CSE-403: Machine Learning

Chapter 7: Convolutional Neural Networks (CNNs)

Md Atikuzzaman

Lecturer
Department of Computer Science & Engineering
atik@cse.green.edu.bd

GREEN UNIVERSITY OF BANGLADESH

Outline

1) The Neuron Model and Activation Functions
2) Architecture of Neural Networks
3) Forward Propagation
4) Loss Functions for Neural Networks
5) Backpropagation and Gradient Descent
6) Initialization and Normalization Techniques
7) Vanishing and Exploding Gradients
8) Regularization in Neural Networks (Dropout, Weight Decay)
9) Optimization Algorithms (SGD, Adam, RMSprop)
10) Batch, Mini-batch, and Stochastic Training
11) Hyperparameter Tuning
12) Performance Evaluation and Model Interpretation

Why Convolutional Neural Networks (CNNs)?

CNNs are used because they are **specifically designed to handle image and grid-like data efficiently and accurately**. Here are the main reasons:

1. Automatic Feature Extraction

Traditional methods require manual feature design (edges, textures, shapes).
CNNs **learn features automatically** from raw data.

- Early layers → detect edges
- Middle layers → detect shapes
- Deep layers → detect objects

2. Preserves Spatial Information

Instead of flattening the image, CNNs process the image in its natural 2D or 3D grid. By using filters that look at small "patches" of pixels at a time, the network understands that neighboring pixels relate to one another to form edges, curves, and textures.

3. Parameter Efficiency (Less Computation)

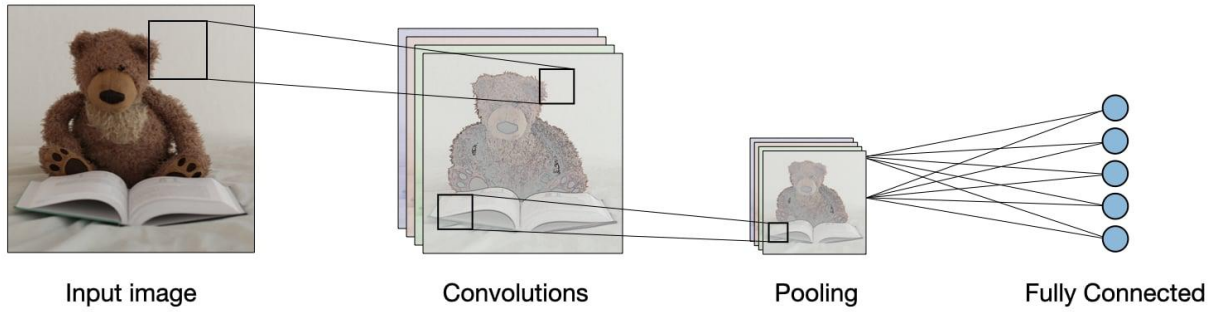
CNN uses **filters (kernels)** instead of connecting every neuron.

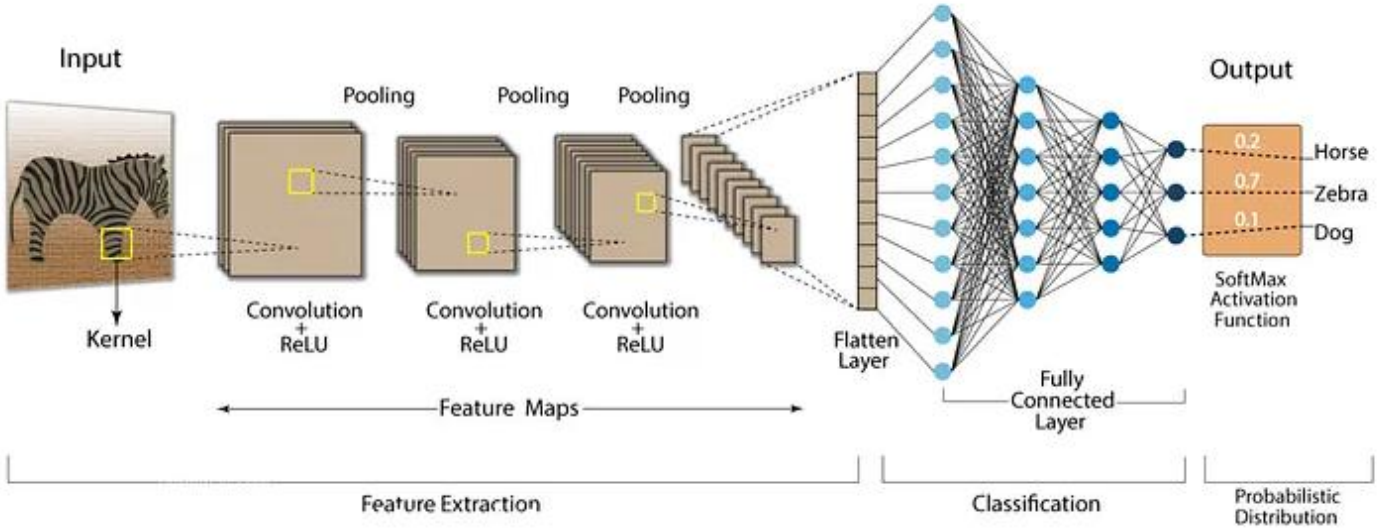
- Fewer parameters
- Faster training
- Less memory usage

What is a Convolutional Neural Network (CNN)?

A Convolutional Neural Network (CNN) is a specialized type of artificial intelligence deep learning algorithm primarily used to analyze and recognize patterns in visual data, such as images and videos. CNNs are composed of **three primary layers**: the convolutional layer, the pooling layer and the fully connected layer.

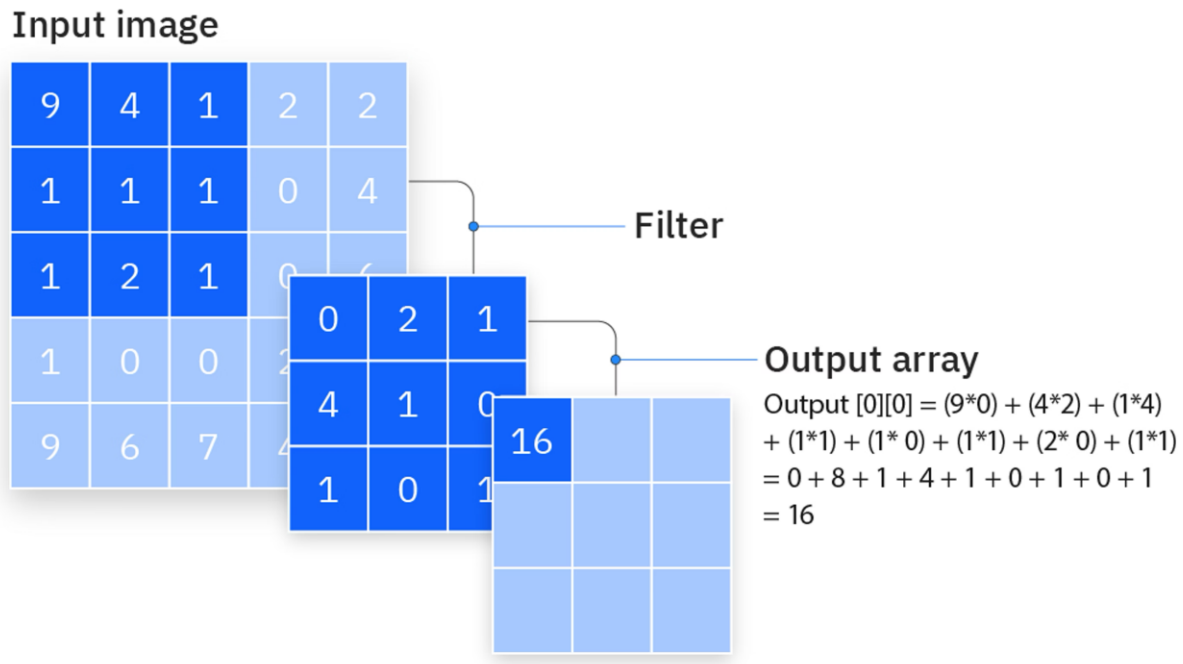
- **Convolutional layer**: Using filters or kernels, this layer finds local patterns and features from the input image.
- **Pooling layer**: This layer downsamples the spatial dimensions of the input, reducing the network's computational complexity and increasing its ability to recognise patterns regardless of scale and position.
- **Fully Connected layer**: This layer integrates previously extracted features into a traditional neural network, in efforts to learn global relationships and generate task-specific outputs.





What is a Convolutional Neural Network (CNN)?

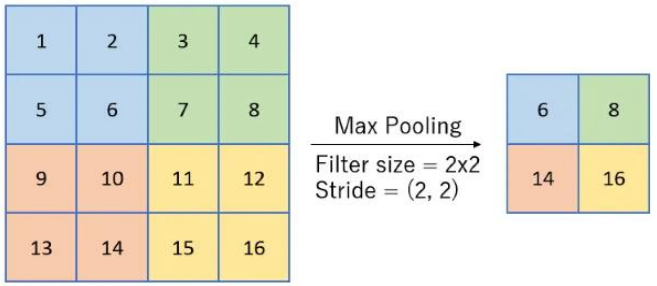
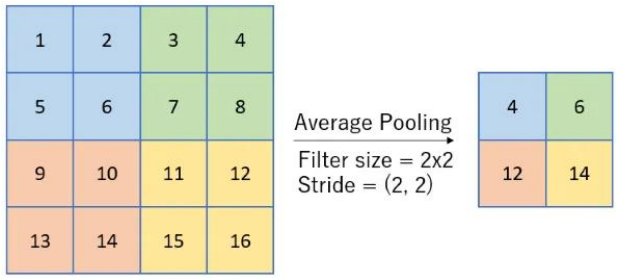
Convolution layer (CONV): The convolution layer (CONV) uses filters that perform convolution operations as it is scanning the input I with respect to its dimensions. Its hyperparameters include the filter size F and stride S. The resulting output O is called *feature map* or *activation map*.



Remark: the convolution step can be generalized to the 1D and 3D cases as well.

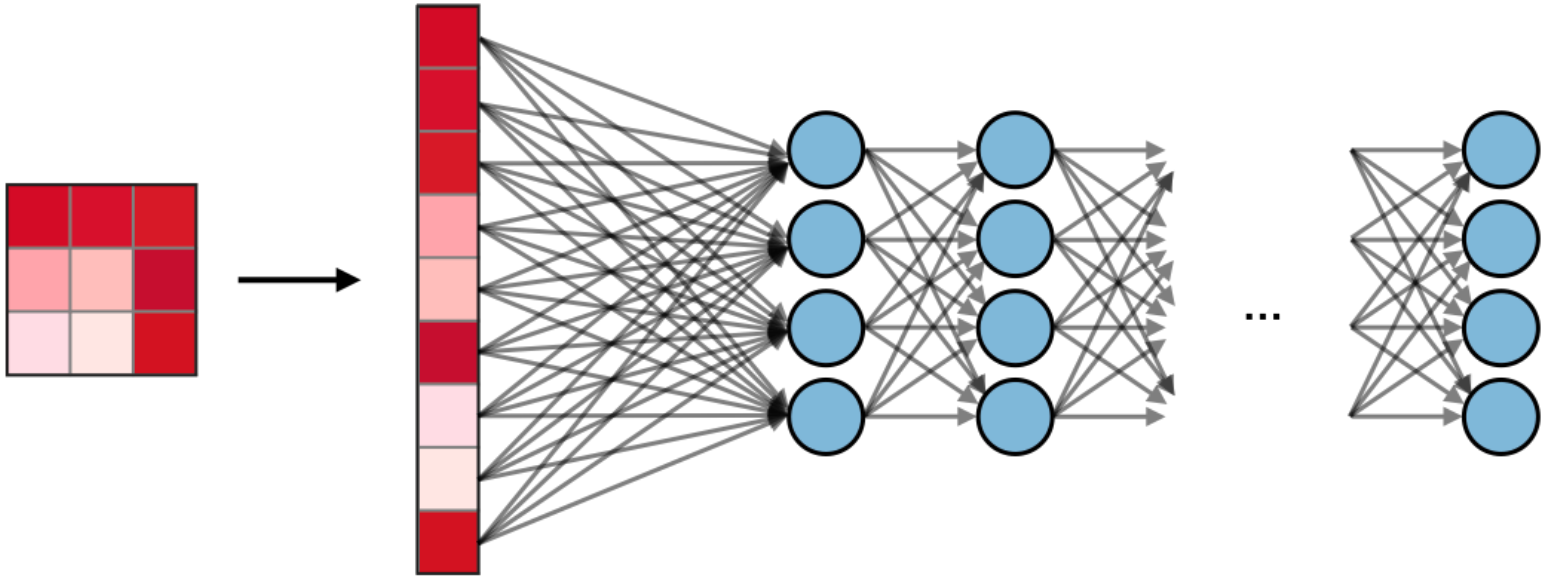
What is a Convolutional Neural Network (CNN)?

Pooling (POOL) The pooling layer (POOL) is a downsampling operation, typically applied after a convolution layer, which does some spatial invariance. In particular, max and average pooling are special kinds of pooling where the maximum and average value is taken, respectively.

Type	Max pooling	Average pooling
Purpose	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view
		
Comments	• Preserves detected features • Most commonly used	• Downsamples feature map • Used in LeNet

What is a Convolutional Neural Network (CNN)?

Fully Connected (FC) The fully connected layer (FC) operates on a flattened input where each input is connected to all neurons. If present, FC layers are usually found towards the end of CNN architectures and can be used to optimize objectives such as class scores.



What is Convolution?

Convolution is a mathematical operation used in CNNs to **extract features from an image**.
Instead of looking at the whole image at once, convolution focuses on **small regions** and scans the image step by step.
A small matrix called a **filter (kernel)** moves across the image and performs element-wise multiplication, then sums the result to produce a new value.

Mathematical Representation

$$S(i, j) = (I * K)(i, j) = \sum_{m} \sum_{n} I\,(i, +mj + n) \cdot K(m, n)$$

where:

- I= input image
- K= filter (kernel)
- S= output (feature map)

What is a Filter (Kernel)?

A **filter** is a small matrix (e.g., 3×3 or 5×5) used to detect specific patterns in an image.
Each filter is designed (or learned) to capture features such as:

- edges
- lines
- textures
- patterns

Example of a simple 3×3 filter:

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

This filter helps detect **vertical edges**.

Step 1: Take a small region of the image
Example (3×3 region):

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

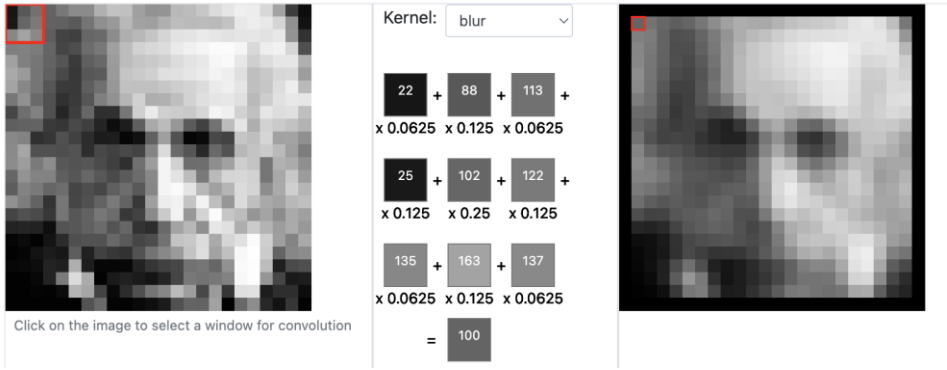
Step 2: Apply the filter

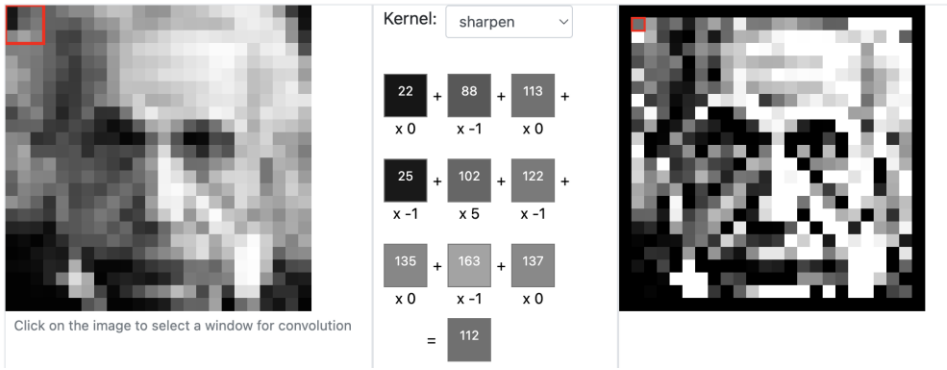
$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Step 3: Multiply element-wise and sum

$$\big(1 \times 1 + 2 \times 0 + 3 \times (-1)\big) + \big(4 \times 1 + 5 \times 0 + 6 \times (-1)\big) + \big(7 \times 1 + 8 \times 0 + 9 \times (-1)\big)$$
$$= (1 + 0 - 3) + (4 + 0 - 6) + (7 + 0 - 9)$$
$$= -2 - 2 - 2 = -6$$

This value becomes one pixel in the output feature map.





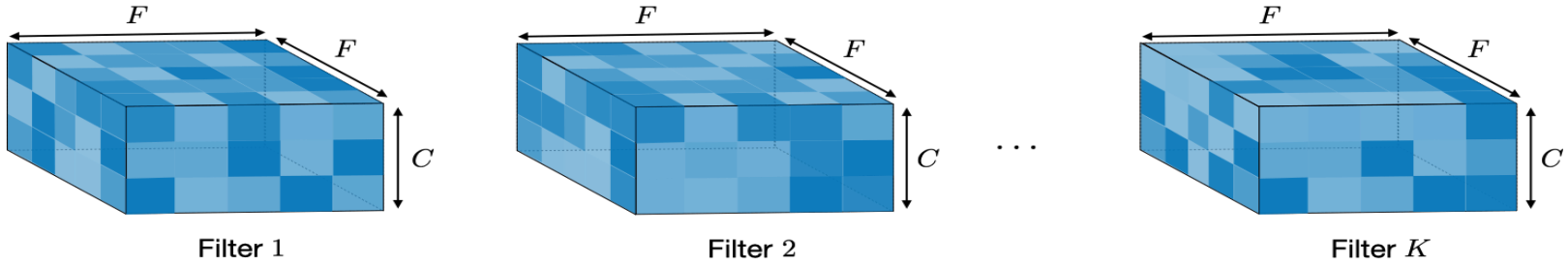
Step 4: Slide the filter
The filter moves across the image (left to right, top to bottom), repeating the same process.

Output: Feature Map
After applying the filter across the entire image, we get a **feature map**, which highlights specific patterns like edges.

Filter hyperparameters

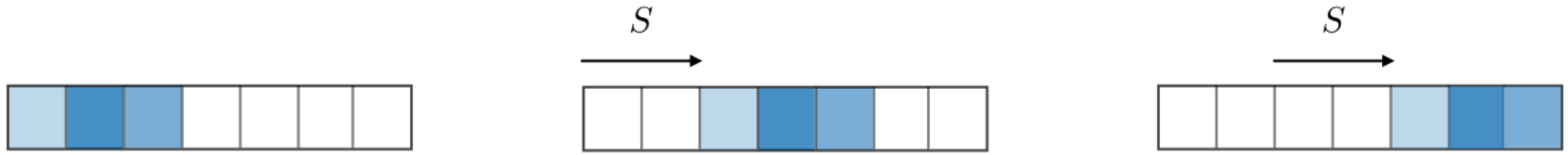
The convolution layer contains filters for which it is important to know the meaning behind its hyperparameters.

Dimensions of a filter: A filter of size $F \times F$ applied to an input containing C channels is a $F \times F \times C$ volume that performs convolutions on an input of size $I \times I \times C$ and produces an output feature map (also called activation map) of size $O \times O \times 1$.



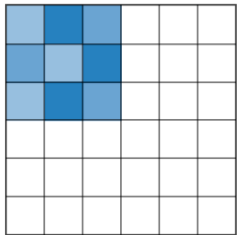
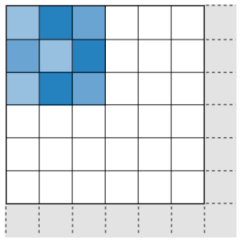
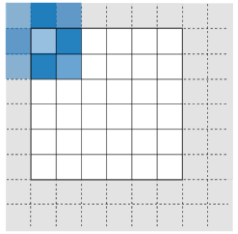
Remark: the application of K filters of size $F \times F$ results in an output feature map of size $O \times O \times K$.

Stride: For a convolutional or a pooling operation, the stride S denotes the number of pixels by which the window moves after each operation.



Filter hyperparameters

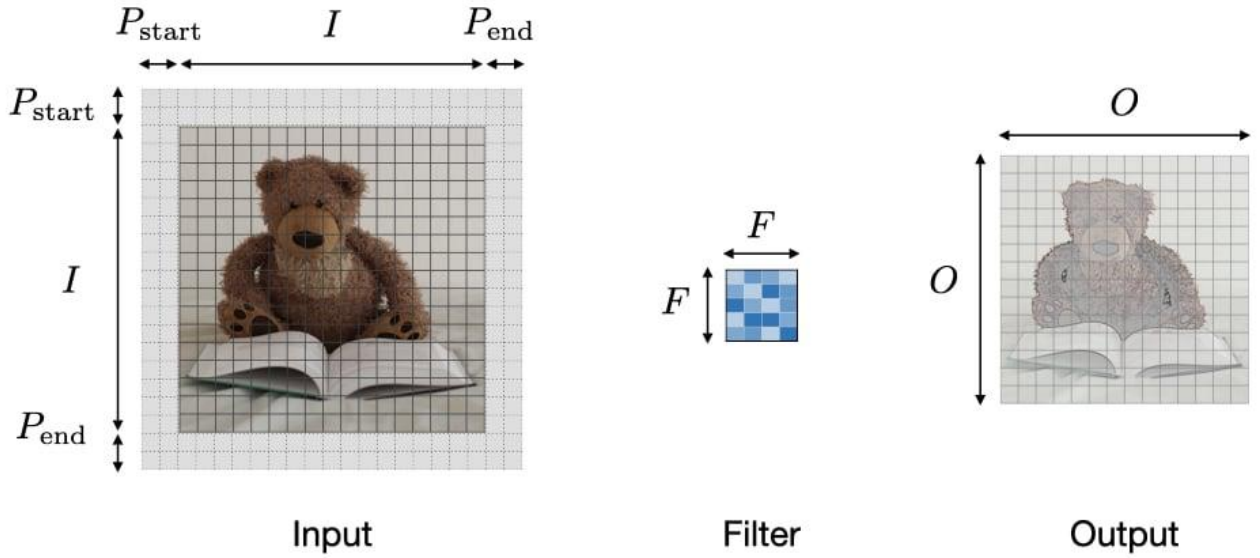
Zero-padding: Zero-padding denotes the process of adding P zeroes to each side of the boundaries of the input. This value can either be manually specified or automatically set through one of the three modes detailed below:

Mode	Valid	Same	Full
Value	$P = 0$	$P_{start} = \left\lfloor \frac{S\left\lceil \frac{I}{S} \right\rceil - I + F - S}{2} \right\rfloor$ $P_{end} = \left\lceil \frac{S\left\lceil \frac{I}{S} \right\rceil - I + F - S}{2} \right\rceil$	$P_{start} \in [[0, F-1]]$ $P_{end} = F - 1$
			
Purpose	• No padding • Drops last convolution if dimensions do not match	• Padding such that feature map size has size $\left\lceil \frac{I}{S} \right\rceil$ •Output size is mathematically convenient • Also called 'half' padding	• Maximum padding such that end convolutions are applied on the limits of the input • Filter 'sees' the input end-to-end

Filter hyperparameters

Parameter compatibility in convolution layer: By noting I the length of the input volume size, F the length of the filter, P the amount of zero padding, S the stride, then the output size O of the feature map along that dimension is given by:

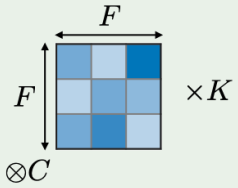
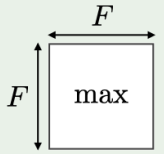
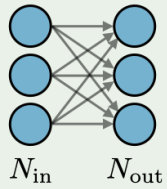
$$\boxed{O = \frac{I - F + P_{start} + P_{end}}{S} + 1}$$



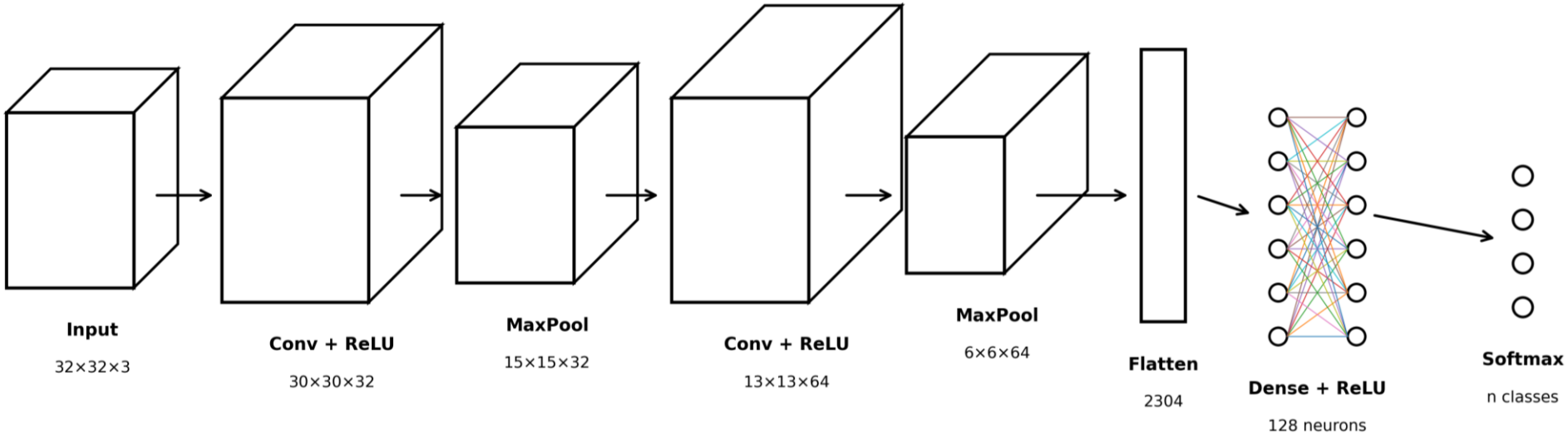
Remark: often times, $P_{start} = P_{end} \triangleq P$,in which case we can replace $P_{start} + P_{end}$ by $2P$ in the formula above.

Filter hyperparameters

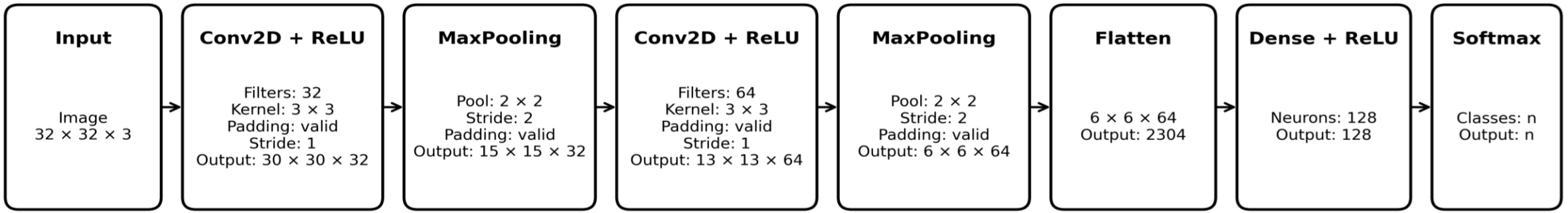
Understanding the complexity of the model: In order to assess the complexity of a model, it is often useful to determine the number of parameters that its architecture will have. In a given layer of a convolutional neural network, it is done as follows:

	CONV	POOL	FC
Illustration			
Input size	$I \times I \times C$	$I \times I \times C$	N_{in}
Output size	$O \times O \times K$	$O \times O \times C$	N_{out}
Number of parameters	$(F \times F \times C + 1) \cdot K$	0	$(N_{\text{in}} + 1) \times N_{\text{out}}$
Remarks	• One bias parameter per filter • In most cases, $S < F$ • A common choice for K is $2C$	• Pooling operation done channel-wise • In most cases, $S = F$	• Input is flattened • One bias parameter per neuron • The number of FC neurons is free of structural constraints

Simple CNN Architecture Example

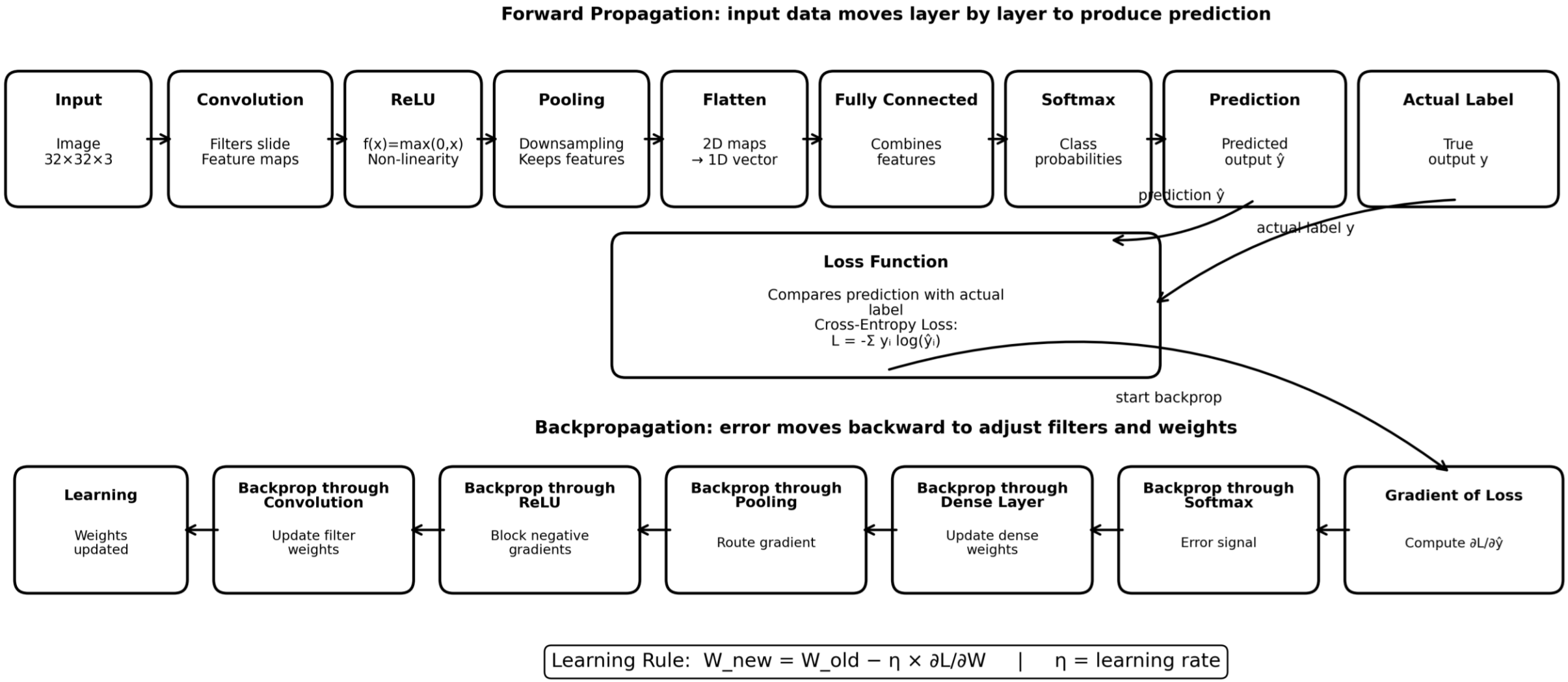


Architecture: Input → Conv(3×3, 32 filters, valid padding, stride 1) → MaxPool(2×2, stride 2)
→ Conv(3×3, 64 filters, valid padding, stride 1) → MaxPool(2×2, stride 2) → Flatten → Dense → Softmax

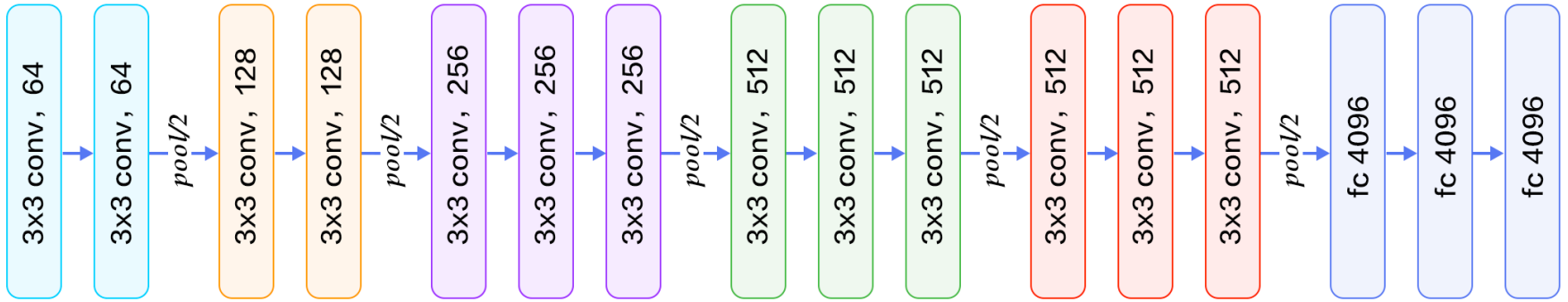


Convolution Output: ((Input Size - Kernel Size + 2 × Padding) / Stride) + 1
Pooling Output: floor((Input Size - Pool Size) / Stride) + 1

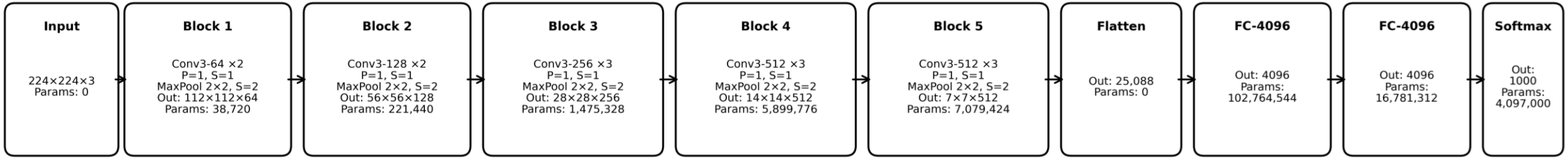
Forward Propagations and Backpropagations



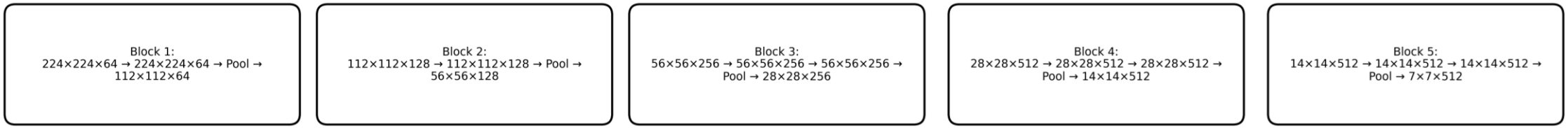
VGG Architectures



Convolution: 3×3 kernel, same padding, stride 1 | MaxPooling: 2×2 pool, stride 2



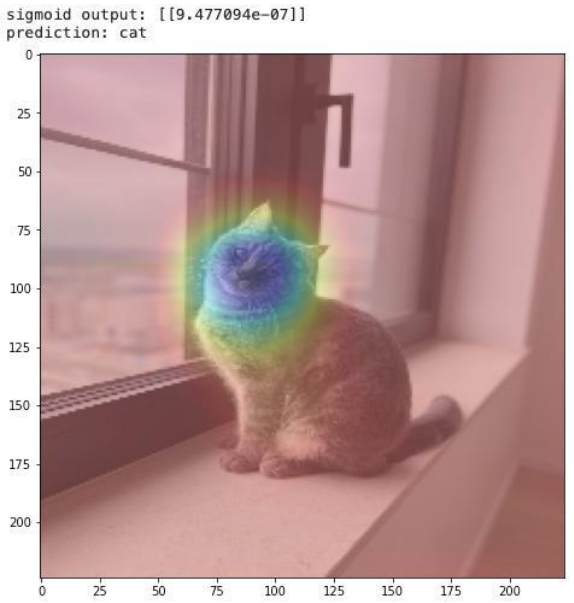
Detailed Layer Outputs



Total Parameters ≈ 138 million | Same padding keeps spatial size unchanged inside each convolution block | MaxPool reduces width and height by half

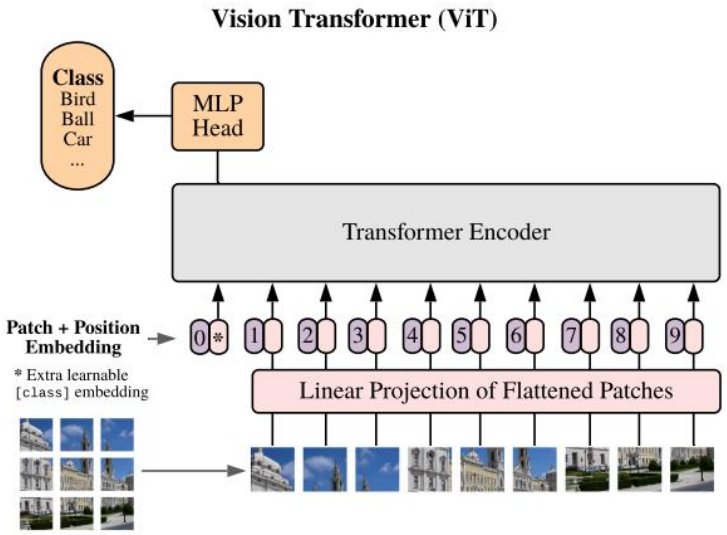
Attention Mechanism

Attention in computer vision means the model learns **which region of an image is important** for a specific task.
For example, if the task is to identify a dog in a photo, the model should focus more on the dog rather than the background trees or sky.



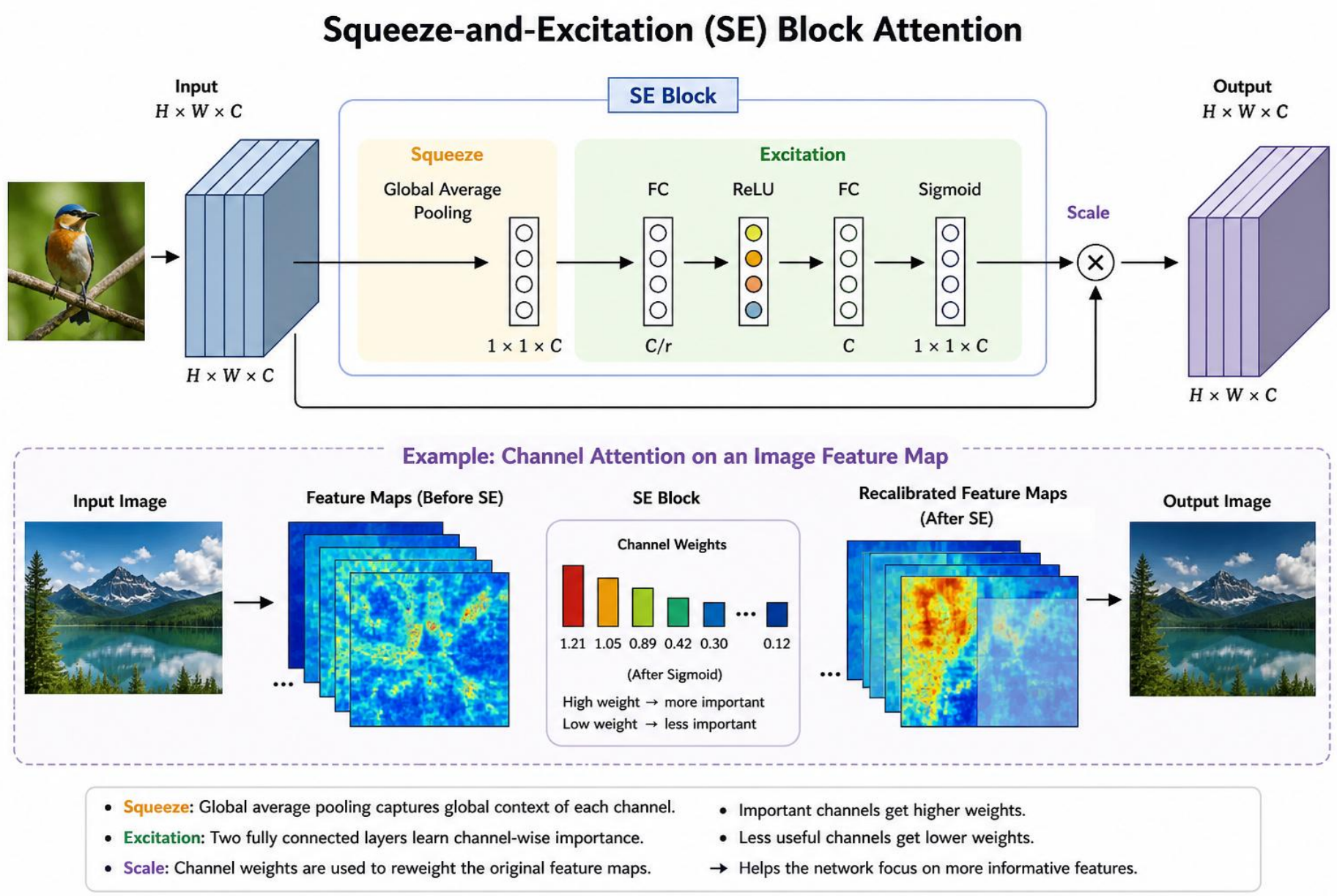
Attention in images works like human vision:

- Humans naturally focus on important regions.
- Attention mechanisms help AI models do the same thing computationally.



Self-Attention in Vision Transformers (ViT): Modern models like Vision Transformers divide an image into patches and learn relationships between different regions.

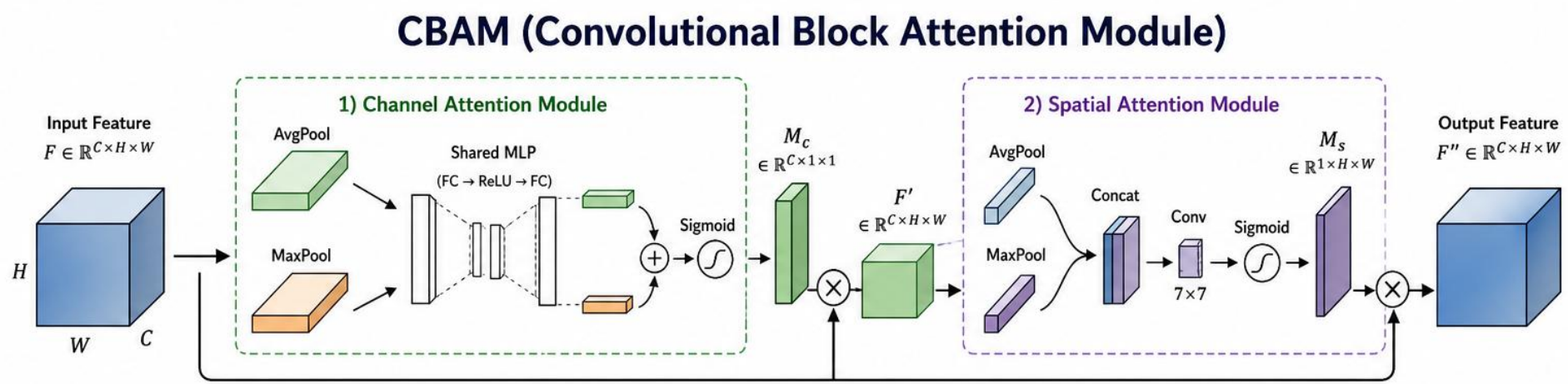
Attention Mechanism



- **Squeeze**: Global average pooling captures global context of each channel.
- **Excitation**: Two fully connected layers learn channel-wise importance.
- **Scale**: Channel weights are used to reweight the original feature maps.
- Important channels get higher weights.
- Less useful channels get lower weights.

→ Helps the network focus on more informative features.

Attention Mechanism



1) Channel Attention (What is important?)

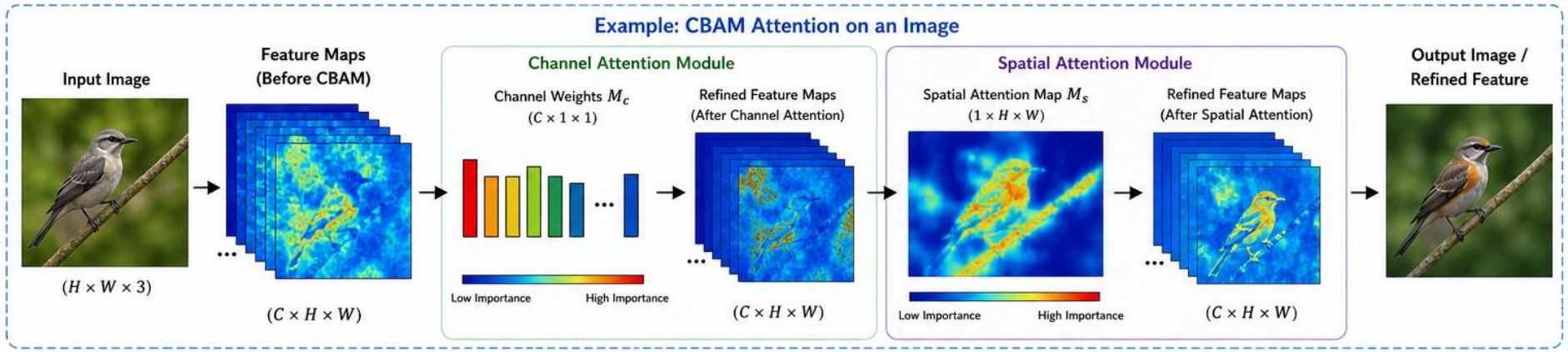
- Uses both AvgPool and MaxPool to capture rich channel-wise statistics.
- Shared MLP learns channel importance.
- Output M_c assigns a weight to each channel.

2) Spatial Attention (Where is important?)

- Uses AvgPool and MaxPool across channel to capture spatial information.
- 7×7 Conv generates spatial attention map M_s.
- Highlights important spatial locations.

Sequential Application

- First refine channels (global "what").
- Then refine spatial locations (local "where").
- Result: focus on informative features in both channel and spatial dimensions.



CBAM = Channel Attention + Spatial Attention (Sequential)
Helps the network focus on "what" is important and "where" it is important.

- Lightweight and easy to plug into existing CNNs (e.g., ResNet, MobileNet).
- Improves performance by suppressing irrelevant features and highlighting meaningful ones.

Transfer Learning and Fine-Tuning of Pre-trained CNNs

Transfer Learning

- Transfer Learning is a deep learning technique where knowledge from a pre-trained model is reused for a new task.
- Instead of training a CNN from scratch, an already trained model is adapted for another problem.
- Commonly used in computer vision tasks.

Pre-trained Knowledge→Adaptation→New Task Performance

Why Transfer Learning?

Training CNNs from scratch requires:

- Large datasets
- High GPU power
- Long training time

Transfer Learning Solves These Problems

- Reuses learned image features
- Faster training
- Better accuracy on small datasets
- Reduces overfitting

Benefit

- Faster convergence
- Better performance with limited data

Examples of Pre-trained CNNs

- ResNet
- VGG16
- MobileNet
- EfficientNet

General Workflow

1) Load pre-trained model
2) Remove original classifier layer
3) Add new classifier layer
4) Train on target dataset

Transfer Learning and Fine-Tuning of Pre-trained CNNs

Feature Extraction Approach

- Freeze convolution layers
- Train only the final classification layer

Characteristics

- Faster training
- Requires less data
- Lower computational cost

Suitable For

- Small datasets
- Similar tasks

Fine-Tuning Approach

- Unfreeze some pre-trained layers
- Retrain selected layers with new dataset

Purpose

- Adapt the model more specifically to the target task

Characteristics

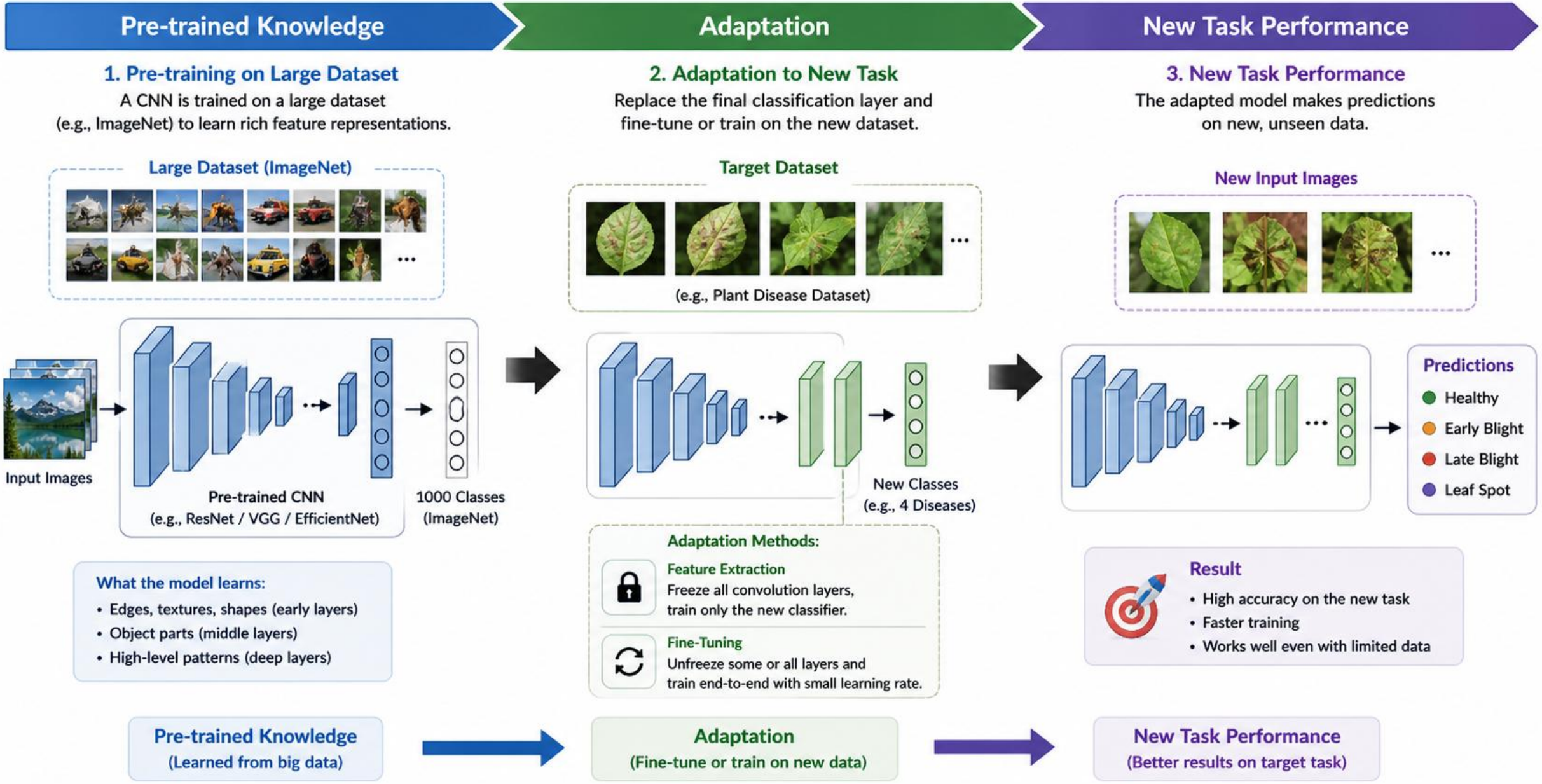
- Higher accuracy
- More training time
- Requires more data

Feature Extraction	Fine-Tuning
Freeze most layers	Retrain some layers
Faster training	Slower training
Small dataset sufficient	Larger dataset preferred
Lower computation	Higher computation
Less flexible	More adaptable

Benefits of Transfer Learning

- Faster model training
- Improved accuracy
- Requires fewer labeled images
- Reduces overfitting
- Saves computational resources

Transfer Learning and Fine-Tuning of Pre-trained CNNs



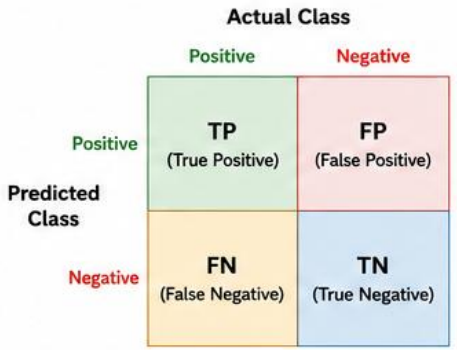
Performance Evaluation

1. Classification Metrics

Accuracy	Overall correctness	$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$
Precision	How many predicted positives are correct	$\text{Precision} = \frac{TP}{TP + FP}$
Recall (Sensitivity)	How many actual positives are detected	$\text{Recall} = \frac{TP}{TP + FN}$
F1 Score	Harmonic mean of Precision and Recall	$F1\ \text{Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$
Specificity	How many actual negatives are correctly identified	$\text{Specificity} = \frac{TN}{TN + FP}$

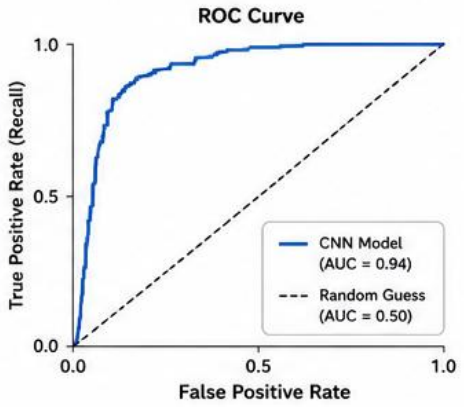
TP: True Positive **TN:** True Negative **FP:** False Positive **FN:** False Negative

2. Confusion Matrix



Provides detailed insight into model performance on each class.

3. ROC Curve and AUC

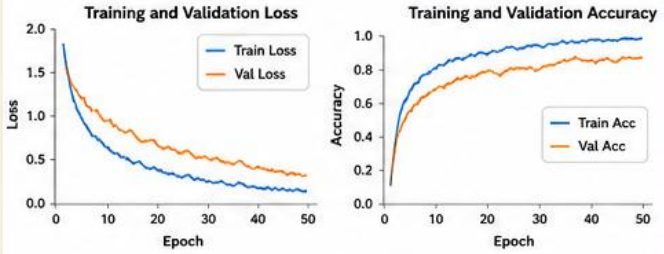


ROC Curve: Shows trade-off between TPR and FPR at different thresholds.

AUC (Area Under Curve): Measures the ability of the model to distinguish between classes.

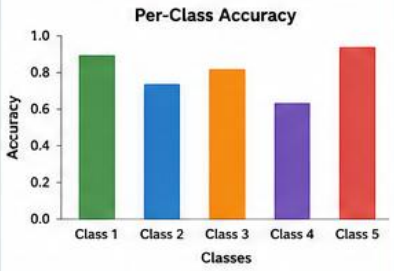
AUC = 1.0 → Perfect
AUC = 0.5 → Random

4. Loss and Accuracy Curves



- Train and validation curves help detect overfitting or underfitting.
- Good model: Low loss and high accuracy on both training and validation.

5. Per-Class Performance

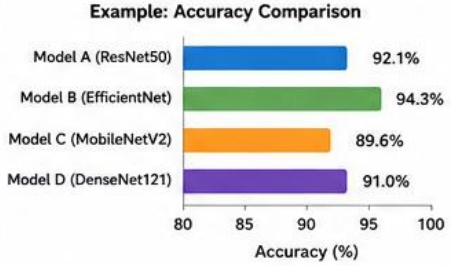


Helps identify classes where the model performs well or poorly.

6. Other Useful Metrics

- **Top-K Accuracy:** True label in top K predictions.
- **MCC (Matthews Correlation Coefficient):** Balanced metric for imbalanced datasets.
- **Balanced Accuracy:** Average of recall obtained on each class.
- **Log Loss (Cross Entropy):** Penalizes confident wrong predictions.

7. Model Comparison



Compare multiple models using the same metrics on the same test set.

Evaluation Workflow

Best Practices

- Use a separate test set for final evaluation.
- Look at multiple metrics, not just accuracy.
- Check per-class performance for imbalanced data.
- Use training curves to monitor overfitting or underfitting.
- Compare models and choose the best trade-off.

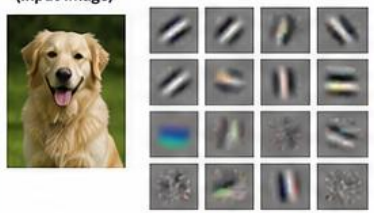
Key Takeaway: Rely on a combination of metrics, curves, and confusion matrix to get a complete understanding of your CNN model's performance.

Model Interpretation and Visualization Techniques

1. Filter/Feature Visualization

Visualizes what each convolution filter has learned (patterns that activate it).

Example (Input Image) | Learned Filters (First Conv Layer)



Insight: Early layers learn edges, colors, and textures.

2. Activation Maps (Feature Maps)

Shows the activation output of filters at different layers for a given input.

Input Image | Activation Maps (Layer Example): Conv1, Conv2, Conv3

Insight: Deeper layers capture more complex and abstract features.

3. Class Activation Map (CAM)

Highlights discriminative regions used by the model to predict a particular class.

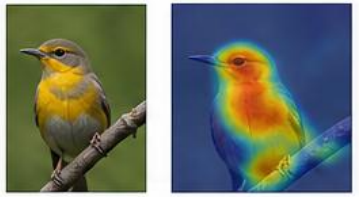
Input Image | CAM Heatmap (for "German Shepherd") | Overlay on Original Image

Insight: Model focuses on the dog (not the background) for the correct prediction.

4. Grad-CAM

Uses gradients from the target class to produce a coarse localization map.

Input Image | Grad-CAM Heatmap (for "Bird")



Insight: Important region (bird body and head) influences the prediction.

5. Guided Backpropagation

Backpropagates the gradients to the input image to show important pixels.

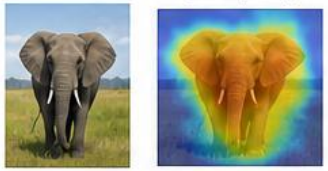
Input Image | Guided Backprop (Edge-like Saliency)

Insight: Shows fine-grained details (edges and important pixels).

6. Grad-CAM++

Improved version of Grad-CAM that provides better localization.

Input Image | Grad-CAM++ Heatmap (for "Elephant")

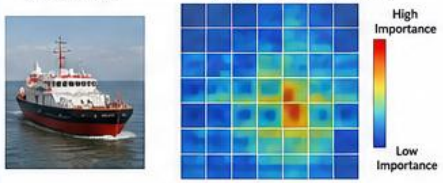


Insight: More accurate localization of the entire elephant.

7. Occlusion Sensitivity

Part of the image is occluded (patched) to see how prediction changes.

Input Image | Occlusion Sensitivity Map (Higher = More Important)



Insight: Red regions are critical; occluding them reduces confidence.

8. LIME (Local Interpretable Model-agnostic Explanations)

Explains individual predictions by approximating the model locally and highlighting important super-pixels.

Input Image | LIME Super-pixels (Highlighted = Important) | Explanation Heatmap

Insight: Shows which regions (super-pixels) are most influential for the prediction.

9. Integrated Gradients

Attributes the prediction to each pixel by integrating gradients along the path from a baseline (e.g., black image) to the input.

Input Image | Integrated Gradients (Attribution Map)

Insight: Red pixels push prediction higher; blue pixels push it lower.

How These Techniques Help

- Understand what the model learns
- Identify important regions
- Increase model transparency
- Detect model bias or failure cases
- Build trust and reliability

Best Practices

- Use multiple techniques together for better understanding.
- Validate with domain knowledge.
- Interpret both correct and incorrect predictions.
- Be careful: explanations are model-dependent, not absolute truth.

Popular Tools and Libraries

TensorFlow (tf-explain) | Keras (Visualizations) | Pytorch (Captum) | OpenCV (Heatmaps) | SHAP | LIME

Key Takeaway: Visualization helps open the "black box" of CNNs, improves interpretability, and supports better decision-making.