Convolutional Neural Networks (CNNs / ConvNets)

1. Introduction to CNNs

- **What are they?** Convolutional Neural Networks (CNNs or ConvNets) are a class of deep neural networks, most commonly applied to analyzing visual imagery. They are inspired by the biological processes in the animal visual cortex.
- **Primary Applications:**
    - Image Classification (e.g., identifying objects in pictures)
    - Object Detection (e.g., locating objects within an image)
    - Image Segmentation (e.g., partitioning an image into regions)
    - Facial Recognition
    - Video Analysis
    - Natural Language Processing (NLP) for tasks like text classification (1D CNNs)
    - Medical Image Analysis

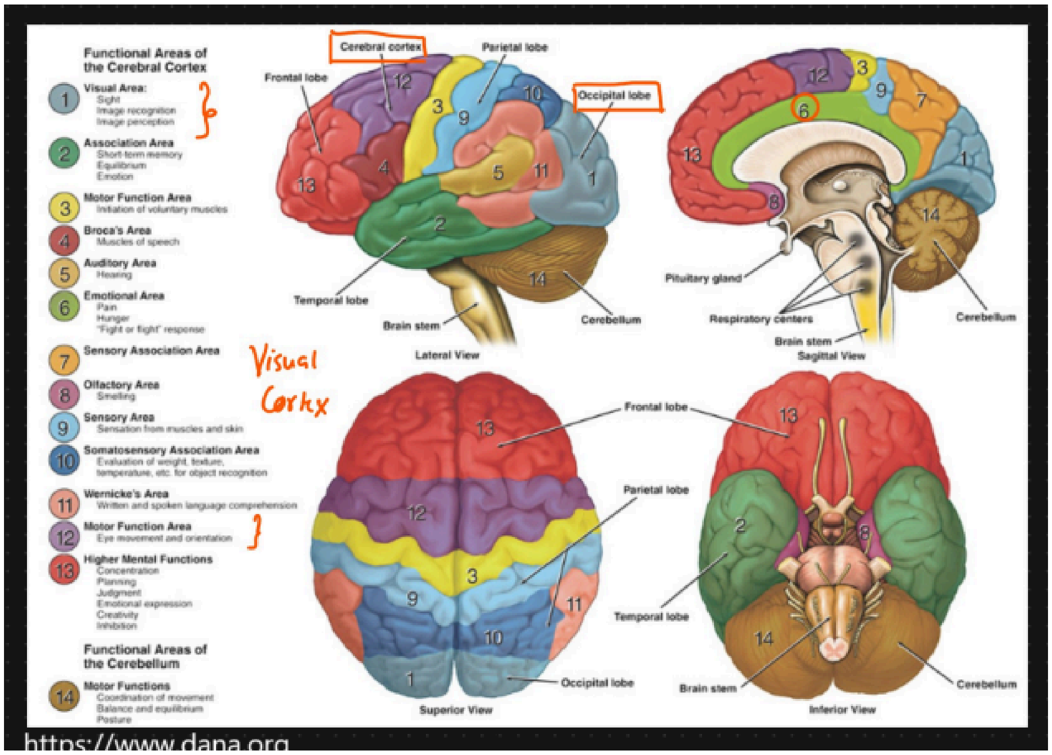
2. Why CNNs for Images (Limitations of Standard NNs/MLPs)?

Standard Neural Networks (Multi-Layer Perceptrons - MLPs) have limitations when dealing with image data:

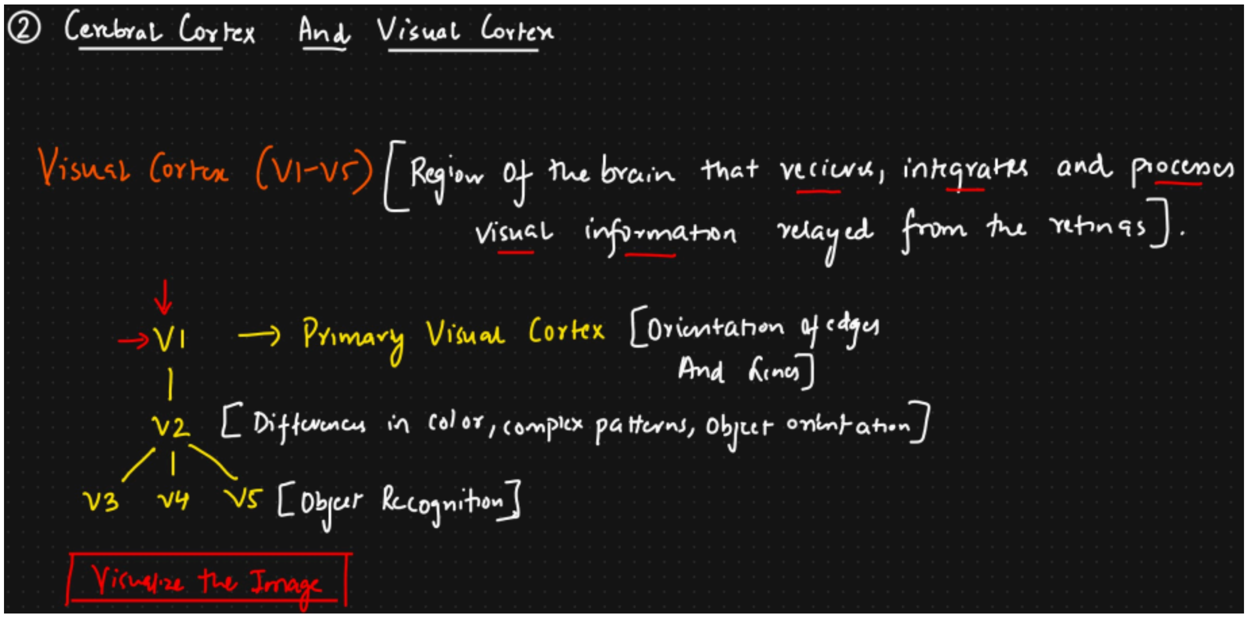
- **High Dimensionality:** Images have a large number of pixels. For example, a 224x224 RGB image has $224 \times 224 \times 3 = 150{,}528$ input features. An MLP would require a huge number of parameters in the first hidden layer, leading to:
    - High computational cost.
    - High memory requirement.
    - High risk of overfitting.
- **Loss of Spatial Information:** MLPs treat input features as a flat vector, losing the spatial relationships between pixels (e.g., how pixels are arranged to form shapes, edges, etc.).
- **Not Robust to Variations:** MLPs are not inherently robust to variations like translation, rotation, or scaling of objects in an image.

CNNs address these issues through:

- **Local Connectivity:** Neurons in a layer are only connected to a small region of the layer before it (receptive field).
- **Parameter (Weight) Sharing:** The same set of weights (a filter or kernel) is used across different locations in the input image, significantly reducing the number of parameters.
- **Hierarchical Feature Learning:** They learn a hierarchy of features, from simple edges and textures in early layers to more complex patterns and objects in deeper layers.


Functional Areas of the Cerebral Cortex
1 Visual Area: Sight, Image recognition, Image perception
2 Association Area: Short-term memory, Equilibrium, Emotion
3 Motor Function Area: Initiation of voluntary muscles
4 Broca's Area: Muscles of speech
5 Auditory Area: Hearing
6 Emotional Area: Pain, Hunger, "Fight or flight" response
7 Sensory Association Area
8 Olfactory Area: Smelling
9 Sensory Area: Sensation from muscles and skin
10 Somatosensory Association Area: Evaluation of weight, texture, temperature, etc. for object recognition
11 Wernicke's Area: Written and spoken language comprehension
12 Motor Function Area: Eye movement and orientation
13 Higher Mental Functions: Concentration, Planning, Judgment, Emotional expression, Creativity, Inhibition
Functional Areas of the Cerebellum
14 Motor Functions: Coordination of movement, Balance and equilibrium, Posture
Cerebral cortex
Parietal lobe
Frontal lobe
Occipital lobe
Temporal lobe
Brain stem
Cerebellum
Lateral View
Pituitary gland
Respiratory centers
Sagittal View
Visual Cortex
Superior View
Inferior View

https://www.dana.org

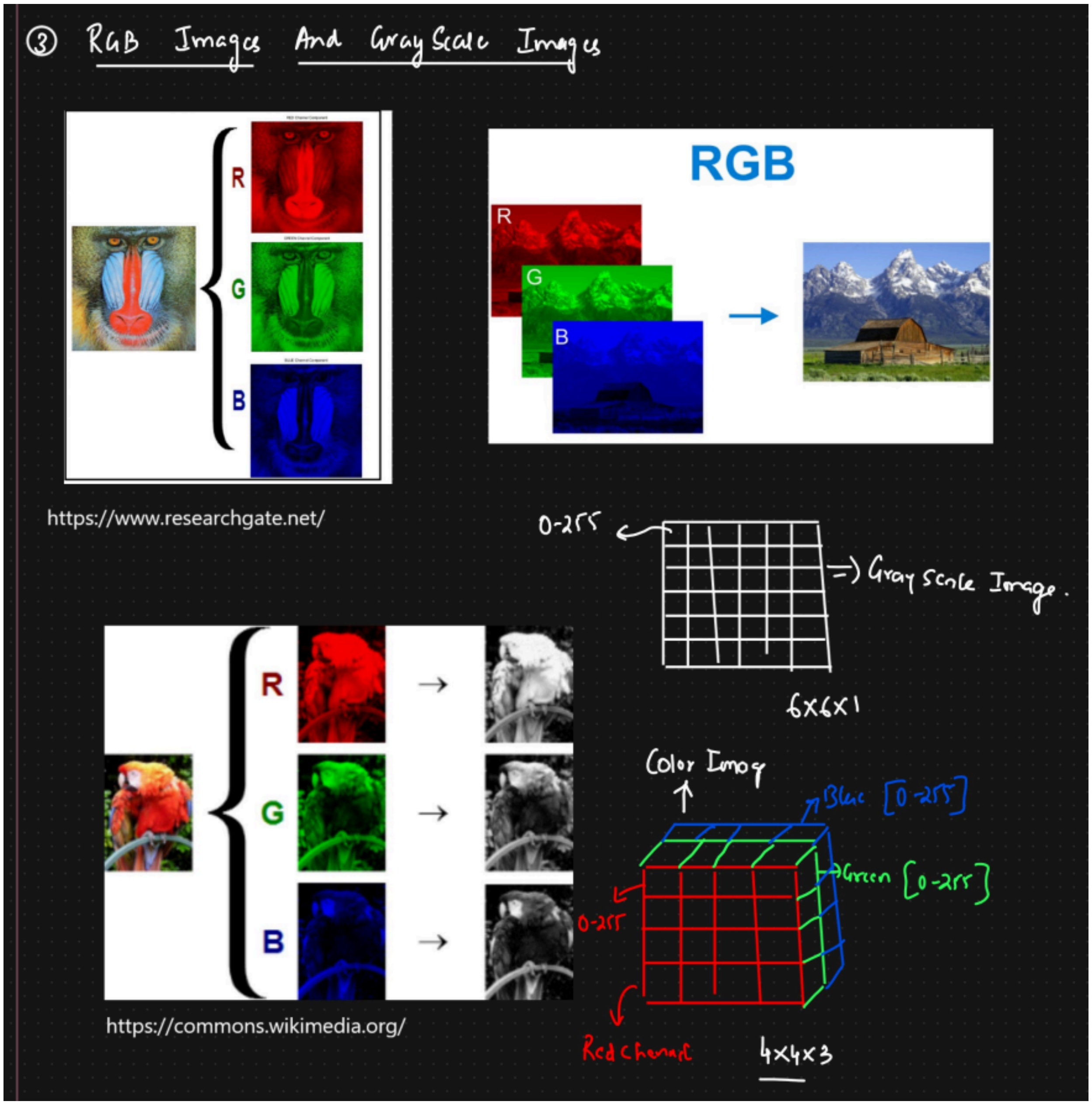


② Cerebral Cortex And Visual Cortex

Visual Cortex (V1-V5) [Region of the brain that recieves, integrates and processes visual information relayed from the retinas].

$$\begin{tikzcd}
& {} \arrow[d] & & \\
{} \arrow[r] & V1 \arrow[r] \arrow[d, dash] & \text{Primary Visual Cortex [Orientation of edges And lines]} & \\
& V2 \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \text{[Difference in color, complex patterns, Object orientation]} & \\
V3 & V4 & V5 \text{ [Object Recognition]} &
\end{tikzcd}$$

$\boxed{\text{Visualize the Image}}$

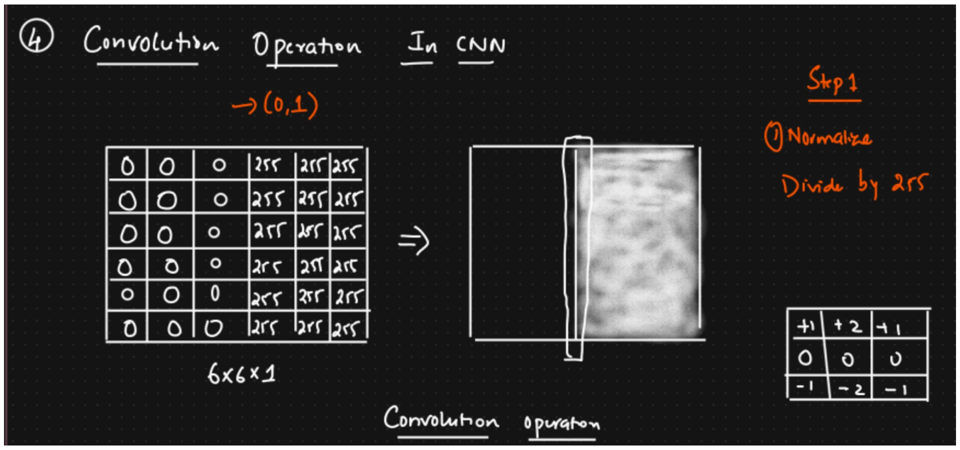


3. Core Components/Layers of a CNN

A CNN typically consists of three main types of layers: Convolutional Layers, Pooling Layers, and Fully Connected Layers.

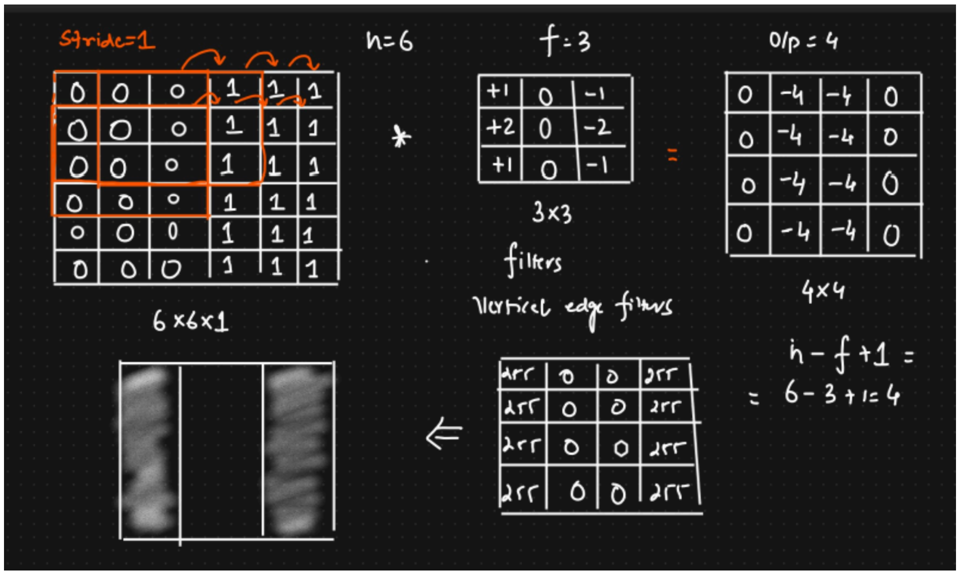
a. Convolutional Layer

- **Purpose:** To extract features (e.g., edges, corners, textures) from the input image or feature maps from previous layers.
- **Key Concepts:**

- **Filters/Kernels:** Small matrices of learnable weights that slide (convolve) over the input volume. Each filter is designed to detect a specific feature. For example, a filter might detect vertical edges, another horizontal edges, etc.
  - The depth of the filter matches the depth of the input volume.
  - 

0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255

+1	+2	+1
0	0	0
-1	-2	-1

  - 

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

$*$

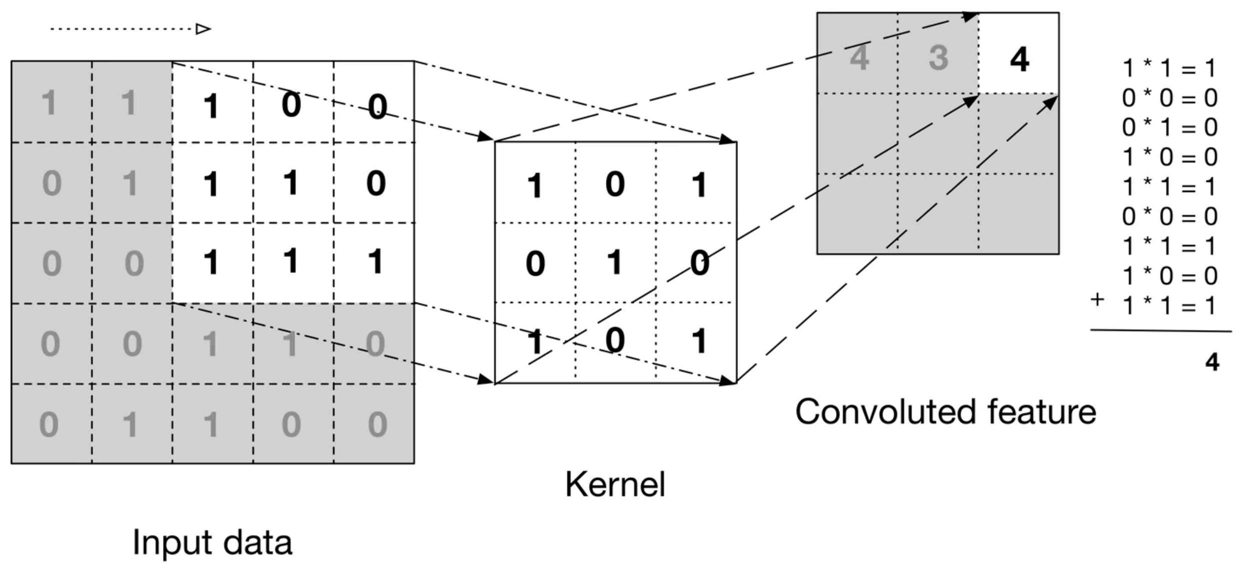
+1	0	-1
+2	0	-2
+1	0	-1

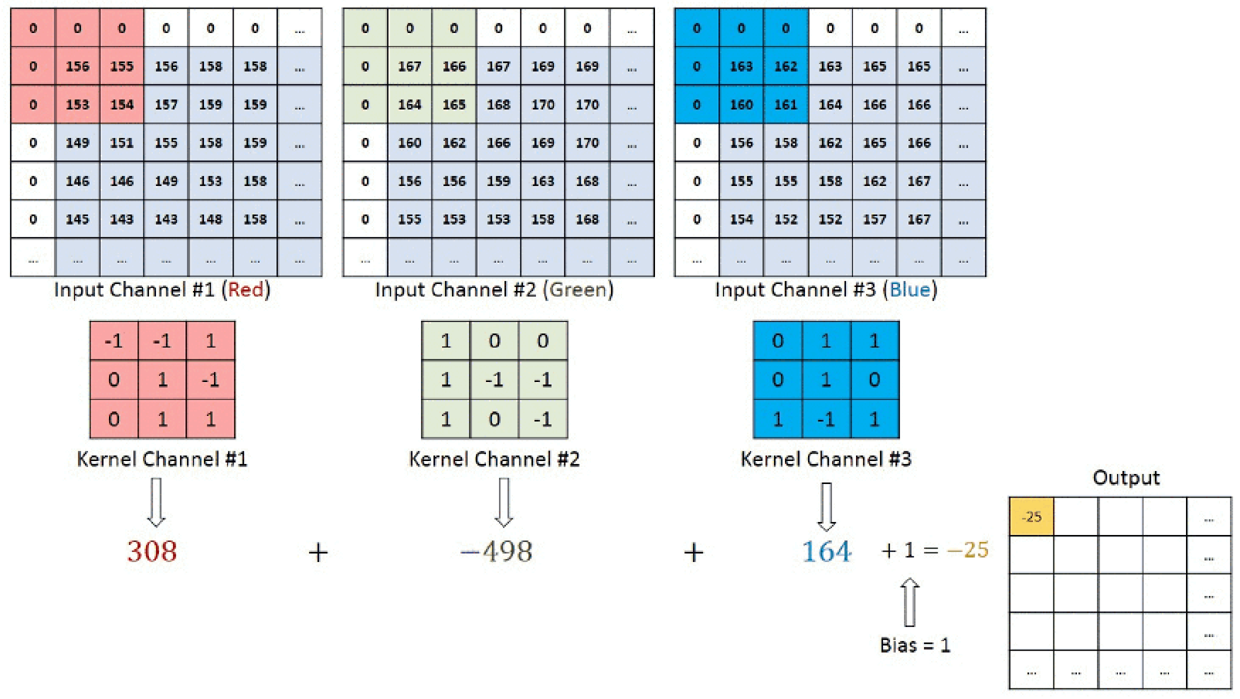
$=$

0	-4	-4	0
0	-4	-4	0
0	-4	-4	0
0	-4	-4	0

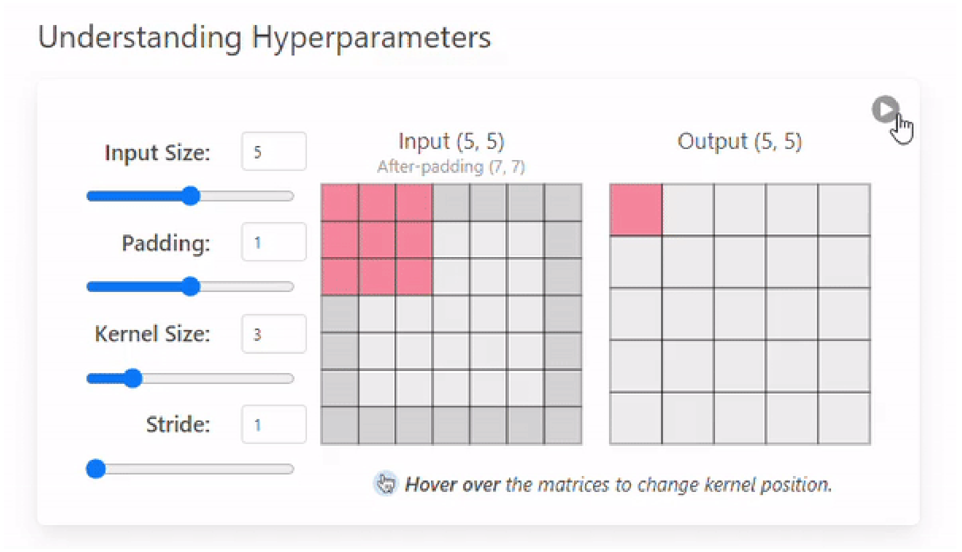
$$n - f + 1 = 6 - 3 + 1 = 4$$

255	0	0	255
255	0	0	255
255	0	0	255
255	0	0	255





- **Convolution Operation:** The element-wise product between the filter and a local region of the input, summed up to produce a single value in the output feature map. (f∗g)(i,j)=m∑n∑f(m,n)g(i−m,j−n) (In practice, deep learning libraries often implement cross-correlation but still call it convolution.)
- **Feature Map (Activation Map):** The output of a filter after convolving over the input. It represents the presence and strength of the detected feature at different spatial locations. A convolutional layer typically learns multiple feature maps, each corresponding to a different filter.
- **Receptive Field:** The region in the input volume that a particular neuron in the convolutional layer is looking at.

- **Parameters:**
    - **Number of Filters (Depth of Output Volume):** Determines how many different features are learned at this stage.
    - **Filter Size (Kernel Size):** Dimensions of the filter (e.g., 3x3, 5x5). Smaller sizes are common in modern architectures.
    - **Stride:** The number of pixels by which the filter slides over the input volume.
        - Stride 1: Filter moves one pixel at a time.
        - Stride > 1: Results in downsampling of the feature map.
    - **Padding:** Adding pixels (usually zeros) around the border of the input volume.
        - **Valid Padding (No Padding):** The output feature map dimensions will be smaller than the input.
        - **Same Padding (Zero Padding):** The output feature map will have the same spatial dimensions as the input (for stride 1). This helps preserve information at the borders.
        - 
          Understanding Hyperparameters

          Input Size: 5

          Padding: 1

          Kernel Size: 3

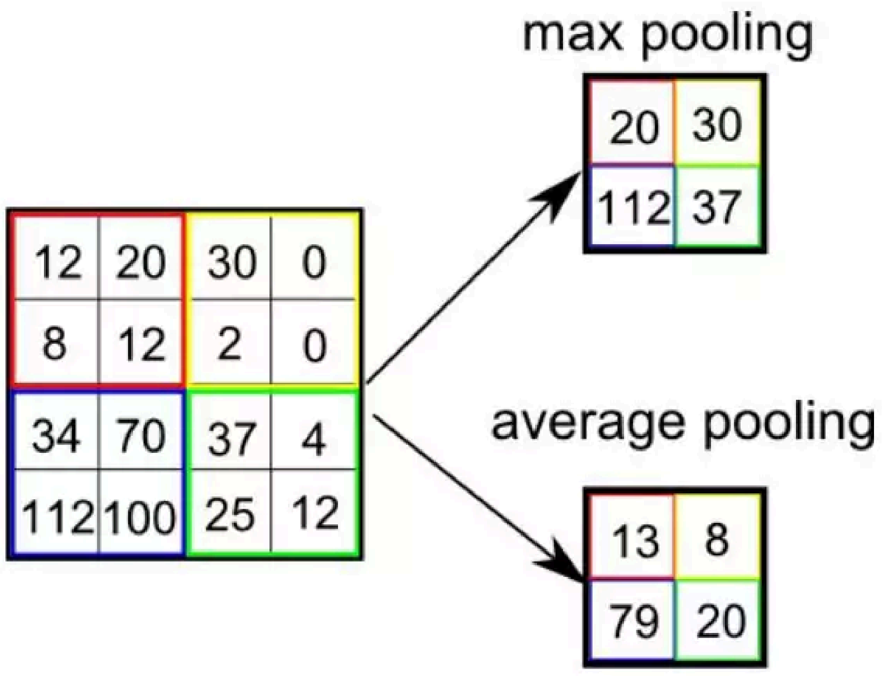
          Stride: 1

- **Activation Function:** Typically, a non-linear activation function (e.g., ReLU - Rectified Linear Unit) is applied element-wise to the feature map after the convolution operation to introduce non-linearity.
- **Shared Weights:** The core idea that the weights of a single filter are used across all spatial locations of the input. This:
    - Drastically reduces the number of learnable parameters.
    - Makes the network equivariant to the translation of features (if a feature is useful in one part of the image, it's likely useful in another).

b. Pooling Layer (Subsampling Layer)

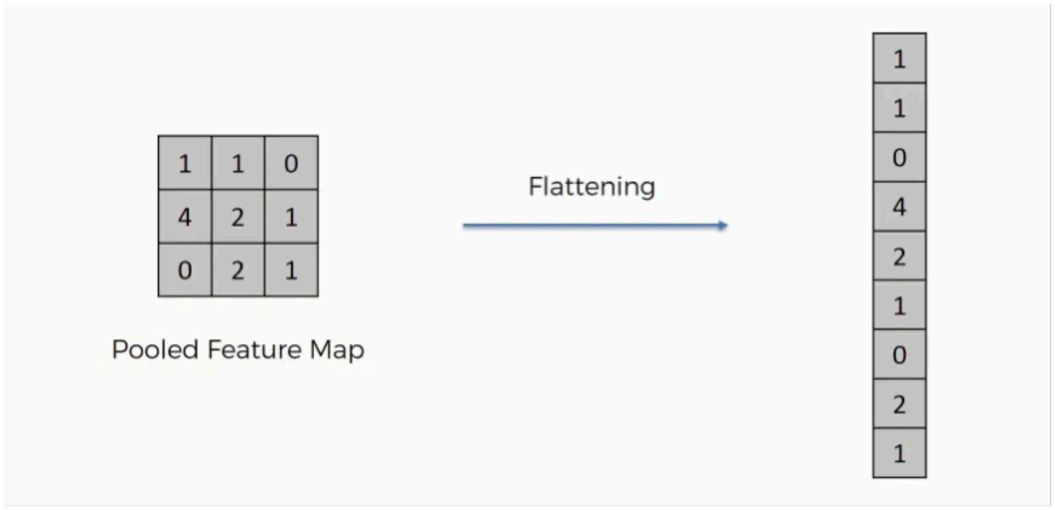
- **Purpose:**
    - **Dimensionality Reduction:** Reduces the spatial dimensions (width and height) of the feature maps, leading to fewer parameters and computations in subsequent layers.

    - **Translation Invariance (Local):** Makes the representations somewhat invariant to small translations in the input. If a feature shifts slightly, the pooled output is likely to remain similar.
- **Common Types:**
    - **Max Pooling:** Selects the maximum value from the pooling window (a small rectangular region) of the feature map. It's the most common type as it often performs better by preserving the strongest activation for a feature.
    - **Average Pooling:** Calculates the average value from the pooling window.
- **Parameters:**
    - **Pool Size (Window Size):** The size of the window over which to pool (e.g., 2x2).
    - **Stride:** The number of pixels by which the pooling window slides. Often, stride is equal to pool size (non-overlapping windows).
- **Note:** Pooling layers operate independently on each feature map (i.e., on each depth slice). They do not have learnable parameters.



c. Fully Connected Layer (Dense Layer)

- **Purpose:** To perform high-level reasoning and classification based on the features extracted by the convolutional and pooling layers.
- **How it works:**
    - The output feature maps from the final convolutional or pooling layer are **flattened** into a 1D vector.
    - This vector then serves as input to one or more fully connected layers, similar to those in a standard MLP.
    - Each neuron in a fully connected layer is connected to all neurons in the previous layer.

- 

- **Activation:**
  - Hidden FC layers often use ReLU.
  - The **final output layer** typically uses:
    - **Softmax:** For multi-class classification (outputs probabilities for each class).
    - **Sigmoid:** For binary classification (outputs a probability between 0 and 1).
    - Linear: For regression tasks.

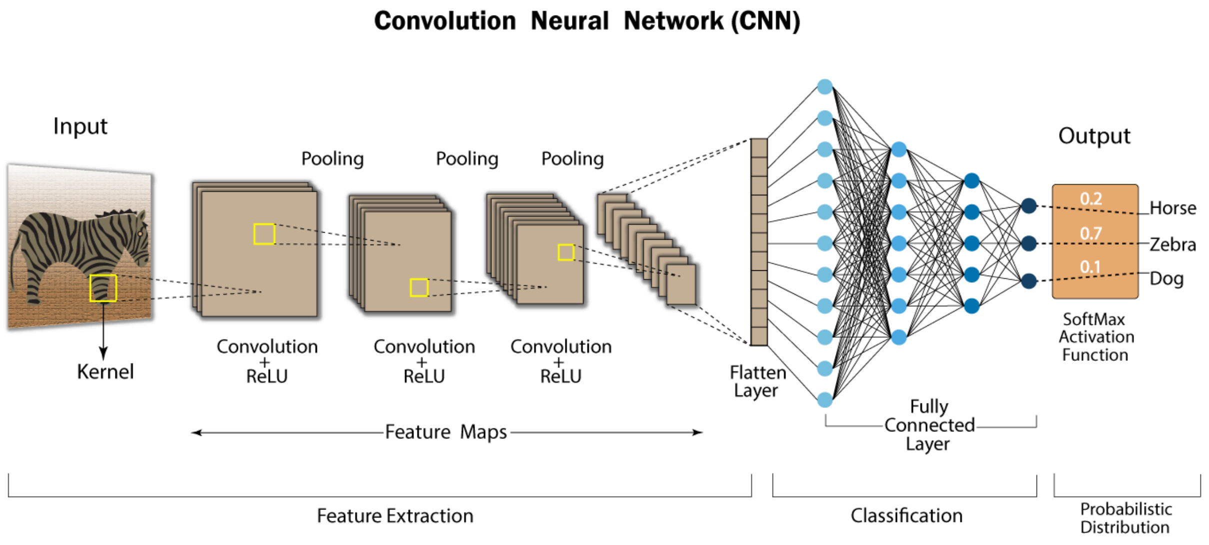
d. Activation Functions (Reiteration)

- **Purpose:** Introduce non-linearity into the model, allowing it to learn complex relationships.
- **Common choices:**
  - **ReLU (Rectified Linear Unit):** f(x)=max(0,x). Most popular due to its simplicity and ability to mitigate vanishing gradient problem.
  - **Leaky ReLU, Parametric ReLU (PReLU), Exponential ReLU (ELU):** Variants of ReLU addressing the "dying ReLU" problem.
  - **Sigmoid:** f(x)=1+e−x1. Squashes values between 0 and 1. Used in output layers for binary classification. Prone to vanishing gradients in deep networks.
  - **Tanh (Hyperbolic Tangent):** f(x)=ex+e−xex−e−x. Squashes values between -1 and 1. Also prone to vanishing gradients.
  - **Softmax:** Converts a vector of K real numbers into a probability distribution of K possible outcomes. Used in the output layer for multi-class classification.

e. Dropout Layer

- **Purpose:** A regularization technique to prevent overfitting.
- **How it works:** During training, randomly "drops out" (sets to zero) a fraction of neurons (and their connections) in a layer for each training batch.
- This forces the network to learn more robust features that are not overly reliant on any single neuron.

- During testing/inference, all neurons are used, but their outputs are scaled down by the dropout rate.



4. Basic CNN Architecture (Example)

A common CNN architecture stacks these layers:

```
INPUT -> [CONV -> ReLU -> POOL]*N -> [FC -> ReLU]*M -> FC
(Softmax/Sigmoid/Linear)
```

Where:

- `*N` and `*M` indicate repeating the block N or M times.
- Early layers (CONV + POOL) are feature extractors.
- Later layers (FC) are classifiers or regressors.

Example Flow:

1. **Input Image:** (e.g., 32x32x3 pixels)
2. **CONV1:** 16 filters of size 3x3, stride 1, same padding. Output: 32x32x16
3. **ReLU1:** Activation. Output: 32x32x16
4. **POOL1:** Max pooling with 2x2 window, stride 2. Output: 16x16x16
5. **CONV2:** 32 filters of size 3x3, stride 1, same padding. Output: 16x16x32
6. **ReLU2:** Activation. Output: 16x16x32
7. **POOL2:** Max pooling with 2x2 window, stride 2. Output: 8x8x32
8. **Flatten:** Convert 8x8x32 feature map into a vector of 8×8×32=2048 elements.
9. **FC1 (Hidden):** 128 neurons, ReLU activation. Output: 128
10. **Dropout:** (Optional)

11. **FC2 (Output):** `num_classes` neurons, Softmax activation (for classification). Output: `num_classes` (probabilities)

5. Key Concepts in CNNs

- **Parameter Sharing:** As mentioned, filters are shared across the input. This drastically reduces the number of parameters compared to an MLP, making CNNs more efficient and less prone to overfitting with limited data.
- **Local Connectivity / Receptive Fields:** Each neuron in a convolutional layer processes information only from a local region of the input. This allows the network to focus on local features before aggregating them into more global ones.
- **Hierarchical Feature Learning:** Early layers learn low-level features (edges, corners, textures). Subsequent layers combine these to learn mid-level features (parts of objects, simple shapes). Deeper layers learn high-level features (entire objects or complex patterns).
- **Translation Equivariance / Invariance:**
  - **Equivariance (Convolution):** If the input shifts, the feature map also shifts by the same amount. The features detected are the same, just in a different location.
  - **Invariance (Pooling):** Pooling helps to make the representation somewhat invariant to small translations. If a feature is detected and then slightly shifted, the max/average pooling operation over that region is likely to produce a similar output. This means the network can recognize an object even if its position changes slightly in the image.

6. Training CNNs

- **Backpropagation:** The standard algorithm for training neural networks. It involves:
  - **Forward Pass:** Input data is fed through the network to compute the output and the loss.
  - **Backward Pass:** Gradients of the loss function with respect to the network's weights are computed by propagating errors backward through the network.
- **Gradient Descent (and its variants):** Optimization algorithms used to update the network weights based on the computed gradients.
  - **Stochastic Gradient Descent (SGD):** Updates weights using one training example (or a small batch) at a time.
  - **Adam, RMSprop, Adagrad:** More advanced optimizers that adapt the learning rate for each parameter, often leading to faster convergence.
- **Loss Functions (Cost Functions):** Measure the difference between the network's predictions and the true labels.
  - **Cross-Entropy Loss (Categorical Cross-Entropy):** Commonly used for multi-class classification.
  - **Binary Cross-Entropy Loss:** Used for binary classification.
  - **Mean Squared Error (MSE):** Commonly used for regression tasks.

- **Batch Normalization (Often used):** Normalizes the activations of a layer across the current batch. This can help stabilize training, allow for higher learning rates, and act as a regularizer. It's typically applied before the activation function in a convolutional or fully connected layer.

7. Advantages of CNNs

- **Excellent performance on visual tasks:** State-of-the-art results in image recognition, object detection, etc.
- **Automatic Feature Extraction:** Learns relevant features directly from the data, eliminating the need for manual feature engineering.
- **Parameter Sharing & Local Connections:** Leads to fewer parameters than MLPs for image data, making them more computationally efficient and less prone to overfitting.
- **Robustness to some variations:** Achieves a degree of translation invariance due to pooling and shared weights.

8. Popular CNN Architectures (Brief Mention - Good for further reading)

- **LeNet-5 (1998):** One of the earliest successful CNNs, used for handwritten digit recognition.
- **AlexNet (2012):** Revolutionized the field by winning the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Deeper and wider than LeNet. Used ReLU.
- **VGGNets (2014):** Showed that depth is critical. Used very small (3x3) convolutional filters stacked on top of each other. Simple and uniform architecture.
- **GoogLeNet / Inception (2014):** Introduced "Inception modules" which run convolutions of different sizes in parallel and concatenate their outputs. Focused on computational efficiency.
- **ResNet (Residual Networks) (2015):** Introduced "residual connections" or "skip connections" to allow training of extremely deep networks (hundreds or even thousands of layers) by addressing the vanishing gradient problem in very deep networks.
- **DenseNet (Densely Connected Convolutional Networks) (2016):** Each layer is connected to every other layer in a feed-forward fashion. Encourages feature reuse and strengthens feature propagation.
- **EfficientNet (2019):** Uses a compound scaling method to systematically scale network depth, width, and resolution for better efficiency and accuracy.

https://medium.com/thedeephub/convolutional-neural-networks-a-comprehensive-guide-5cc0b5eae175