Understanding of AlexNet Architecture

PREPARED BY: SUBASH SAH

Introduction

Introduced by Alex Krizhevsky et al. in 2012.

Won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC 2012).

Marked a major breakthrough in deep learning and computer vision.

Key Achievements

Reduced classification error rate from nearly 26% to 15%.

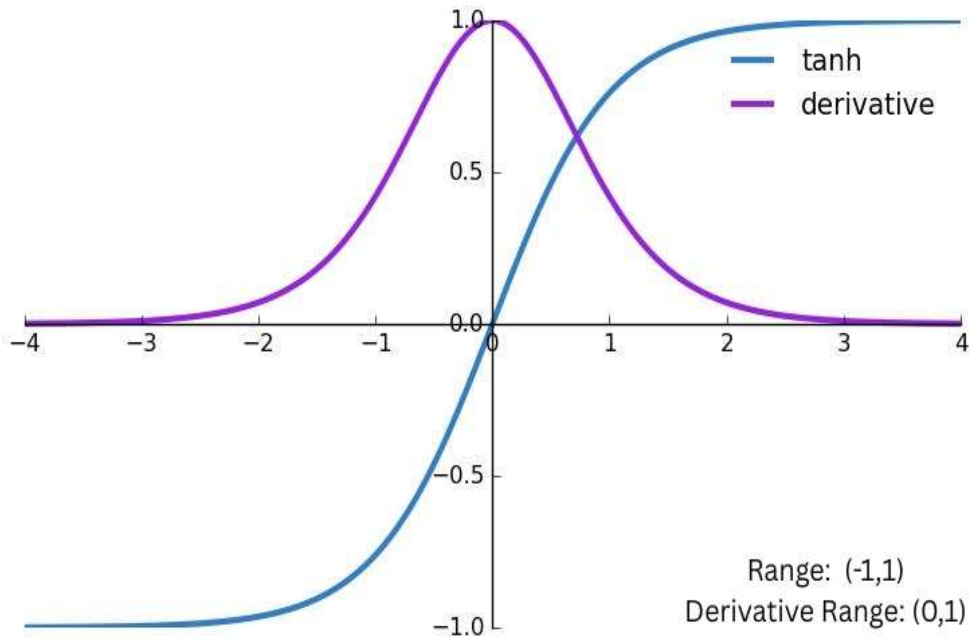
Used **ReLU** activation instead of sigmoid/tanh.

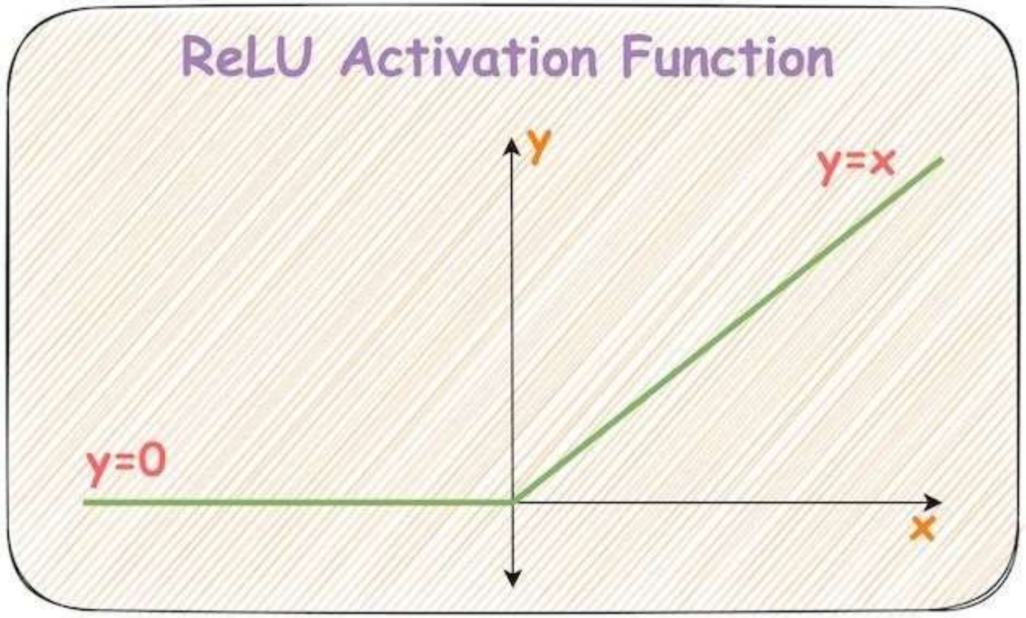
First CNN trained on GPUs – huge performance boost.

Introduced **Dropout** and **Data Augmentation** to prevent overfitting.

Dropout: A regularization technique that randomly deactivates a fraction of neurons during training to prevent overfitting and improve model generalization.

Data Augmentation: A method of artificially increasing the training dataset by applying transformations (like rotation, flipping, cropping or scaling) to existing images to improve model robustness.


1.0
0.5
0.0
−0.5
−1.0
−4
−3
−2
−1
0
1
2
3
4
tanh
derivative
Range: (-1,1)
Derivative Range: (0,1)


ReLU Activation Function
y
y=x
y=0
x

Disadvantages of Tanh:

Saturating function -> Creates Vanishing Gradient Problem

Computationally expensive -> Involves exponential calculation

Advantages of ReLU over Tanh:

Non-saturating in +ve region.

Computationally inexpensive.

Converges faster than sigmoid and tanh.

Some terms before moving to architecture:

Convolution Operation

Padding

Strides

Pooling layer

Convolution Operation

Image

Filter (Horizontal)

feature map

$$
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
255 & 255 & 255 & 255 & 255 & 255 \\
255 & 255 & 255 & 255 & 255 & 255 \\
255 & 255 & 255 & 255 & 255 & 255
\end{pmatrix}
*
\begin{pmatrix}
-1 & -1 & -1 \\
0 & 0 & 0 \\
1 & 1 & 1
\end{pmatrix}
=
\begin{pmatrix}
0 & 0 & 0 & 0 \\
255 & 255 & 255 & 255 \\
255 & 255 & 255 & 255 \\
0 & 0 & 0 & 0
\end{pmatrix}
$$

6*6 | 3*3 | **4*4**

n*n | m*m | (n-m+1)*(n-m+1)

For RGB Image:

$$\underset{n*n*c}{6*6*3} \;*\; \underset{m*m*c}{3*3*3} \;=\; \underset{(n-m+1)(n-m+1)}{4*4}$$

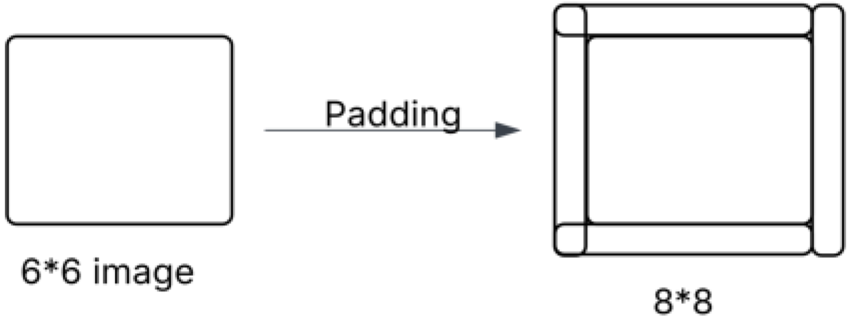
For multiple filters:

$$6*6*3 \;*\; \left.\begin{array}{l} 3*3*3 \text{ (horizontal edge)} = 4*4 \\ 3*3*3 \text{ (vertical edge)} = 4*4 \end{array}\right\} 4*4*2$$

If 10 filters then 4*4*10 feature map and so on.

Padding

After applying filter, the size of the image is reduced. So, padding is done so that the size doesn't get reduced. In this, before applying filter, we add padding (normally zero padding) to the image and then apply filter so that the size doesn't get reduced.



$*$ 3*3 filter $= (8-3+1)*(8-3+1)$
$= 6*6$

Dimension of feature map after padding is:

$n*n \; * \; m*m \; = \; (n+2p-m+1)*(n+2p-m+1)$

Keras provide two types of padding:

1. Valid: image size is reduced.
2. same: image size remains same

Stride

When not mentioned, the value of stride is 1 by default.

If strides = (2,2), then the sliding window slides 2 units horizontal as well as 2 units vertical.

After applying stride (along with padding), the dimension of feature map will be:

$$n*n \; * \; m*m \; = \; \left(\frac{n+2p-m}{s}+1\right)*\left(\frac{n+2p-m}{s}+1\right)$$

Pooling layer

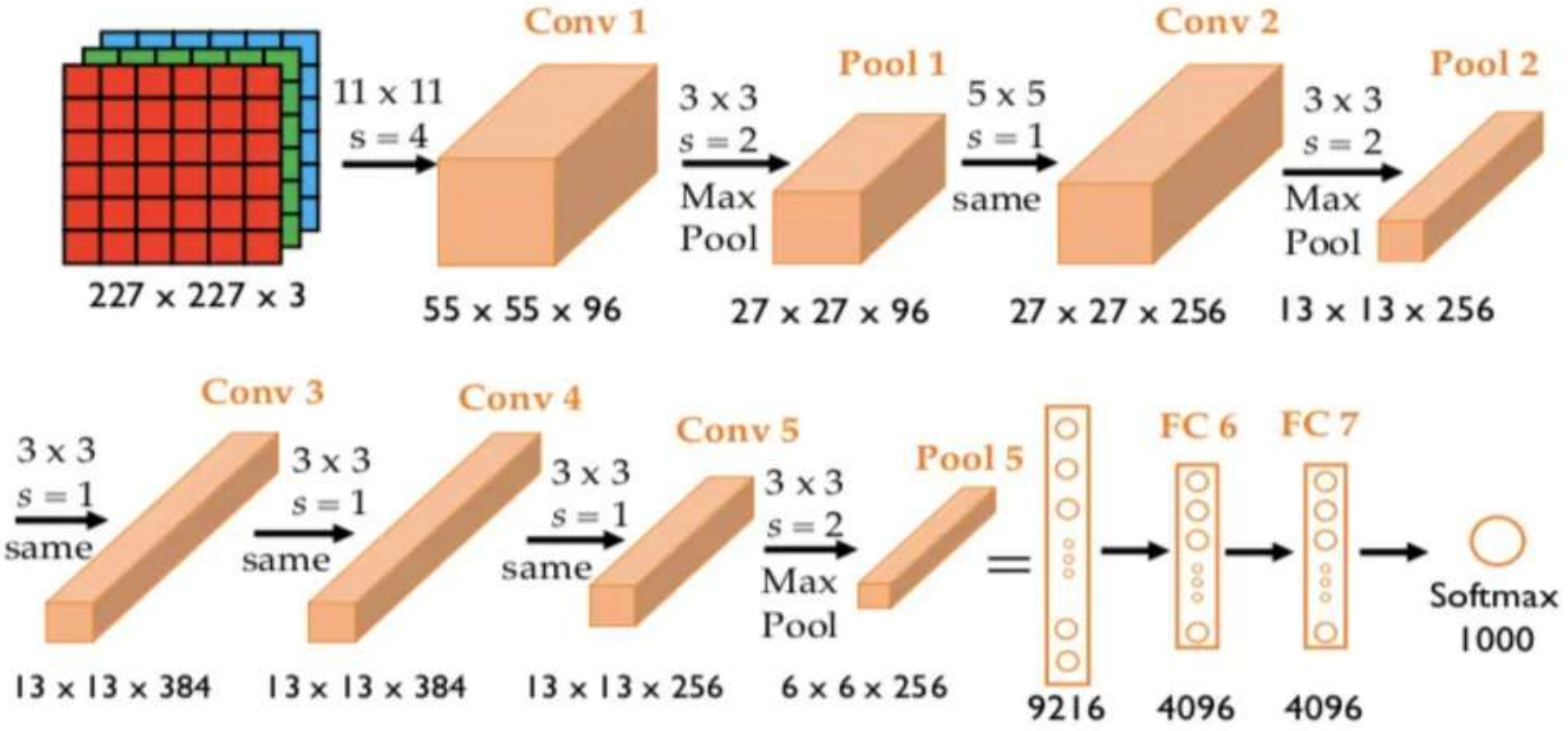
Pooling is a way to down sample the feature map.

Types of pooling:

- Max pooling
- Min pooling
- Avg pooling
- Global pooling

$$\begin{pmatrix} 3 & 1 & 1 & 3 \\ 2 & 5 & 0 & 2 \\ 1 & 4 & 2 & 1 \\ 4 & 7 & 2 & 4 \end{pmatrix} \xrightarrow[\substack{\text{size} = (2,2) \\ \text{stride} = (2,2) \\ \text{type} = \max}]{\text{Pooling}} \begin{pmatrix} 5 & 3 \\ 7 & 4 \end{pmatrix}$$

Architecture



```python
model = Sequential()

# Convolution layer 1
model.add(Conv2D(filters=96,kernel_size=(11,11),strides=(4,4),padding='valid',activation='relu',input_shape=(227,227,3)))
# Pooling layer
model.add(MaxPooling2D(pool_size=(3,3), strides=(2,2)))
# Convolution layer 2
model.add(Conv2D(filters=256,kernel_size=(5,5),strides=(1,1),padding='same',activation='relu'))
# Pooling layer 2
model.add(MaxPooling2D(pool_size=(3,3), strides=(2,2)))
# Convolution layer 3
model.add(Conv2D(filters=384,kernel_size=(3,3),strides=(1,1),padding='same',activation='relu'))
# Convolution layer 4
model.add(Conv2D(filters=384,kernel_size=(3,3),strides=(1,1),padding='same',activation='relu'))
# Convolution layer 5
model.add(Conv2D(filters=256,kernel_size=(3,3),strides=(1,1),padding='same',activation='relu'))
# Pooling layer 3
model.add(MaxPooling2D(pool_size=(3,3), strides=(2,2)))

# Flatten
model.add(Flatten())

# Dense layer 1
model.add(Dense(4096,activation='relu'))
# Dense layer 2
model.add(Dense(4096,activation='relu'))
# Output layer
model.add(Dense(1000,activation='softmax'))

model.summary()
```

Layer (type)	Output shape	Param #
conv2d (Conv2D)	(None, 55, 55, 96)	34,944
max_pooling2d (MaxPooling2D)	(None, 27, 27, 96)	0
conv2d_1 (Conv2D)	(None, 27, 27, 256)	614,656
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 256)	0
conv2d_2 (Conv2D)	(None, 13, 13, 384)	885,120
conv2d_3 (Conv2D)	(None, 13, 13, 384)	1,327,488
conv2d_4 (Conv2D)	(None, 13, 13, 256)	884,992
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 256)	0
flatten (Flatten)	(None, 9216)	0
dense (Dense)	(None, 4096)	37,752,832
dense_1 (Dense)	(None, 4096)	16,781,312
dense_2 (Dense)	(None, 1000)	4,097,000

Total params: 62,378,344 (237.95 MB)
Trainable params: 62,378,344 (237.95 MB)
Non-trainable params: 0 (0.00 B)

Explanation for parameters

Layer 1: Conv2D(96 filters, 11×11 kernel, input 3 channels)

- $(11 \times 11 \times 3) \times 96 + 96 = 34{,}944$

Layer 2: Conv2D(256 filters, 5×5 kernel, input 96 channels)

- $(5 \times 5 \times 96) \times 256 + 256 = 614{,}656$

Layer 3: Conv2D(384 filters, 3×3 kernel, input 256 channels)

- $(3 \times 3 \times 256) \times 384 + 384 = 884{,}992 + 384 = 885{,}120$

Layer 4: Conv2D(384 filters, 3×3 kernel, input 384 channels)

- $(3 \times 3 \times 384) \times 384 + 384 = 1{,}327{,}104 + 384 = 1{,}327{,}488$

Layer 5: Conv2D(256 filters, 3×3 kernel, input 384 channels)

- $(3 \times 3 \times 384) \times 256 + 256 = 884{,}736 + 256 = 884{,}992$

Fully connected(Dense) layers

Dense(9216 → 4096):

(9216×4096)+4096=37,752,832

Dense(4096 → 4096):

(4096×4096)+4096=16,781,312

Dense(4096 → 1000):

(4096×1000)+1000=4,097,000

Total: 34,944 + 614,656 + 885120 + 1,327,488 + 884,992 + 37,752,832 + 16,781,312 + 4,097,000

= 62,378,344

Evolution on CNN Architectures on ImaeNet

Year	Model	Error Rate
1998	LeNet-5	~25%
2012	AlexNet	15.3%
2013	ZFNet	11.7%
2014	VGGNet	7.3%
2014	GoogLeNet (Inception v1)	6.7%
2015	ResNet	3.6%
2016	Inception-v3 / Inception-v4	3.5% – 3.1%
2017	DenseNet	3.5%
2019	EffiecientNet	<2.0%
2021+	Vision Transformers (ViT, Swin, etc)	<1.0%

Thank You