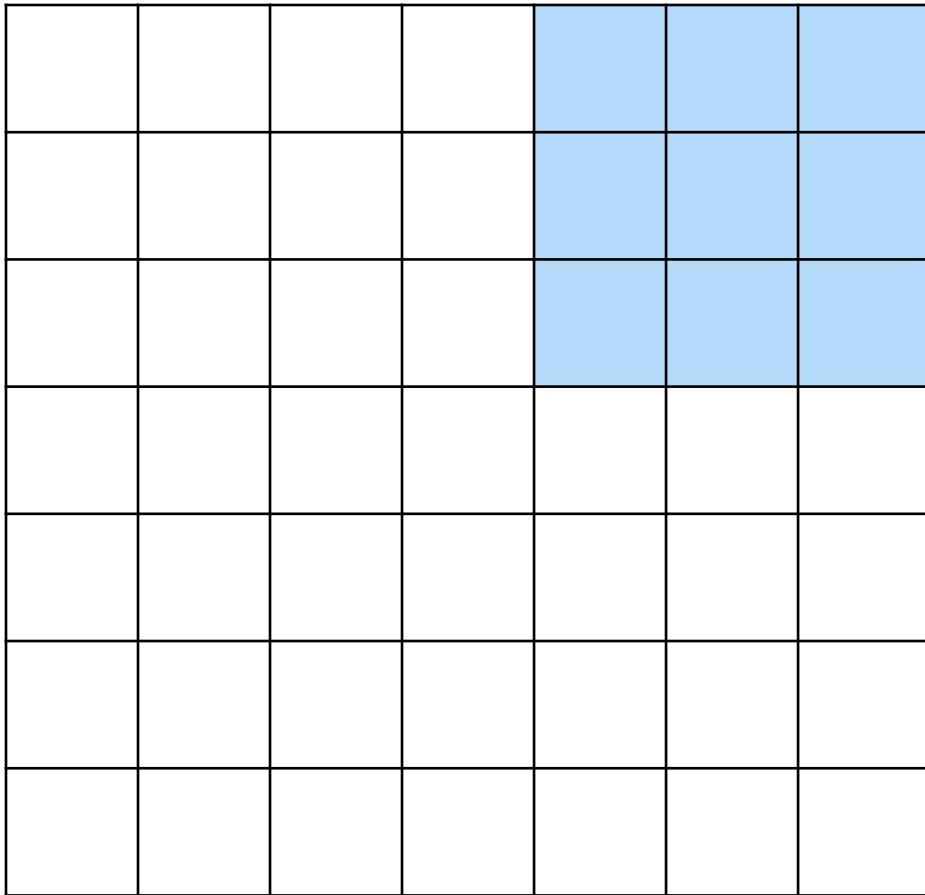
# How Filter Works



- Imagine a 7X7 pixel image and assume a 3X3 filter

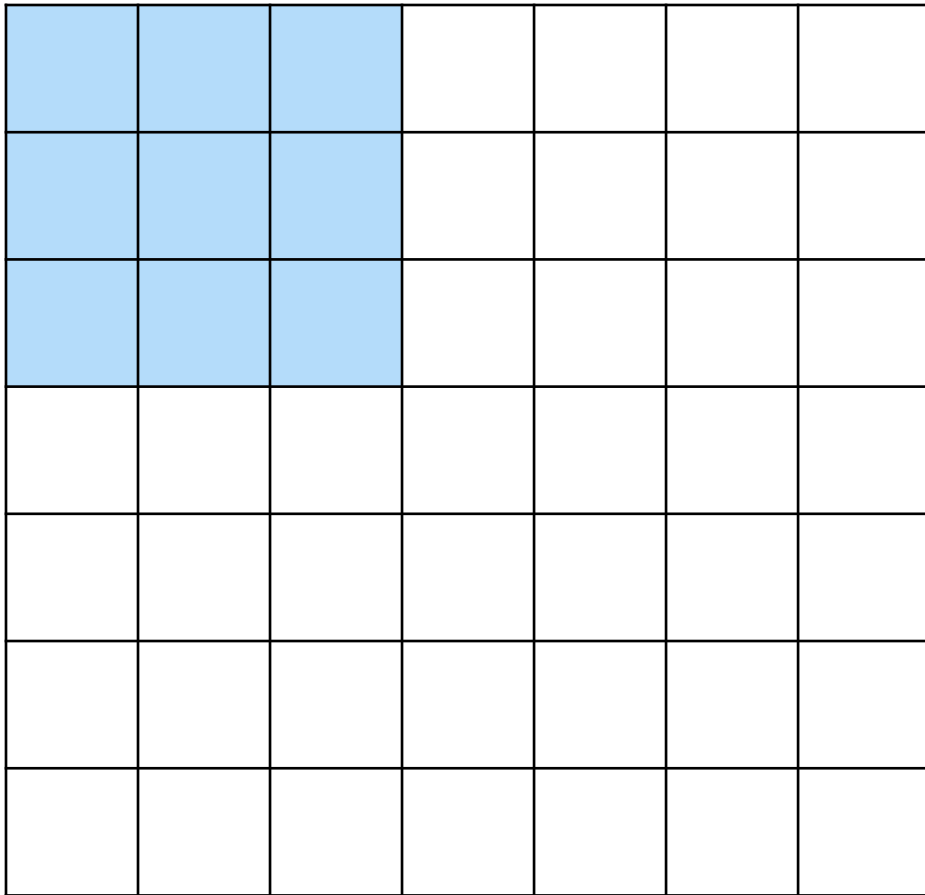


# How Filter Works



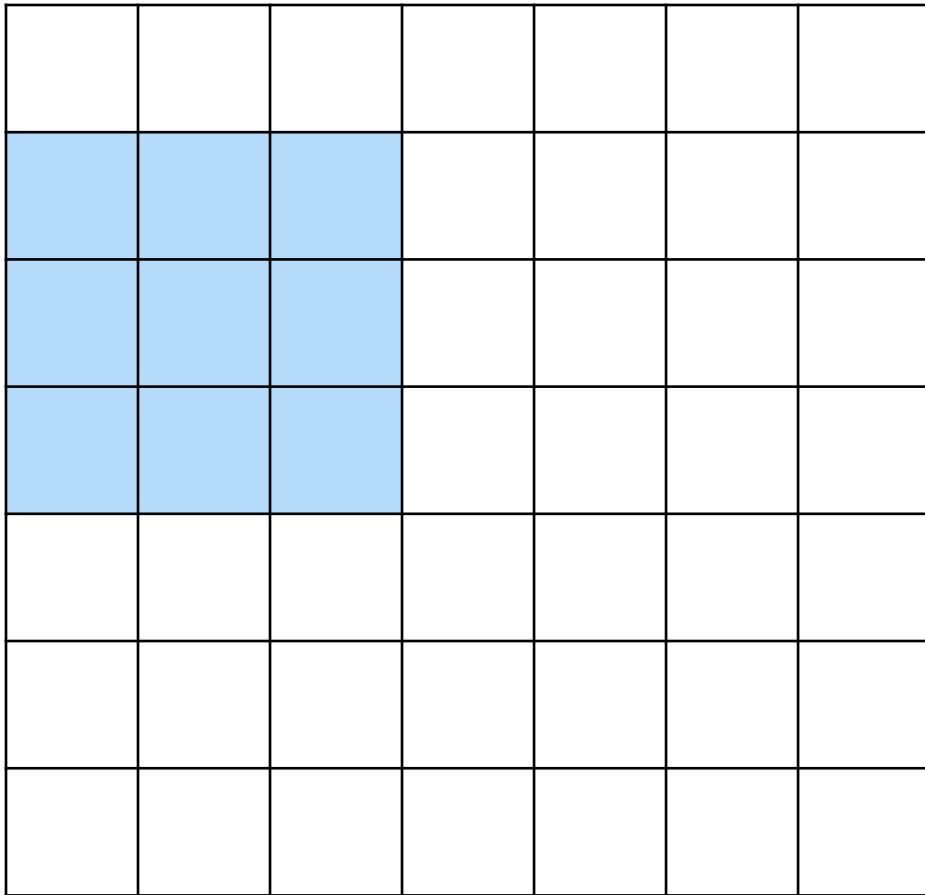
- Imagine a 7X7 pixel image and assume a 3X3 filter

# How Filter Works



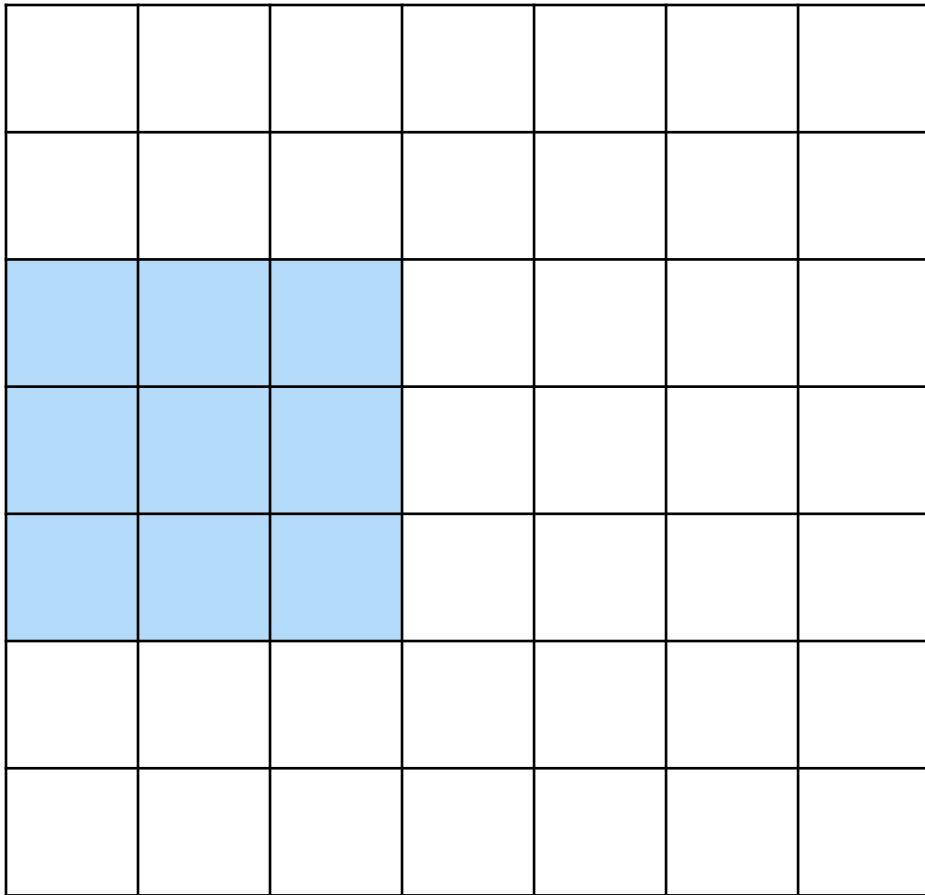
- Imagine a 7X7 pixel image and assume a 3X3 filter

# How Filter Works



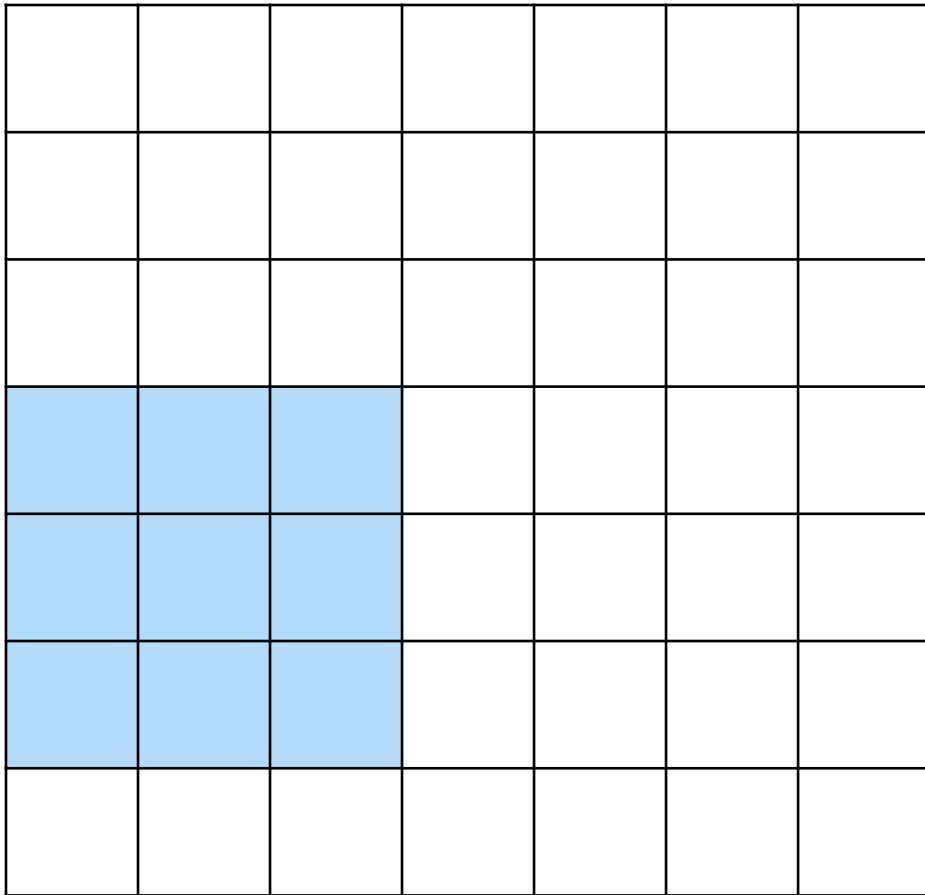
- Imagine a 7X7 pixel image and assume a 3X3 filter

# How Filter Works



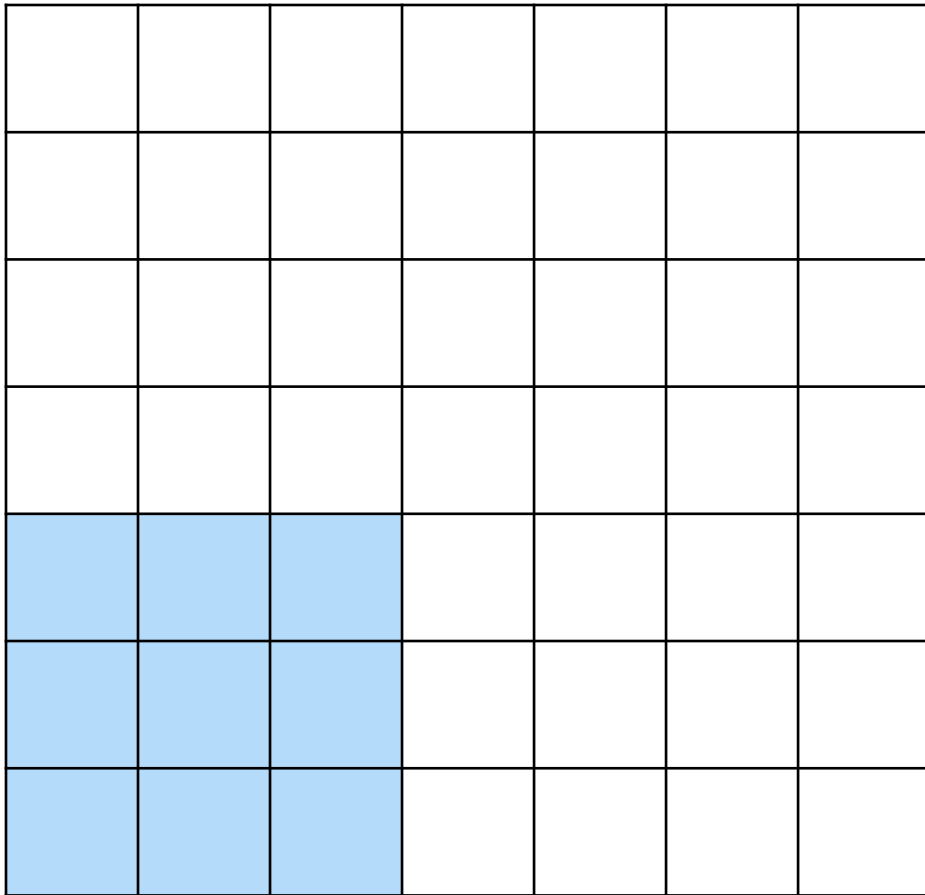
- Imagine a 7X7 pixel image and assume a 3X3 filter

# How Filter Works



- Imagine a 7X7 pixel image and assume a 3X3 filter

# How Filter Works



- Imagine a 7X7 pixel image and assume a 3X3 filter
- A 7X7 image becomes 5X5 Image in the next layer.
- We moved the filter by stride-1 across the original image
- We can also move it by stride-2

- A dot product with the filter region

6				

- A dot product with the filter region

6	5			



# Pixel convolved to new features

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

- A dot product with the filter region

1	1	1
1	1	1
1	1	1

6	5	4		

# Pixel convolved to new features

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

- A dot product with the filter region

1	1	1
1	1	1
1	1	1

6	5	4	6	

# Pixel convolved to new features

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

- A dot product with the filter region

1	1	1
1	1	1
1	1	1

6	5	4	6	6
3	3	3	6	6
3	2	2	4	5
3	2	2	4	5
6	4	3	4	5

# Pixel convolved to new features

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

1	1	1
1	1	1
1	1	1

6	5	4	6	6
3	3	3	6	6
3	2	2	4	5
3	2	2	4	5
6	4	3	4	5

- Also known as kernel matrix
- Kernel with all 1's is an unweighted smoothing kernel

# Effective Convolution with Kernel Matrix

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

kernel matrix

1	1	1
1	1	1
1	1	1

- Use Kernel
  - To enhance intensity
  - or to make the image smooth
  - or to make it sharp

6	5	4	6	6
3	3	3	6	6
3	2	2	4	5
3	2	2	4	5
6	4	3	4	5

kernel matrix

- There are different versions of kernels

3				

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

kernel matrix

0	1	0
1	1	1
0	1	0

- There are different versions of kernels

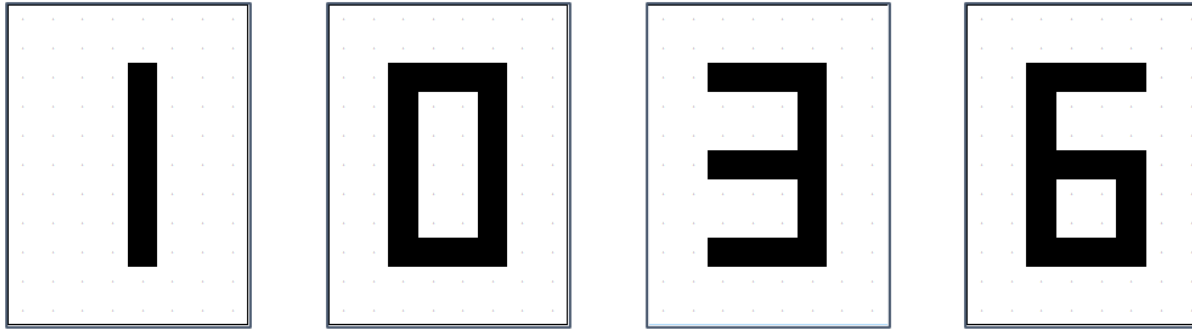
3	4			

# Many Filters in Convolution Layer

- Each Kernel matrix captures a specific type of pattern in input data
- Apply different filters by changing the kernel function
- There are many kernel functions/filters
  - Filter for detecting the curves
  - Filter for detecting the circles
  - Filter for detecting the sharp edges
  - Filter for detecting the straight lines



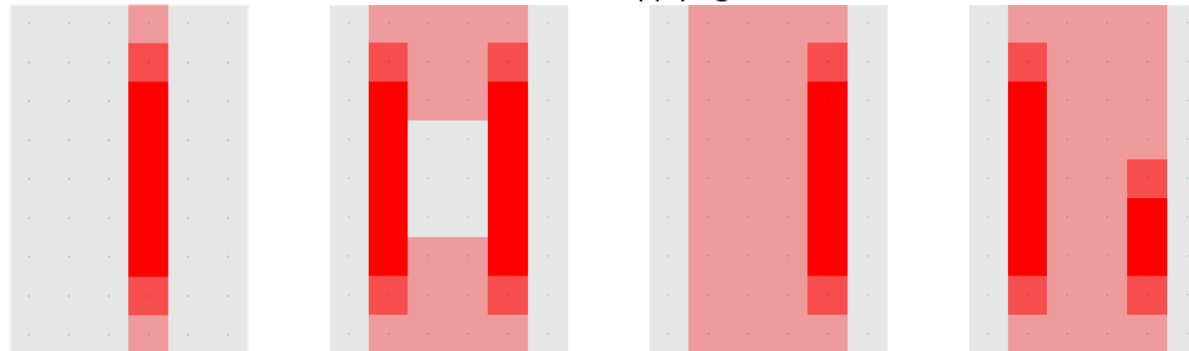
# Kernels for detecting features



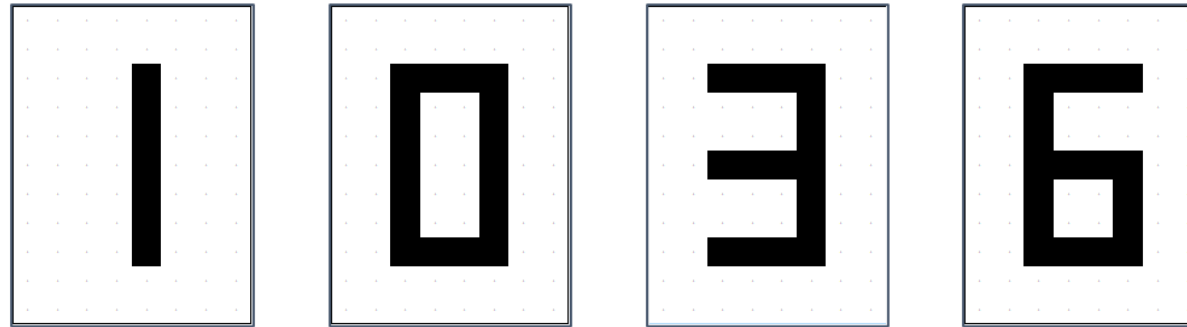
Kernel for detecting vertical lines in the images

0	1	0
0	1	0
0	1	0

Convolved features after applying the above filter



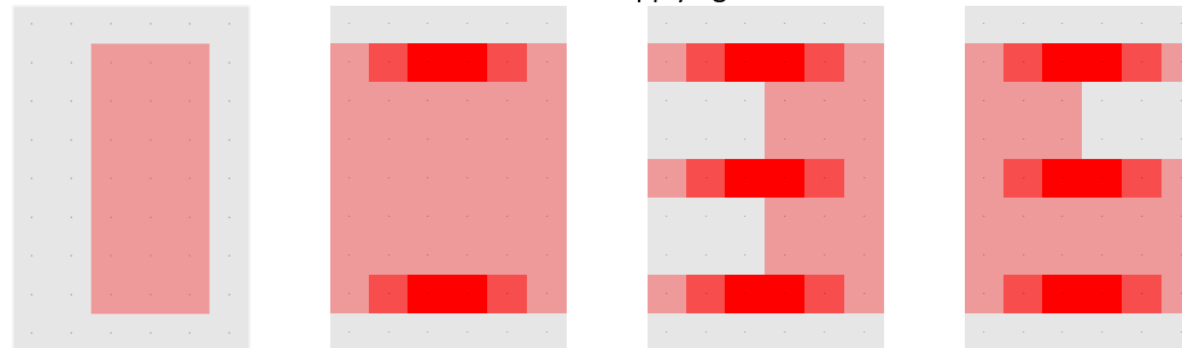
# Kernels for detecting features



Kernel for detecting Horizontal lines in the images

0	0	0
1	1	1
0	0	0

Convolved features after applying the above filter



# Important note

- There are many kernel functions/filters
  - Filter for detecting the curves
  - Filter for detecting the circles
  - Filter for detecting the sharp edges
  - Filter for detecting the straight lines
- **We don't need to provide kernel matrix manually**
- **If we give kernel with random values, it will automatically detect the features.**
- **That is the major advantage of CNN Algorithm.**

# Features are detected based on the data

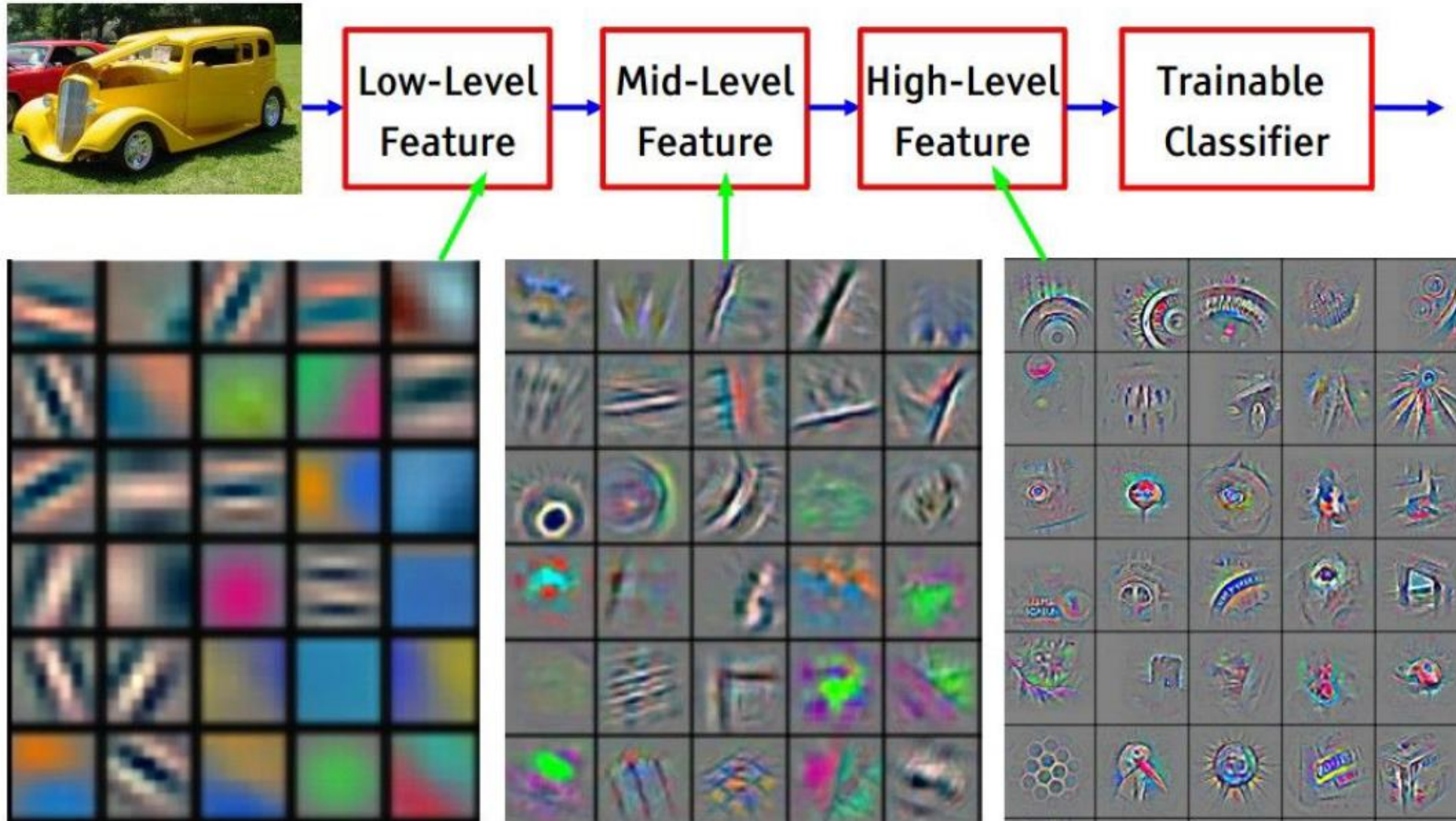
1	1	1	0	0	1	1
1	1	0	1	0	1	1
1	0	0	1	1	1	0
0	0	0	0	0	1	1
1	1	0	0	0	0	1
1	0	0	1	1	1	0
1	1	1	0	0	1	1

kernel matrix

w1	w2	w3
w4	w5	w6
w7	w8	w9


- We don't need to provide the values of kernel matrix manually
- If we give kernel with random values, it will automatically detect the features based on data

# Convolved features



# Multiple Filters

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

0	0	1
0	1	1
1	1	1

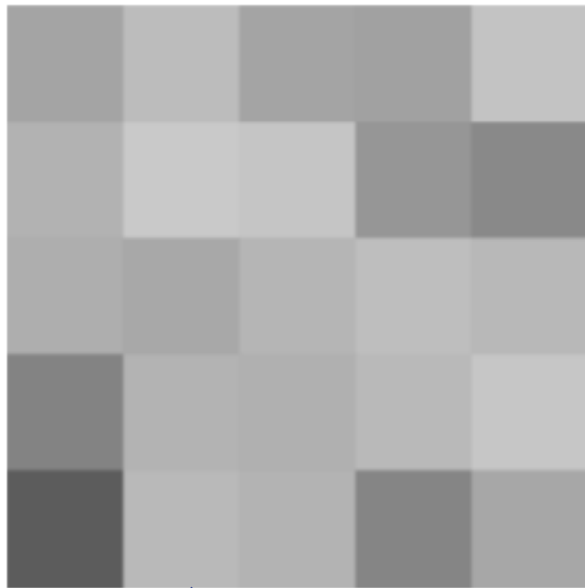
1	1	1
1	1	0
1	0	0

0	0	15
0	15	0
15	0	0

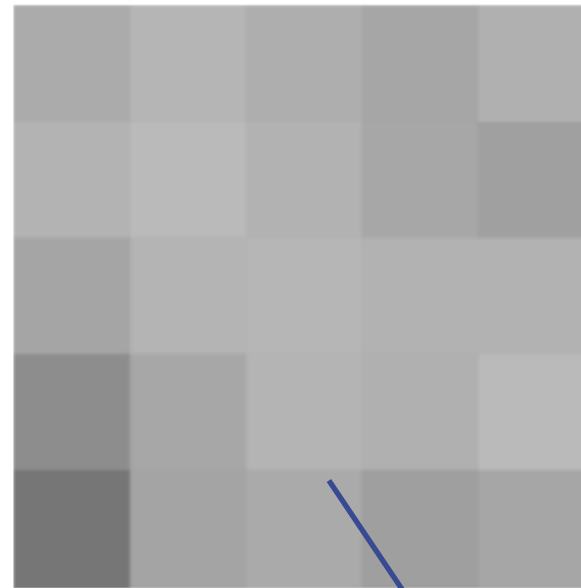
0	10	0
10	10	10
0	10	0

Different kernel matrices

# Convolved features



Actual Pixels



Convolved features- Smoothed image

# Kernel Matrix examples

1	1	1
1	1	1
1	1	1

Unweighted 3x3 smoothing kernel

0	1	0
1	4	1
0	1	0

Weighted 3x3 smoothing kernel with Gaussian blur

0	-1	0
-1	5	-1
0	-1	0

Kernel to make image sharper

-1	-1	-1
-1	9	-1
-1	-1	-1

Intensified sharper image



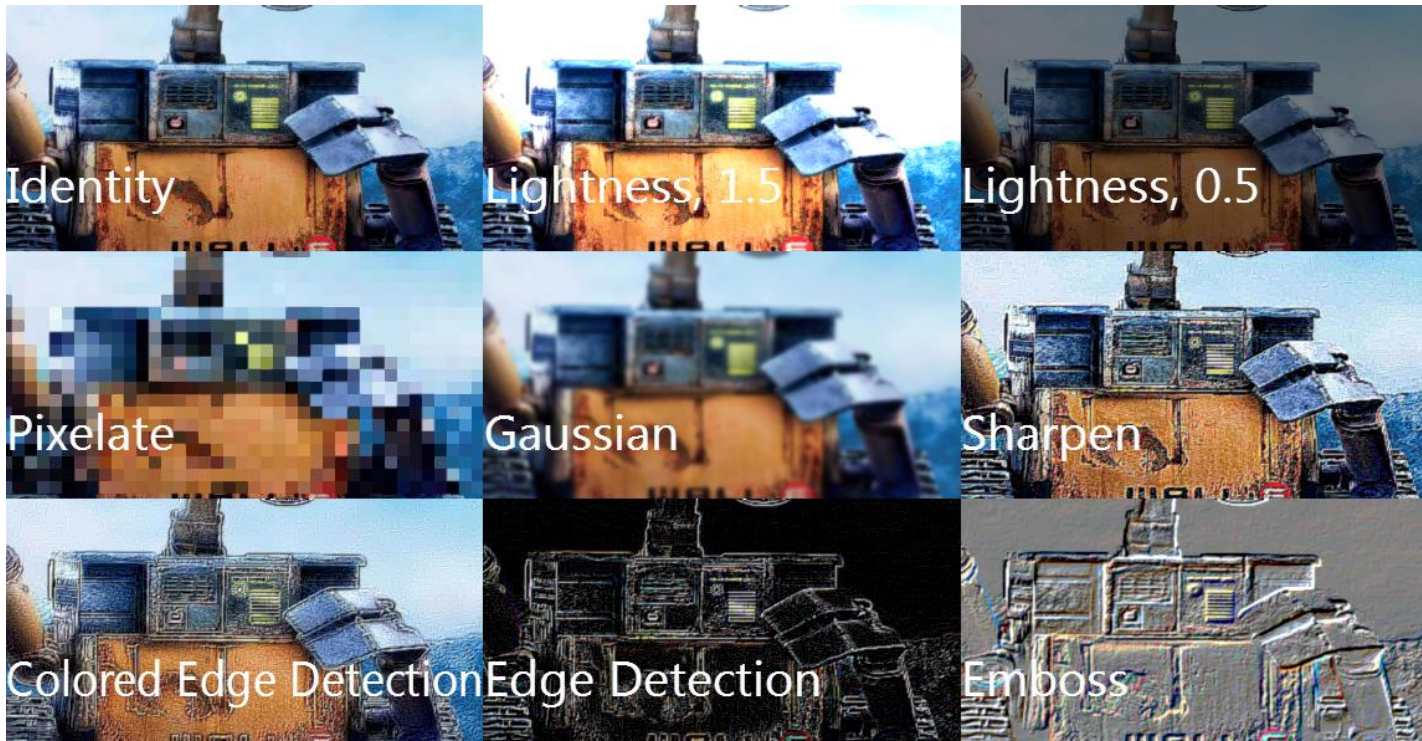
Gaussian Blur



Sharpened image



# Kernel Matrix examples



- Sharpen – 
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- Blur – 
$$\begin{bmatrix} 0 & 0.2 & 0 \\ 0.2 & 0.2 & 0.2 \\ 0 & 0.2 & 0 \end{bmatrix}$$

- Horizontal Motion Blur – 
$$\begin{bmatrix} 0 & 0 & 0 \\ 0.2 & 0.2 & 0.2 \\ 0 & 0 & 0 \end{bmatrix}$$

- Edge detect – 
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

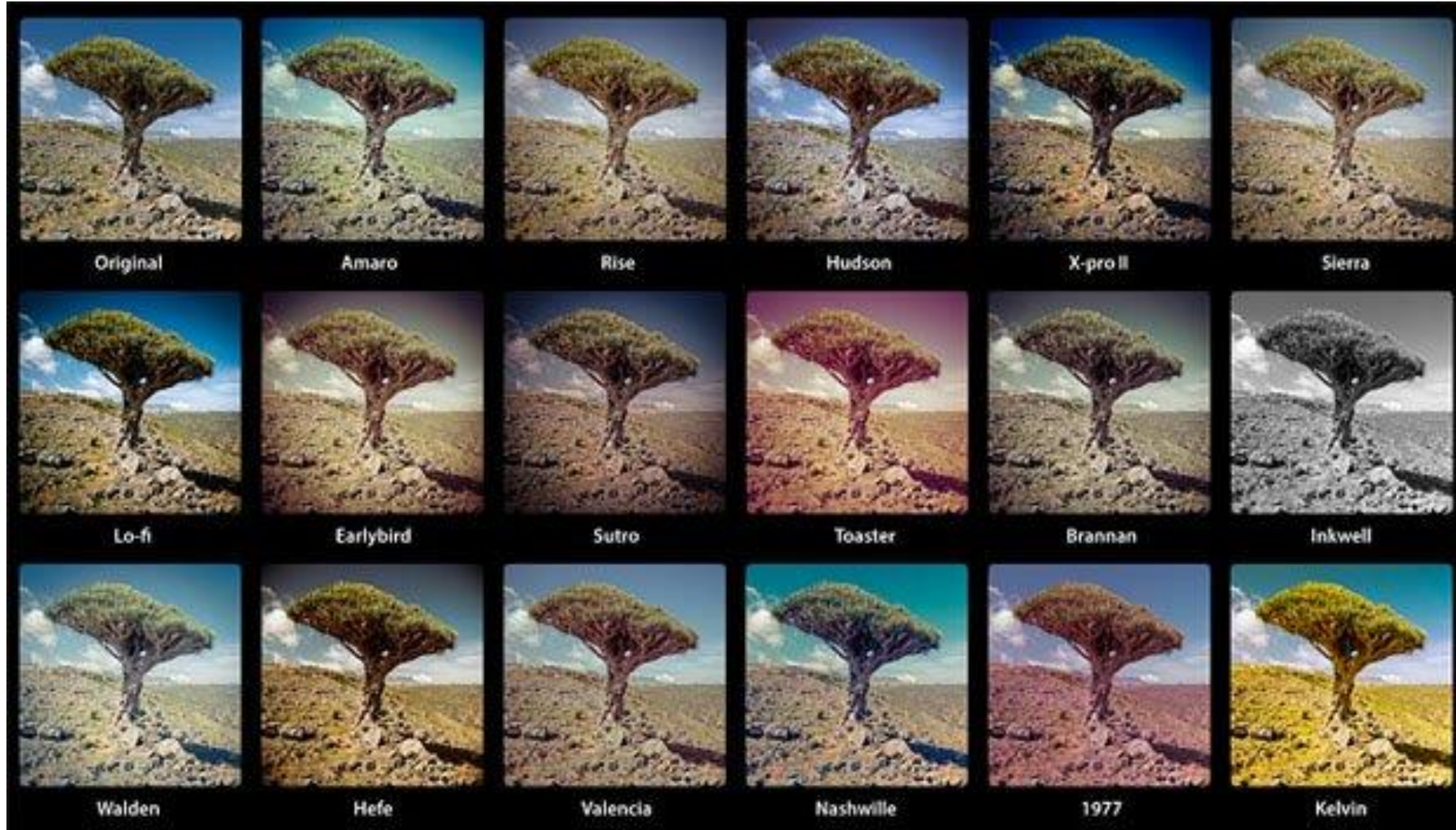
# Convolved features



We are looking for this type  
of convolved features



# Kernel Filters in Instagram ?





# Kernel Filters in Instagram ?





# Remember ...features are detected based on the data

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

kernel matrix

w1	w2	w3
w4	w5	w6
w7	w8	w9


- We don't need to provide the values of kernel matrix manually
- If we give kernel with random values, it will automatically detect the features based on data

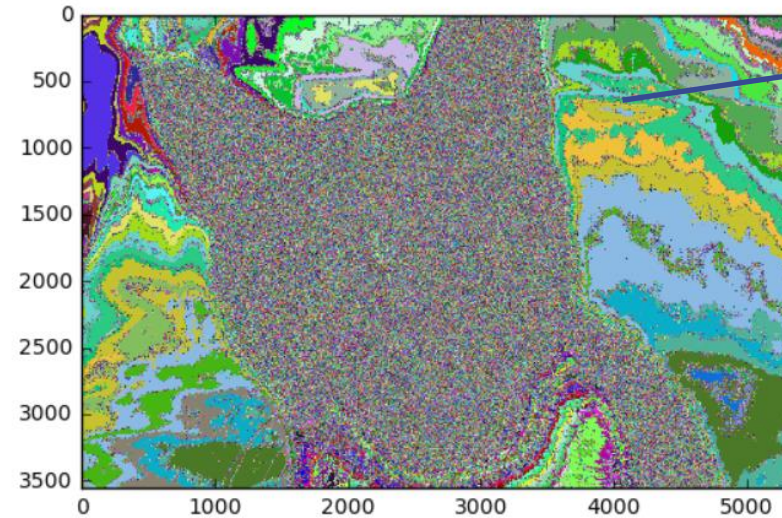
# What is happening in Convolution Layer

- What is a convolution layer
  - A new feature created from input layer by applying a filter
- What is the advantage of convolution layer
  - Spatial dependency is preserved.
  - Lesser data and lesser weights to optimise
- Are we losing any data at convolution layer
  - Yes, but we preserved the local correlation

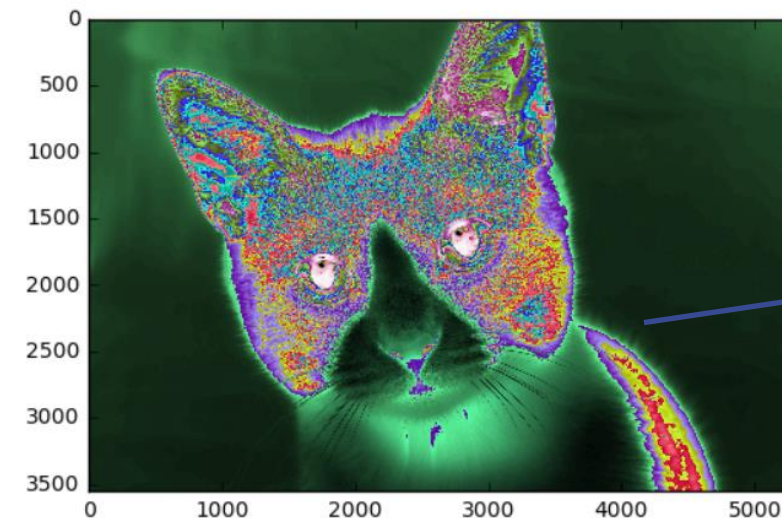


# Convoluted features

Different filters capture different features



A filter to capture overall shape and edges



A filter to ignore white spaces on image



# Note: Colour Images have depth

- Black and white images have just intensity value
- Colour images have depth, 3 colours Intensity of RGB.
  - Red
  - Green
  - Blue
- Actual images are considered as 3 dimensional in image processing terminology

# LAB: Convolution Layer

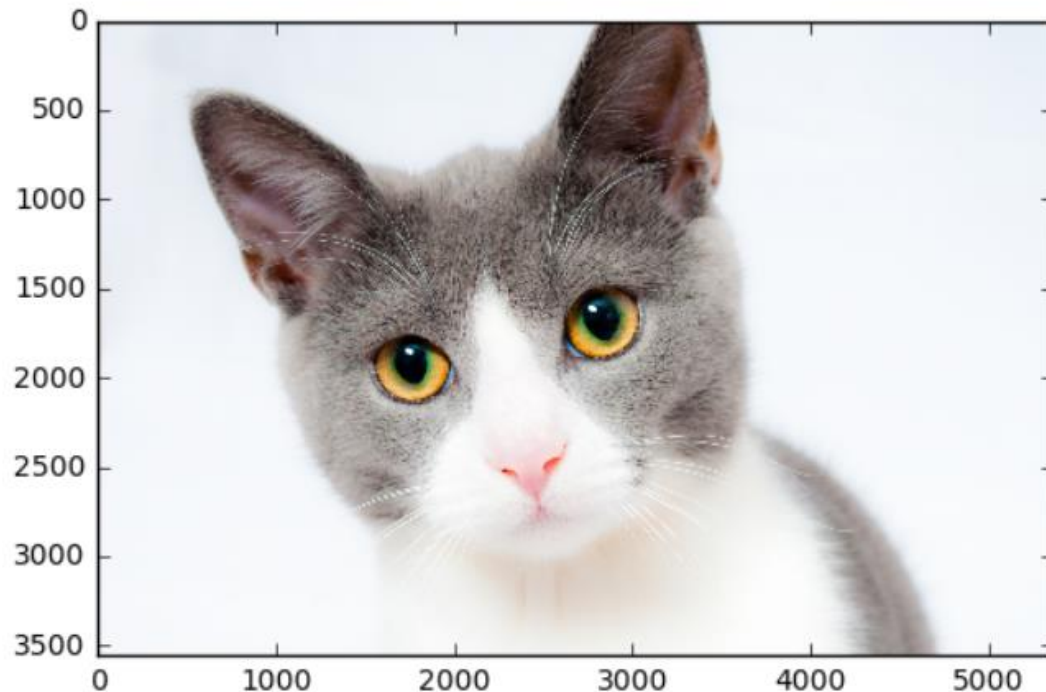
- Import cat.jpeg
- How many pixels are there in the image?
- Apply a 5X5 filter with random values as filter values.

# Code: Convolution Layer

```
import numpy as np
import matplotlib.pyplot as plt
```

```
%matplotlib inline
x=plt.imread('D:\\Dropbox\\39. Deep Learning\\First Version References\\4. CNN\\Datasets\\cat.jpeg')
plt.imshow(x)
#Pixels in this image
print('Shape of the image',x.shape)
```

Shape of the image (3560, 5360, 3)

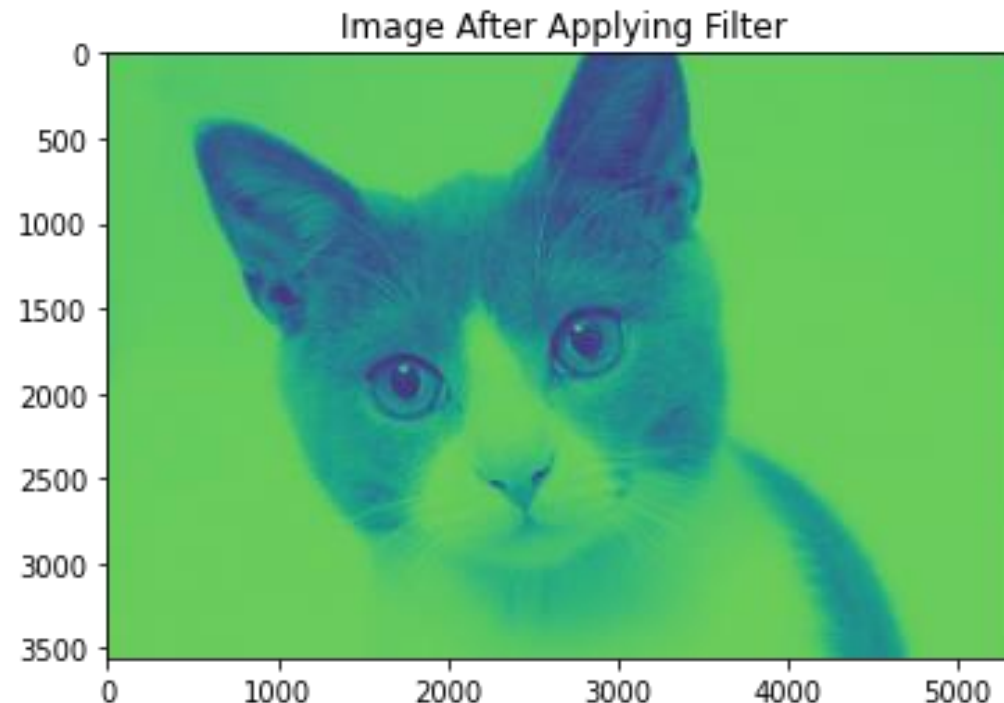


# Code: Convolution Layer

```
model=Sequential()
```

```
model.add(Conv2D(filters=1, kernel_size=(5,5), input_shape=x.shape, kernel_in  
initializer='random_uniform'))
```

```
model.summary()
```





# The depth in convolution layer

---

# The depth in convolution layer

- Every filter gives us a resultant matrix (activation map)

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

1	1	1
1	1	1
1	1	1

4	5	4	2	3	4	4
5	6	5	4	6	6	5
3	3	3	3	6	6	5
3	3	2	2	4	5	4
3	3	2	2	4	5	4
5	6	4	3	4	5	4
3	4	3	3	4	4	3

# The depth in convolution layer

- With two filters we will get two activation maps i.e depth=2

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

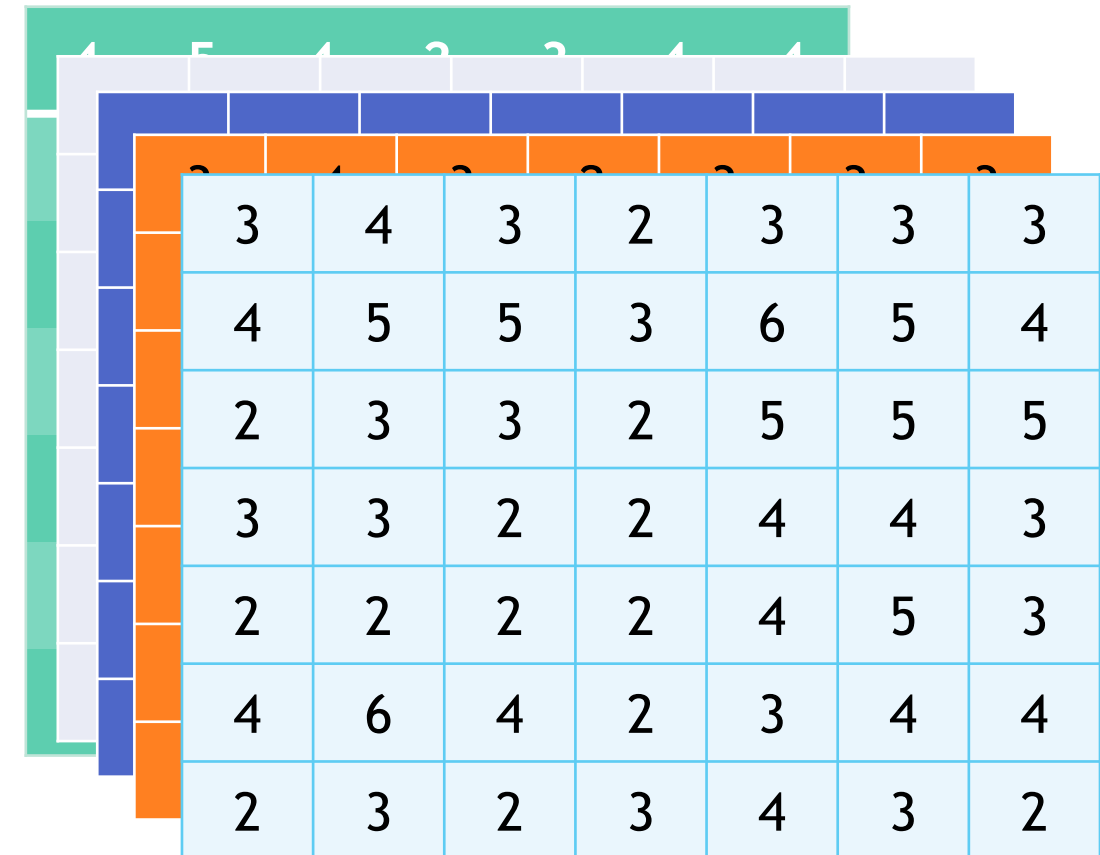
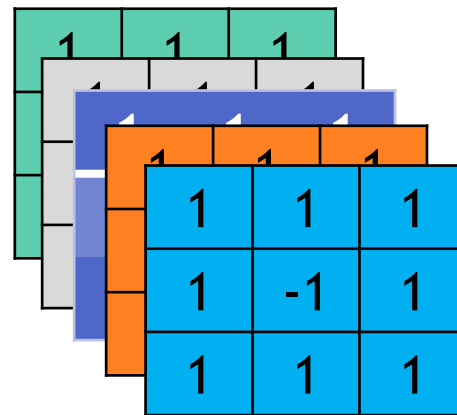
1	1	1
1	1	1
1	1	1
1	1	1
1	1	1
1	-1	1
1	1	1

4	5	4	2	3	4	4	
5	3	4	3	2	3	3	3
3	4	5	5	3	6	5	4
3	2	3	3	2	5	5	5
3	3	3	2	2	4	4	3
3	2	2	2	2	4	5	3
5	4	6	4	2	3	4	4
3	2	3	2	3	4	3	2

# The depth in convolution layer

- If we apply 10 different filters then we will get 10 resultant matrices. The depth is 10

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

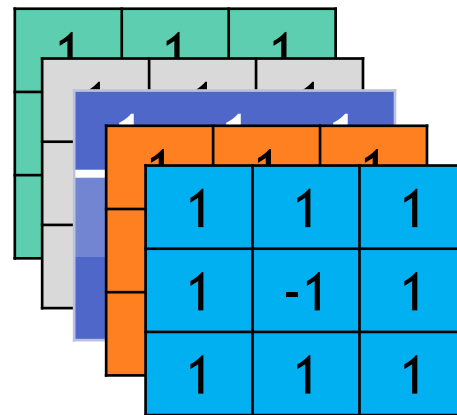
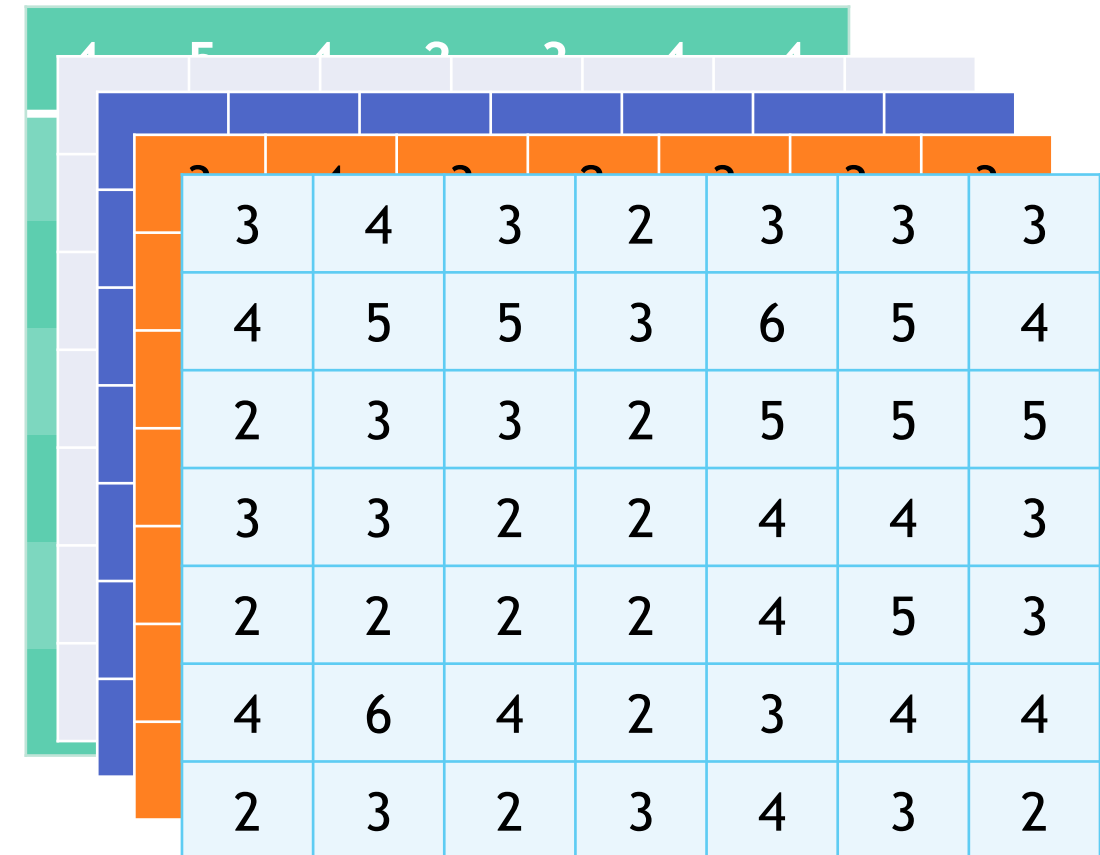




# The depth in convolution layer

- In this example the input is 7X7 with 1 padding and five filters. Output volume size will be 7X7X5.

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

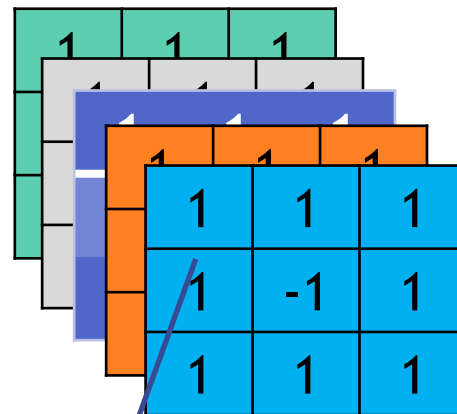



4	5	4	3	3	4	4
3	4	3	2	3	3	3
4	5	5	3	6	5	4
2	3	3	2	5	5	5
3	3	2	2	4	4	3
2	2	2	2	4	5	3
4	6	4	2	3	4	4
2	3	2	3	4	3	2

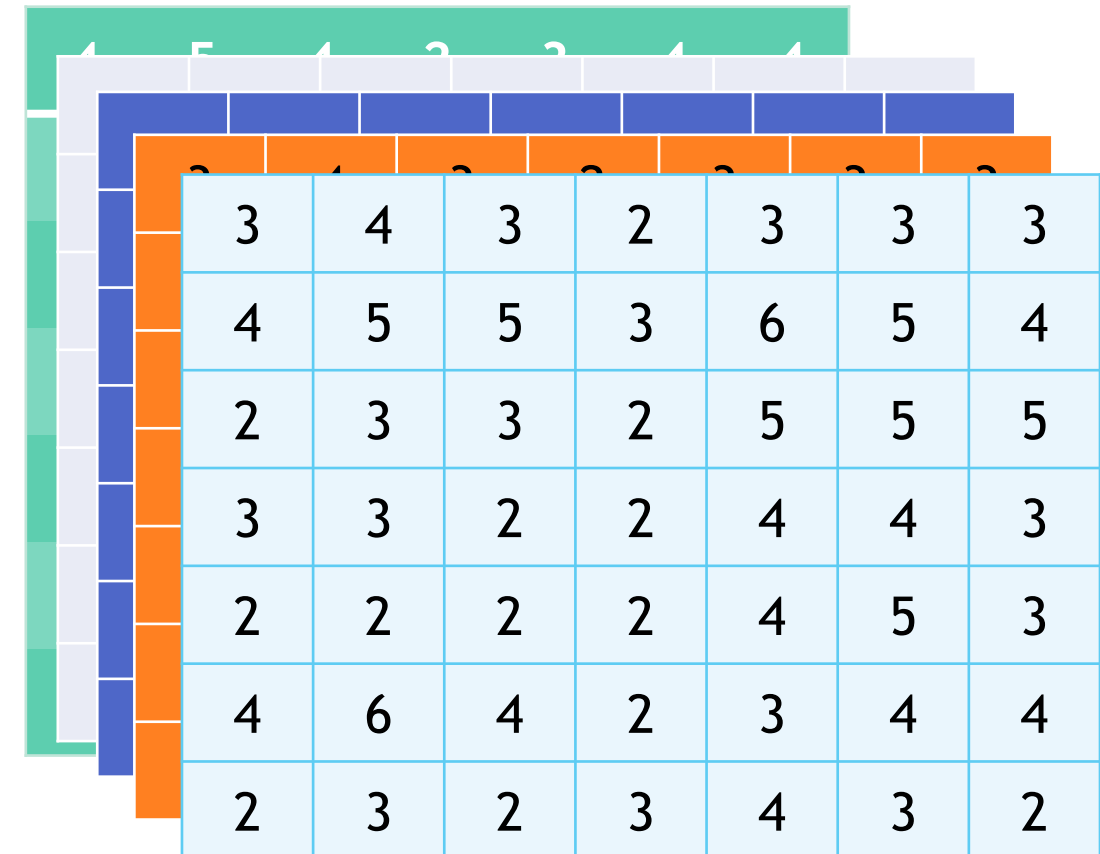
# The number of weights

- In this example the input is 7X7 with 1 padding and five filters. Output volume size will be 7X7X5.

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



Each value from filter matrix will have one weight



3	4	3	2	3	3	3
4	5	5	3	6	5	4
2	3	3	2	5	5	5
3	3	2	2	4	4	3
2	2	2	2	4	5	3
4	6	4	2	3	4	4
2	3	2	3	4	3	2

# The number of weights in conv layer

- In this example we have a filter of size 3X3. The number of weights will be  $3 \times 3 = 9$  weights.

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

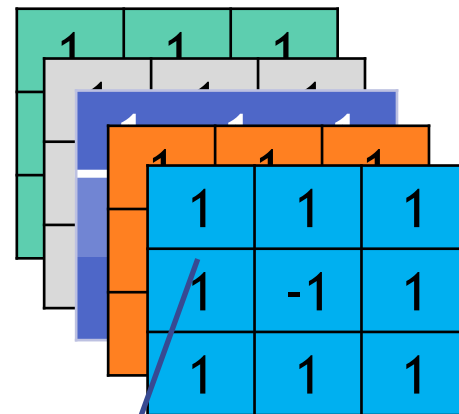
1	1	1
1	1	1
1	1	1

4	5	4	2	3	4	4
5	6	5	4	6	6	5
3	3	3	3	6	6	5
3	3	2	2	4	5	4
3	3	2	2	4	5	4
5	6	4	3	4	5	4
3	4	3	3	4	4	3

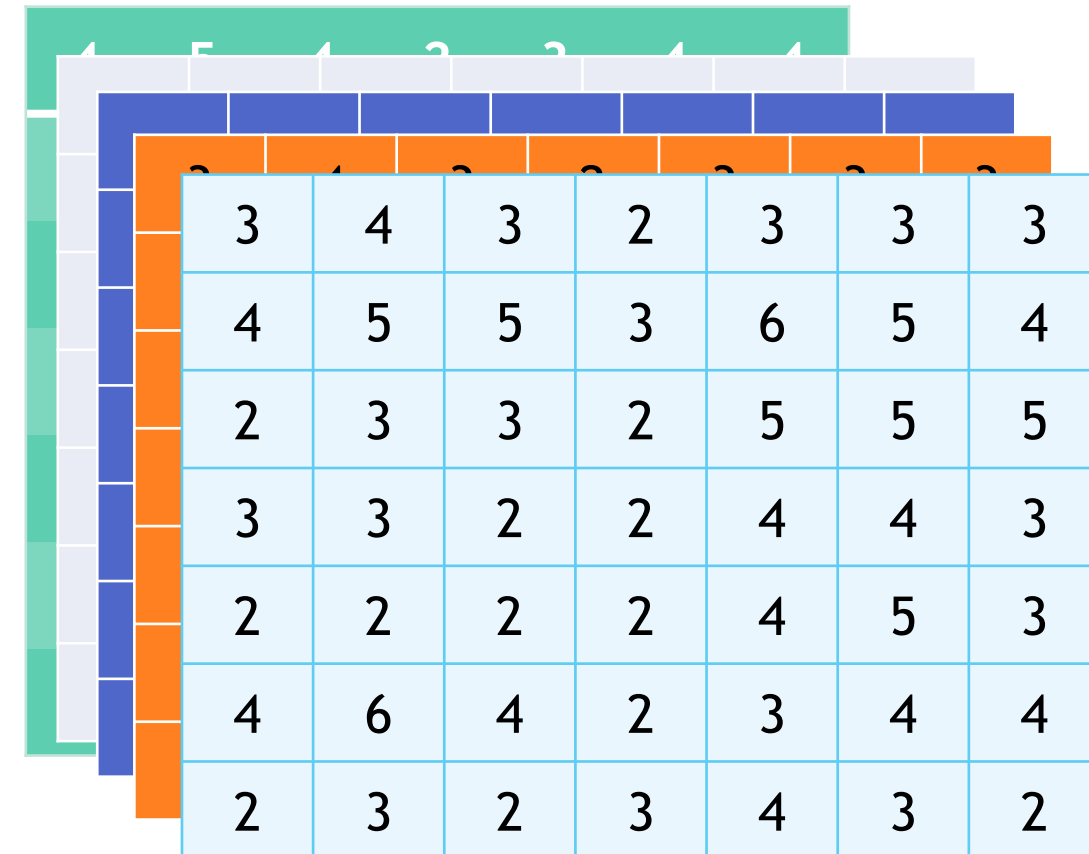
# The number of weights in conv layer

- In this example we have five filters of size 3X3. The number of weights will be  $3 \times 3 \times 5 = 45$  weights.
- We also need to add bias terms. So for the below example the number of weights will be  $45 + 5 = 50$

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



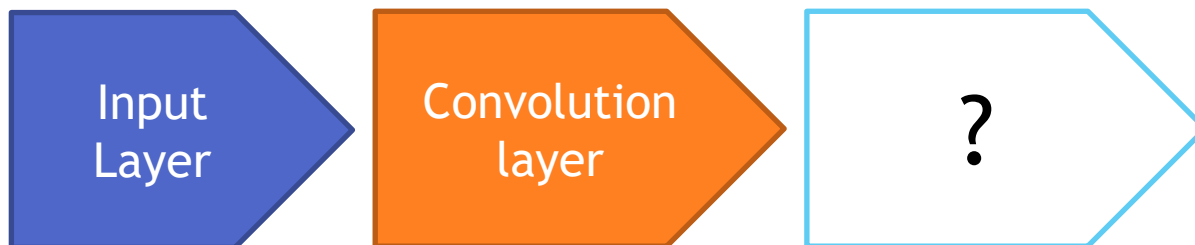
Each value from filter matrix will have one weight



3	4	3	2	3	3	3
4	5	5	3	6	5	4
2	3	3	2	5	5	5
3	3	2	2	4	4	3
2	2	2	2	4	5	3
4	6	4	2	3	4	4
2	3	2	3	4	3	2

# The Story So Far...

- ANN is a powerful algorithm. But there are too many parameters to train in mega pixel images.
- It is unwise to consider each pixel as input, the training time is overlong
- We need an alternative to ANN's fully connected network architecture
- We used a filter and converted the actual input pixels(input features) to convoluted features. Several convolution filters are used to capture as much of the data as possible.
- Sub regions are captured i.e some specific features are extracted. What are the next steps ?



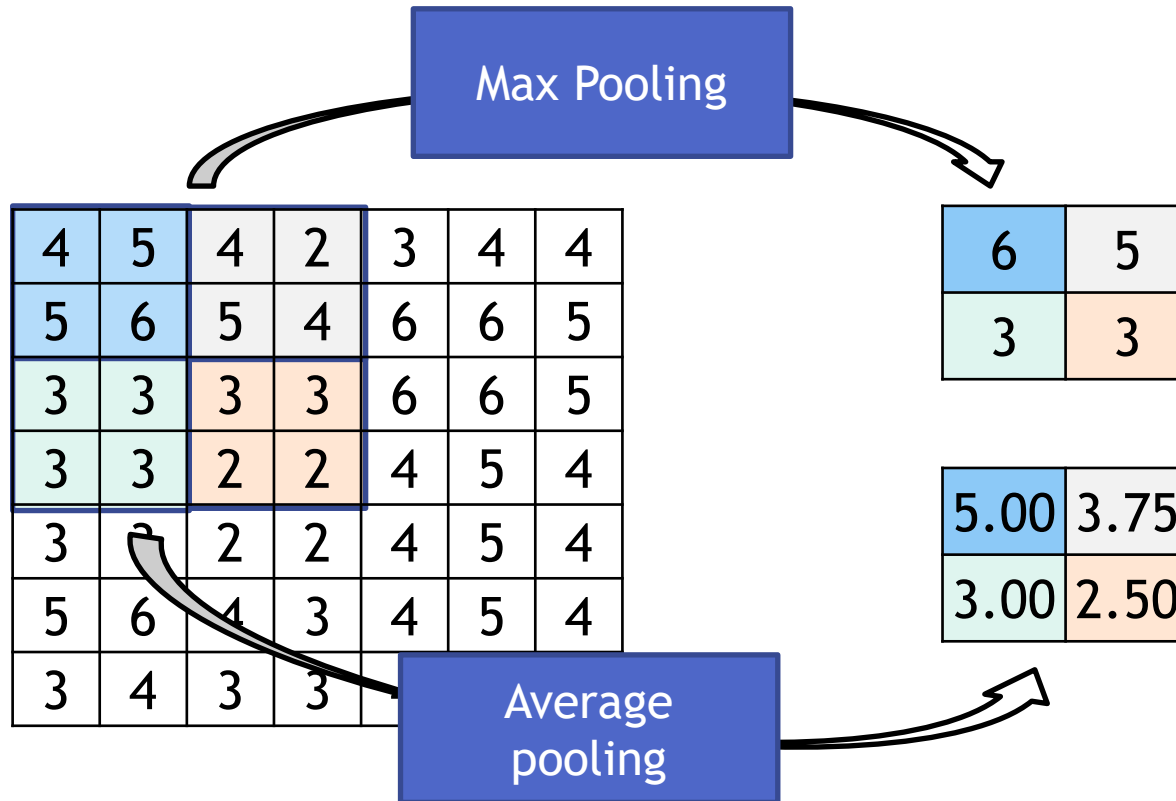
# Down sampling

- Each primitive element in convolution layer is capturing more information than a pixel. We can comfortably down sample it.
- Down sampling of the data is also known as pooling
- Down sampling reduces the number of free parameters considerably

# Pooling layer

- Pooling helps in further downsizing the data
- Pooling is used to avoid overfitting and increase the robustness
- Reduces lot of computation time
- Gives importance to feature presence not to the position in the image
- Several pooling techniques
  - Max pooling
  - Average pooling
- Pooling doesn't really ensure the spatial integrity, convolution takes care of it. We can work with sampled data now.

# How Pooling works





# What happens in pooling layer

- Down sampling of the data.
- Since we preserved lot of information in each convolution layer, we can comfortably down sample it, in this pooling layer
- Yes, we do loose some information, but we will not loose the overall integrity of the data
- It is still good enough for a classification model
- Generally we try max pooling with 2X2 filter with stride 2

# How many parameters in pooling layer

- No parameters
- We just perform down sampling, that's it. No further activation and no further parameters
- Number of parameters in pooling layer=0

# LAB: Pooling layer

- Image “cat.jpeg”
- Apply 3X3 convolution filter and then apply 3X3 max pooling matrix
- Apply 3X3 convolution filter and then apply 3X3 max pooling matrix

# Code: Pooling layer

```
def Visualize(model, cat):  
    '''prints the cat as a 2d array'''  
    cat_batch = np.expand_dims(cat,axis=0)  
    conv_cat2 = model.predict(cat_batch)  
  
    conv_cat2 = np.squeeze(conv_cat2, axis=0)  
    print (conv_cat2.shape)  
    conv_cat2 = conv_cat2.reshape(conv_cat2.shape[:2])  
  
    print (conv_cat2.shape)  
    plt.imshow(conv_cat2)
```

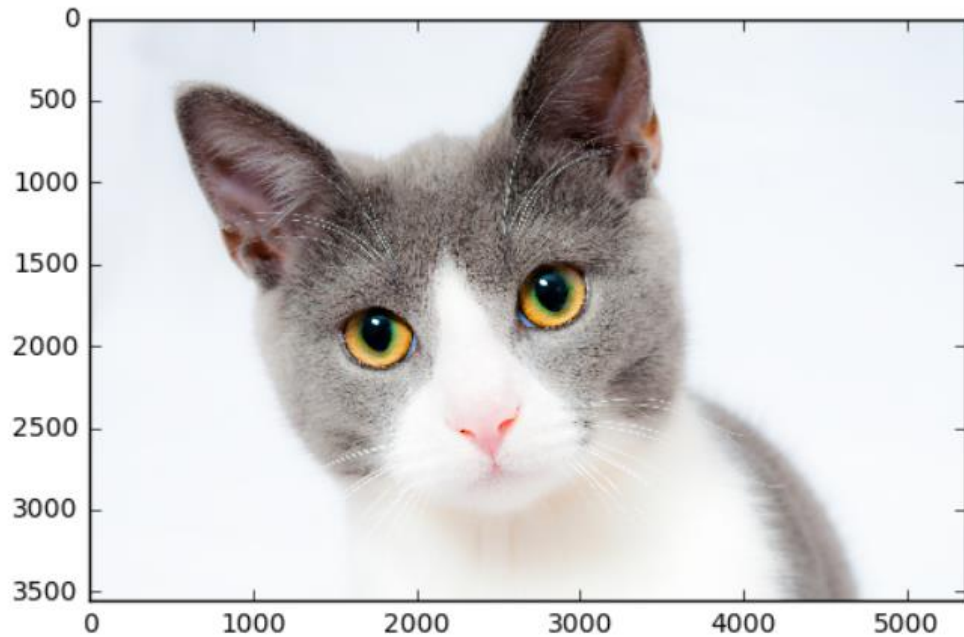
```
model=Sequential()  
model.add(Conv2D(1,  
                (3,3),  
                input_shape=x.shape,  
                ))  
model.add(MaxPooling2D(pool_size=(3,3)))  
  
#We visualize change the filter size to see the difference  
Visualize(model, x)
```

- User defined function for printing the image.
- Takes input as a vector and prints the image

- Convolution and pooling

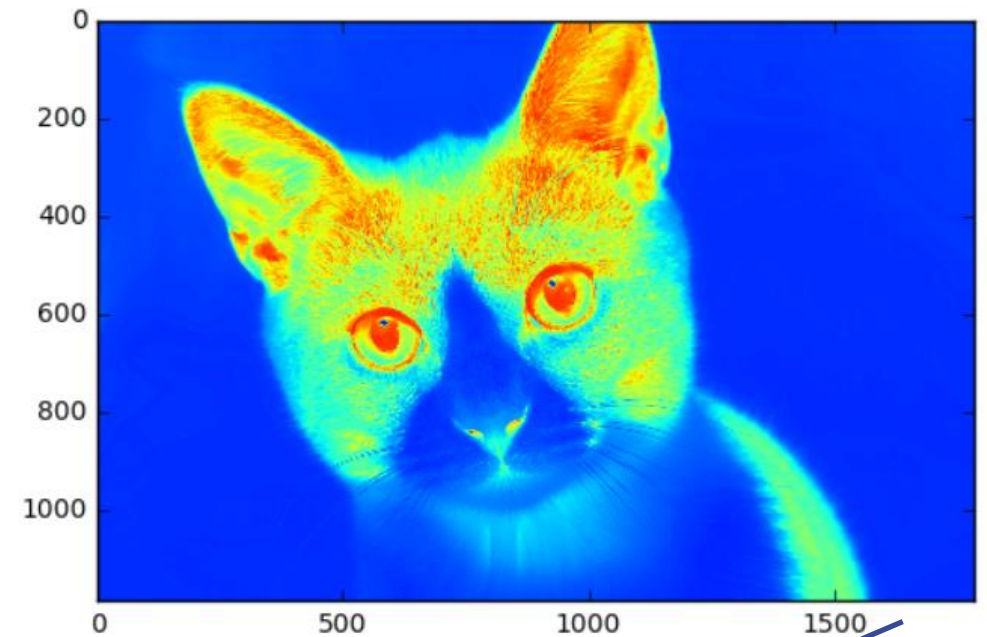
# Code: Pooling layer

Original Image



Shape before pooling

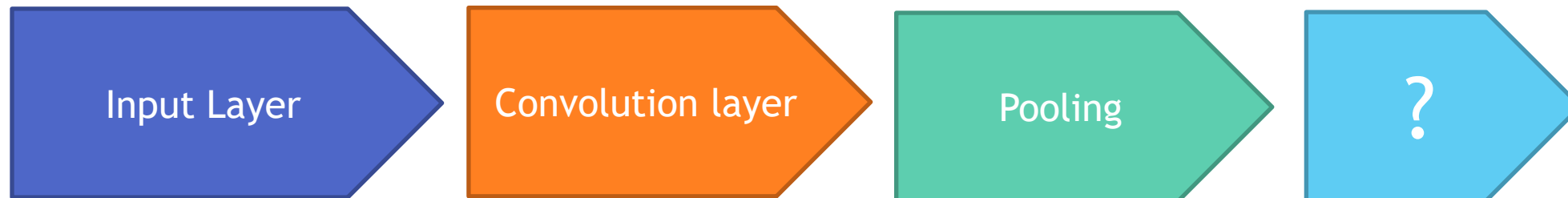
Image after pooling



Shape after pooling

# The Story So Far...

- The convolution and pooling was performed
  - To preserve the spatial dependence
  - To keep data size low
  - To reduce number of free parameters
  - To create new features without losing lot of information
- Now we are almost ready to go with usual ANN style computation
- So the preceding layer of the output layer is fully connected layer
- This is also known as the final feature layer



# ANN to CNN

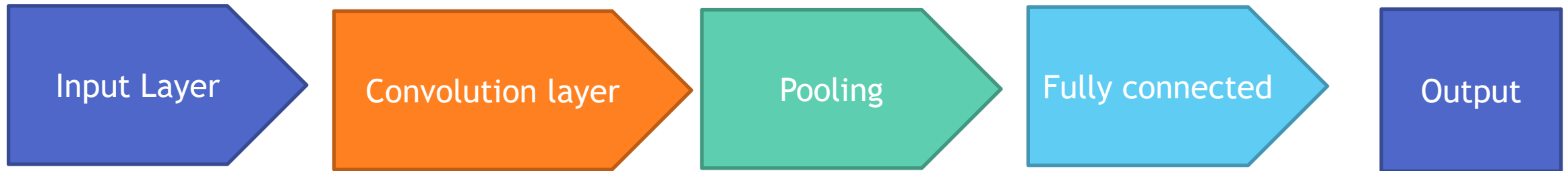
- ANN has three types of layers
  - Input layer
  - Hidden layer
  - Output layer
- We changed the network architecture of ANN
  - We created convolutional layer by adding filters
  - We down sampled the data in pooling layer
- The new network is very different. This network is known as Convolutional Neural Network (CNN)

# Convolutional Neural Networks

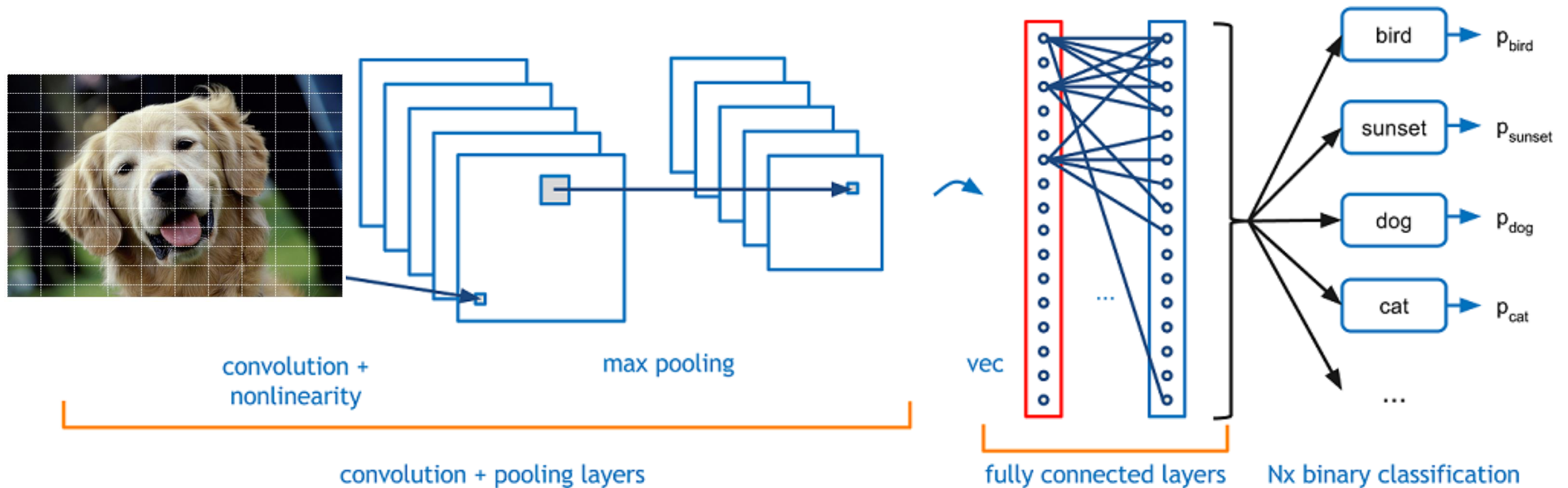
- ANN has three types of layers
  - Input layer
  - Hidden layer
  - Output layer
- CNN has five types of layers
  - Input layer
  - Convolutional layer
  - Pooling layer
  - Fully connected hidden layer
  - Output layer



# CNN Layers and Architecture



# Final Fully Connected layer



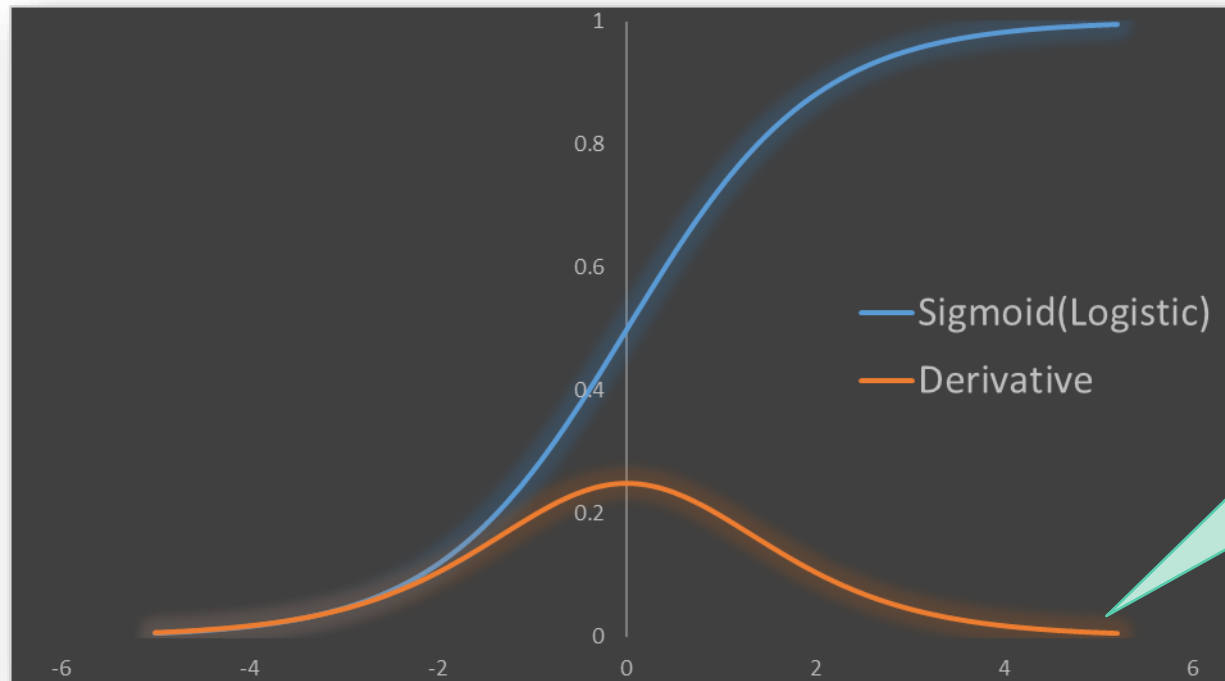
Once we have the final features ready the problem becomes a simple ANN

# Activation Functions in CNN

- CNNs are very deep by default
- They may suffer from vanishing gradients issue if we use sigmoid or tanh

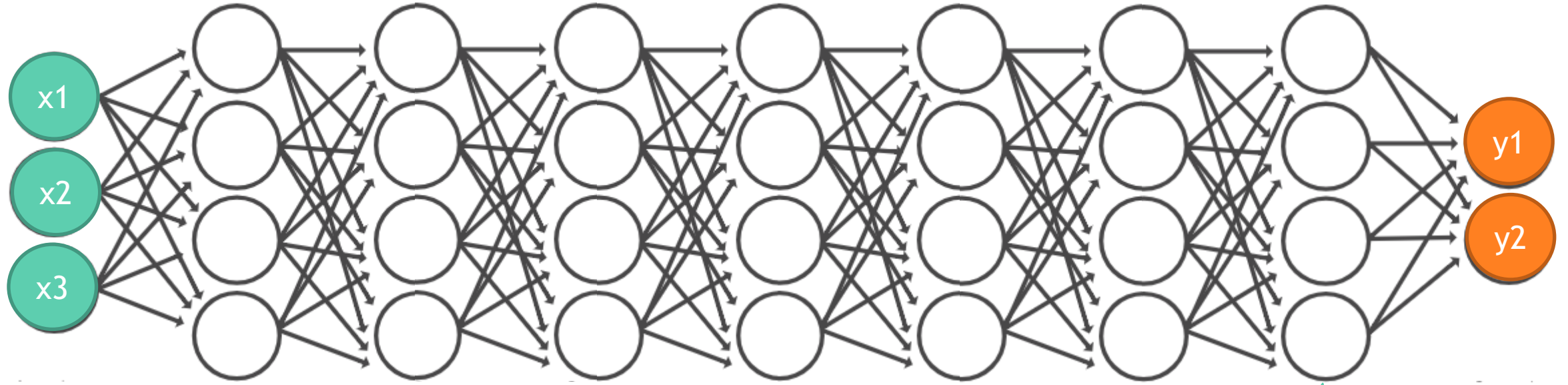
# The problem of Vanishing gradients

- Your deep neural network might be suffering from a problem of vanishing gradients
- Computationally sigmoid values and gradient values and their multiplications are very small
- For a deep network with many hidden layers sigmoid gradients vanish very quickly



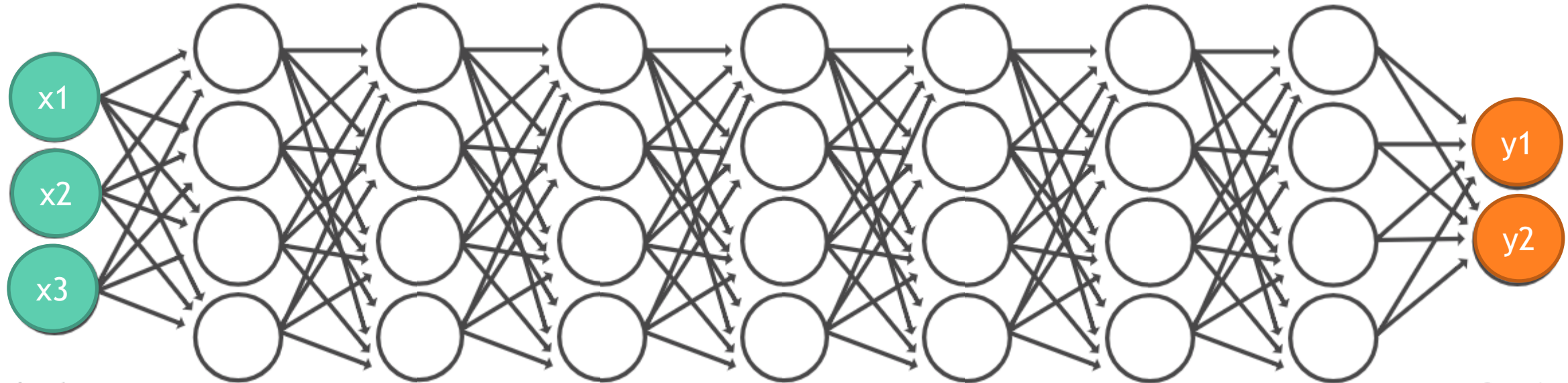
The derivative of sigmoid is zero on both ends

# The problem of vanishing gradients



The gradient values are already low here

# The problem of vanishing gradients



If the values of gradients at final layers are already low, then gradient will be almost zero for the starting layers

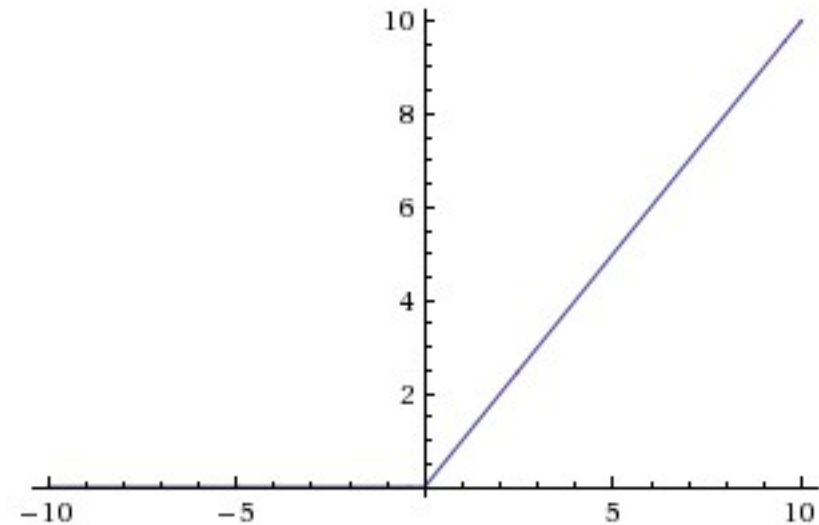
The gradient values are already low here

- With small values in the matrix and several matrix multiplications (from layer  $k$  to 1) the gradient values shrink exponentially fast. Eventually vanishing completely after a few layers.

# ReLU - The Rectified Linear Unit

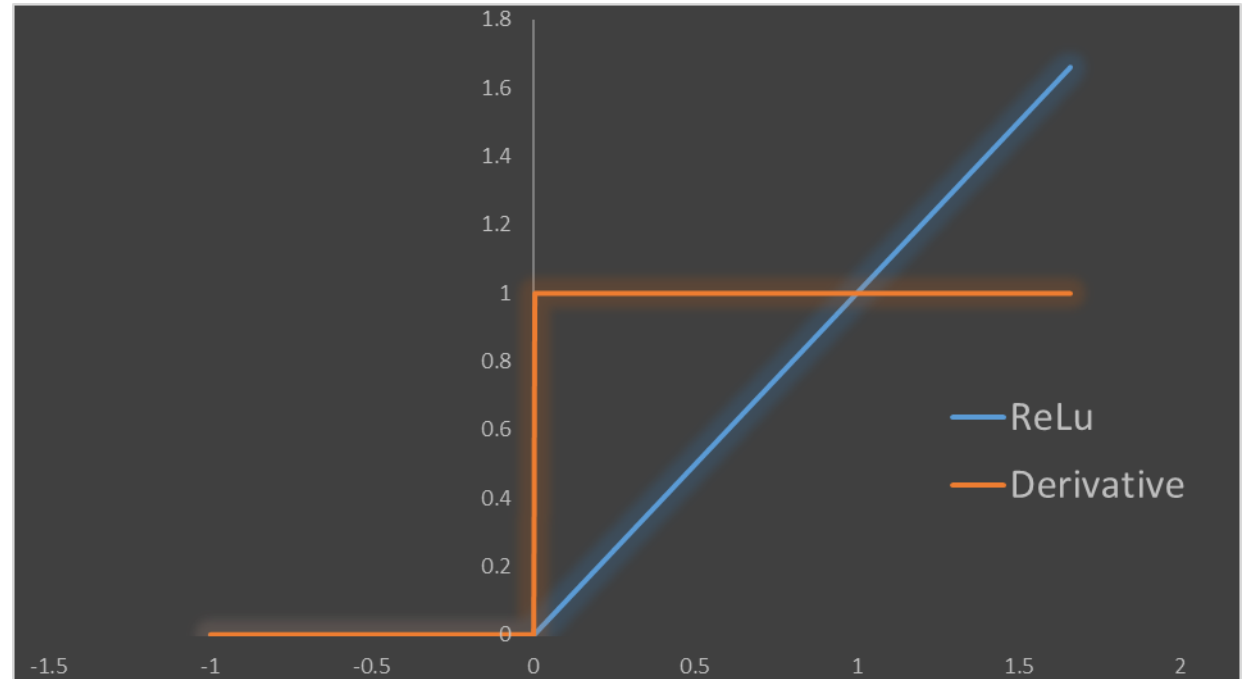
- Very popular activation function in recent times
- $f(x) = \max(0, x)$
- In other words, the activation is simply thresholder at zero
- Works very well for a certain class of problems
- Doesn't have vanishing gradient problem. It can be used for modelling real values

$$f(x) = \max(0, x)$$
$$f(x) = \log(1 + ex)$$



# ReLU – Handles Vanishing gradients

- The derivative of ReLU is 1 if  $x > 0$  else zero
- The derivatives are not small fractions
- Matrix multiplications won't result in vanishing gradients







# LAB: CNN Model building

---

# CIFAR10 Data

- The CIFAR-10 dataset (Canadian Institute For Advanced Research) is a collection of images
- CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes.
- The 10 different classes represent - **airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.**
- There are 6,000 images of each class

# LAB: CNN on CIFAR10 Data

- Download CIFAR10 data
- Build a CNN model
- What is the accuracy of the model?

# Code: CNN on CIFAR10 Data

```
model = models.Sequential()

model.add(layers.Conv2D(16, (3, 3), activation='relu', input_shape=(32, 32, 3)))
#For Detecting low level features
model.add(layers.MaxPooling2D((2, 2)))

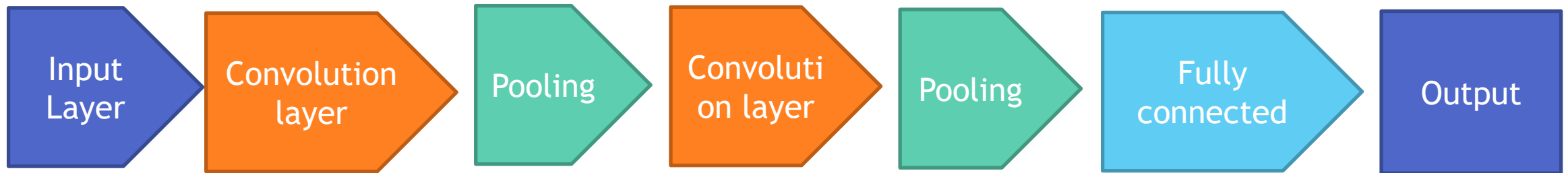
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
#For Detecting Mid level features - Higher than previous conv layer
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
#High level features - Higher than previous conv layer
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
model.summary()
```

# Layers - Workflow

- We might repeat the convolution and pooling steps multiple times in real-life



# ImageNet project

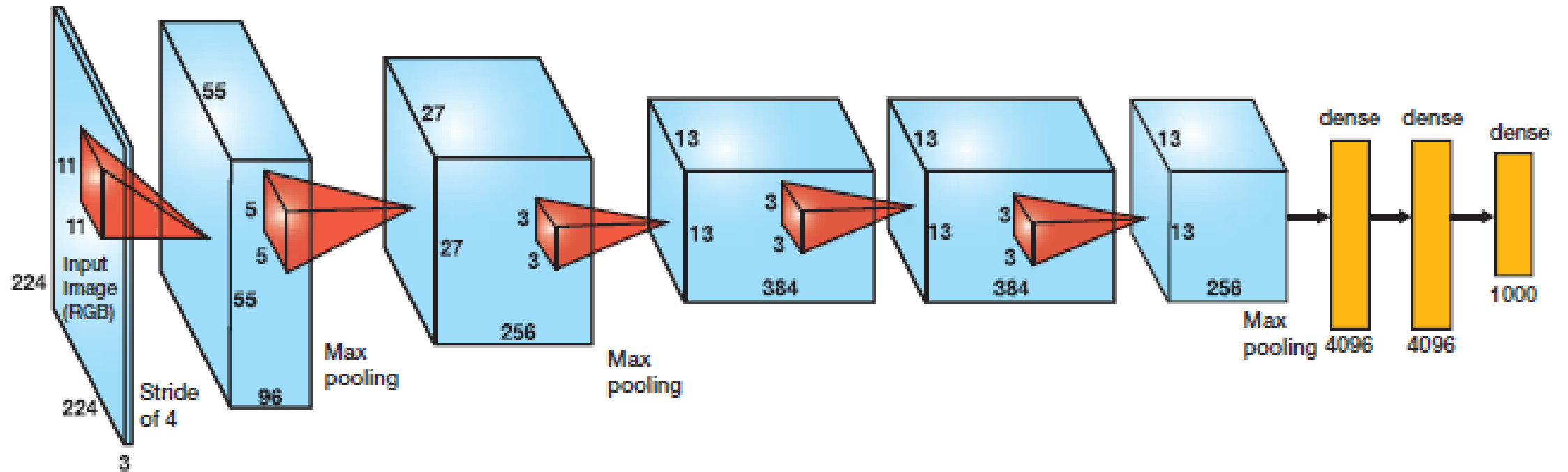
- The ImageNet is a large visual database designed for use in visual object recognition software research.
- More than 14 million images have been hand-annotated
- ImageNet contains more than 20,000 categories
- typical category, such as "balloon" or "strawberry", consisting of several hundred images.
- Since 2010, the ImageNet project runs an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where software programs compete to correctly classify and detect objects and scenes.

# AlexNet

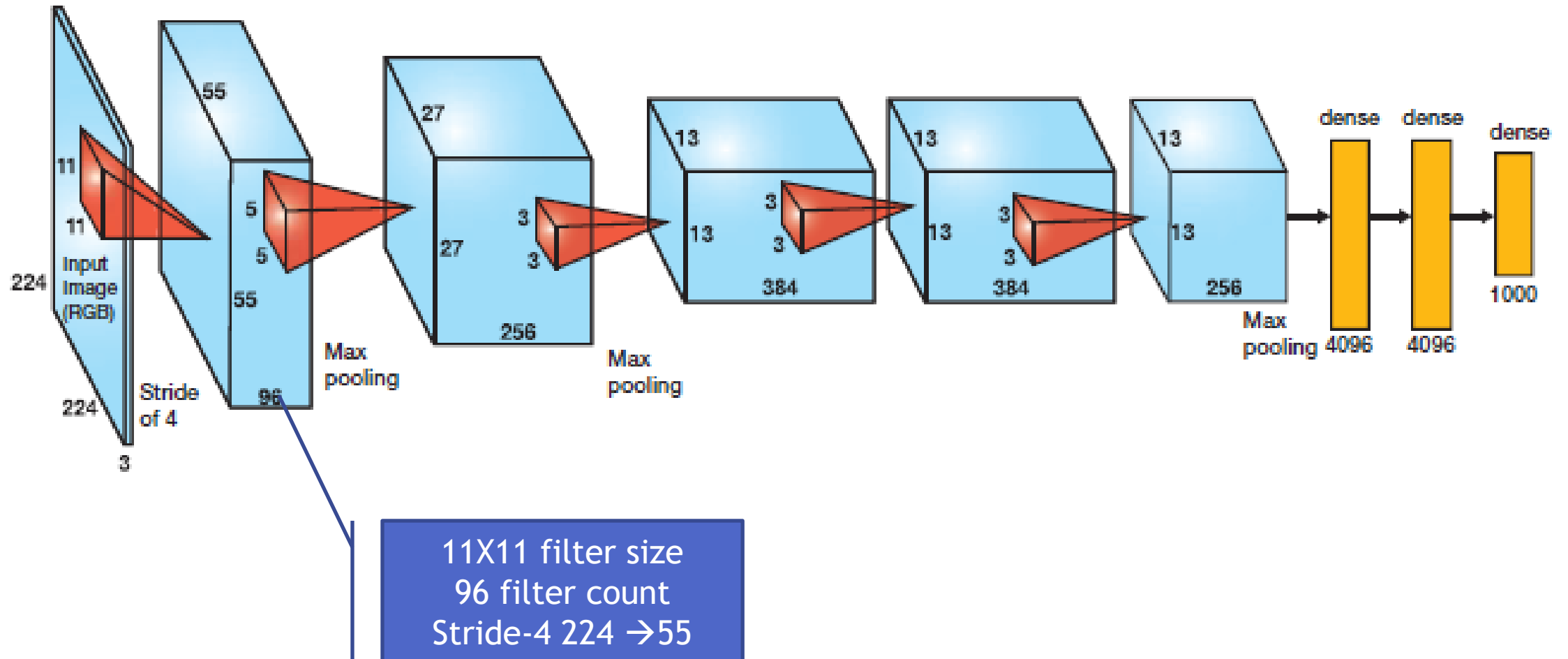
- AlexNet is the name of a CNN, which competed in the ImageNet challenge in 2012
- The network achieved a top-5 error of 15.3%, more than 10.8 percentage points ahead of the runner up
- It was designed by the SuperVision group, consisting of Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever
- Alexnet contained 8 layers, first 5 were convolutional layers followed by fully connected layers



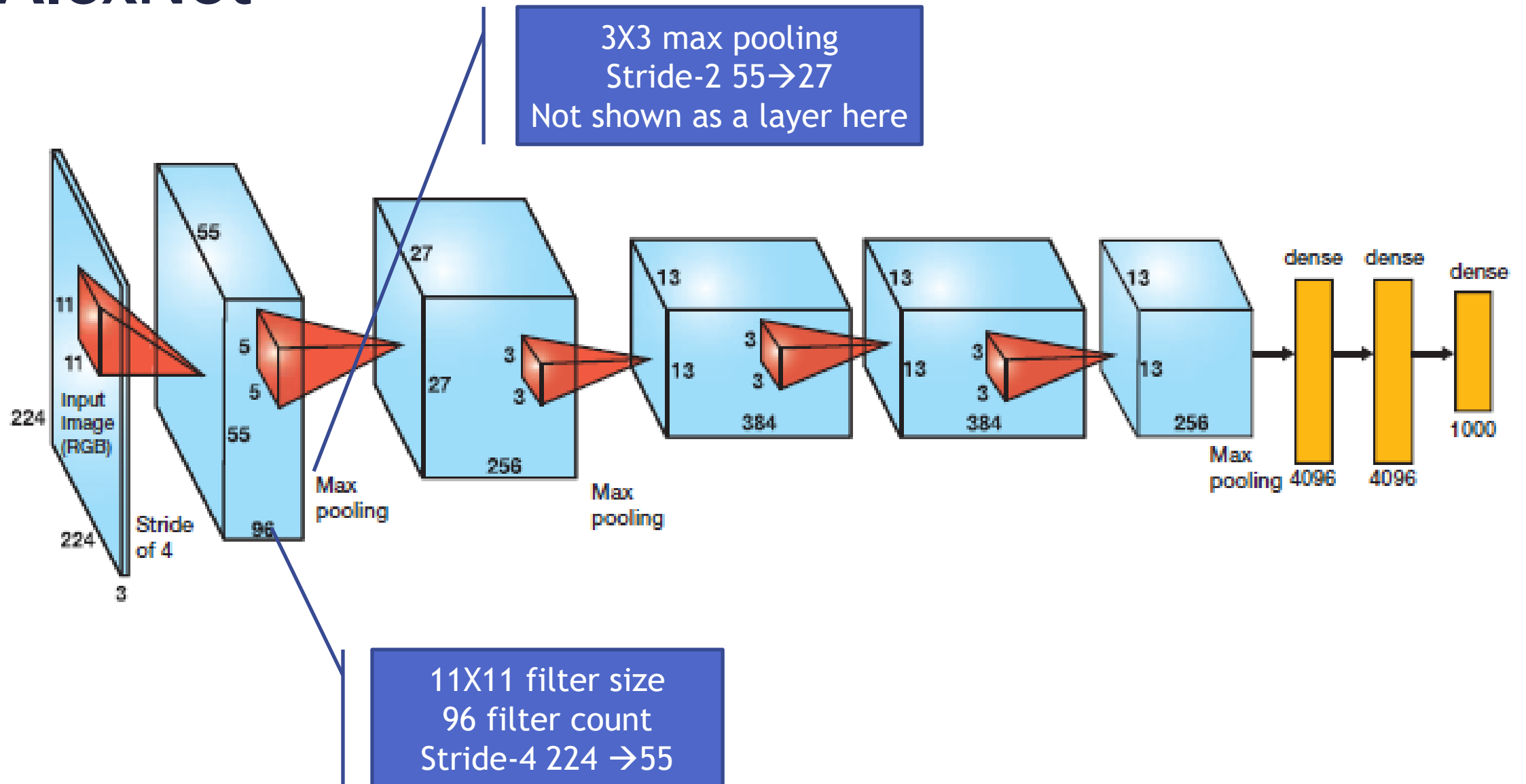
# AlexNet



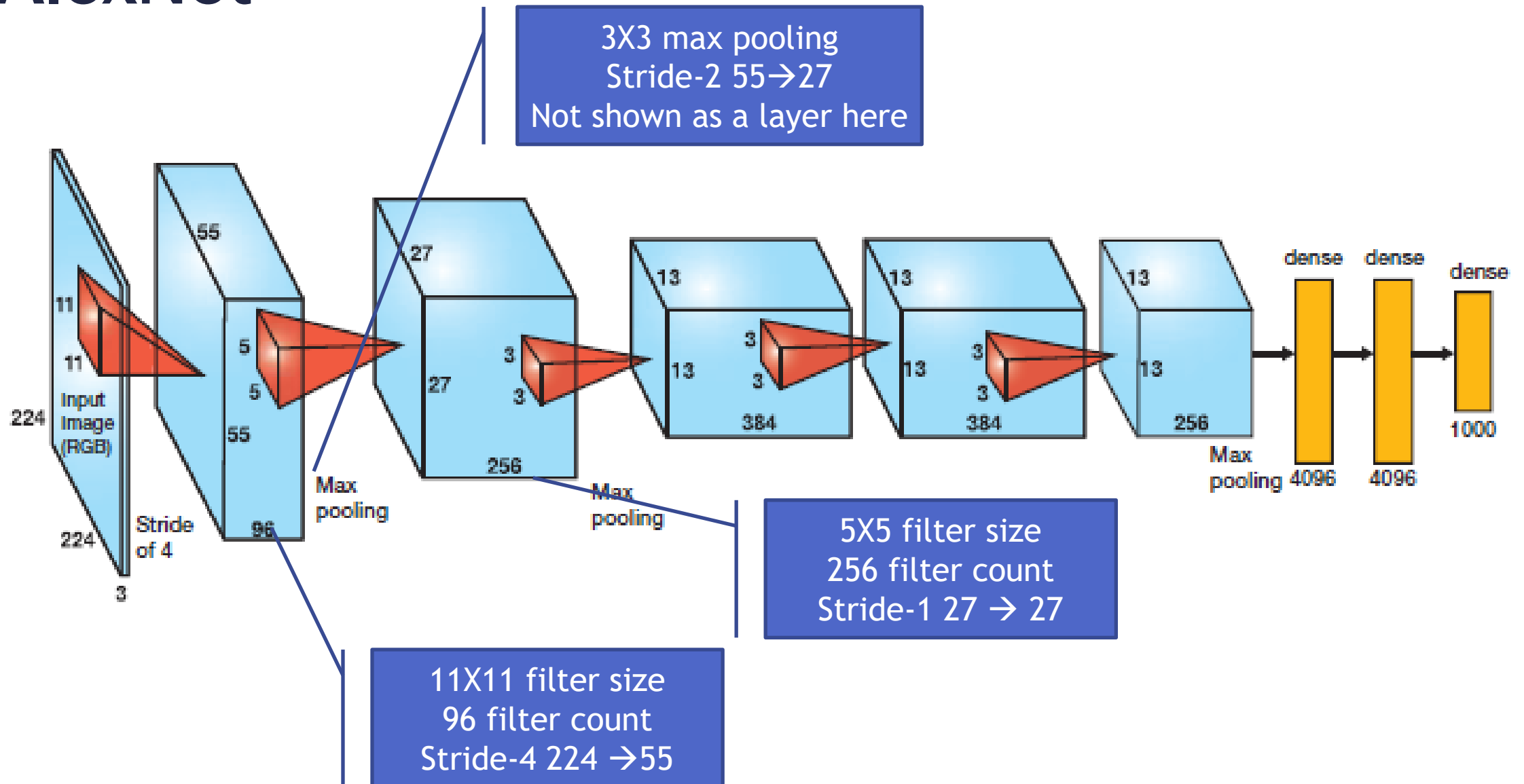
# AlexNet



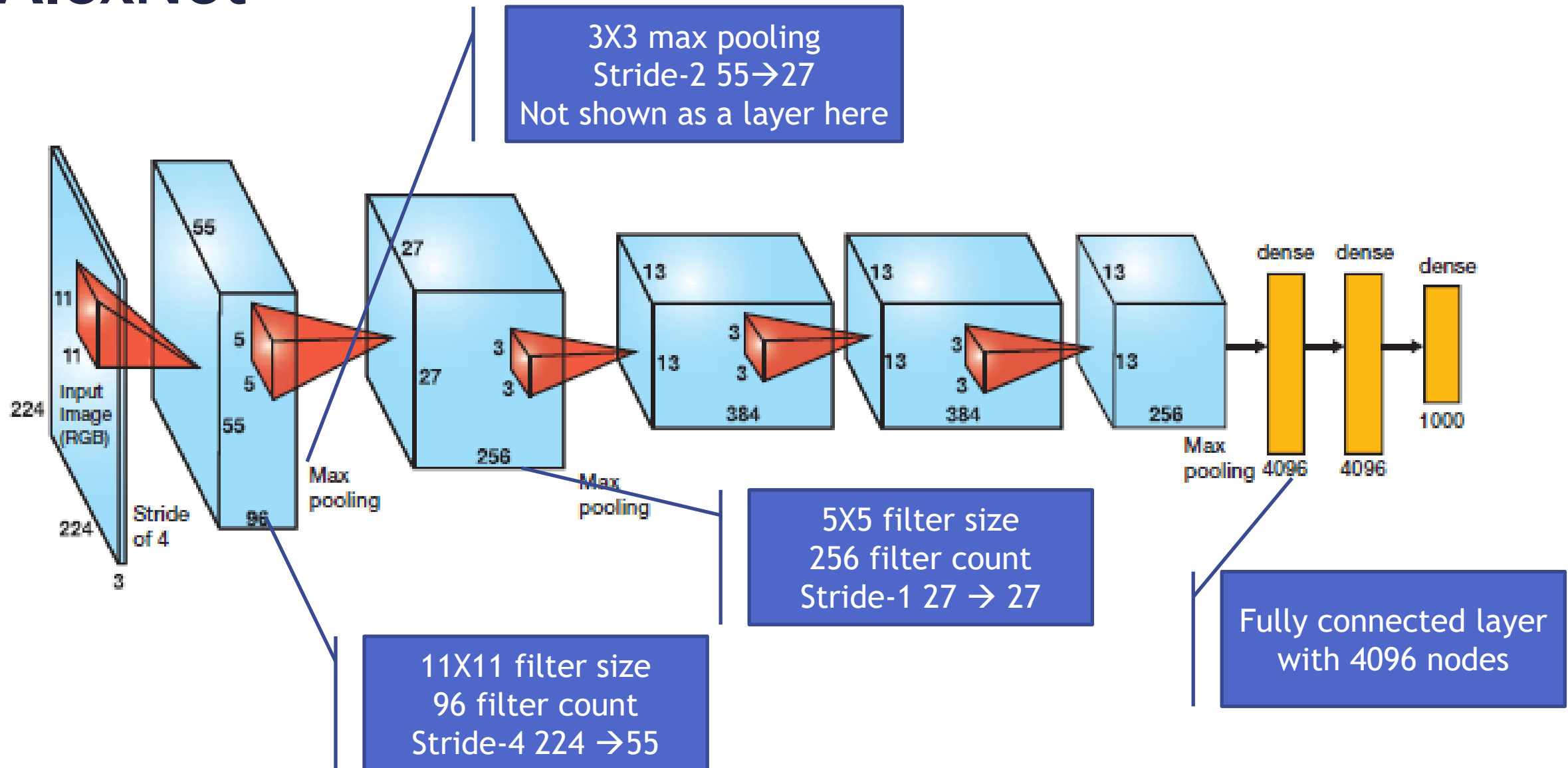
# AlexNet



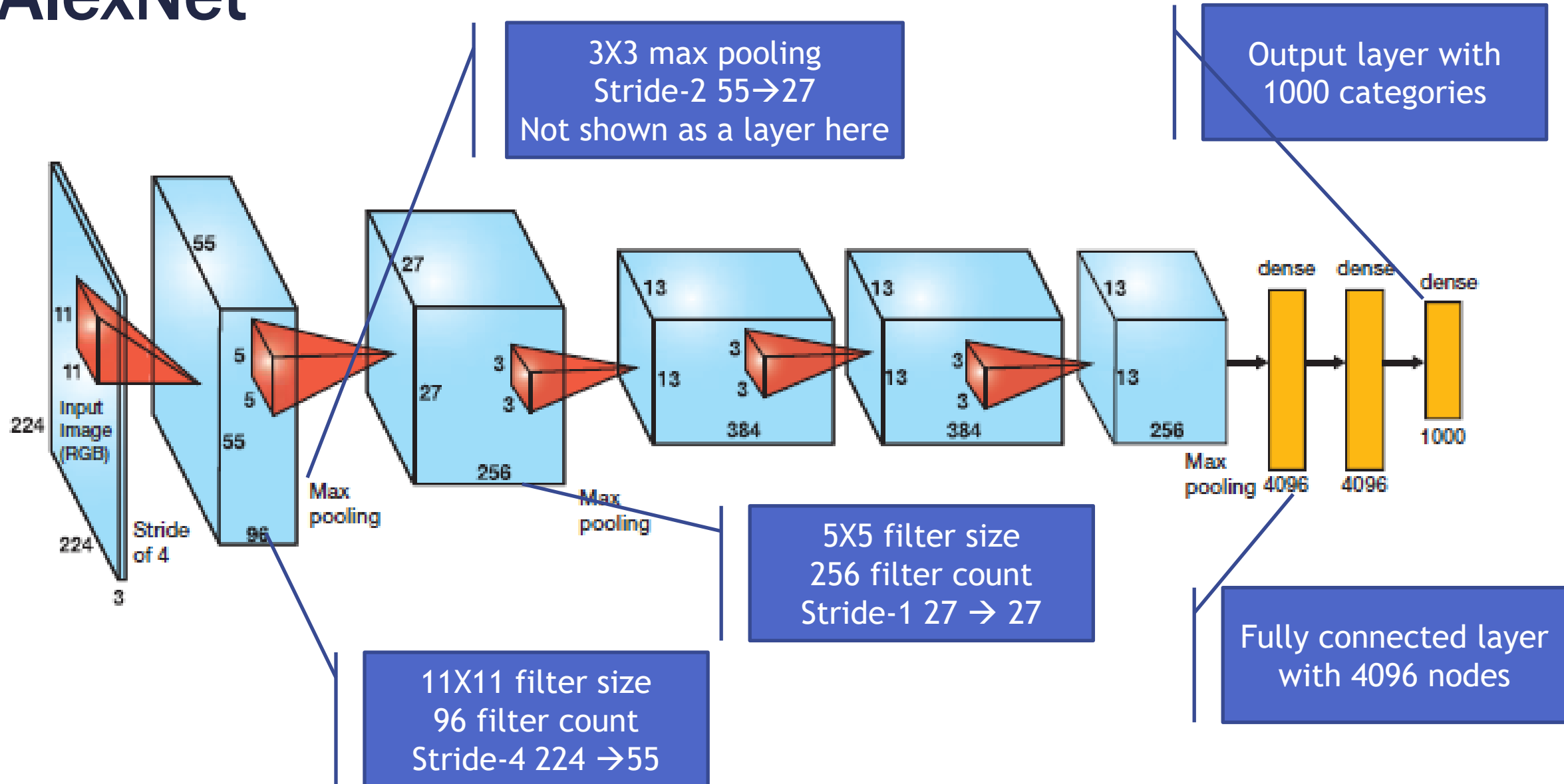
# AlexNet



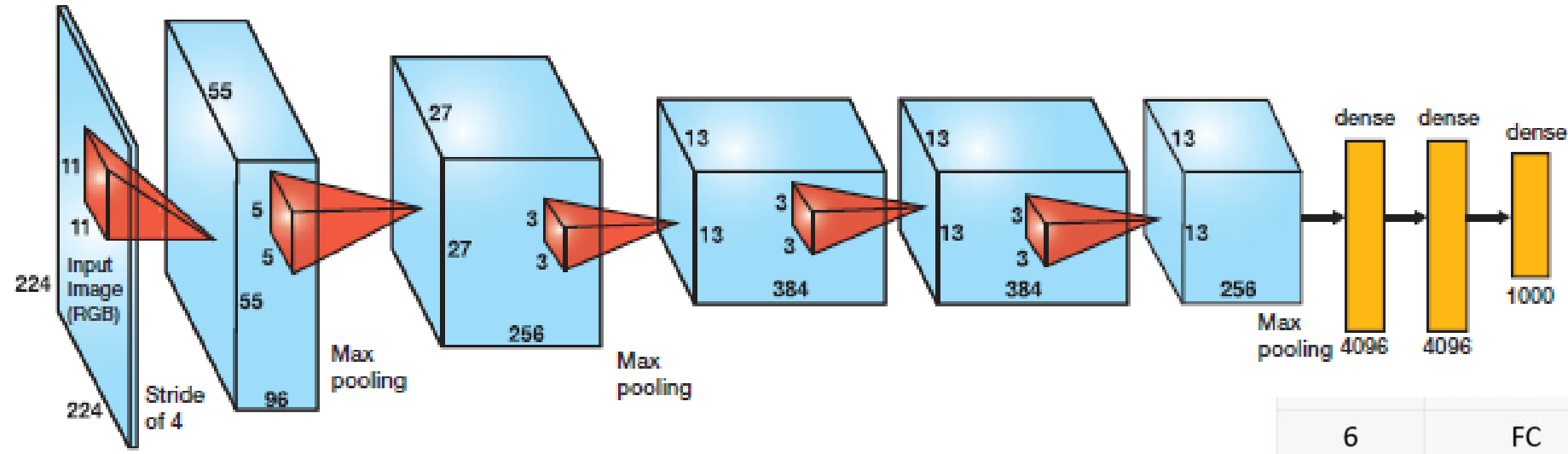
# AlexNet



# AlexNet



# AlexNet



Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	224 x 224 x 3	-	-	-
1	Convolution	96	55 x 55 x 96	11x11	4	relu
	Max Pooling	96	27 x 27 x 96	3x3	2	relu
2	Convolution	256	27 x 27 x 256	5x5	1	relu
	Max Pooling	256	13 x 13 x 256	3x3	2	relu
3	Convolution	384	13 x 13 x 384	3x3	1	relu
4	Convolution	384	13 x 13 x 384	3x3	1	relu
5	Convolution	256	13 x 13 x 256	3x3	1	relu
	Max Pooling	256	6 x 6 x 256	3x3	2	relu

6	FC	-	9216
7	FC	-	4096
8	FC	-	4096
Output	FC	-	1000

# Pre-trained model

- Using a standard model on our data.
- A standard model has been already built on multiple classes



# Pre-trained Keras Models

- The [Keras](#) currently supports five models that have been pre-trained on ImageNet:
- [VGG16](#) A convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”.
- VGG19 - It is essentially the VGG16 model with three additional weight layers.
- [Inception V3](#) - A convolutional neural network model from Google that scales up networks in ways that aim at utilizing the added computation as efficiently as possible by suitably factorized convolutions and aggressive regularization.
- [ResNet50](#) - A residual learning framework from Microsoft Research that eases the training of deep networks. It reformulates the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions.
- [Xception](#) - Xception is an extension of the Inception architecture which replaces the standard Inception modules with depthwise separable convolutions.

# Pre-trained model-Demo

```
from keras.applications.resnet50 import ResNet50
from keras.preprocessing import image
from keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from PIL import Image
from PIL import ImageFont
from PIL import ImageDraw
```

Using TensorFlow backend.

```
model = ResNet50(weights='imagenet')
#This code will download the pretrained weights
#You can manually copy the file to cache folders
#The temp folder location C:\Users\StatInfer\.keras
```

# ResNet50

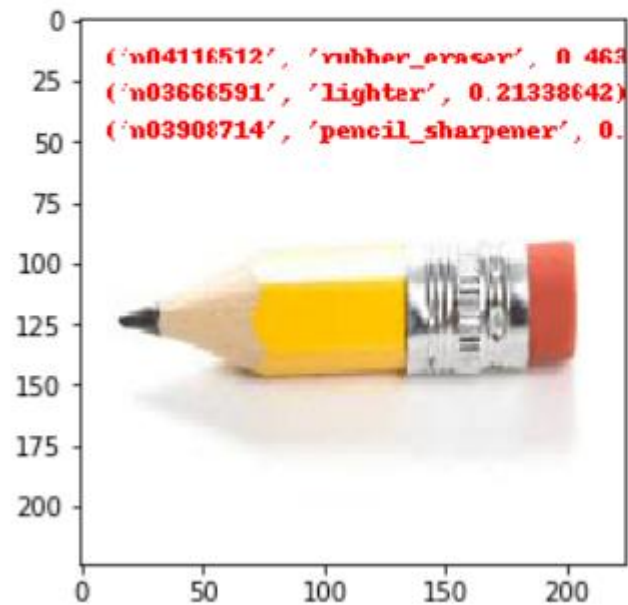
```
### Load Image
img_path = 'D:\\Dropbox\\39. Deep Learning\\First Version References\\8. Deep Learning Models\\Data\\15.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

### Prediction
preds = model.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
prediction = '\n'.join(map(str, decode_predictions(preds, top=3)[0]))
print(prediction)
draw = ImageDraw.Draw(img)
draw.text((10, 10), prediction, (250, 0, 0))
img.save('sample-out.jpg')
plt.imshow(img)
```

# ResNet50

```
( 'n04116512', 'rubber_eraser', 0.46310714)
( 'n03666591', 'lighter', 0.21338642)
( 'n03908714', 'pencil_sharpener', 0.09975958)
```

```
<matplotlib.image.AxesImage at 0x221d70b25c0>
```





# CNN Project

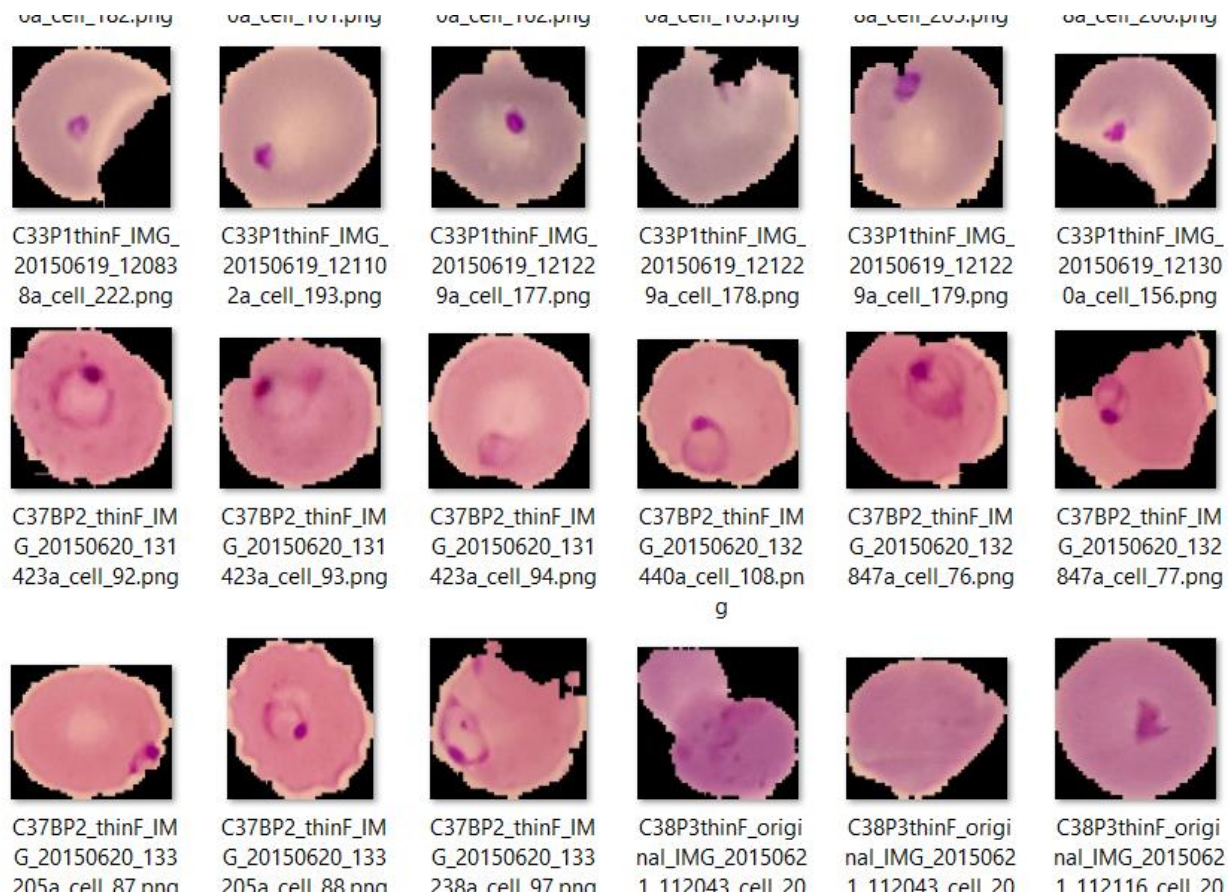
---

# Detect Malaria based on Cell images

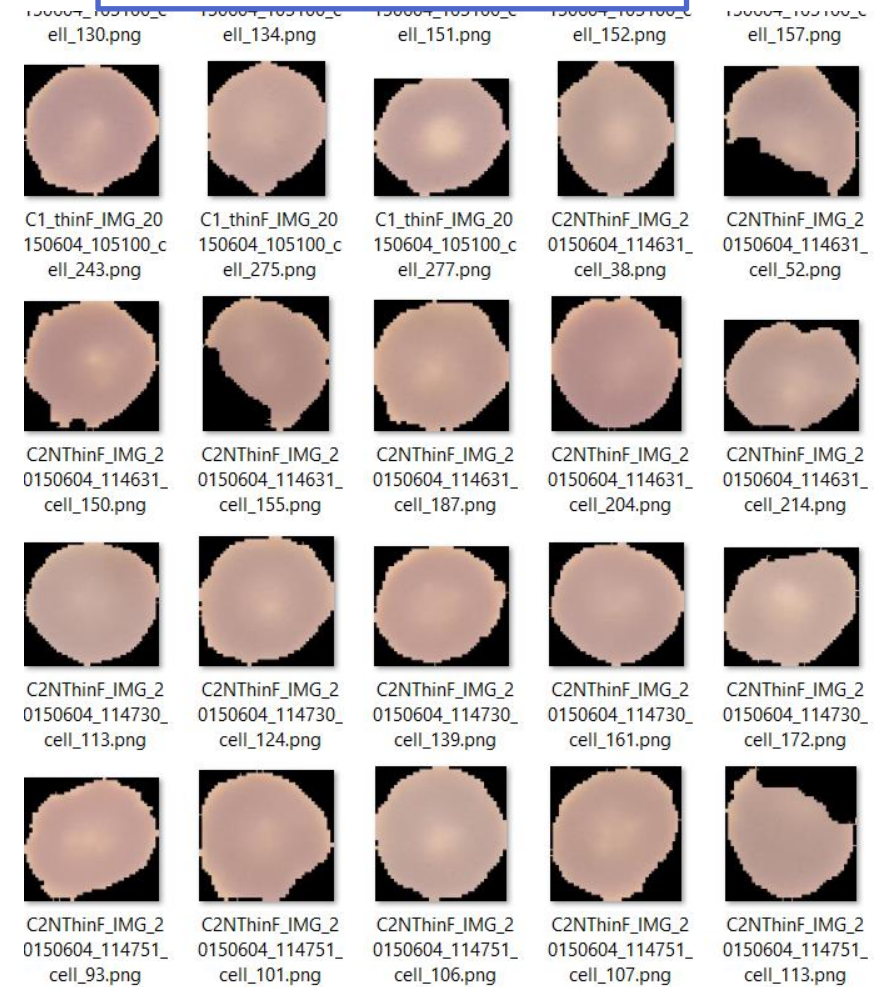
- The dataset contains 2 folders; A total of 27,558 images.
  - Infected Cell Images
  - Uninfected Cell Images
- This Dataset is taken from the official NIH  
Website: <https://ceb.nlm.nih.gov/repositories/malaria-datasets/>
- Build a image recognition model that detects malaria based on cell images

# Detect Malaria based on Cell images

Infected Cell images



Un-infected Cell images



# Steps

1. Get the data
2. Print the sample images
3. Create train and test data
4. Build the model ( use GPU )
5. Validate the model
6. Make the predictions on new cell images



# Conclusion

- This session we discussed basics details of CNN
- CNN works well when there is a special importance between the data values
- You may have to carefully set the hyper parameters like filter size, the number of filters, strides, zero Padding etc.,
- There is a lot of research going on right now on CNN. Read the tech news and CNN related papers to stay updated.
- There is some spatial dependency in text data. CNN algorithms work well on text classification as well.



# Thank you

---



# Appendix

---

# What happens to edges in Convolution layer?

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

1	1	1
1	1	1
1	1	1

	6	5	4	6	6	
	3	3	3	6	6	
	3	2	2	4	5	
	3	2	2	4	5	
	6	4	3	4	5	

- A 7X7 image becomes 5X5 Image in the next layer. With 3X3 filter
- If the filter size is 5X5 then the resultant image will be just 3X3
- We are losing the information on the edges.

# What happens to edges in Convolution layer?

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

1	1	1
1	1	1
1	1	1

?						
	6	5	4	6	6	
	3	3	3	6	6	
	3	2	2	4	5	
	3	2	2	4	5	
	6	4	3	4	5	

- How to preserve the shape of input matrix?

# How to deal with pixels on the edge?

?							
	1	1	1	0	0	1	1
	1	1	0	1	0	1	1
	1	0	0	1	1	1	0
	0	0	0	0	0	1	1
	1	1	0	0	0	0	1
	1	0	0	1	1	1	0
	1	1	1	0	0	1	1

1	1	1
1	1	1
1	1	1

?						
	6	5	4	6	6	
	3	3	3	6	6	
	3	2	2	4	5	
	3	2	2	4	5	
	6	4	3	4	5	
						?

- How to deal with edge pixels

- How to derive values on the edges?

# Zero padding for pixels on the edge

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

1	1	1
1	1	1
1	1	1

	6	5	4	6	6	
	3	3	3	6	6	
	3	2	2	4	5	
	3	2	2	4	5	
	6	4	3	4	5	

- Add zero padding

# Zero padding for pixels on the edge

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

1	1	1
1	1	1
1	1	1

4	5	4	2	3	4	4
5	6	5	4	6	6	5
3	3	3	3	6	6	5
3	3	2	2	4	5	4
3	3	2	2	4	5	4
5	6	4	3	4	5	4
3	4	3	3	4	4	3

- Continue the calculation as usual with zero padding



# Zero padding

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

- Zero padding is the solution for the edges
- A single layer zero padding is sufficient for 3X3 filter
- We need double layer zero padding for 5X5 filter

# Data Augmentation

- Data Augmentation - Creating more data points from the source data.
- Insufficient data leads to over-fitting, reduces the generalization performance of the model while testing.
- Also, obtaining valid data is a time consuming and laborious process.
- When we don't have enough data to train our model, Data augmentation comes to our rescue.
- With data augmentation we can generate new labelled data based on existing data with some transformations, keeping the labels preserved.

# Data Augmentation

- Random Crop
- Rotation
- Blurring
- Image Mirroring(flips)
- Color Shifting / Whitening
- Distortions
- Cutout

# How Augmented Data May Look Like



Augmentations





# How Augmented Data May Look Like

