1. **Activation Function**

* **Sigmoid**: The sigmoid function maps any real-valued number to the range (0, 1). It is often used as an activation function in the output layer for binary classification problems, where the output is interpreted as the probability of the positive class.
* Pros:
  + It is a smooth function that is easy to understand and compute.
  + It maps the input to a range between 0 and 1, making it useful for binary classification tasks.
* Cons:
  + It suffers from the vanishing gradient problem, which can cause the network to converge slowly or not at all.
  + It is not zero-centered, which can slow down learning in some cases.
* **ReLU** (Rectified Linear Unit): The ReLU function is defined as max(0, x). It is one of the most popular activation functions in deep learning because it is simple and computationally efficient. It has been shown to work well in many different types of neural networks.
* Pros:
  + It is simple and computationally efficient, making it popular in deep learning.
  + It overcomes the vanishing gradient problem and can accelerate the convergence of the network.
  + It has been shown to work well in many different types of neural networks.
* Cons:
  + It is not differentiable at x = 0, which can cause some optimization algorithms to fail.
  + It can be too "aggressive" and lead to dead neurons, where the output is always zero.
* **LeakyReLU**: The LeakyReLU function is similar to ReLU, but instead of being zero for negative inputs, it has a small negative slope. This can help prevent the "dying ReLU" problem, where ReLU neurons become permanently inactive and stop contributing to the output.
* Pros:
  + It overcomes the dead neuron problem by allowing a small gradient for negative inputs.
  + It has been shown to work well in many different types of neural networks.
* Cons:
  + It introduces another hyperparameter (the leak slope) that needs to be tuned.
  + It can be slower to compute than ReLU.
* **Softmax**: The softmax function is used in the output layer for multi-class classification problems, where the output is interpreted as the probability of each class. It maps a vector of real-valued numbers to a probability distribution, so that the sum of the probabilities is 1.
* Pros:
  + It maps the input to a probability distribution, making it useful for multi-class classification tasks.
  + It is differentiable, making it easy to use in optimization algorithms.
* Cons:
  + It can suffer from numerical stability issues when the inputs are large.
  + It can be slow to compute when the number of classes is large.
* **Tanh**: The tanh function maps any real-valued number to the range (-1, 1). It is similar to the sigmoid function, but it has a range that is symmetric around zero.
* Pros:
  + It is a smooth function that is easy to understand and compute.
  + It maps the input to a range between -1 and 1, making it useful for tasks where the output should be centered around zero.
* Cons:
  + It suffers from the vanishing gradient problem, which can cause the network to converge slowly or not at all.
  + It is not as popular as ReLU and its variants in deep learning.
* **Swish**: The swish function is a relatively new activation function that has been shown to work well in deep neural networks. It is defined as x \* sigmoid(x).
* Pros:
  + It has been shown to work well in deep neural networks.
  + It has a smooth curve that is similar to ReLU but with a non-zero derivative at negative inputs.
* Cons:
  + It introduces another hyperparameter (the beta parameter) that needs to be tuned.
  + It can be slower to compute than ReLU.
* **ELU** (Exponential Linear Unit): The ELU function is similar to ReLU, but it has a smooth exponential curve for negative inputs instead of being zero. This can help prevent the "dying ReLU"
* Pros:
  + It overcomes the dead neuron problem by having a smooth exponential curve for negative inputs.
  + It has been shown to work well in deep neural networks.
* Cons:
  + It introduces another hyperparameter (the alpha parameter) that needs to be tuned.
  + It can be slower to compute than ReLU.

1. **Padding**

* **Valid Padding:** This means no padding is added to the input edges. The output will therefore have smaller dimensions than the input because the convolution does not extend all the way to the edges
* **Same Padding:** input tensor is padded with zeros are the edges so that the output has the same spatial dimensions as the input