Convolutional Neural Networks (CNN)

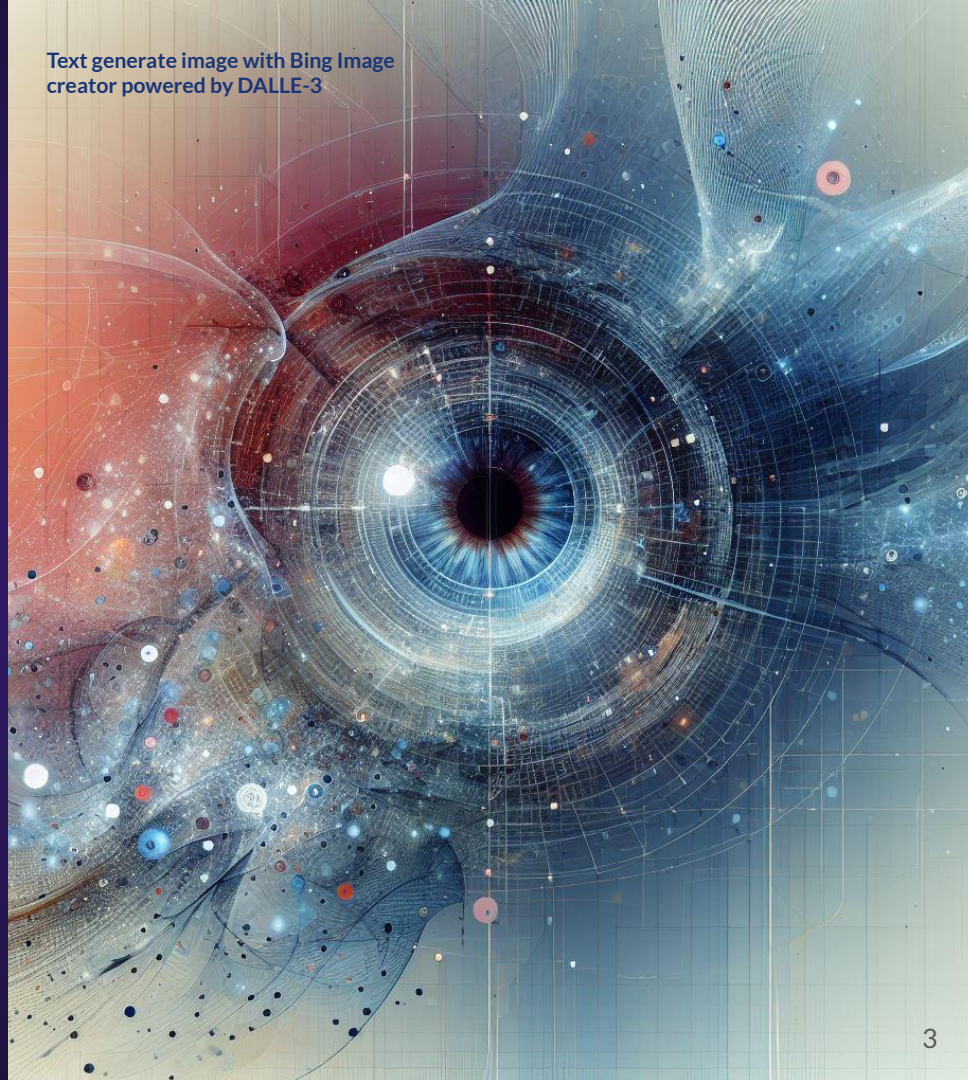
EE6350: Artificial Intelligence

Mr. M.W.G.C.K Moremada
Lecturer, Department of Electrical and Information Engineering, Faculty of Engineering, University of Ruhuna

Lecture Outline

1. **Introduction to CNN**
   a. Convolutional Neural Networks (CNN)
   b. CNN vs Traditional Feed-Forward ANN
   c. Why CNN?
2. **Core Components of CNN Architecture**
   a. Convolutional Layers
   b. Kernels
   c. Convolution Operation
   d. Padding
3. **Importance of the Convolutional Operation**
   a. Sparse Interactions
   b. Parameter Sharing
   c. Equivariance
4. **Non-Linear Activation Functions**
   a. Sigmoid
   b. Tanh (Hyperbolic Tangent)
   c. ReLU (Rectifier Linear Unit)
   d. Leaky ReLU
   e. Other Non-Linear Activation Functions
5. **Core Components of CNN Architecture Cont.**
   a. Pooling Layers
   b. Fully Connected Layers
   c. What Information are Learned by Different Layers of CNN?
6. **CNN: Regularization**
   a. Dropout & Drop Connect
   b. L1 and L2 Regularization
   c. Early Stopping
   d. Data Augmentation
   e. Batch Normalization
7. **CNN: Applications & Associated Architectures**
   a. Image Classification, Object Detection and Image Segmentation.

Introduction to CNN

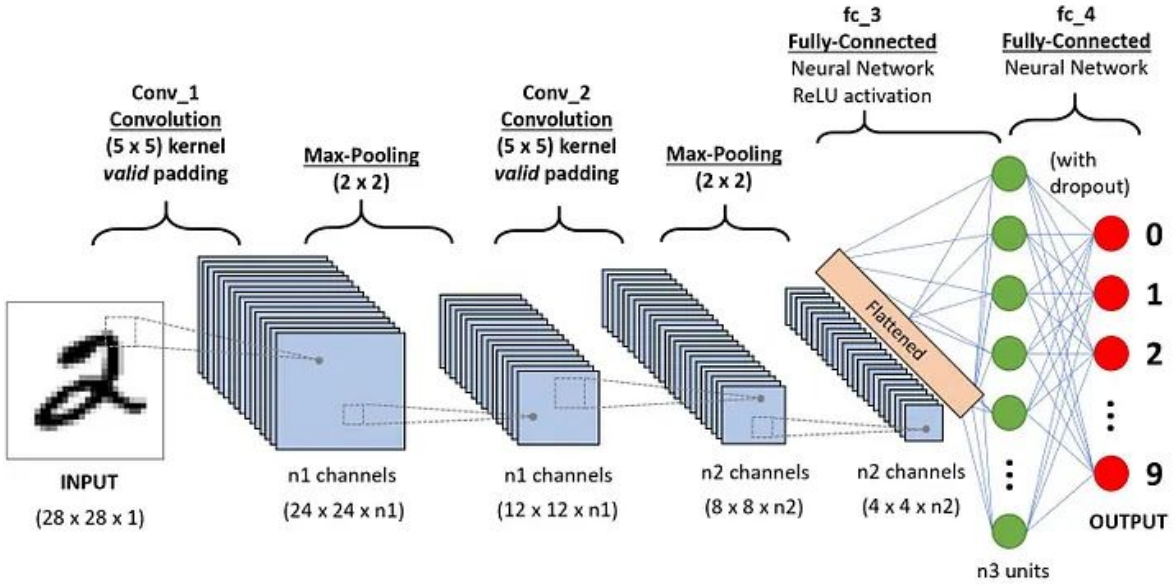


Convolutional Neural Networks (CNN)

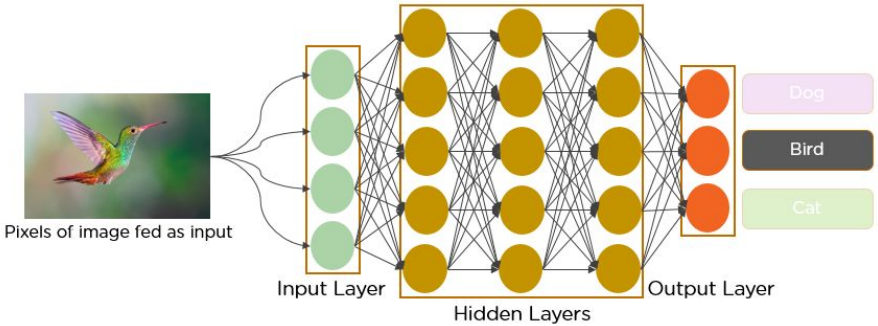
- Convolutional Networks or CNNs are specific type of neural networks that specifically designed for processing data having **grid-like topology** (e.g., **images**).
- The convolutional neural networks employs the mathematical operation known as **convolution**.

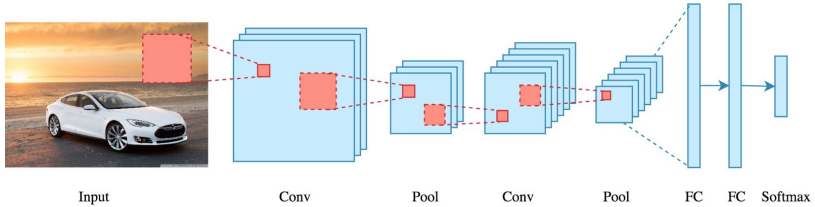
Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

Convolutional Neural Networks (CNN)

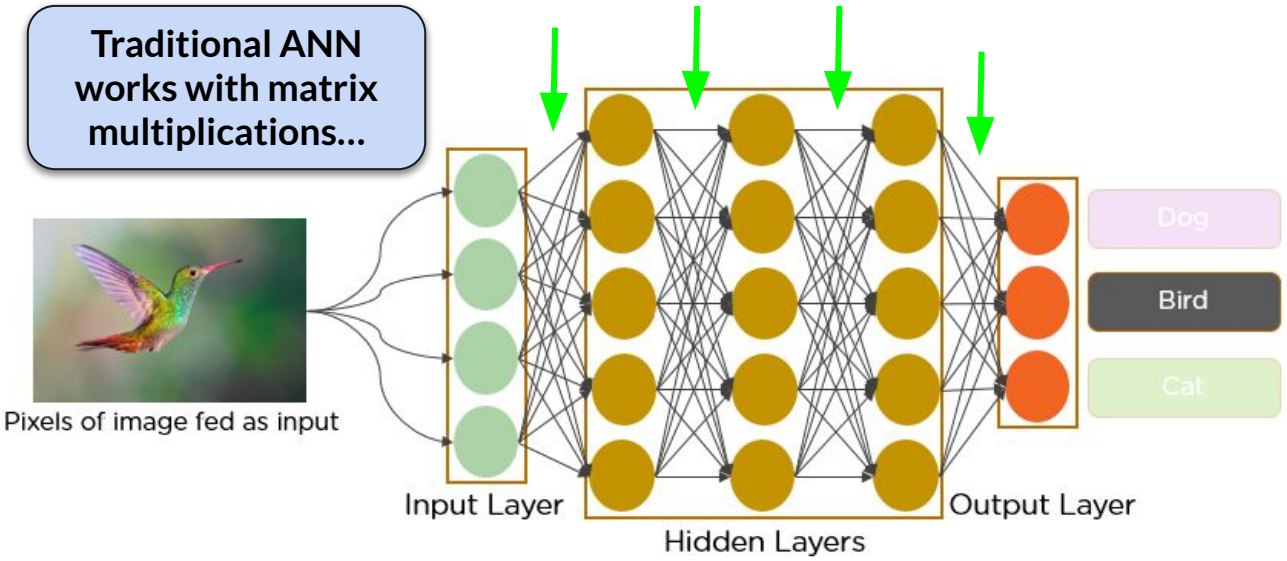


CNN vs Traditional Feed-Forward ANN





Why CNN?

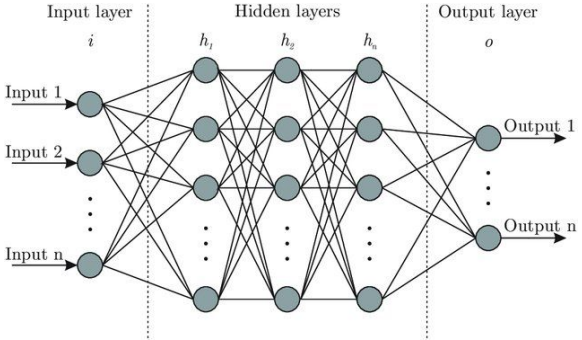


In traditional ANNs, every output unit interacts with every input unit (under fully connected cases).

But with image like data there can be thousands to million pixels.

Why CNN?

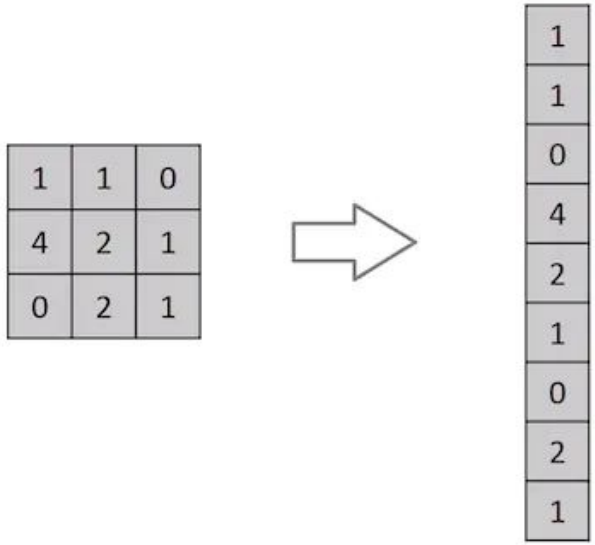
- Therefore, if there are m inputs and n outputs, then there will be m × n parameters/weights.
- Considering an input layer, an image with dimensions **24 × 24 × 3** will resulting in **1728 weights/parameters** for a single neuron and if there are same number of neurons in the first hidden layer of the ANN, it will result in **2,985,984 parameters/weights** without bias terms!



This may leads to overfitting and demands huge computational resources.

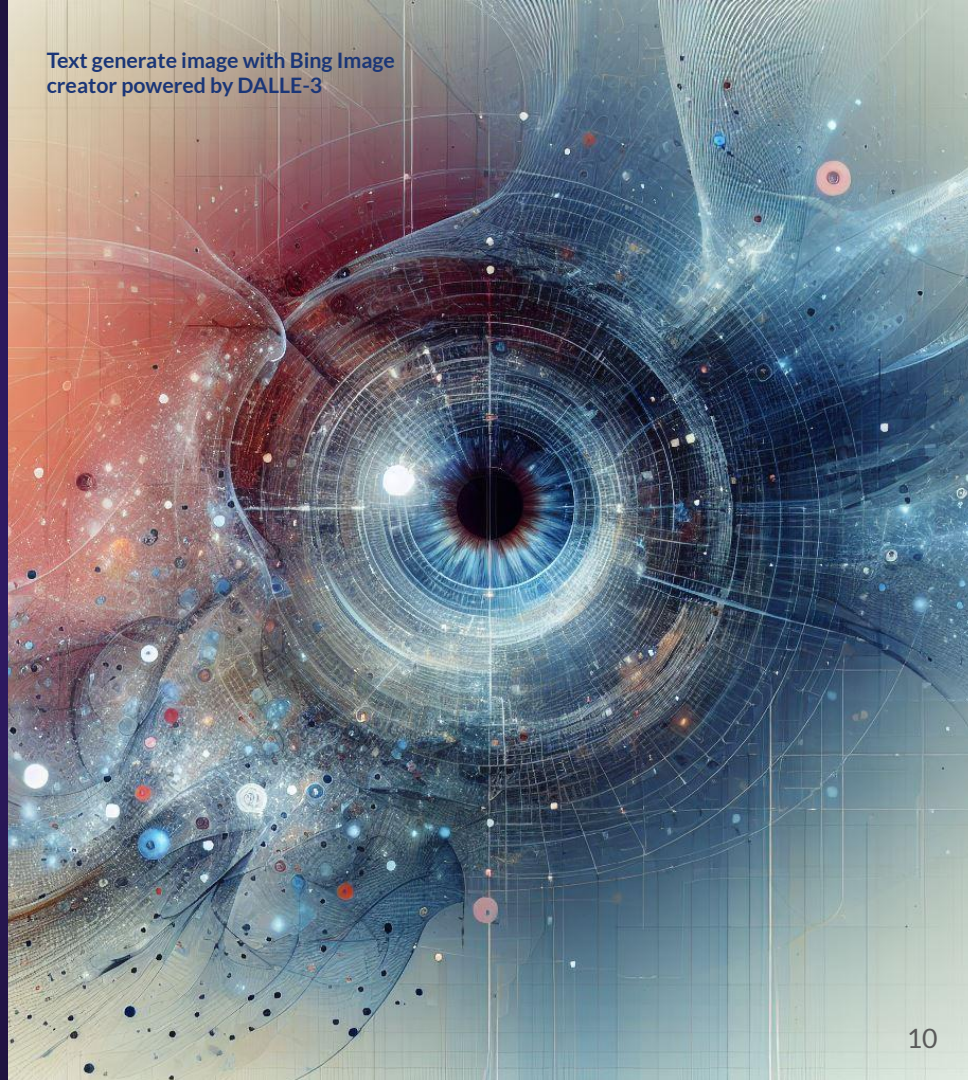
Why CNN?

- Furthermore, if we are trying to feed the images by after flattening as of the image, we will loss the **spatial dependances** in the image data which may resulting in low accuracies at the end.



More information on the importance of CNN for image like data will be provided in upcoming slides…

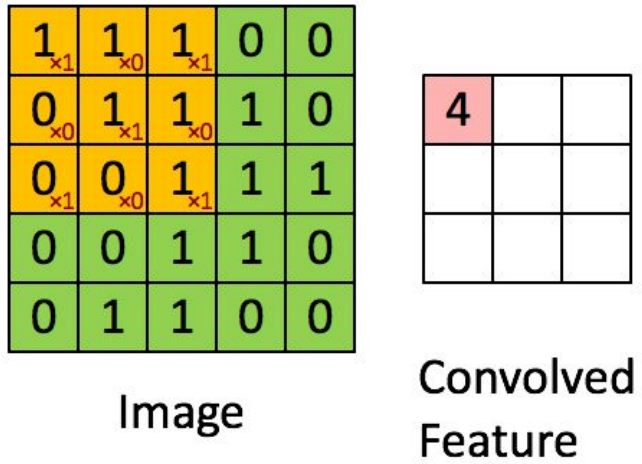
Core Components of CNN Architecture



Convolutional Layers

- Can be recognized as the most important core component in CNN.
- Convolutional layers consist of set of convolutional **kernels** (i.e, **filters**).
- After performing the convolution with an input images these kernels will generate feature maps.

Example of the Convolutional Operation in CNNs



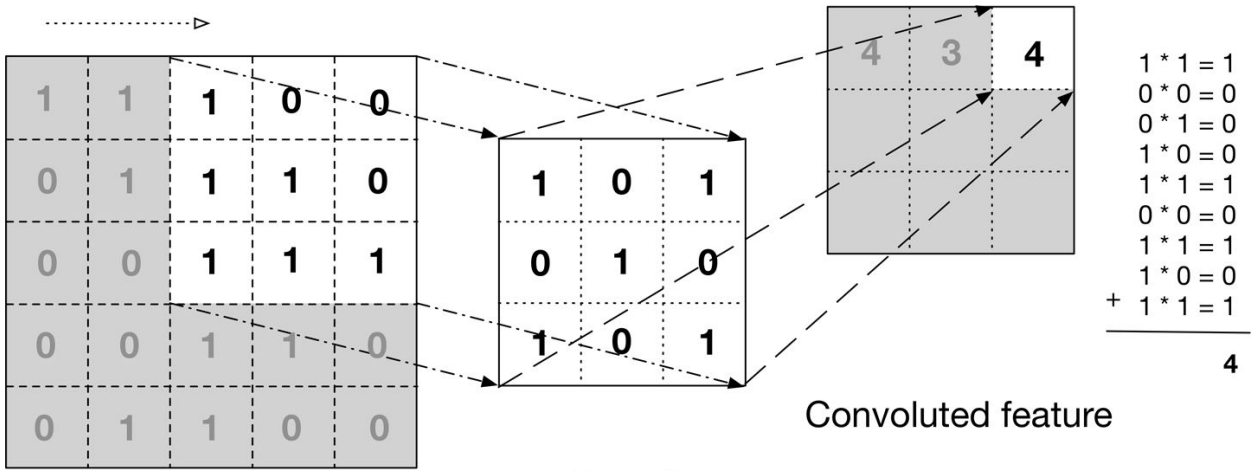
Convolutional Layers: Kernels

- A kernel can be described as a grid of discrete values or numbers, where each value is known as the **weight** of this kernel.
- At the beginning, the wights are assigned randomly and there are also other algorithms to initialize the weights as well.
- Afterwards, under each epoch the weights are tuned and the kernel learns to **extract meaningful features**.
- **CNN will uses a set of multiple filters in each convolutional layers so that each filter can extract the different types of features.**

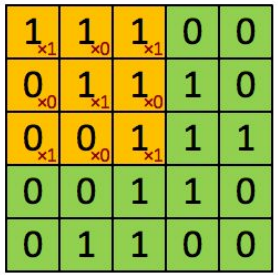
Example of a 2D Kernel

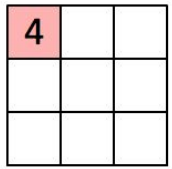
0	1
-1	2

Convolutional Layers: Convolution Operation (Grayscale Image with 1 Channel)



Dot product between kernel and the corresponding values of the image will be taken and sum up all values to generate one scalar value in the output feature map.





Convolutional Layers: Convolution Operation

In the previous case we considered a **zero padding** with **stride = 1** case.

- **Padding:** Padding involves adding extra pixels around the border of the input feature map before convolution.
- **Stride:** The distance, or number of pixels, that the kernel moves over the input matrix.

In this case, size of the generated feature map has been reduced in compared to the input image. If we further increase the stride (e.g., **Stride = 2**) the feature map size will be further reduces.

The size of the output feature maps can be calculated using following equations the (the width and height)

$$h' = \left\lfloor \frac{h - f + 2p}{s} + 1 \right\rfloor$$

$$w' = \left\lfloor \frac{w - f + 2p}{s} + 1 \right\rfloor$$

Convolutional Layers: Convolution Operation

$$h' = \left\lfloor \frac{h - f + 2p}{s} + 1 \right\rfloor$$

$$w' = \left\lfloor \frac{w - f + 2p}{s} + 1 \right\rfloor$$

- In this case
  - $h' = Height\ of\ the\ output\ feature\ map$
  - $\omega' = Width\ of\ the\ output\ feature\ map$
  - $h = Height\ of\ the\ input\ image\ or\ feature\ map$
  - $\omega = Width\ of\ the\ input\ image\ or\ feature\ map$
  - $f = Filter\ size$
  - $p = Padding$
  - $s = Stride$

Convolutional Layers: Convolution Operation

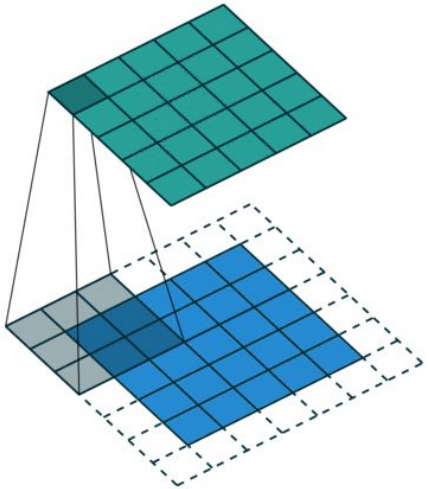
Applying the equation for the previous example we can obtain:

$$h'/w' = \left\lfloor \frac{5 - 3 + 2 * 0}{1} + 1 \right\rfloor = 3$$

So the **dimensions have been reduced!**

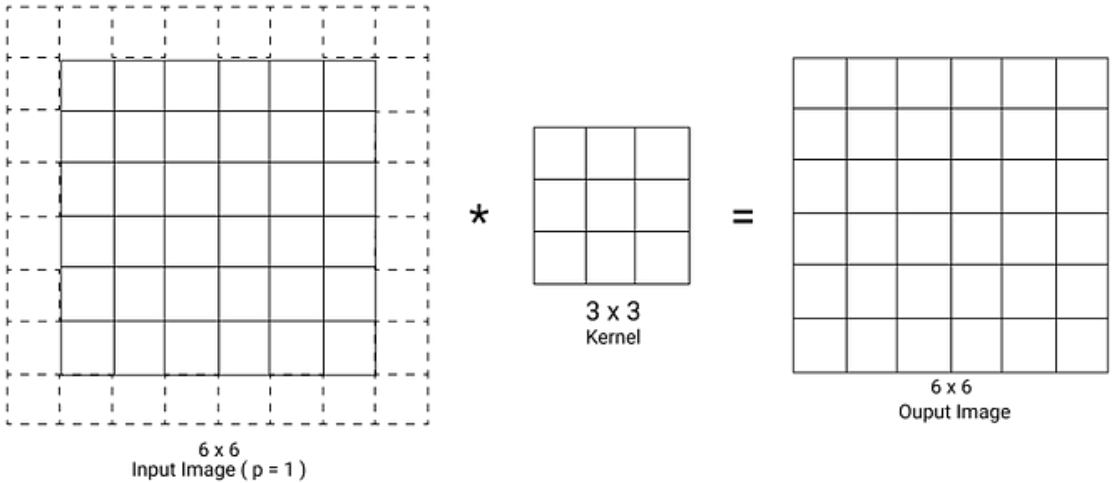
Convolutional Layers: Convolution Operation - Padding (Types)

- **Valid Padding:** This is also known as no padding.
- **Same Padding:** This padding ensures that the output layer has the same size as the input layer.



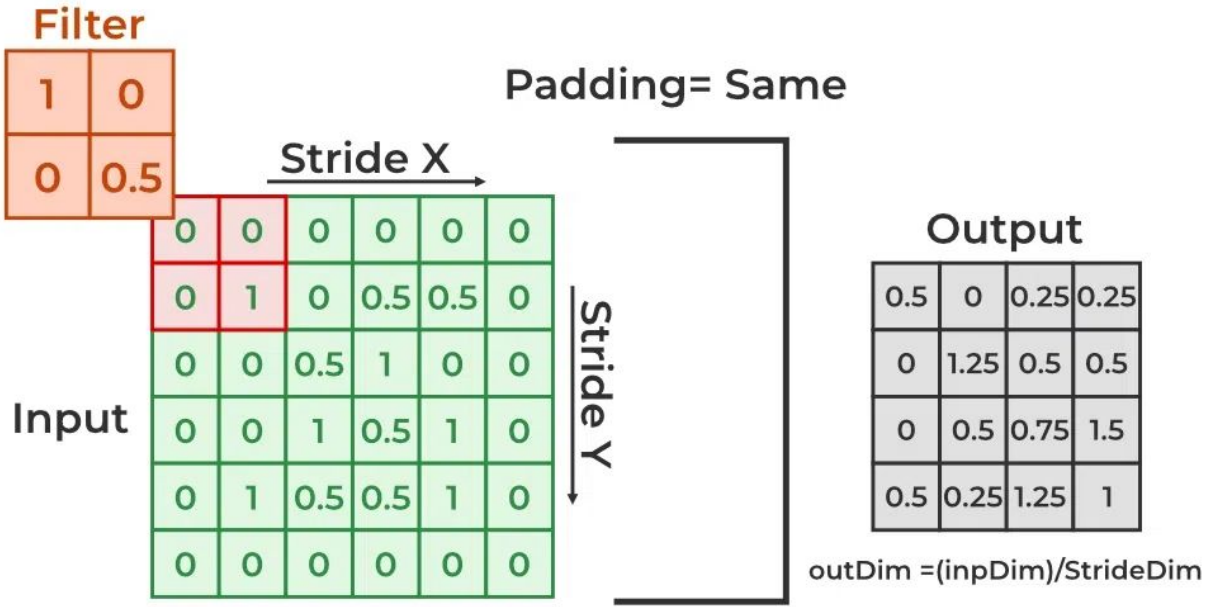
What are the other types of Padding available in relation to the CNN?

Convolutional Layers: Convolution Operation - Padding



$$h'/w' = \left\lfloor \frac{6 - 3 + 2 \times 1}{1} + 1 \right\rfloor = 6$$

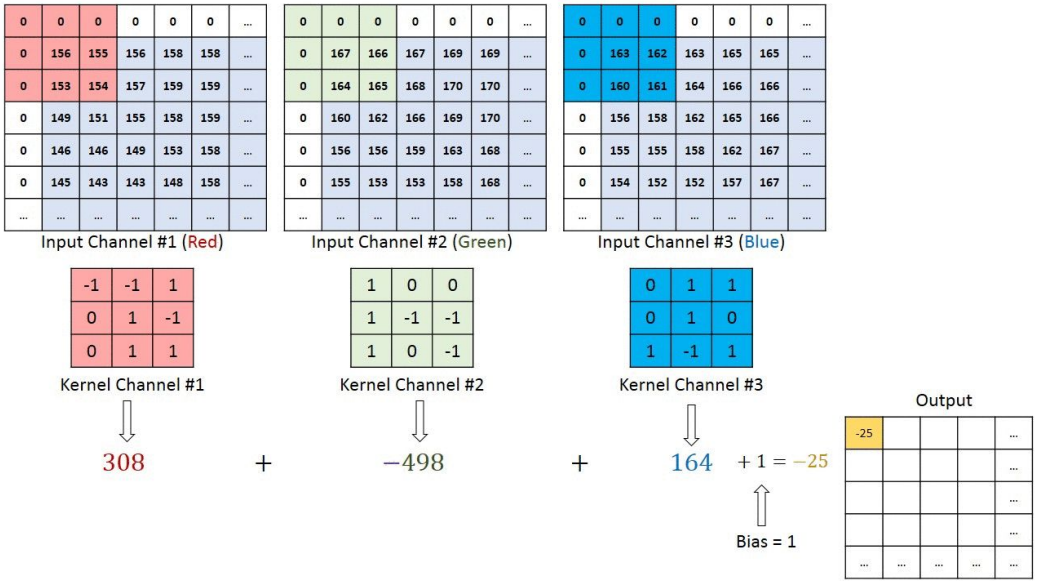
Convolutional Layers: Convolution Operation - With Same Padding



Convolutional Layers: Convolution Operation - Padding

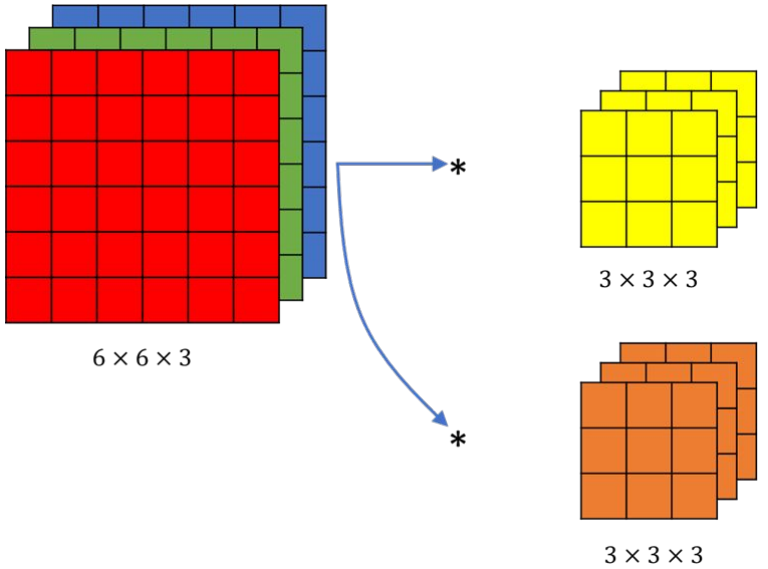
- The padding is important to **give border size information of the input image more importance,** otherwise without using any padding t**he border side features are gets washed away too quickly**.
- Sometimes the **original size of the image(no shrink)has to be preserved** to extract some low-level features.
- The padding is also used to increase the input image size, as a result the output feature map size also get increased.

Convolutional Layers: Convolution Operation - For an RGB Image (3 Channels)

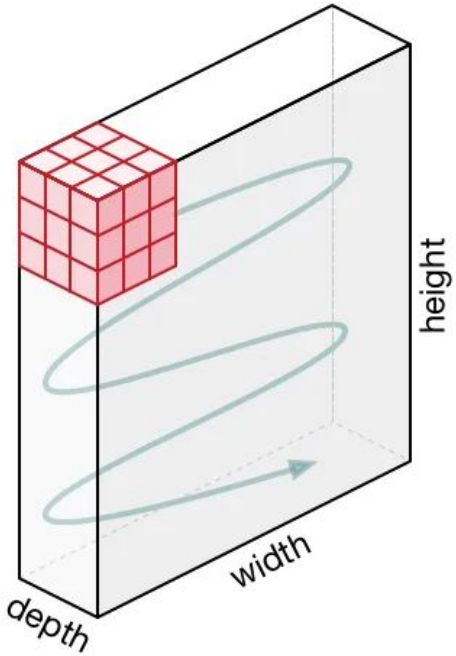


Convolutional Layers Cont.

- **We can have more than one filter in a convolutional layer.**
- **The depth of a filter/kernel should be the same as of the input.**
- The example is for a RGB image.



Convolutional Layers Cont.



0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

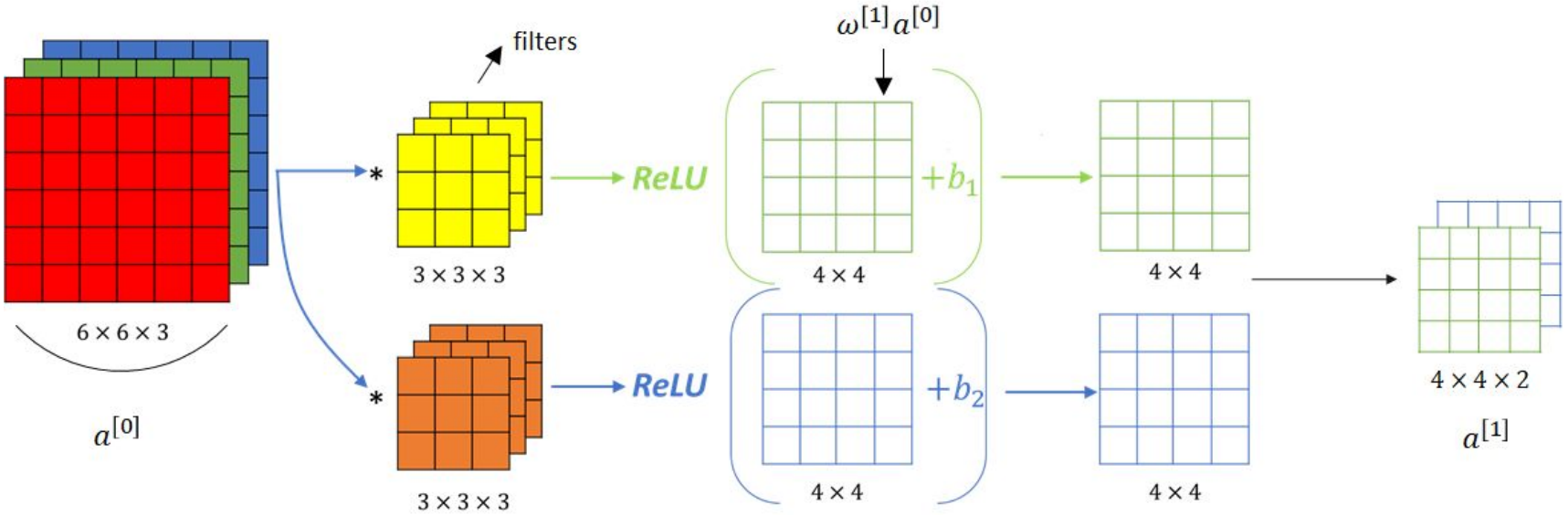
$308 + (-498) + 164 + 1 = -25$

Bias = 1

Output

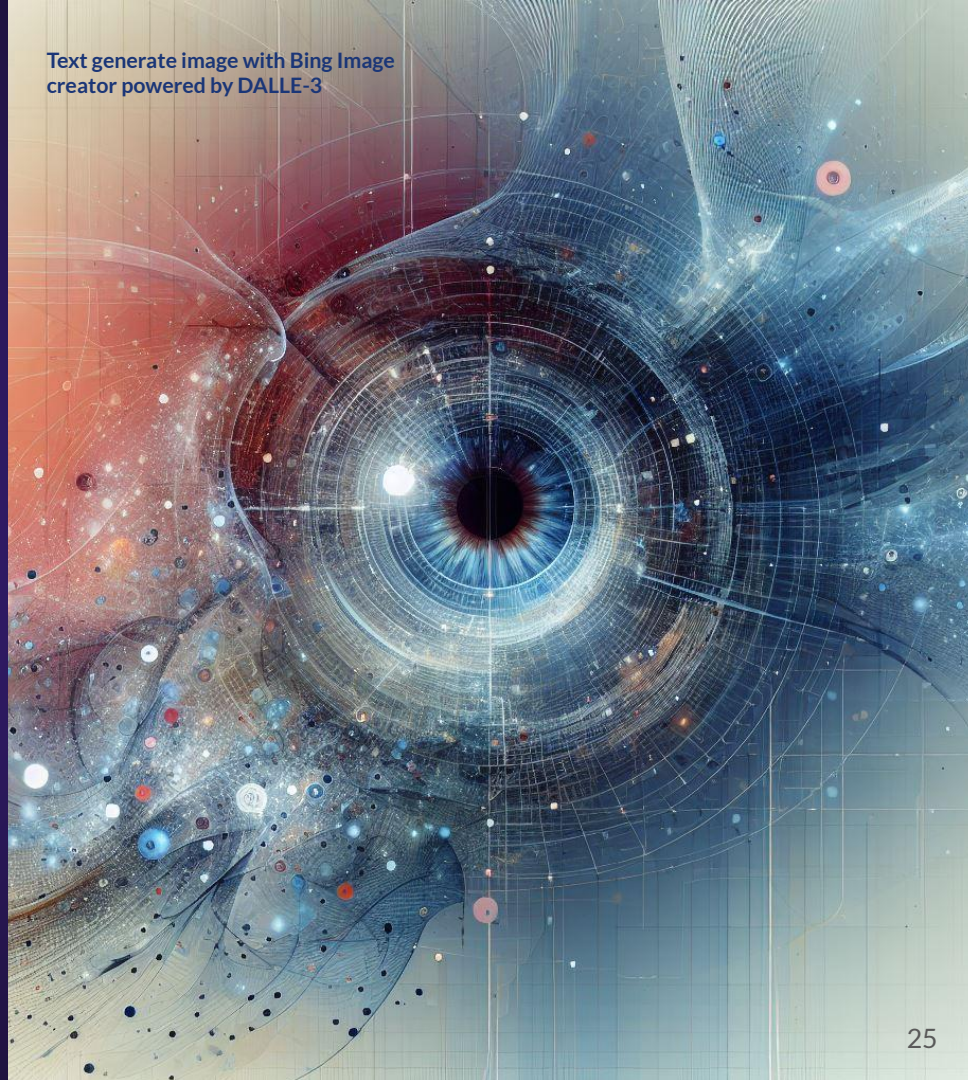
-25				...
				...
				...
				...
...	...	...	...	...

Result of the Convolution



Importance of the Convolutional Operation

The Motivation



Motivation Behind the Convolutional Operation

- Convolution operation leverages three important ideas which resulting in better performance of CNNs.
  - **Sparse interactions.**
  - **Parameter sharing.**
  - **Equivariant representations.**

Motivation: Sparse Interactions

- This is also referred as **sparse connectivity** or **sparse weights**.
- This has been achieved by making the kernel/s smaller than the input.

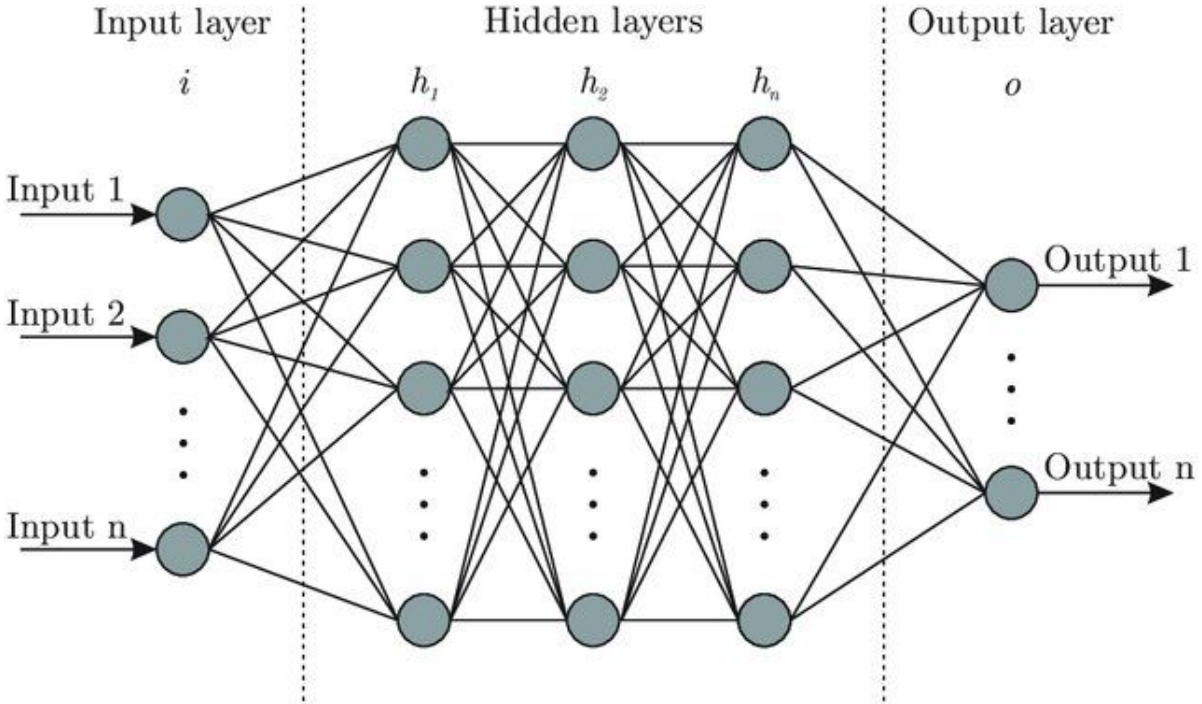
Basic idea is even though images contains thousand to millions of pixels. But there can be only a few meaningful features such as edges which can be a order of magnitude lower than the original number of pixels. So, we only need to store only a fewer amount of parameters.

- This will **reduce computing and memory requirements** and while improving the statistical efficiency.

Motivation: Sparse Interactions

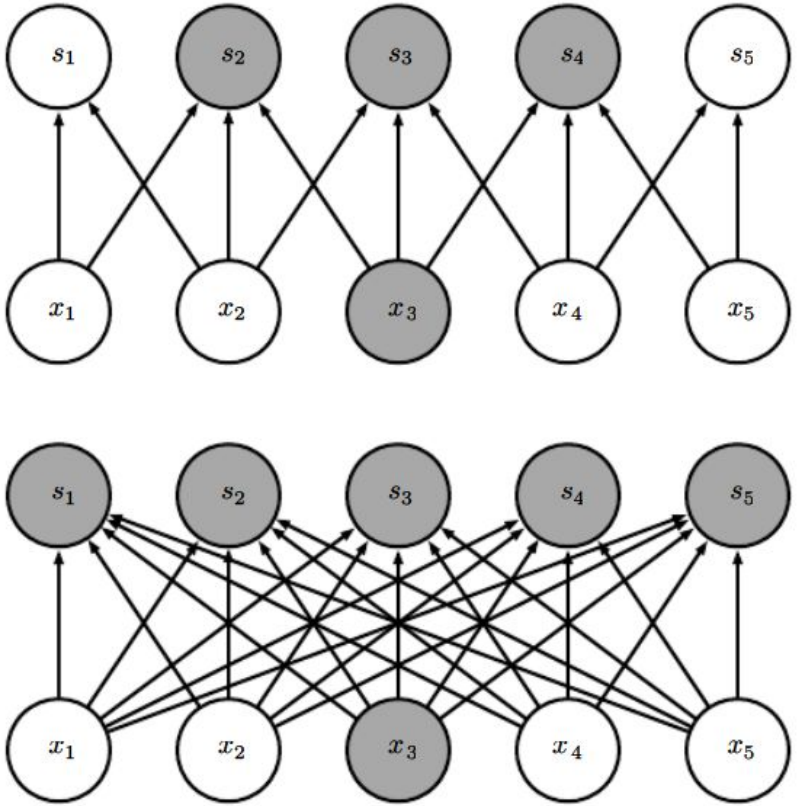
- Traditional NNs utilize **matrix multiplications to describe the interaction between each input and output units.**
- For fully connected cases, every input unit interacts with every output units.
- Therefore with **m** input units and **n** output units, then the matrix multiplication accompany **m × n** parameters and **O(m × n)** runtime in the algorithm.
- If we limit number of connections each output may have to **k which is order of magnitude lower than m**, then there will be **k × n** parameters and only **O(k × n).**

Problem With Tradition Fully Connected ANNs



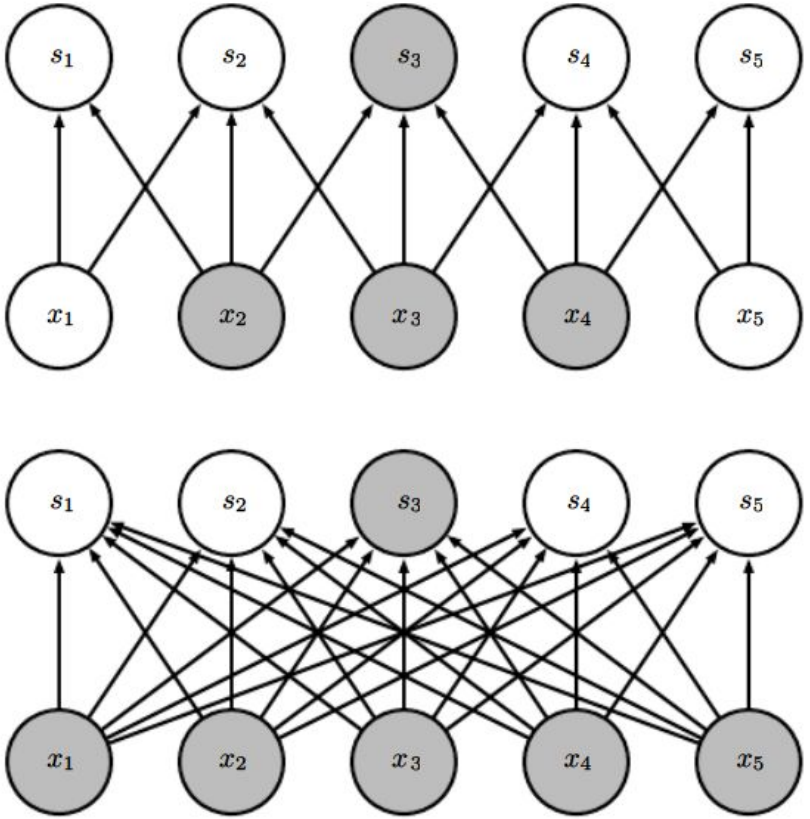
Motivation: Sparse Interactions

- Kernel of with 3: **Only 3 outputs are affected** by x_3.
- With matrix multiplication all the inputs outputs are affected by all the inputs.



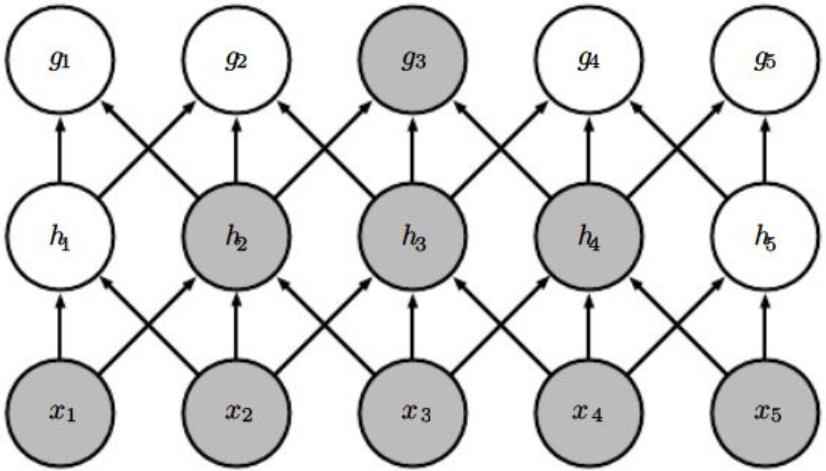
Motivation: Sparse interactions

- With the usage of kernel only three inputs are affecting a single output.
- With matrix multiplication all the inputs are affecting as of the example.



Motivation: Sparse Interactions

Even though direct connections in a convolutional net are very sparse, **units in the deeper layers can be indirectly connected** to all or most of the input image.

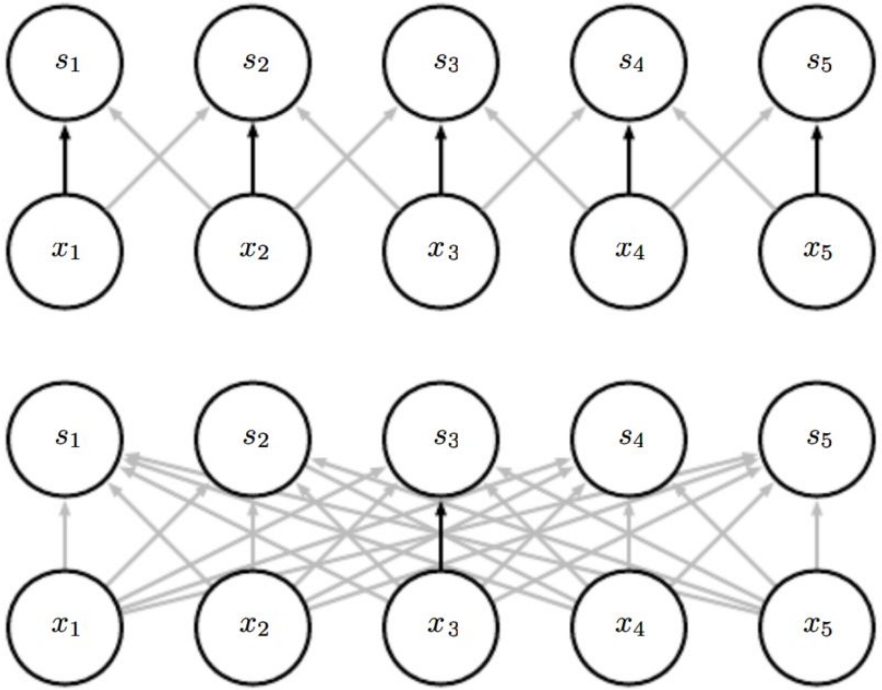


Motivation: Parameter Sharing

- Under parameter sharing, **same parameter use for more than one function in a model.**
- In a traditional neural net, **each element of the weight matrix is used exactly once** when computing the output of a layer.
- The parameter sharing used by the convolution operation means that **rather than learning a separate set of parameters for every location, we learn only one set.**
- Convolution is thus dramatically more efficient than dense matrix multiplication in terms of the memory requirements and statistical efficiency.

Motivation: Parameter Sharing

Under the first case, the central element of the 3-element kernel has been utilized in all the input locations. But in a fully connected model a parameter have utilized only once.

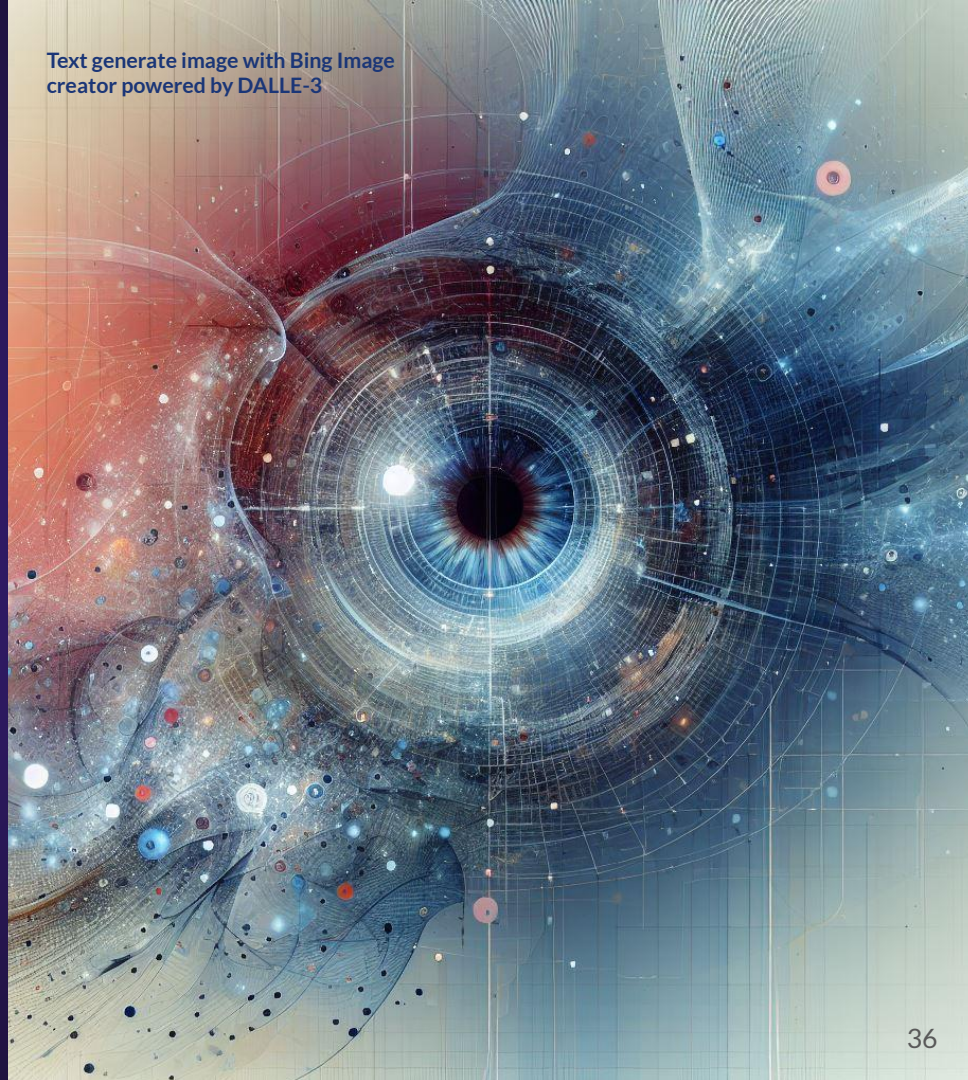


Motivation: Equivariance

- To say a function is equivariant means that **if the input changes, the output changes in the same way.**
- A function ***f(x)*** is equivariant to a function ***g*** if: ***f(g(x)) = g(f(x))*.**
- E.g., With images, convolution creates a 2-D map of where certain features appear in the input. If we move the object in the input, its representation will move the same amount in the output.

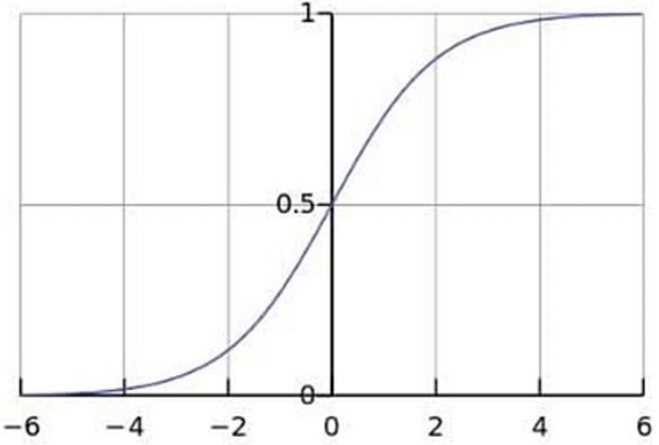
Ref: https://fabianfuchsml.github.io/equivariance1of2/#:~:text=CNNs%20are%20famously%20equivariant%20with,parameters%20by%20orders%20of%20magnitude. | https://youtu.be/goWAFBYOtoU

Non-Linear Activation Functions



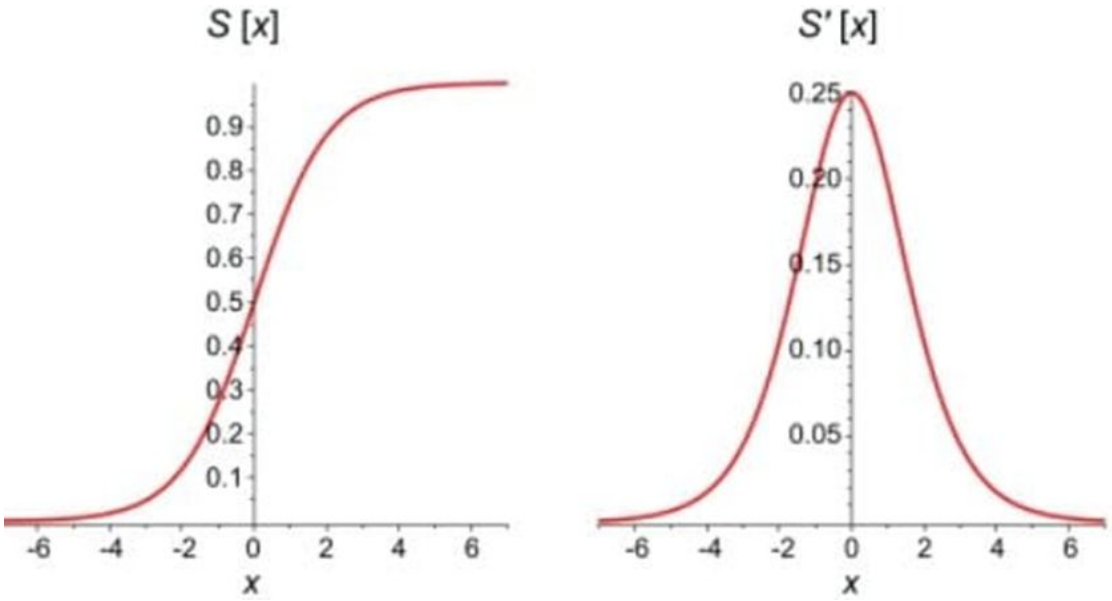
Sigmoid

- Also known as **logistic function**.
- Useful as an activation function when one is interested in **probability mapping**.
- Can lead to **slow learning** and the **model getting trapped in local minima** during training.
- Furthermore, this is **subjected to vanishing gradient problem** in deep neural networks and not a zero-centric function (always give positive values)



$$\phi(z) = \frac{1}{1 + e^{-z}}$$

Sigmoid

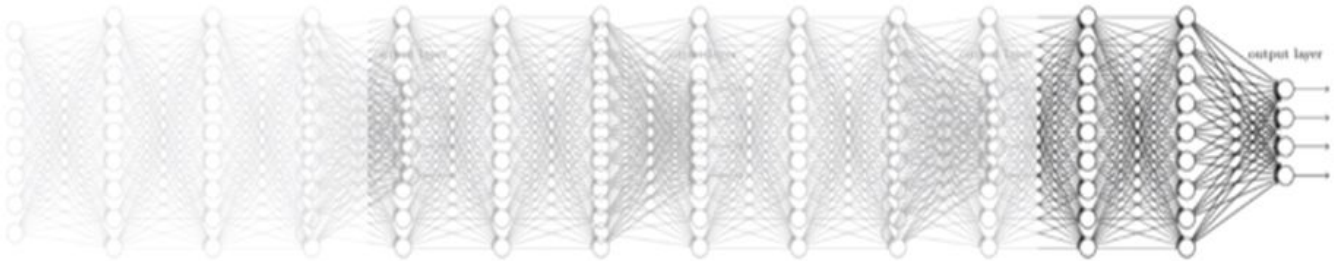


Derivative of the Sigmoid Function

Info Slide

Vanishing Gradient Problem

- The vanishing gradient problem is a phenomenon that occurs during the training of deep neural networks, **where the gradients that are used to update the network become extremely small** or **"vanish"** as they are **backpropagated** from the output layers to the earlier layers.



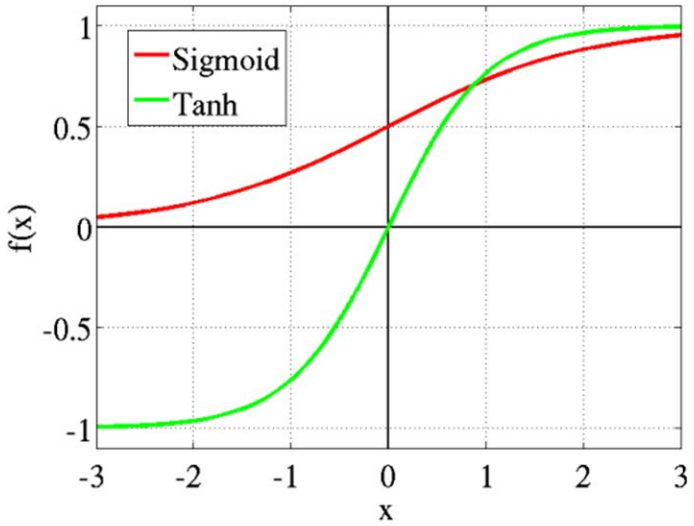
Info Slide

Vanishing Gradient Problem

- During the training process of the **NNs are trying to minimize the loss function** by adjusting the weights and the **backpropagation algorithm computes these gradients by propagating error from the output layer to the input.**
- Because of the vanishing gradient problem, the NNs may **display slow convergence** and **may get stuck at local minima** while indicating impaired learning of deep representations.

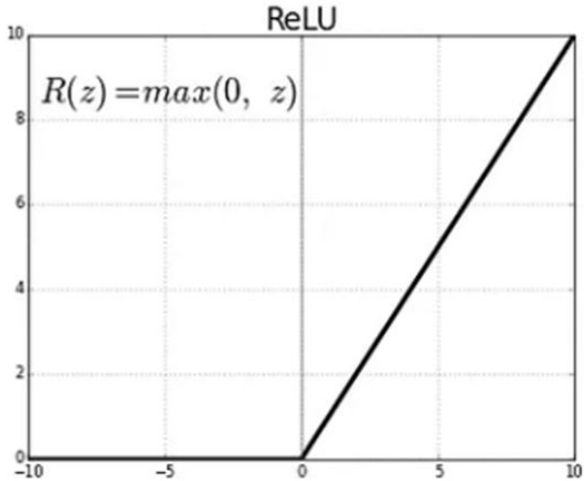
Tanh (Hyperbolic Tangent)

- The range of the tanh function is from (-1 to 1).
- This is a zero-centric function which **removes the requirement for center the data under certain cases.**
- However, this can be **still subjected to the vanishing gradient problem** and can be computationally more expensive.



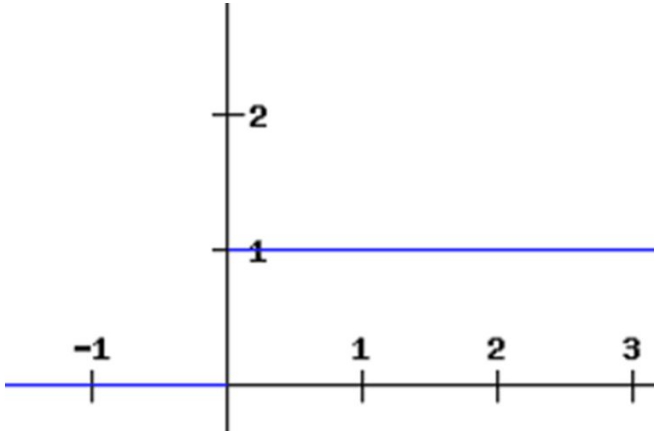
ReLU (Rectifier Linear Unit)

- This function ranges from **0 to +infinity.**
- The ReLU function is non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
- It **does not activate all the neurons at the same time**. Since the output of some neurons is zero, **only a few neurons are activated** making the network sparse, efficient, and easy for computation.
- **Reduce** the vanishing gradient issue.



ReLU (Rectifier Linear Unit)

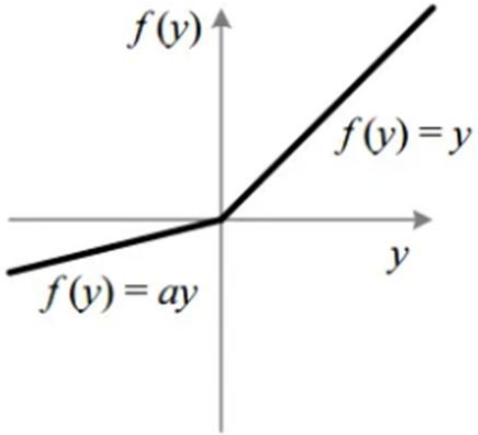
- **Non-differentiable at zero.**
- The gradients for negative input are zero, which means for **activations in that region, the weights are not updated during backpropagation**. This can create **dead neurons** that **never get activated**. This can be handled by reducing the learning rate and bias.
- ReLU output is **not zero-centered** and it does hurt the neural network performance.



Derivative of the ReLU Function

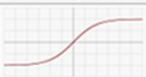
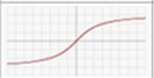
Leaky ReLU

- With this function it has been attempted to solve the dying ReLU problem.
- Range: -Infinity to +infinity.
- If the output of a ReLU is consistently 0 (for example, if the ReLU has a large negative bias), then the gradient through it will consistently be 0. **The error signal backpropagated from later layers gets multiplied by this 0, so no error signal ever passes to earlier layers. The ReLU has died.**
- No dying ReLU units with Leaky ReLU and speed up the training process, however, this can be saturate for large negative values.

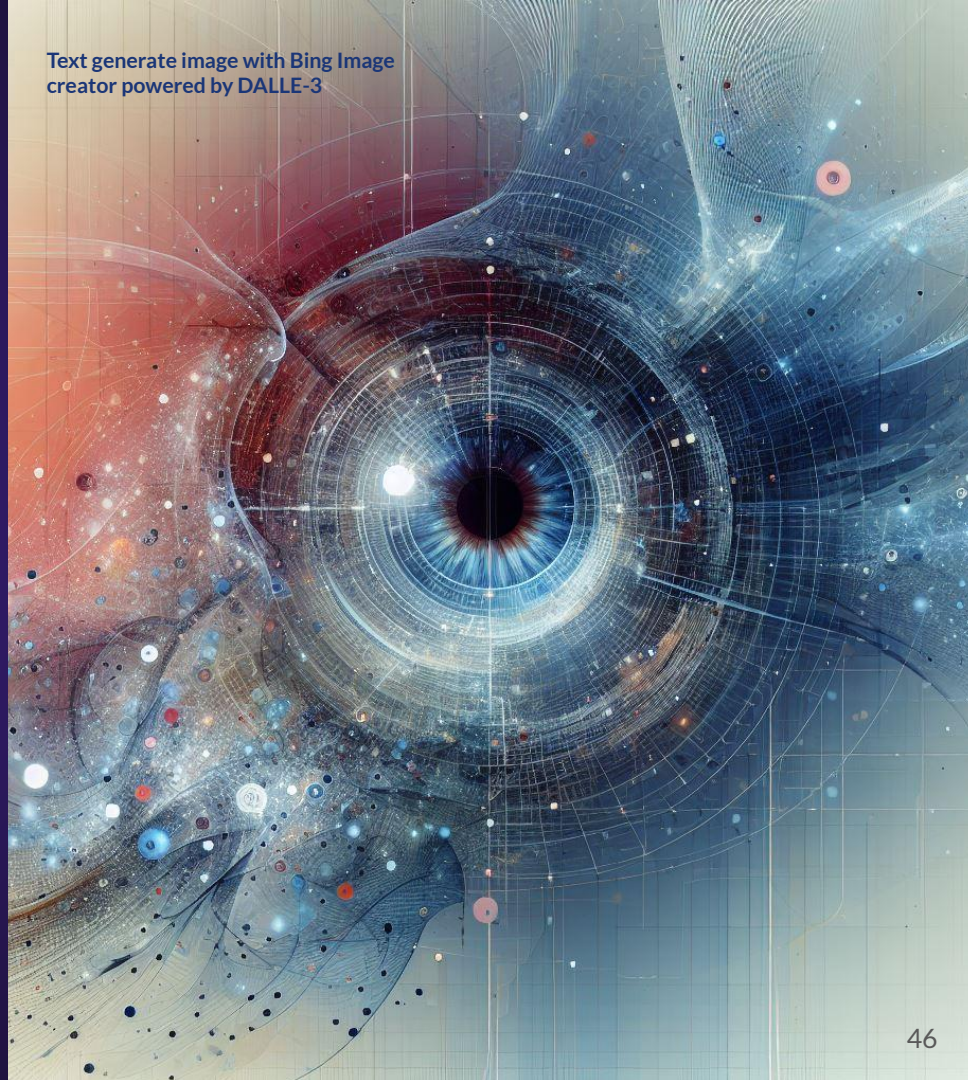


Other Non-Linear Activation Functions

Study on the use-case, advantages and disadvantages of the other non-linear activation functions.

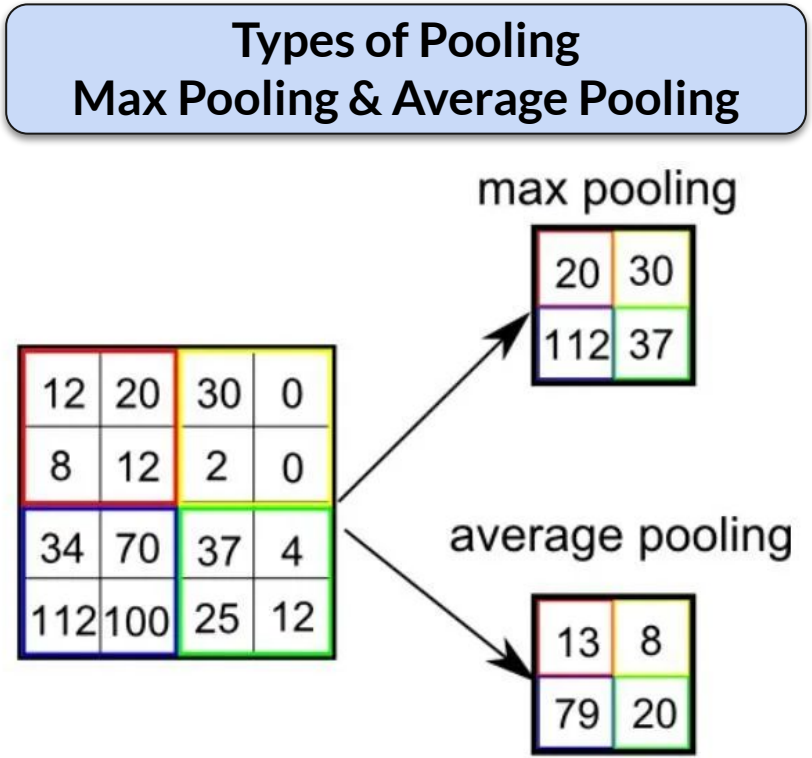
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Core Components of CNN Architecture Cont.

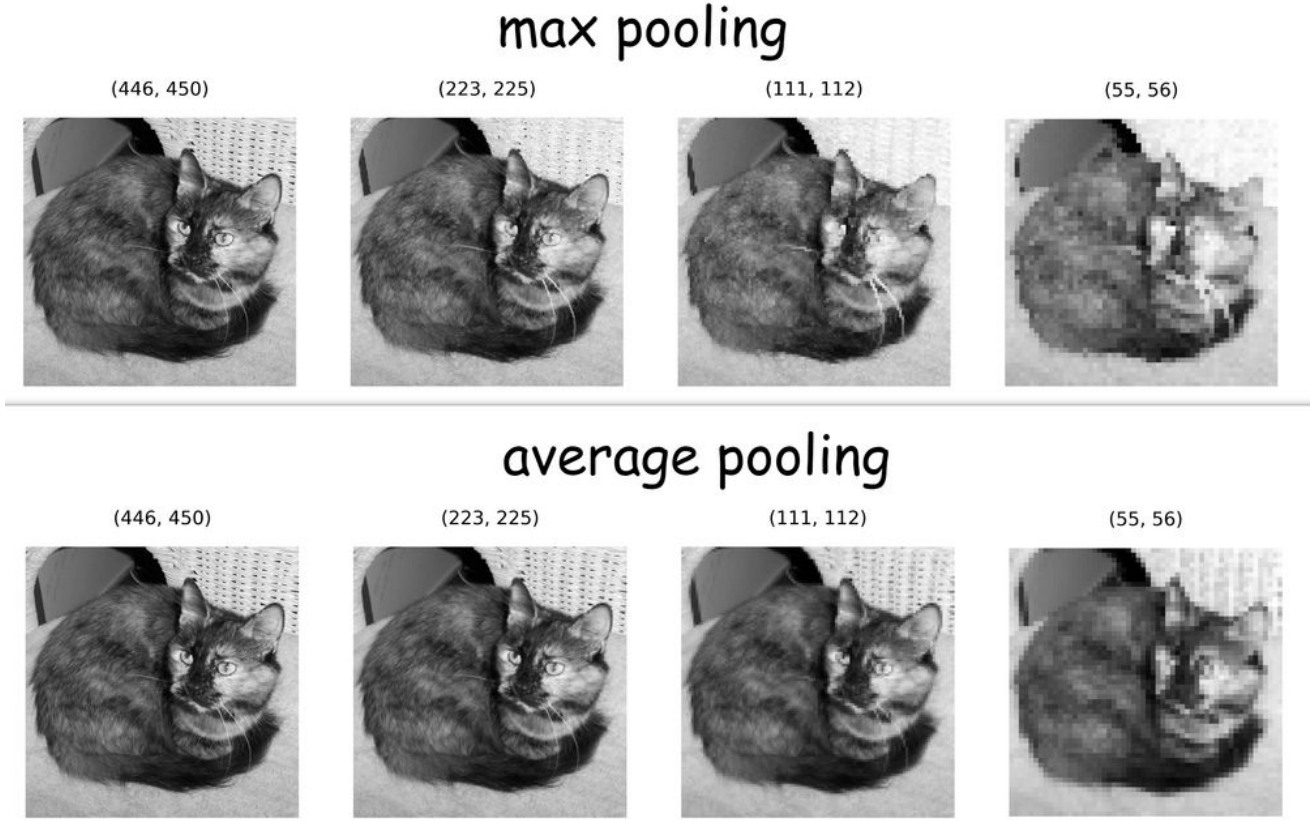


Pooling Layers

- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.
- This will **reduce the spacial size** and **decrease the computational power** required.
- Also useful in **extracting the dominant features**.



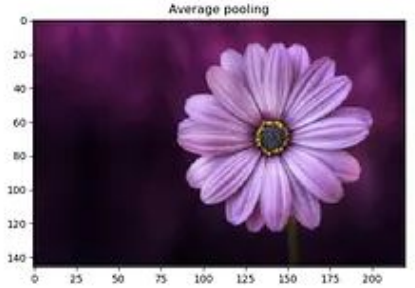
Pooling Layers

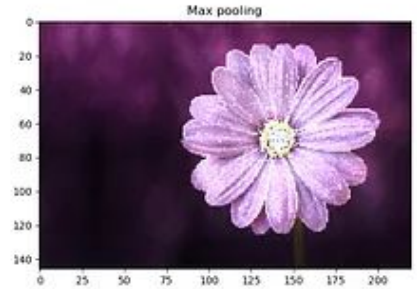


Pooling Layers

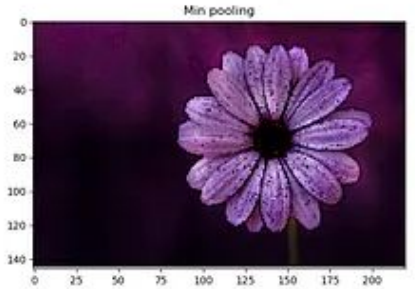
Original Image

Average Pooling





Max Pooling



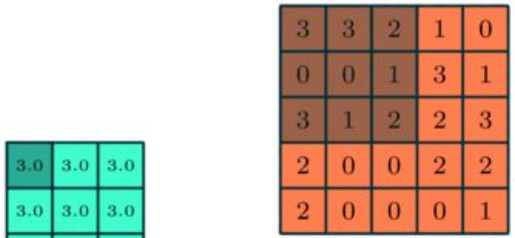
Min Pooling

Pooling Layers

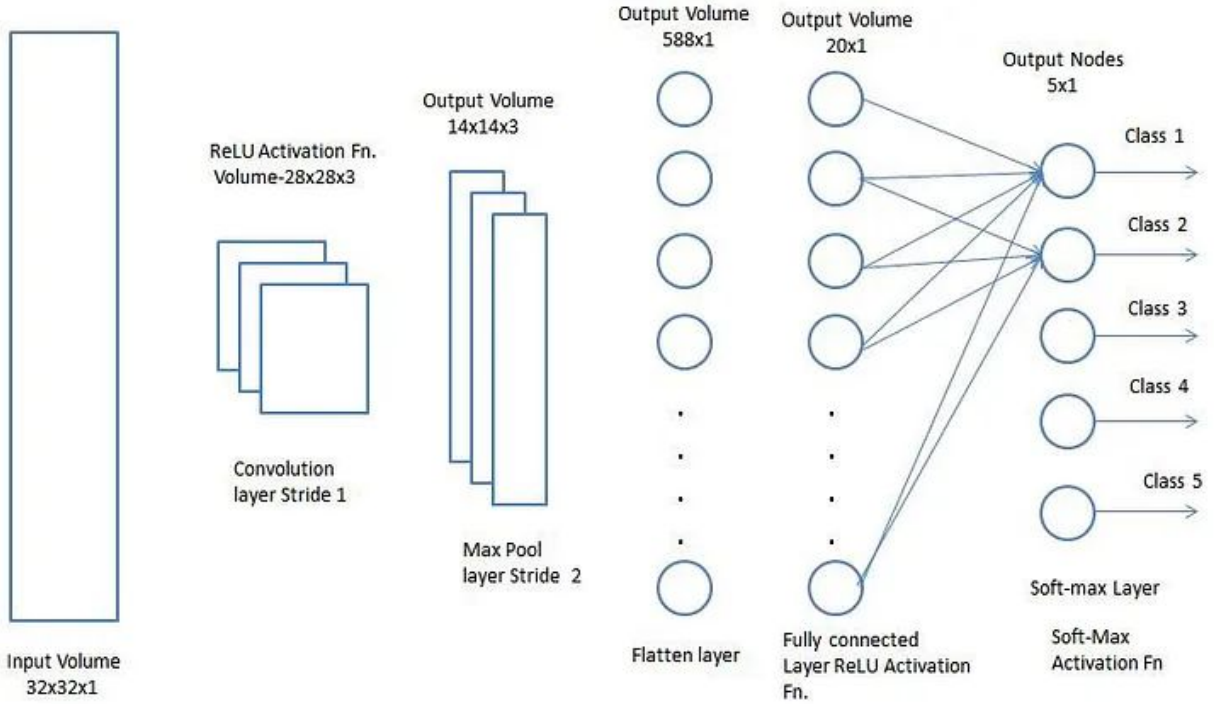
- Pooling **sub-sample** the generated feature maps.
- They **preserve the most most dominant features/information**. However this may leads to some contextual information loss under some cases.

$$h' = \left\lfloor \frac{h - f}{s} + 1 \right\rfloor$$

$$w' = \left\lfloor \frac{w - f}{s} + 1 \right\rfloor$$



Fully Connected Layers



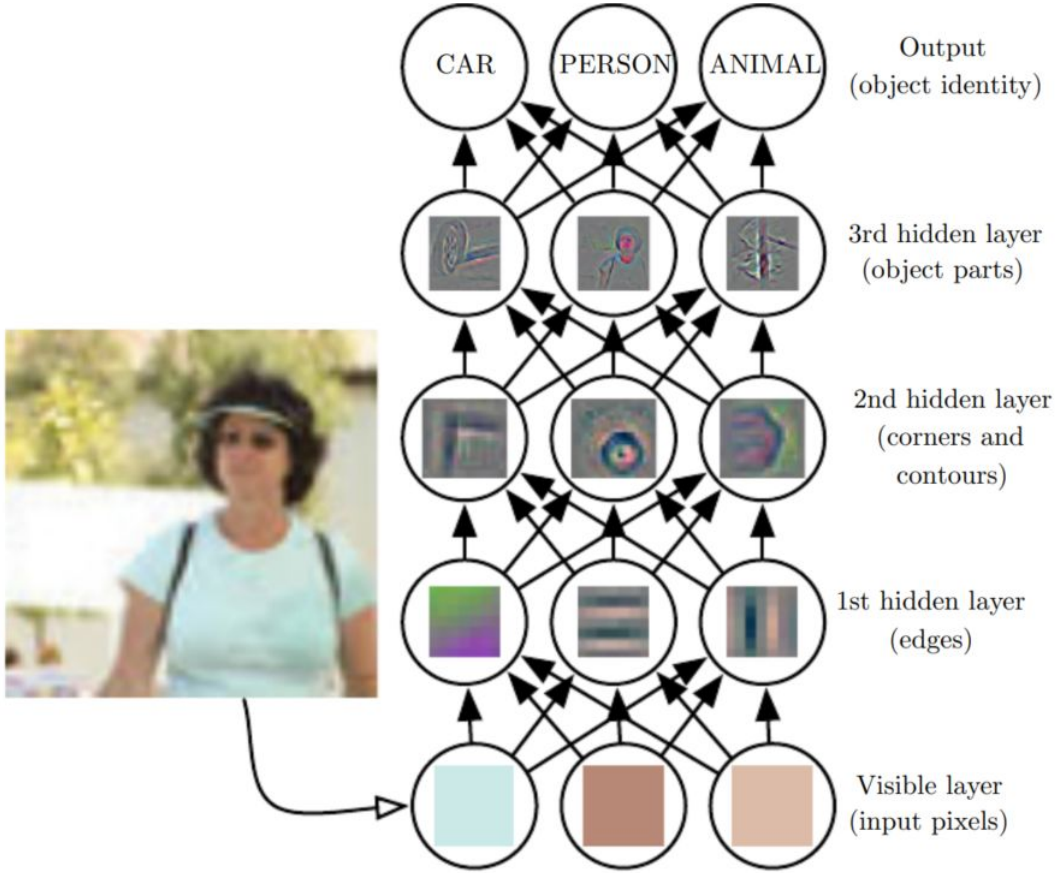
Fully Connected Layers

- With fully connected layers we can learn **non-linear combinations** of the high level features.
- In CNN we need to **flatten** the outcomes from the previous layers into a column vector before feeding into the fully connected layers.
- The learning of the whole network will be happening by means of **backpropagation** algorithm in series of **epochs** in general.
- The model is able to distinguish between dominating and certain low-level features in images and classify them using the Softmax Classification technique.

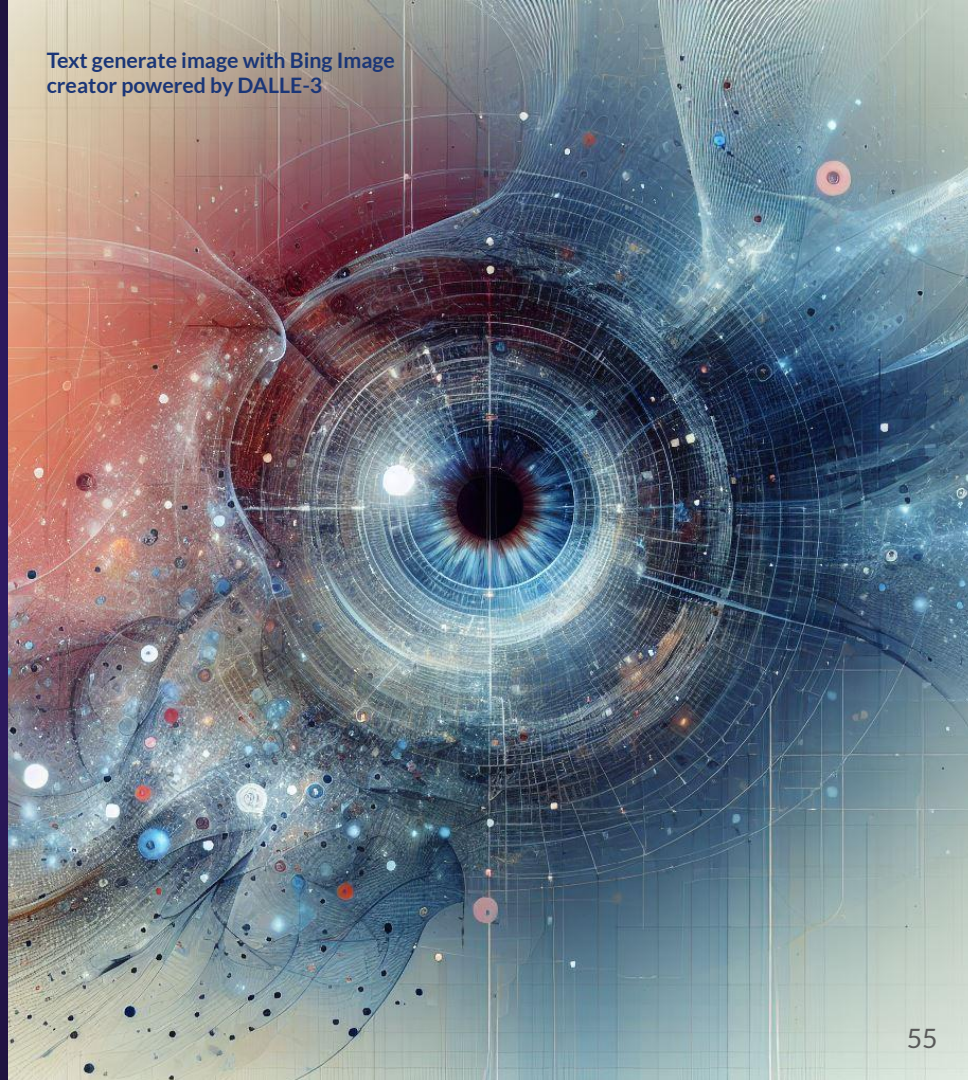
What Information are Learned by Different Layers of CNN?

- **Early Convolutional Layers:** Focus on the basic features such as fundamental shapes, edges, lines and color gradients.
- **Middle Convolutional Layers:** Combining simple features; these layers learn how to combine the basic features from earlier layers into more complex patterns (e.g., textures, corners, simple shapes like circles or squares).
- **Later Convolutional Layers:** Recognizing complex objects; these layers learn to identify highly specific objects or parts of objects (e.g., eyes, noses, and other facial features based on the application).

What Information are Learned by Different Layers of CNN?

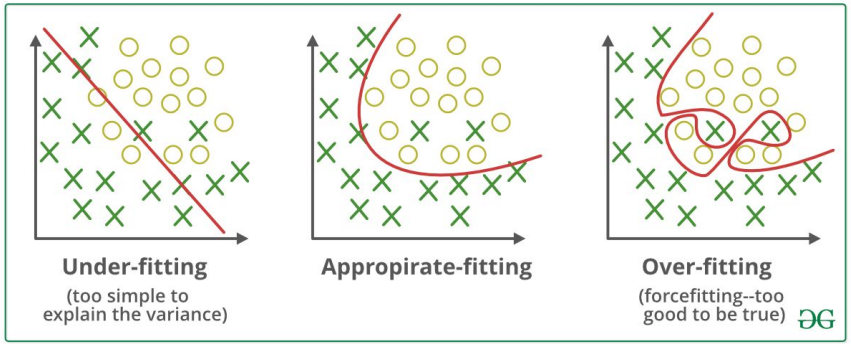


CNN: Regularization



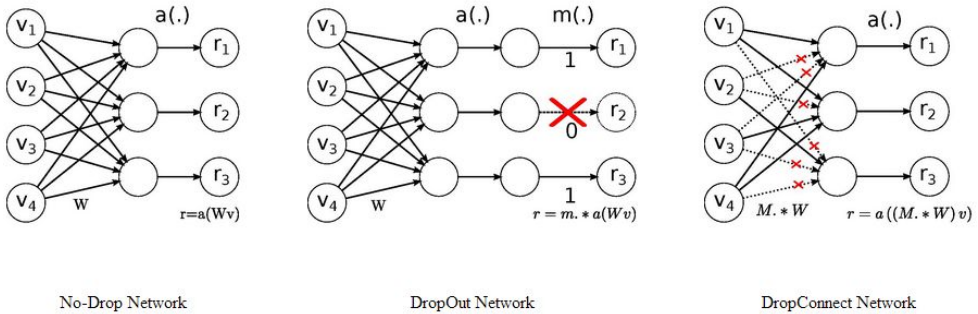
Regularization

- **The core challenge of deep learning or machine learning problems is to adapt to new or previously unseen data.**
- This process known as model **generalization.**
- In this case we have to avoid **overfitting** and **underfitting** scenarios.
- The model **regularization help us to ensure model generalization**.



Regularization: Dropout & Drop Connect

- When training with Dropout, **a randomly selected subset of activations are set to zero within each layer.**
- DropConnect instead sets a **randomly selected sub-set of weights within the network to zero**. Each unit thus receives input from a random subset of units in the previous layer.



No-Drop Network DropOut Network DropConnect Network

Regularization: Dropout & Drop Connect

A Note on Dropout:

- With dropout dropped units would not take part in both forward propagation or backward propagation during the training process.
- But in the case of testing process, the full-scale network is used to perform prediction.

Regularization: L^1 and L^2 Regularization

- L1 and L2 regularization are methods in machine learning that a**dd a penalty term to the loss function.** L1 regularization is also known as **lasso regression**, and L2 regularization is also known as **ridge regression**.
- L1 regularization generates **sparse solutions and is helpful for feature selection.** L2 regularization **yields non-sparse solutions and is beneficial for building simpler models.**

Objective Function for L^2 Reg.

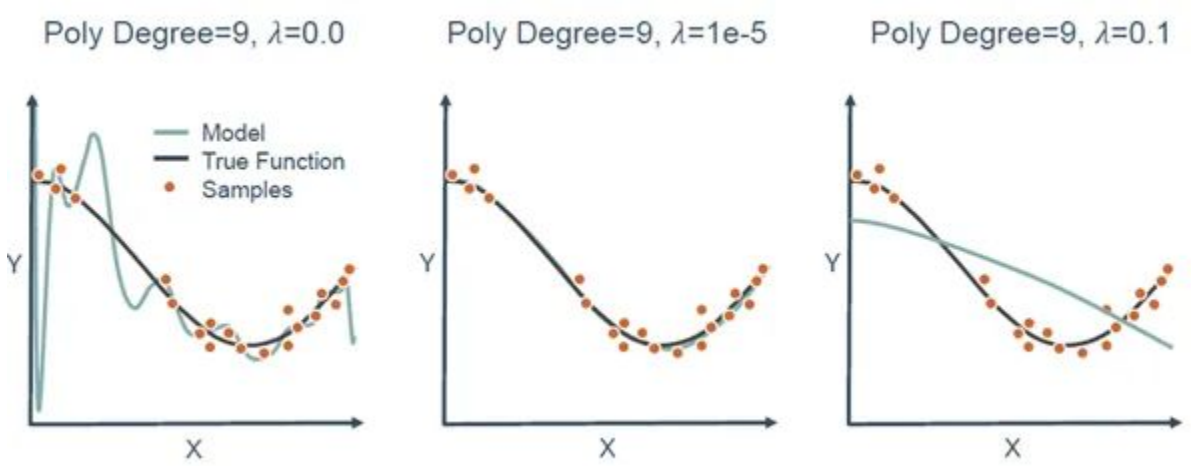
$$Cost\ Function = loss + \frac{1}{2}\lambda||w||_2^2$$

Objective Function for L^1 Reg.

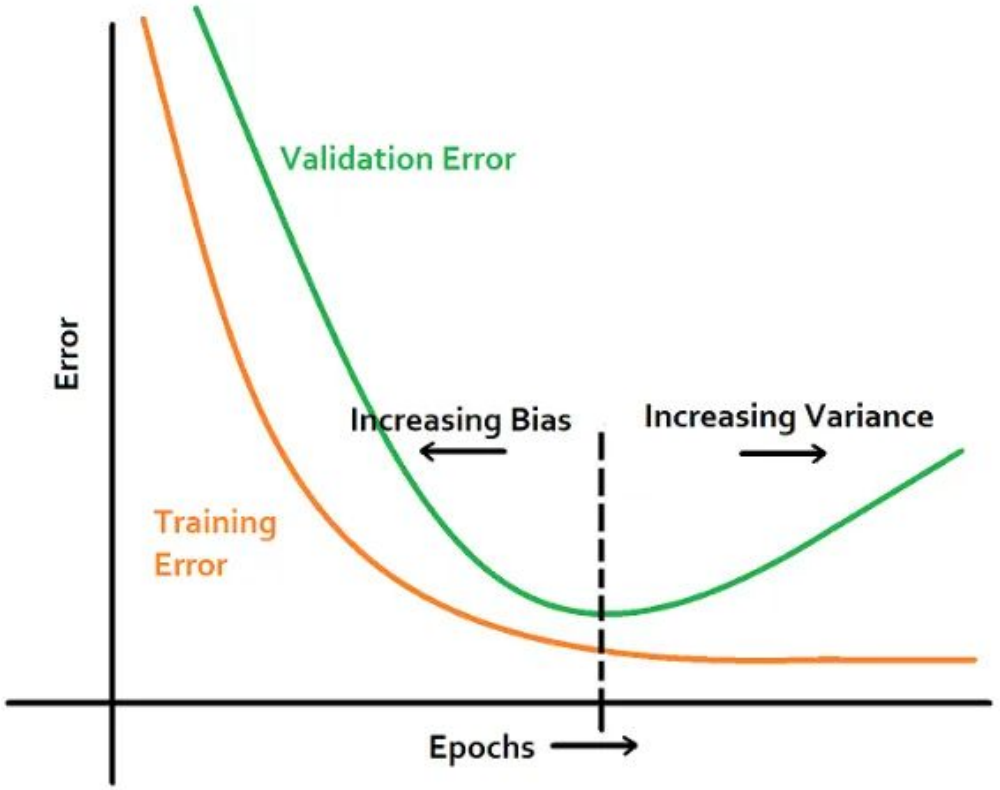
$$Cost\ Function = loss + \lambda||w||_1$$

λ is a hyperparameter decides the strength of the penalization.

Regularization: L^1 and L^2 Regularization



Regularization: Early Stopping



Regularization: Early Stopping

- When training large models with **sufficient representational capacity** to overfit the task, we often observe that **training error decreases steadily over time, but validation set error begins to rise again.**
- This means we can obtain a model with better validation set error (and thus, hopefully better test set error) by returning to the parameter setting at the point in time with the lowest validation set error.
- Under the early stopping strategy, we would terminates the algorithm and return the trained parameters **when no parameters have improved over the best recorded validation error for some pre-specified number of iterations.**

Regularization: Data Augmentation

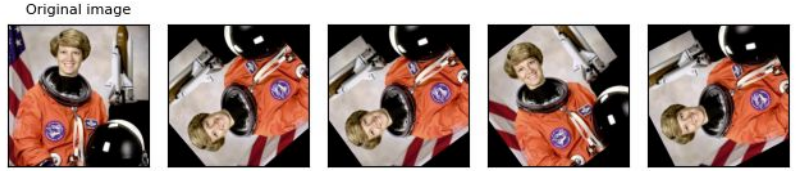
- Under data augmentation we artificially **expand the training dataset** size by different techniques.
- In this way, it has been **targeted to avoid overfitting by training the data on a large amount of data with several varieties.**

See available options in PyTorch:
https://pytorch.org/vision/0.15/auto_examples/plot_transforms.html#sphx-glr-auto-examples-plot-transforms-py

Resize

Center Crop

Random Rotation



Regularization: Batch Normalization

- Normalization is a pre-processing technique used to **standardize data.** In other words, **having different sources of data inside the same range.**
- Batch Norm is a normalization technique **done between the layers of a Neural Network instead of in the raw data.** It is done along mini-batches instead of the full data set.
- Batch normalization helps to:
  - Speed up training.
  - Use higher learning rates.
  - Make learning process easier.

m_z and s_z are mean and standard deviation respectively for neuron's output

$$z^N = \left(\frac{z - m_z}{s_z} \right)$$

CNN: Applications & Associated Architectures

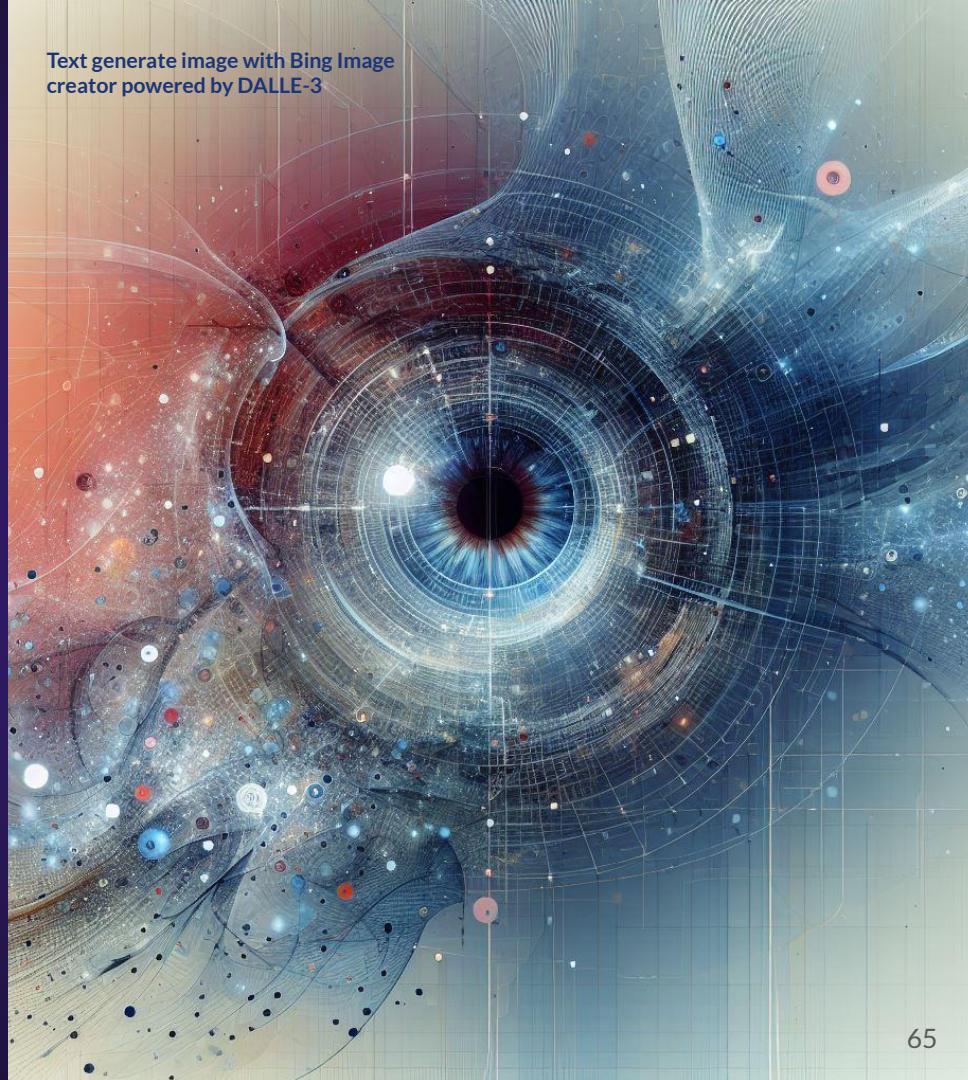
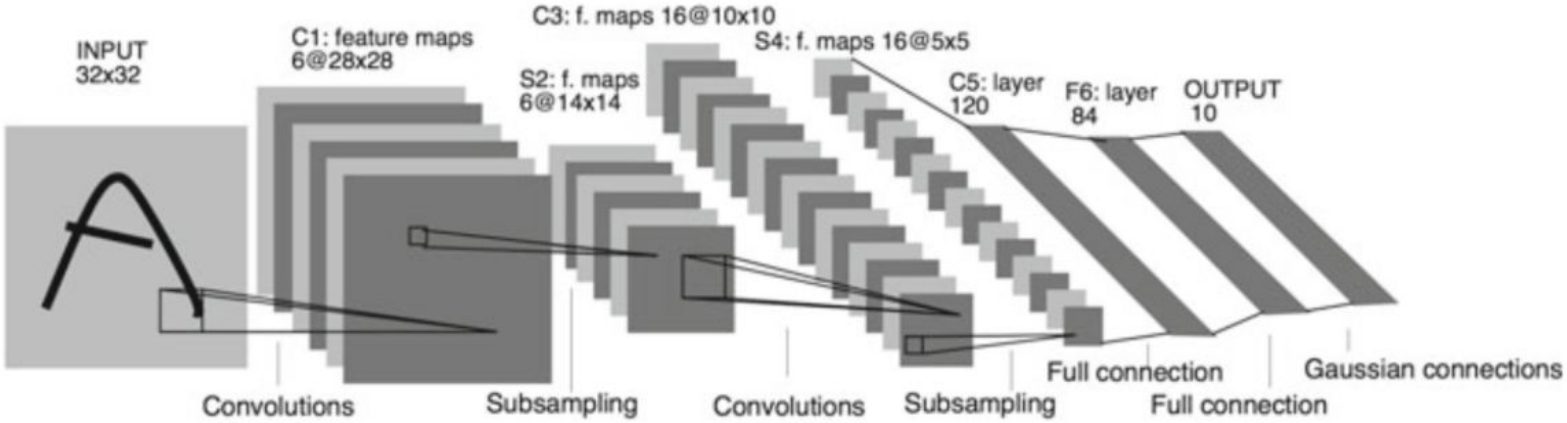
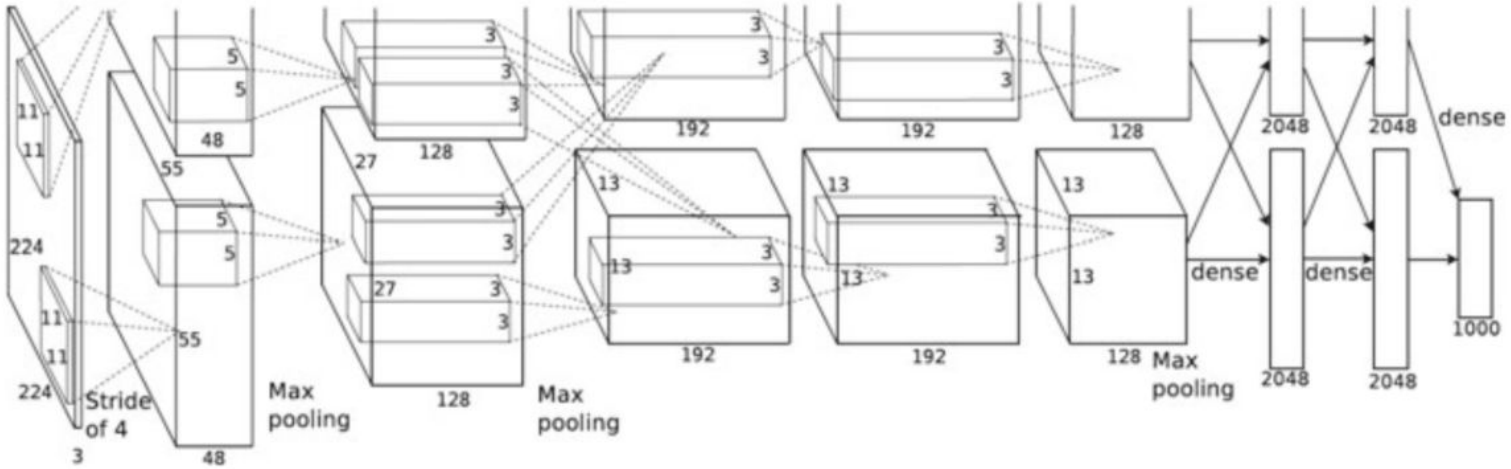


Image Classification



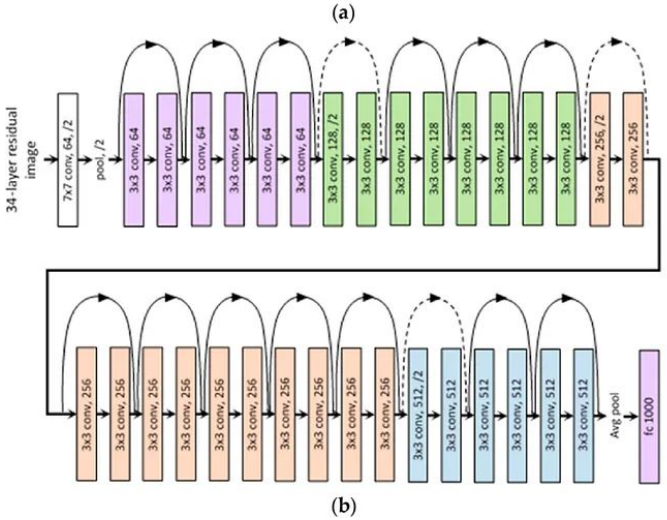
LeNet-5

Image Classification



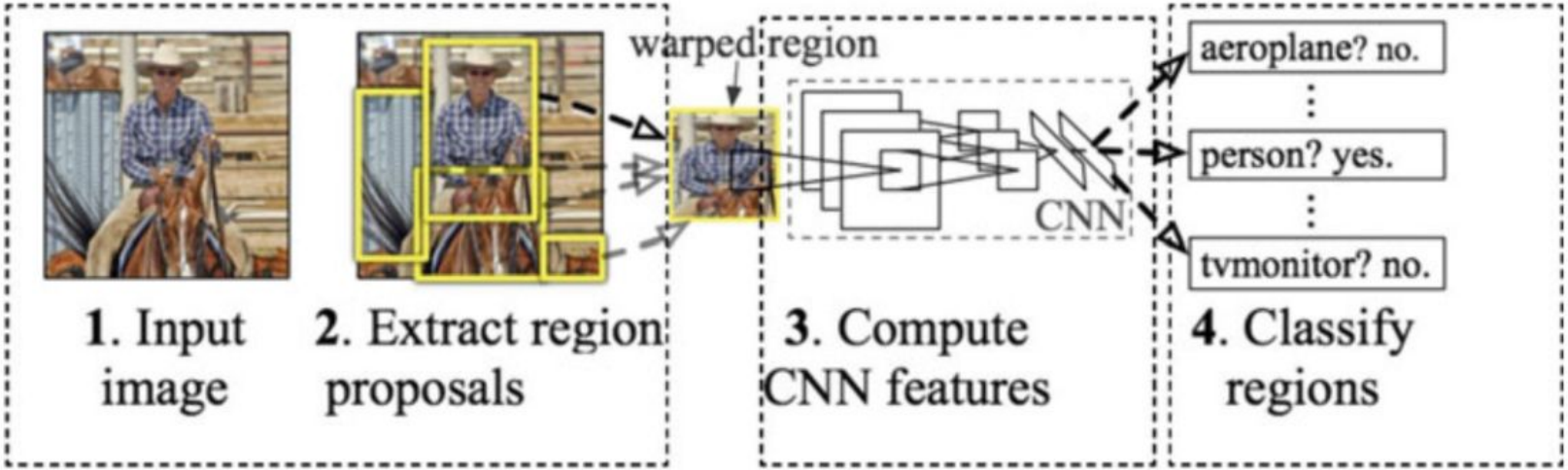
AlexNet

Image Classification



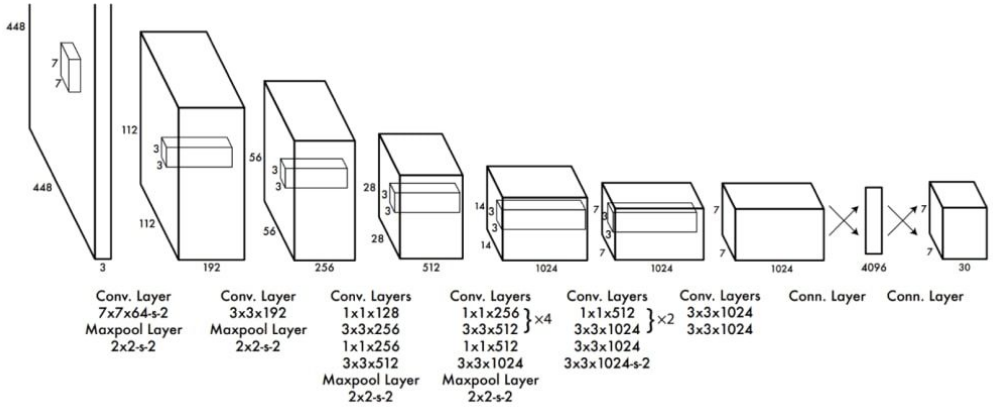
ResNet-34

Object Detection



R-CNN (Region-Based CNN)

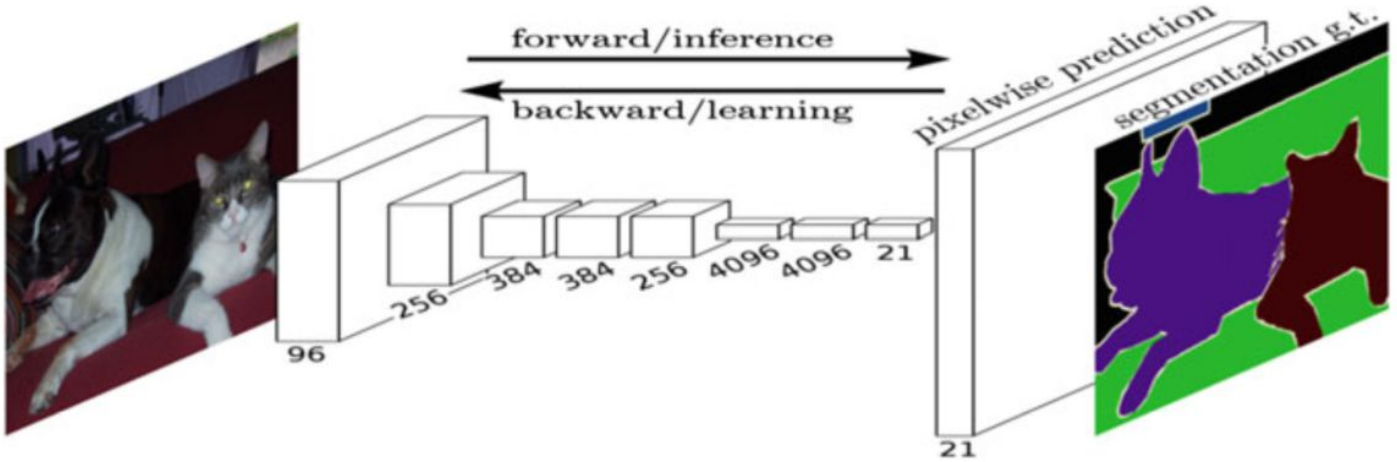
Object Detection



The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

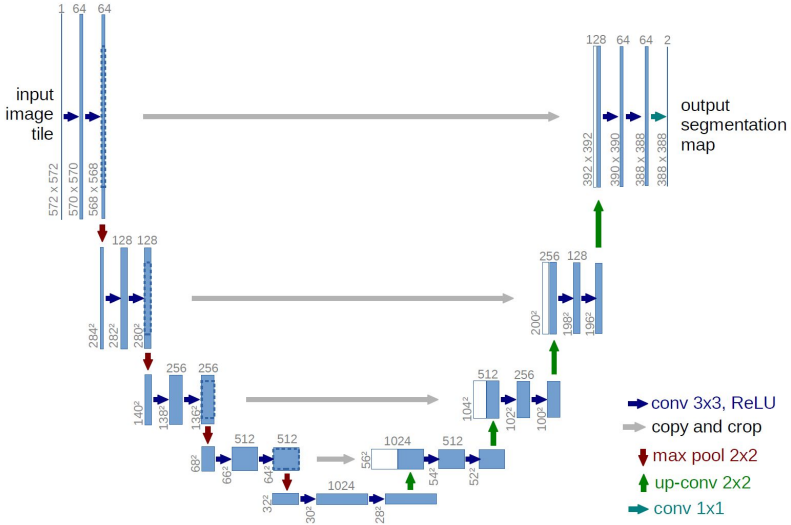
YOLO

Image Segmentation



End-to-end model of Fully Convolutional Network (FCN)

Image Segmentation



U-Net

Explore other CNN applications and associated architectures

References

1. I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, Massachusetts: The Mit Press, 2016.
2. S. Saha, “A Comprehensive Guide to Convolutional Neural Networks—the ELI5 way,” Towards Data Science, Dec. 15, 2018. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
3. M. Mandal, “CNN for Deep Learning | Convolutional Neural Networks (CNN),” Analytics Vidhya, May 01, 2021. https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/
4. F. Bre, J. M. Gimenez, and V. D. Fachinotti, “Prediction of wind pressure coefficients on building surfaces using artificial neural networks,” Energy and Buildings, vol. 158, pp. 1429–1441, Jan. 2018, doi: https://doi.org/10.1016/j.enbuild.2017.11.045.
5. A. Ghosh, A. Sufian, F. Sultana, A. Chakrabarti, and D. De, “Fundamental Concepts of Convolutional Neural Network,” Intelligent Systems Reference Library, vol. 172, pp. 519–567, Nov. 2019, doi: https://doi.org/10.1007/978-3-030-32644-9_36.
6. IBM, “What are Convolutional Neural Networks? | IBM,” www.ibm.com, 2023. https://www.ibm.com/topics/convolutional-neural-networks
7.]R. S, “Convolution Neural Networks (CNN),” Medium, Apr. 16, 2019. https://medium.com/@rathna211994/convolution-neural-networks-cnn-b6fe90214b1e (accessed Mar. 05, 2024).
8. “CNN | Introduction to Padding,” GeeksforGeeks, Jul. 23, 2019. https://www.geeksforgeeks.org/cnn-introduction-to-padding/
9. P. Antoniadis, “Calculate the Output Size of a Convolutional Layer | Baeldung on Computer Science,” www.baeldung.com, Feb. 11, 2022. https://www.baeldung.com/cs/convolutional-layer-size
10. datahacker.rs, “#007 CNN One Layer of A ConvNet,” Master Data Science, Nov. 07, 2018. https://datahacker.rs/one-layer-covolutional-neural-network/
11. S. SHARMA, “Activation Functions in Neural Networks,” Medium, Sep. 06, 2017. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
12. “Top 10 Activation Function’s Advantages & Disadvantages,” www.linkedin.com. https://www.linkedin.com/pulse/top-10-activation-functions-advantages-disadvantages-dash.
13. “Vanishing Gradient: 기울기 소실,” Velog.io, 2022. https://velog.io/@yunyoseob/Gradient-Vanishing-%EA%B8%B0%EC%9A%B8%EA%B8%B0-%EC%86%8C%EC%8B%A4 (accessed Mar. 07, 2024).

References

14. T. Jacob, “Vanishing Gradient Problem, Explained,” KDnuggets, Feb. 25, 2022. https://www.kdnuggets.com/2022/02/vanishing-gradient-problem.html.
15. “Vanishing gradient problem,” Engati. https://www.engati.com/glossary/vanishing-gradient-problem#:~:text=Vanishing%20gradient%20problem%20is%20a
16. D. Nautiyal, “Underfitting and Overfitting in Machine Learning,” GeeksforGeeks, Nov. 23, 2017. https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/
17. “Regularization of Neural Networks Using DropConnect,” NYU Center for Data Science. https://cds.nyu.edu/projects/regularization-neural-networks-using-dropconnect/#:~:text=We%20introduce%20DropConnect%2C%20a%20generalization (accessed Mar. 16, 2024).
18. M. Chablani, “L1 regularization and L2 regularization intuitive explanation,” Medium, Sep. 30, 2023. https://medium.com/@ManishChablani/l1-regularization-and-l2-regularization-intuitive-explanation-229157a64dc3#:~:text=Starting%20with%20definition%3A (accessed Mar. 16, 2024).
19. M. Riva, “Batch Normalization in Convolutional Neural Networks | Baeldung on Computer Science,” www.baeldung.com, Oct. 29, 2020. https://www.baeldung.com/cs/batch-normalization-cnn
20. S. Bangar, “Resnet Architecture Explained,” Medium, Jul. 05, 2022. https://medium.com/@siddheshb008/resnet-architecture-explained-47309ea9283d
21. R. Kundu, “YOLO: Real-Time Object Detection Explained,” www.v7labs.com, Jan. 17, 2023. https://www.v7labs.com/blog/yolo-object-detection.
22. “U-Net: Convolutional Networks for Biomedical Image Segmentation,” Uni-freiburg.de, 2015. https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/
23. “Pooling In Convolutional Neural Networks,” Paperspace Blog, Oct. 08, 2022. https://blog.paperspace.com/pooling-in-convolutional-neural-networks/
24. M. Basavarajaiah, “Which pooling method is better? Maxpooling vs minpooling vs average pooling,” Medium, Aug. 22, 2019. https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-pooling-95fb03f45a9
25. S. Mukherjee, “Regularization and Gradient Descent Cheat Sheet,” The Startup, Jan. 19, 2021. https://medium.com/swlh/regularization-and-gradient-descent-cheat-sheet-d1be74a4ee53
26. R. JAIN, “Why ‘early-stopping’ works as Regularization?,” Medium, Feb. 08, 2020. https://medium.com/@rahuljain13101999/why-early-stopping-works-as-regularization-b9f0a6c2772

Thank You