📌 Convolution

Convolution is an operation that applies a filter (kernel) to an input (like an image) to extract features such as edges, textures, or patterns.

✒️ Modes of Convolution:

- **Valid Convolution:**
  - No padding is added, so the output is smaller than the input.
  - The filter moves horizontally and vertically without going outside the input's borders.

    Example: Input size 5x5 convolved with a 3x3 filter results in a 3x3 output.

- **Same Convolution:**
  - Padding is added to keep the output size the same as the input.
  - The filter slides horizontally and vertically, including padded borders.

    Example: Input size 5x5 with a 3x3 filter and padding of 1 results in a 5x5 output.

- **Full Convolution:**
  - Padding is added so that every element of the input is visited by the filter, resulting in a larger output.
  - The filter starts outside the input boundary, moving horizontally and vertically over fully padded edges.

    Example: Input size 5x5 with a 3x3 filter produces a 7x7 output.

📌 Dropout layers:

A regularization technique that randomly drops a fraction of neurons during training to prevent overfitting.

Purpose:

- Prevents over-reliance on specific neurons.
- Improves model generalization.

📌 Batch Normalization Layer:

Normalizes the inputs of a layer to maintain a stable distribution of activations, speeding up training.

Purpose:

- Accelerates convergence and improves stability.
- Reduces sensitivity to initialization and allows higher learning rates.

📌 Pooling Layer:

Pooling layers reduce the spatial dimensions (height and width) of feature maps, retaining important features while minimizing computational complexity.

Types of Pooling Layer:

- **Max Pooling:** Selects the maximum value within a pool, highlighting the most prominent features.
- **Average Pooling:** Computes the average value within a pool, preserving background information.

Purpose:

- Reduces overfitting by downsampling.
- Increases computational efficiency and translation invariance.

The general formula to calculate the result of a single neuron in a layer is:

$$z = (w1*x1 + w2*x2 + ... + wn*xn) + b$$

Where:

z: The weighted sum of the inputs plus the bias. This is the value that will be passed to the activation function.

w1, w2, ..., wn: The weights connecting the neuron to the inputs from the previous layer.

x1, x2, ..., xn: The inputs from the previous layer.
b: The bias term.

In matrix form:

$$Z = W * X + B$$

Where:

Z = Weighted sum (before activation)

W = Weight matrix

X = Input vector

B = Bias vector

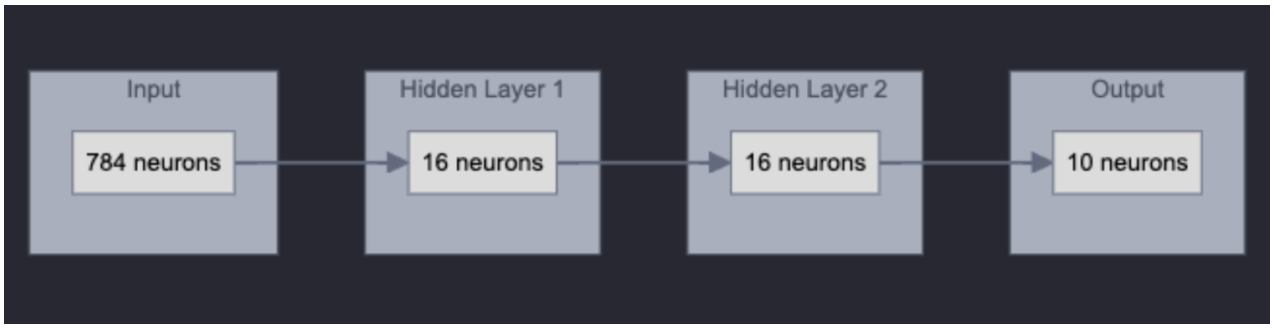
✒️ Calculation of Number of Parameters:

Consider a layer with:

1 input layer with 784 neurons (from an image of 28x28 pixels)

2 hidden layers with 16 neurons each

1 output layer with 10 neurons



Weights: (neurons of input layer x neurons of hidden layer 1 + neurons of hidden layer 1 x neurons of hidden layer 2 + neurons of hidden layer 2 x neurons of output layer)

= (784 x 16 + 16 x 16 + 16 x 10) = 12544 + 256 + 160 = 12960

Bias: (16 + 16 + 10) = 42

Total number of parameters : 12960 + 42 = 13002

✒️ weights and biases in a convolutional neural network (CNN) :

In CNN, we have **filters (which have weights) and biases.**

```
If you have a 3×3 filter:
- Weights: 9 parameters (3×3)
- Bias: 1 parameter
Total parameters per filter = 9 + 1 = 10 parameters
```

So if you have 32 filters in a convolutional layer:

- Total weights = 32 × (3×3) = 288
- Total biases = 32 × 1 = 32
- Total parameters = 288 + 32 = 320

Each filter shares a single bias term across all spatial locations.