Module – III

Convolution Operation in CNN :

Convolution is a mathematical operation mostly used in Convolution Neural Networks - CNNs and also in various fields such as signal processing, image processing, and deep learning .

It is widely used in deep learning for extracting the features from images, and it helps detect patterns like edges, textures, and objects.

Importance of Convolution in Image Processing :

- Feature Extraction: Helps identify patterns such as edges, corners, and textures.
- Blurring & Sharpening: Used to smooth or enhance image details.
- Edge Detection: Highlights boundaries of objects in an image.
- Noise Reduction: Removes unwanted variations in an image.
- Used in Deep Learning (CNNs): Helps recognize patterns in images for classification and object detection.

2. Convolution in Image Processing :

Generally in images are represented in pixels with in the range 0 to 255. But in image processing The pixels can be normalized and scaled and represented in numerical values like 0 to 1.

In image processing, convolution is performed using a kernel (filter) that moves over an image, applying element-wise multiplication and and addition.

How Convolution Works in Images :

1. A small kernel (filter matrix) slides over the image.
2. Each element in the kernel multiplies with the corresponding pixel values.

3. The results are summed up to create a new pixel in the output image.
4. The filter moves across the entire image to generate a transformed output.

Mathematical Representation for 2D Convolution

For an image I and a kernel K, the convolution is:

$$\boxed{S(i, j) = \sum m \sum n\, I(i-m, j-n)\, K(m, n)}$$

where:

- I (i,j) is the input image.
- K(m,n) is the kernel.
- S(i,j) is the convolved output.

4. Properties of Convolution :

1. Commutative Property: $f*g = g*f$
2. Associative Property: $f * (g * h) = (f * g) * h$
3. Distributive Property: $f * (g+h) = (f*g) + (f*h)$

5. Types of Convolution in Deep Learning :

1. Valid Convolution (No Padding)

- Only the parts of the image where the kernel fully overlaps are computed.
- The output size is smaller than the input.
- Formula for output size:
- Output Size = (Input Size - Kernel Size) / Stride+1

2. Same Convolution (Zero Padding)

- Adds zero-padding around the image to keep the output size the same as input.
- Commonly used in CNNs.

Nearest Convolution (zero or one Padding) :

- Adds zero-padding around the image to keep the output size the same as input.
- Adds nearest padding around the image to keep the output size the same as input.

3. Strided Convolution :

- Instead of moving one pixel at a time, the kernel moves in steps (stride > 1).
- Reduces the size of the output.

6. Applications of Convolution :

1. Image Processing
   - Edge Detection (Sobel, Laplacian filters)
   - Blurring (Gaussian blur)
   - Sharpening
2. Deep Learning (Convolutional Neural Networks - CNNs)
   - Feature extraction in images
   - Object detection
   - Face recognition
3. Audio Processing
   - Noise reduction
   - Speech recognition
4. Signal Processing
   - Filtering signals
   - Smoothing

Example of Convolution in Images

Consider a 3×3 kernel applied to a 6×6 image.

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

6×6 image

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

3×3 kernel (Filter)

Output :

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

Convoled output

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

after applying min max scale

Finally the zero values can be showing strong edge detection between the 0 and 1 values.

Motivation Behind Convolutional Neural Networks (CNNs) :

Convolutional Neural Networks (CNNs) are specialized deep learning models designed for image processing and computer vision tasks.

They were developed to overcome the limitations of traditional neural networks in handling high-dimensional image data efficiently.

1. Why Do We Need CNNs? (Motivation) :

A) Challenges of Fully Connected Neural Networks (FCNs) for Images

Before CNNs, traditional fully connected networks (FCNs) were used for image classification. However, they faced several issues:

1.High Number of Parameters :

- A simple 100×100 image has 10,000 pixels per channel. If it has 3 color channels (RGB), the input size becomes 30,000.
- If the first layer has 1,000 neurons, this leads to 30 million weights—which is computationally expensive.

2. Loss of Spatial Information :

- FCNs flatten the image into a 1D vector, losing spatial structure (e.g., nearby pixels that form edges and textures).
- This makes it harder for FCNs to recognize objects correctly.

3.Poor Generalization to Translations :

- In FCNs, if an object appears at a different position, the network may fail to recognize it.
- CNNs solve this with translation invariance, meaning they can detect patterns anywhere in the image.

2. How CNNs Solve These Challenges :

CNNs introduce two key concepts:

1. Convolution – Extracts features while preserving spatial relationships.
2. Pooling – Reduces dimensions while keeping important information.

These techniques solve the problems of FCNs:

Problem in FCNs	Solution in CNNs
Too many parameters	Convolution shares weights, reducing parameters
Loss of spatial info	Filters preserve spatial relationships
No translation invariance	Pooling & weight sharing allow detection anywhere

3. Biological Inspiration :

CNNs are inspired by the human visual system:
Hubel & Wiesel's experiments (1962) showed that the brain processes images using hierarchical layers of neurons:

- Early layers detect simple patterns (edges, textures)
- Deeper layers recognize complex objects (faces, shapes, animals, etc.)

4. Key Advantages of CNNs :

Efficient Feature Extraction:

- CNNs automatically learn important features, unlike traditional methods that require hand-crafted features.

Reduces Computational Complexity:

- Weight sharing in convolution layers drastically reduces the number of parameters compared to FCNs.

Translation Invariance:

- CNNs recognize objects regardless of position in the image.

Works Well for Image Data:

- CNNs are state-of-the-art in image classification, object detection, and segmentation.

5. Real-World Applications of CNNs :

- Image Classification – Recognizing objects (e.g., cats vs. dogs)
- Autonomous Vehicles – Detecting pedestrians, lanes, and traffic signs
- Facial Recognition – Used in security systems
- Medical Diagnosis – Identifying diseases from X-rays, MRIs, etc.
- Gaming & AR – Motion tracking and real-time image processing

Pooling in Convolutional Neural Networks (CNNs) :

Pooling is an essential operation in Convolutional Neural Networks (CNNs) that reduces the spatial dimensions (height and width) of feature maps while retaining the most important information.

A feature map is the output of a convolution operation applied to an input image in a convolutional neural network (CNN). It represents the important features detected by a filter (or kernel), such as edges, textures, or patterns.

It helps in making the model computationally efficient and less sensitive to small variations in the input.

1. Why is Pooling Used?

Pooling is used in CNNs for the following reasons:

- Reduces computational complexity by decreasing the number of parameters and operations.
- Prevents over fitting by reducing the size of the feature map.
- Enhances feature detection by making the model more robust to translation and distortion in images.

2. Types of Pooling :

There are different types of pooling operations used in CNNs:

a) Max Pooling

- Takes the maximum value from a small region of the feature map.
- Helps in detecting the most important features.
- Commonly used in CNN architectures.

Example:

If we apply a 2×2 max pooling operation on the following feature map:

1 3 2 1

4 2 1 5

6 8 3 2

7 3 4 6

The result will be:

4   5

8   6

Key Points:

- Selects the strongest feature in each region.
- Helps in maintaining sharp edges.

b) Average Pooling :

- Takes the average value from a small region of the feature map.
- Useful when we need smoother feature maps.
- Less commonly used than max pooling.

Example:

If we apply a 2×2 average pooling operation on the same feature map:

1 3 2 1

4 2 1 5

6 8 3 2

7 3 4 6

(1+3+4+2) / 4   (2+1+1+5) / 4

(6+8+7+3) / 4   (3+2+4+6) / 4

The result will be:

2.5 2.25

6 3.75

Key Points:

- Provides a more generalized representation.
- Less effective at retaining important features.

c) Global Pooling

- Averages or takes the maximum value of the entire feature map.
- Converts the feature map into a single value per channel.
- Used in fully convolutional networks (FCNs) instead of dense layers.

EX : Given Feature Map (4×4 matrix):

1 3 2 1

4 2 1 5

6 8 3 2

7 3 4 6

1. Global Max Pooling (GMP) :

Find the maximum value in the entire feature map:
max(1,3,2,1,4,2,1,5,6,8,3,2,7,3,4,6) = Output: 8

2. Global Average Pooling (GAP):

Find the average of all values in the feature map:

1+3+2+1+4+2+1+5+6+8+3+2+7+3+4+6 / 16 = 57 / 16 = 3.56
Output: 3.56

3. Pooling Parameters :

When applying pooling, we define:

- Pool size: The size of the window (e.g., 2×2 or 3×3).
- Stride: The step size to move the pooling window (e.g., 1 or 2).
- Padding: Whether to keep the same dimensions (if needed).

4. Advantages of Pooling :

- Reduces dimensions, making computation faster.
- Increases translation invariance, meaning small shifts in input don’t affect output.
- Extracts dominant features from images.

Convolution and Pooling as an Infinitely Strong Prior :

In Deep learning, particularly in Convolution Neural Networks (CNNs), convolution and pooling act as strong priors that impose specific assumptions on the data.

1. Understanding Prior in Machine Learning :

A prior in machine learning refers to assumptions about data before seeing the actual observations.

- A strong prior imposes strict conditions on how the model learns.
- A weak prior allows more flexibility, requiring the model to learn patterns from data.

For example:

- In a fully connected neural network (FCN), the model is very flexible but inefficient for images.
- In a CNN, convolution and pooling restrict how features are extracted, which can be beneficial.

2. Convolution as a Strong Prior :

Convolution imposes the assumption that:

- Local patterns matter: Each filter scans small regions instead of processing the entire image at once.
- Spatial hierarchy exists: Lower layers detect edges and textures, while deeper layers detect shapes and objects.
- Translation invariance: The same filter applies across different image locations, assuming features are location-independent (e.g., a cat's ear looks the same anywhere in the image).

Why is this a strong prior?

- CNNs cannot easily learn features that require non-local dependencies (e.g., relationships between distant parts of an image).
- If features depend on absolute position, convolution may struggle (e.g., distinguishing left from right in medical imaging).

3. Pooling as a Strong Prior :

Pooling assumes that:

- Precise location is not important: Max or average pooling discards the exact position of features.

- Small shifts in input do not change output: This reduces sensitivity to minor variations in the image.

Why is this a strong prior?

- It forces the network to focus on what features exist rather than where they are.
- However, this can be problematic for tasks requiring fine-grained localization (e.g., image segmentation, object detection).

4. Infinitely Strong Prior: The Limitations :

Since convolution and pooling force these assumptions on all images, they can become too restrictive, leading to:

1. Loss of global context – CNNs struggle with relationships between distant objects.
2. Inflexibility to domain-specific tasks – Some tasks require precise positioning of features.
3. Inefficiency for non-grid data – CNNs assume a structured grid (like images), making them less effective for other types of data (e.g., graphs).

5. Conclusion: Is It Good or Bad?

- Good: If the task fits the assumptions (e.g., image classification, natural images), convolution and pooling make learning more efficient.
- Bad: If the assumptions are too strong (e.g., requiring spatial precision or global context), CNNs struggle to generalize.

<u>**Variants of the Basic Convolution Function in CNNs :**</u>

Convolution Neural Networks (CNNs) use the convolution operation to extract features from images. However, several variants of the basic convolution function have been developed to improve performance, capture different patterns, and handle diverse data types efficiently.

1. Standard Convolution (Basic Convolution) :

In the standard convolution operation:

- A filter (kernel) slides over the input image.
- The dot product between the filter and the corresponding part of the image is computed.
- This produces a feature map that highlights important patterns like edges and textures.

Mathematical Representation:
For an input image X and a filter W, convolution is given by:

$$\boxed{Y(i,j) = \sum m \sum n \, X(i+m, j+n) \cdot W(m, n)}$$

where:

- X (i, j) is the input at position (i, j).
- W (m, n) is the filter weight.
- Y (i,j) is the resulting feature map value.

Limitations of Standard Convolution:

- Requires many parameters, leading to high computational costs.
- Only captures local spatial relationships.
- May not be efficient for tasks needing global context (e.g., long-range dependencies).

2. Dilated (Atrous) Convolution :

Goal: Capture larger receptive fields without increasing computational cost.

How It Works:

- Introduces gaps (dilation rate) between filter elements, allowing it to cover a wider area.

- The standard convolution slides continuously, while dilated convolution skips pixels in between.

Mathematical Representation:

$$\boxed{Y(i, j) = \sum m \sum n\, X(i+d\cdot m, j+d\cdot n)\cdot W(m, n)}$$

where d is the dilation rate.

Advantages:

Increases receptive field without increasing the number of parameters.
Useful for semantic segmentation and context-aware models.

Use Cases:

- Deep Lab models (used in image segmentation).
- Audio and text processing.

3. Transposed (De convolution) Convolution :

Goal: Perform up sampling (increase image size) while learning filters.

How It Works:

- Unlike standard convolution, which reduces the image size, transposed convolution expands it.
- Used in generative networks and segmentation models to reconstruct high-resolution images.

Use Cases:

- Image segmentation (e.g., U-Net, FCN).
- Super-resolution (enhancing image quality).
- Generative models (e.g., GANs).

4. Depth wise Separable Convolution :

Goal: Reduce computational cost while maintaining feature extraction quality.

How It Works:

Instead of applying one large filter to the entire input, it splits convolution into two steps:

1.Depth wise Convolution – Each channel is convolved separately with a smaller filter.
2.Point wise Convolution (1×1 Convolution) – Combines all channels using a 1×1 filter.

Mathematical Representation:
If K is the filter size and C is the number of channels:

- Standard Convolution: K×K×C multiplications.
- Depth wise Separable Convolution: K×K+1×1 multiplications (much fewer operations).

Use Cases:

- Mobile Nets (optimized for mobile vision tasks).
- Efficient deep learning models for edge devices.

5. Grouped Convolution :

Goal: Improve computational efficiency while maintaining feature extraction quality.

How It Works:

- Instead of applying a single large filter, filters are divided into groups that process different parts of the image separately.
- Each group learns distinct features and then combines them.

Advantages:

Reduces computational cost compared to full convolution.
Useful in multi-channel models and residual networks.

Structured Outputs and Data Types in CNNs :

Convolution Neural Networks (CNNs) are designed to handle structured input data (mainly images) and produce structured output, which is essential for various computer vision tasks. Understanding structured outputs and data types in CNNs helps optimize network design for specific applications.

1. Structured Outputs in CNNs :

What is a Structured Output?

A structured output refers to predictions that maintain a meaningful structure, unlike traditional classification outputs (which are just labels). CNNs generate structured outputs for tasks where the relationships between elements (like pixels in an image) matter.

Types of Structured Outputs in CNNs :

Task	Structured Output Type	Example Applications
Image Classification	Single class label or probability vector	Identifying cats vs. dogs
Object Detection	Bounding boxes + class labels	Detecting cars in images
Image Segmentation	Pixel-wise class labels (mask)	Medical imaging (tumor detection)
Pose Estimation	Key point coordinates	Human motion tracking
Depth Estimation	Depth map (per-pixel depth values)	3D scene reconstruction
Super-Resolution	High-resolution image	Enhancing image quality

1.1 Image Classification (Single-Label & Multi-Label) :

Output Format: A probability distribution over classes
Example:
For a 3-class problem (cat, dog, horse), CNN output might be:

[0.7, 0.2, 0.1] {(70% cat, 20% dog, 10% horse)}

Loss Function: Cross-Entropy Loss
Use Cases: Object recognition, scene classification

1.2 Object Detection (Bounding Boxes + Class Labels) :

Output Format:

(x, y, w, h, class_label)

where:

- (x, y) = coordinates of the bounding box center
- (w, h) = width and height of the bounding box
- class_label = predicted object type

Example: Detecting a dog in an image

(50, 60, 120, 80, "dog")

Loss Functions:

- Classification Loss: Cross-entropy
- Bounding Box Loss: IoU (Intersection over Union)

Use Cases: Self-driving cars, security cameras

1.3 Image Segmentation (Pixel-Wise Classification) :

Output Format: Per-pixel label (e.g., road, car, pedestrian)

Example:
A 256×256 image segmentation map:

```
Mask:  [ 0 1 0 2
         1 1 0 2 ]
```

(0 = background, 1 = car, 2 = pedestrian)

Loss Function: Dice loss, Cross-entropy loss
Use Cases: Medical imaging, satellite image analysis

1.4 Pose Estimation (Key points Prediction) :

Output Format: (x, y) coordinates of body joints
Example: Detecting human pose

Left Hand = (120, 200), Right Hand = (250 , 210)

Loss Function: Mean Squared Error (MSE)
Use Cases: Sports analytics, animation

1.5 Depth Estimation (Predicting Depth for Each Pixel) :

Output Format: Depth map (each pixel has a depth value)
Example: Estimating 3D structure from a 2D image
Loss Function: Mean Squared Error (MSE)
Use Cases: 3D reconstruction, robotics

2. Data Types in CNNs :

CNNs process and output different types of data, which vary based on input format and task requirements.

2.1 Input Data Types

CNNs typically process the following data types:
1.Grayscale Images → Shape: (H, W, 1) (single channel)
2. RGB Images → Shape: (H, W, 3) (3 channels)

3.Videos → Shape: (Frames, H, W, 3)
4.Medical Images (3D scans like MRI, CT) → Shape: (D, H, W, C)

Example:
A 256×256 RGB image is stored as:

(256, 256, 3)

where 3 represents RGB channels.

2.2 Output Data Types

Task	Output Data Type	Example Output
Classification	Vector of probabilities	[0.7, 0.2, 0.1]
Detection	Bounding box coordinates	(x, y, w, h, label)
Segmentation	Pixel-wise labels (mask)	Matrix (H, W)
Pose Estimation	Key point coordinates	[(x1, y1), (x2, y2)]
Depth Estimation	Depth map	Matrix (H, W)

2.3 Data Types Used in CNN Computation :

CNNs work with different data types during computation:

Float 32 (Most Common)

- Standard data type for CNNs
- Used in training & inference

Float 16 (Mixed Precision)

- Reduces memory usage, speeds up training
- Used in Tensor Cores (NVIDIA GPUs)

Int 8 (Quantized CNNs)

- Used in mobile and edge devices for efficiency
- Reduces model size without much accuracy loss

Example:

- Training: Float32 or Float16
- Deployment on mobile: Int8 (quantized model)

3. Summary Table :

Aspect	Details
Structured Output	Maintains relationships in the prediction
Types	Classification, Object Detection, Segmentation, Pose Estimation, Depth Estimation
Input Data	Images (Grayscale, RGB), Videos, Medical Scans
Output Data	Labels, Bounding Boxes, Masks, Key points, Depth Maps
Computation Types	Float 32, Float 16, Int 8 (Quantization for Edge AI)

Efficient Convolution Algorithms in CNNs :

Convolution is a fundamental operation in Convolution Neural Networks (CNNs), but standard convolution is computationally expensive.

Several efficient convolution algorithms have been developed to reduce computation, improve speed, and optimize memory usage without significantly affecting accuracy.

1. Why Do We Need Efficient Convolution?

Challenges of Standard Convolution:

1.High Computational Cost:

- A standard convolution operation involves multiplying and adding many parameters.

- The complexity is O(K^2 × C_in × C_out × H × W), where K is the kernel size, C in and C out are input and output channels, and H,W are feature map dimensions.

2.Large Memory Usage:

- CNNs require high memory bandwidth to store and fetch large feature maps and kernels.

3.Inefficient for Mobile & Edge Devices:

- Standard convolution is slow for real-time applications like self-driving cars, smart phones, IoT devices.

Solution: Efficient convolution algorithms reduce computational cost while maintaining performance.

2. Types of Efficient Convolution Algorithms :

Algorithm	Key Idea	Speedup Factor	Use Cases
Wino grad Convolution	Uses fast matrix multiplications for small kernels	2-4x	Mobile AI
FFT-Based Convolution	Performs convolution in the frequency domain	2-3x	Large kernels
Grouped Convolution	Uses channel-wise convolutions	2-3x	ResNeXt
Sparse Convolution	Skips zero values in sparse matrices	Varies	3D vision, NLP

3. Detailed Explanation of Efficient Convolution Algorithms :

3.1 Wino grad Convolution (Fast Convolution) :

Key Idea:

- Rewrites the convolution operation as matrix multiplications, which can be computed faster.
- Works best for small filters (e.g., 3×3, 5×5).

  Mathematical Explanation:
  For a 3×3 filter applied to a 4×4 input, standard convolution requires 16 multiplications. Winograd reduces this to 8 multiplications using Winograd's minimal filtering algorithm.

Advantages:
2-4× faster than standard convolution for small kernels.
Reduces multiplication operations significantly.

Limitations:
Not efficient for large filters (e.g., 7×7, 9×9).
Requires more memory for intermediate transformations.

Use Cases:

- Mobile AI models (used in Tensor Flow Lite and Qualcomm DSPs).

3.2 FFT-Based Convolution (Fast Fourier Transform) :

Key Idea:

- Uses Fourier Transform to convert convolution into element-wise multiplication in the frequency domain.
- Standard convolution requires $O(K^2 \times H \times W)$ operations, but FFT reduces it to $O(HW \log HW)$.

Mathematical Explanation:
1.Convert the input X and filter W to the frequency domain using FFT.

$Xf = FFT(X)$, $Wf = FFT(W)$

2.Perform element-wise multiplication.

$Yf = Xf \cdot Wf$

3.Convert back to the spatial domain using Inverse FFT (IFFT).

Advantages:
Works well for large kernels (e.g., 7×7, 9×9).
Reduces computation significantly for high-resolution images.

Limitations:
Not efficient for small kernels (e.g., 3×3).
Requires additional processing time for FFT and IFFT.

Use Cases:

- Medical imaging, astronomy, signal processing.

3.4 Grouped Convolution (Used in ResNeXt)

Key Idea:

- Instead of applying a single large filter, divide channels into groups and apply convolution separately.

Example:
For a 64-channel input, instead of using a 64×64 filter, use 4 groups of 16×16 filters.

Advantages:
Reduces computation by 2-3×.
Improves parallelism and memory efficiency.

Limitations:
Requires careful channel grouping for best performance.

Use Cases:

- Res NeXt, Alex Net, Shuffle Net.

3.5 Sparse Convolution (Used in 3D Vision & NLP) :

Key Idea:

- Skips zero values in sparse feature maps, reducing unnecessary computation.

Advantages:

Highly efficient for sparse data (e.g., 3D point clouds, text embeddings).

Use Cases:

- LIDAR-based object detection (self-driving cars).
- Transformers (efficient NLP models).

Random or Unsupervised Features in CNNs :

Convolution Neural Networks (CNNs) typically learn features through supervised learning with labeled datasets. However, in some cases, random or unsupervised features can be used to improve efficiency, reduce reliance on large labeled datasets, and enhance generalization.

1. Random Features in CNNs :

Random features refer to convolution filters initialized randomly and kept fixed without training. These random filters can still extract useful information, especially in early layers.

1.1 Why Use Random Features ?

Avoids expensive training → No back propagation required
Useful for low-data settings → No need for large datasets
Surprisingly effective for feature extraction
Reduces over fitting → No risk of memorizing the training set

1.2 How Random Features Work :

1.Randomly initialize convolution filters (e.g., Gaussian or uniform distribution)
2.Do not update these filters during training
3.Use them as fixed feature extractors
4.Pass extracted features to a classifier (e.g., SVM, fully connected layers)

Example:

- Apply random 3×3 filters to an image
- Extract edges and textures
- Train an SVM classifier on extracted features

2. Unsupervised Features in CNNs :

Unsupervised learning methods allow CNNs to learn features without labeled data. This is useful for applications where labeled datasets are expensive or unavailable.

2.1 Types of Unsupervised Feature Learning in CNNs :

Method	Key Idea
Auto encoders	Learn to reconstruct inputs, forcing CNNs to capture important features
Contrastive Learning	Learn representations by maximizing similarity between augmented views of the same image
Generative Adversarial Networks (GANs)	Train a generator and discriminator to learn data distributions
Self-Supervised Learning	Create pretext tasks (e.g., predicting missing parts of an image) to learn features
K-means Clustering + CNNs	Cluster features and use cluster centers as filters

2.2 Auto encoders for Feature Learning :

- Unsupervised CNN auto encoders learn features by compressing and reconstructing images.
- The encoder extracts meaningful features, while the decoder reconstructs the image.
- Use Case: Pre training CNNs before fine-tuning on a classification task.

2.3 Contrastive Learning :

- Instead of using labels, CNNs learn by comparing similar and different images.
- The network pulls similar images closer in feature space and pushes different images apart.
- Use Case: Self-supervised image classification (e.g., Image Net pre training).

3. Comparison: Random vs. Unsupervised Features :

Feature Type	Pros	Cons	Best For
Random Features	No training needed, computationally cheap	Lower accuracy compared to trained models	Low-resource tasks, small datasets
Unsupervised Features	Learns meaningful representations, works without labels	Requires more computation	Large-scale learning, self-supervised pre training

- End -