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

1. **Optimizer**

**\*model.compile()**

* **Stochastic Gradient Descent:** Classic optimization algorithm that updates model parameters in the direction of the gradient of the loss function, scaled by learning rate
* Pros:
  + Simple and easy to implement.
  + Computationally efficient, especially for small datasets.
* Cons:
  + SGD can get stuck in local minima or saddle points in high-dimensional spaces, which can make convergence slow or impossible.
  + Requires careful tuning of the learning rate to prevent divergence or slow convergence.
* **Adam:** An optimization algorithm that is similar to SGD, but uses adaptive learning rates for each parameter, which can improve performance on non-convex optimization problems.
* Pros:
  + Computes adaptive learning rates for each parameter, which can improve convergence on non-convex optimization problems.
  + Tends to converge faster than traditional optimization algorithms like SGD.
* Cons:
  + Can converge to suboptimal solutions if the learning rate or other hyperparameters are not chosen carefully.
  + Can be sensitive to the choice of the initial learning rate.
* **Adagrad**: an optimization algorithm that adapts the learning rate of each parameter based on the historical gradients of that parameter.
* Pros:
  + Adapts the learning rate of each parameter based on the historical gradients of that parameter, which can lead to good performance on sparse datasets.
* Cons:
  + Can accumulate gradients in the denominator of the learning rate update, which can cause the learning rate to shrink to zero.
  + Can require more memory than other optimization algorithms, since it needs to store the historical gradient for each parameter.
* **RMSProp**: An optimization algorithm that maintains a moving average of the squared gradient for each parameter, and adapts the learning rate based on the root mean squared of these values.
* Pros:
  + Maintains a moving average of the squared gradient for each parameter, which can help prevent the learning rate from decreasing too quickly.
  + Performs well on non-stationary problems, where the gradients can change rapidly over time.
* Cons:
  + Can be sensitive to the choice of hyperparameters, especially the decay rate used to compute the moving average.
  + Can suffer from similar issues as Adagrad if the gradient magnitudes vary widely across different parameters.

1. **Loss**

**\*model.compile()**

* **Mean Squared Error (MSE):** A loss function commonly used for regression problems, which calculates the average of the squared differences between predicted and actual values.
* Pros:
  + Differentiable and easy to optimise.
  + Penalises larger errors more heavily than smaller errors, which can be useful for some regression problems.
* Cons:
  + Can be sensitive to outliers, since the squared term magnifies their effect on the loss.
* **Binary Cross-Entropy**: A loss function used for binary classification problems where the goal is to predict between two classes.
* Pros:
  + Suitable for binary classification problems.
  + Penalises confident incorrect predictions more heavily than uncertain predictions, which can lead to better performance in some cases.
* Cons:
  + Can be more difficult to optimise than other loss functions, especially when the classes are highly imbalanced.
* **Categorical Cross-Entropy:** A loss function used for multi-class classification problems where the goal is to predict among more than two classes.
* Pros:
  + Suitable for multi-class classification problems.
  + Can handle cases where the number of classes is large and the classes are not mutually exclusive.
* Cons:
  + Can be sensitive to imbalanced class distributions, especially when some classes have very few samples.
* **Mean Absolute Error (MAE)**: A loss function used for regression problems which calculates the average absolute difference between predicted and actual values.
* Pros:
  + Resilient to outliers since it doesn't use the squared term.
  + Can be easier to interpret than other loss functions.
* Cons:
  + Can be less stable to optimise than MSE since it is less smooth.
  + Can be less suitable for problems where smaller errors are less important than larger ones.

1. **metrics**

**\*model.compile()**

* **Accuracy**: The proportion of correct predictions out of all predictions made.
* **Precision**: The proportion of true positive predictions out of all positive predictions made. Measures how often the model correctly identifies positive instances.
* **Recall**: The proportion of true positive predictions out of all actual positive instances. **Measures** how well the model identifies all positive instances.
* **f1**: The harmonic mean of precision and recall, which balances both metrics.
* **Auc**: The area under the receiver operating characteristic curve, which measures the trade-off between true positive rate and false positive rate.