

## 1. Autoencoder

An **autoencoder** is a type of **neural network** used for **unsupervised learning**, mainly to **compress data (encode)** and then **reconstruct it (decode)** back.

It has **three parts**:

- **Encoder** – compresses the input into a smaller representation (latent space).
- **Bottleneck** – the compressed, lower-dimensional hidden layer.
- **Decoder** – reconstructs the original data from this compressed form.

### Purpose:

To learn useful patterns or features from data by minimizing the reconstruction error.  
Used in tasks like **anomaly detection**, **denoising**, or **dimensionality reduction**.

---

## 2. Model Parameters

These are **learned by the model during training**.

They are **internal weights and biases** updated after each iteration.

### Example:

In a neural network layer like `Dense(64, activation='relu')`, the **parameters** are:

- **Weights (W)**
- **Biases (b)**

 **They are automatically learned and optimized** through backpropagation.

---

## 3. Hyperparameters

These are **set by the user before training** — not learned by the model.

They control how the model learns.

### Examples:

- Learning rate
- Number of epochs
- Batch size
- Optimizer type (e.g., Adam, SGD)
- Hidden layers, activation functions, etc.

✓ They must be tuned manually or via trial/error or techniques like Grid Search.

---

## ⚡ 4. Activation Function

Activation functions decide **whether a neuron should be activated** or not — they add **non-linearity** to the model.

Common ones:

- **ReLU (Rectified Linear Unit)**

$$f(x) = \max(0, x)$$

→ Keeps positive values as is, replaces negative values with 0.

Used widely in CNNs and deep networks.

- **Sigmoid:**

$$f(x) = \frac{1}{1 + e^{-x}}$$

→ Squashes values between 0 and 1.

Used for binary outputs.

- **Softmax:**

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

→ Converts outputs into probability distribution for multi-class classification.

---

## 🔄 5. Epoch

An **epoch** is **one complete pass** of the entire training dataset through the model.

If you train for 10 epochs, the model has seen the whole dataset 10 times.

📘 **Purpose:** Helps the model gradually learn patterns and reduce loss.

---

## 📦 6. Batch Size

The **number of samples** processed before the model updates its weights once.

Example:

If you have 1000 samples and batch size = 100 → 10 updates per epoch.

✓ **Smaller batch size** → more updates, noisier learning

✓ **Larger batch size** → smoother but slower learning

---

## 💔 7. Loss Function

It measures **how far the model's predictions are from the actual values**.

It's the **error signal** the model tries to minimize.

📘 Examples:

- **Mean Squared Error (MSE)** – Regression tasks
- **Cross-Entropy Loss** – Classification tasks  

$$L = -\sum y_i \log(\hat{y}_i)$$

$$L = -\sum y_i \log(\hat{y}_i)$$
 where  $y_i$  = true label,  $\hat{y}_i$  = predicted probability

✓ The optimizer uses this loss to adjust weights.

---

## 🎲 8. random\_state

Used in data splitting or random processes (e.g., `train_test_split`, `shuffle`) to **ensure reproducibility**.

📘 Example:

```
train_test_split(X, y, test_size=0.2, random_state=42)
```

Means every time you run it, the split will be identical.

---

## 📊 9. Confusion Matrix

A **2D table** that compares **actual labels vs predicted labels** to evaluate classification accuracy.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

📘 From this, you can calculate:

- **Accuracy** =  $(TP + TN) / (TP + TN + FP + FN)$
  - **Precision** =  $TP / (TP + FP)$
  - **Recall** =  $TP / (TP + FN)$
  - **F1-score** =  $2 \times (Precision \times Recall) / (Precision + Recall)$
- 

## 🌌 10. CNN (Convolutional Neural Network)

A **deep learning model** mainly used for **image and video processing**.

It learns **spatial features** using filters and layers like:

- **Convolution Layer** → extracts features using kernels
- **Pooling Layer** → reduces spatial size
- **Flatten Layer** → converts 2D to 1D
- **Dense Layer** → performs classification

✓ CNNs automatically learn **edges, textures, shapes, and objects**.

---

## 11. Transfer Learning


It means **reusing a pre-trained model's knowledge** on a new, similar task.

Example:

Using **MobileNetV2** trained on **ImageNet** to classify your own small image dataset.

✓ Benefits:

- Faster training
- Requires less data
- Better accuracy

 You **freeze early layers** (feature extractors) and **train only the last layers** (your custom classifier).

---

## 12. Optimizer

An **algorithm that updates the model's parameters (weights and biases)** to minimize the loss.

Common types:

- **SGD (Stochastic Gradient Descent)**  
Uses gradient direction with a learning rate to update weights.
- **Adam (Adaptive Moment Estimation)**  
Combines **momentum** (like in SGD) and **RMSProp** (adaptive learning rates).

 **Adam update rule (conceptually):**

$$\theta = \theta - \alpha \frac{m_t}{\sqrt{v_t} + \epsilon}$$

where

- $\theta$  = parameter
- $\alpha$  = learning rate

- `mtm_tmt` = momentum term (moving average of gradients)
- `vtv_tvt` = RMS term (moving average of squared gradients)

✅ Helps converge faster and more smoothly.

The `random_state` is a seed value ensuring reproducible data splitting, guaranteeing the same data points in training and testing sets each time.

## 💡 Definition

**Sparse Categorical Crossentropy** is a **loss function** used for **multi-class classification problems**, when your **labels are integers** (not one-hot encoded).

It measures **how far the model's predicted probability distribution** is from the **true class label**.

---

## 🧮 Mathematical Form

$$L = -\frac{1}{N} \sum_{i=1}^N \log(p_i, y_i) \quad L = -\frac{1}{N} \sum_{i=1}^N \log(p_i, y_i)$$

Where:

- $N$  → number of samples
- $y_i$  → true class label (as an integer, e.g. 0, 1, 2, ...)
- $p_{i, y_i}$  → model's predicted probability for the correct class

✅ The loss is **low** when the model assigns **high probability to the correct class**.

---

## 🧠 In Simple Words

- It compares the model's predicted probabilities (from **Softmax**) with the **correct class index** (like **2** for "cat").
  - It penalizes the model when it predicts wrong or uncertain probabilities.
- 

## 📄 Example

Let's say we have 3 classes → **[dog, cat, horse]**.

**True label:** **1** (i.e., "cat")

**Predicted probabilities:** **[0.1, 0.8, 0.1]**

$$\text{Loss} = -\log(0.8) = 0.223 \quad \text{Loss} = -\log(0.8) = 0.223 \quad \text{Loss} = -\log(0.8) = 0.223$$

✓ Small loss → good prediction.

---

## Difference from “Categorical Crossentropy”

Type	Expected Label Format	Example Label	Example Code
<b>Categorical Crossentropy</b>	One-hot encoded	<code>[0, 1, 0]</code>	<code>loss="categorical_crossentropy"</code>
<b>Sparse Categorical Crossentropy</b>	Integer labels	<code>1</code>	<code>loss="sparse_categorical_crossentropy"</code>

So basically:

“**Sparse**” = you don’t need to one-hot encode your labels.