A Tutorial on Convolutional Neural Networks through Conceptual Discovery and Minimal Examples for Early AI Learners

M. SABRIGIRIRAJ[1], K. MANOHARAN[2]
[1]Department of Information Technology,
Hindusthan College of Engineering and Technology, Coimbatore,
INDIA

[2]Department of ECE, SNS College of Technology, Coimbatore,
INDIA

Abstract: - Convolutional Neural Networks, or CNNs, are now common in image-based artificial intelligence. They are used for image classification, object detection, and many other computer vision tasks. Still, CNNs are not always easy for beginners to understand. In many books and online materials, the topic is introduced through equations, matrix operations, or long coding examples. These details are important, but they can make the first stage of learning difficult. This tutorial therefore starts with the basic idea of how a computer reads an image as pixel values. It then uses small examples to show how a CNN can learn patterns from those values. The discussion covers local image regions, convolution, pattern checking, shared weights, feature maps, activation functions, pooling, and layered feature learning. It also explains why CNNs use repeated convolution and pooling blocks, and why feature extraction is different from the final classification step. A brief section on transfer learning is included to show how features learned from one image task can be useful in another task. The main purpose of this article is to give beginners a clear and practical understanding of CNNs before they study the mathematical details or start implementation.

Key-Words: - Convolutional Neural Networks, Artificial intelligence, Neural Networks

Received: December 5, 2025. Revised: March 9, 2026. Accepted: May 6, 2026. Published: July 1, 2027.

1 Introduction

Images are used in many artificial intelligence applications. These include object recognition, medical image analysis, agricultural monitoring, and autonomous systems. For these tasks, Convolutional Neural Networks, or CNNs, are one of the most commonly used models. A typical CNN architecture is shown in Figure 1. Because CNNs are important in computer vision, they are now included in many basic courses on deep learning and image processing. However, many beginners find CNNs difficult to understand. This is because CNNs are often taught using equations, matrix operations, and coding details. These details are useful at a higher level. But they may be difficult for students who are learning image-based machine learning for the first time. Some students may follow the formulas, but still may not clearly understand what a CNN does to an image. They may also find it hard to understand why the layers are placed in a particular order. This can make learning slow and less interesting.

This tutorial explains CNNs using a concept-first approach. This means that the basic ideas are explained before the mathematics and programming details. The tutorial first explains how an image is stored as pixel values. It then introduces the main parts of a CNN, such as convolution, feature maps, pooling, and hierarchical feature learning. Each idea is explained with small examples and simple reasoning. This order allows readers to follow the idea in a gradual way. Rather than beginning with equations, the tutorial shows what happens to an image as it passes through a CNN layer by layer. The purpose is to give readers a clear picture of the process inside the network.

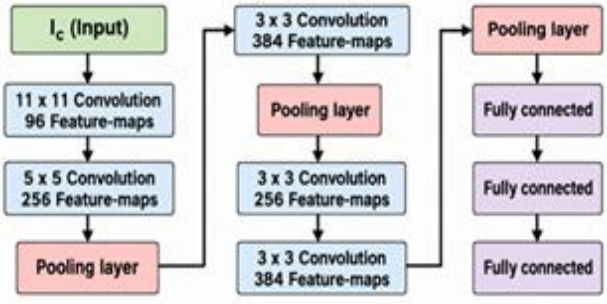

Fig. 1: A typical CNN

The article is mainly intended for undergraduate students, teachers, and self-learners who are new to CNNs. It does not go deeply into mathematical derivations, optimization methods, or complete coding procedures. Those topics are important, but they are better studied after the basic idea is clear. For this reason, the paper gives more attention to the working logic of CNNs than to formal details.

The main contribution of this work is its teaching approach. It introduces CNNs in a concept-first manner, with limited dependence on mathematical formalism. Second, it uses small numerical examples to explain the role of each major CNN component. Third, the tutorial provides an intuitive explanation of important architectural design choices such as network depth, pooling layers, and transfer learning, especially for beginners who are new to deep learning.

1.1 Literature survey

Several surveys and tutorials describe CNN architectures, components, and applications. A concise description of CNN components is presented in [1], but the discussion mainly follows formal definitions and standard explanations, with limited emphasis on intuition or visual understanding of how CNNs process images. Similarly, survey papers on CNNs and transfer learning [2], [3], as well as general overview papers [4], [5], mainly focus on architectures, performance, and applications. These papers are useful for understanding the field, but they do not usually guide beginners through the basic operations such as convolution, feature extraction, and architectural design choices in a step-by-step manner.

Some tutorial-style papers [6]–[11] present mathematical derivations, implementation details, or specific applications such as one-dimensional CNNs. While these works are valuable, they can still be mathematically heavy or abstract for beginners who are trying to build an intuitive understanding of CNN behavior. A few works attempt a more concept-focused explanation. For example, [12] explains CNNs from first principles and includes numerical examples and visualizations. However, its matched-filter interpretation may be difficult for readers who do not have a background in signal processing. Interactive tools such as CNN Explainer [13] allow users to explore convolution and layer activations through visualization, which helps in understanding the internal behavior of CNNs. However, these tools require access to the interface and are usually used as supplementary learning tools rather than as a primary textual explanation. Overall, although many surveys and tutorials exist, very few provide a simple, step-by-step, concept-first explanation that is suitable for complete beginners.

1.2 Contribution and Difference from Previous Tutorial Work

The authors have previously published tutorial-oriented work related to machine learning and introductory neural networks. However, the present manuscript is different in scope and teaching approach. The earlier work discussed general machine learning concepts and basic neural network ideas, whereas this paper focuses specifically on Convolutional Neural Networks and explains their architecture through a step-by-step conceptual approach using small visual and numerical examples. The main purpose of this work is to explain CNN concepts in a gradual and easy-to-follow manner. The discussion begins with how an image is represented in a computer and how small local patterns are detected. It then moves step by step to convolution, feature maps, activation functions, pooling, hierarchical feature learning, and transfer learning. Each idea is explained with very small examples. This makes it easier for beginners to see how a CNN works and why each part of the network is included.

Many CNN tutorials begin with equations or coding details. This paper takes a different route. It first explains the basic idea behind each step. Once this idea is clear, the formal mathematics becomes easier to follow. The tutorial uses a simple order, small examples, and short explanations of design choices. For this reason, the manuscript should be read as a teaching-oriented tutorial. Its main purpose is to explain the basic structure and working of CNNs. It is not intended to provide a broad introduction to machine learning or deep learning.

2 Why Images Need a Different Kind of Network?

Artificial Neural Networks (ANNs) are designed to process numerical data, treating each input as independent. While they work reasonably well for tabular data, they are not well suited for images. A CNN can learn from an image if the image is expressed in a form the network can process. An image is expressed as a structured grid of numbers (referred to as tensors), where each number represents the intensity of a pixel. A grayscale image is represented as a two-dimensional array. A color image is represented as three stacked arrays for 3 colour channels with one channel each for red, green, and blue. These stacked grids preserve the spatial

arrangement of pixels, allowing neighboring values to remain neighbours in the network's input. By keeping this spatial structure intact, CNNs exploit the local patterns and relationships in the image. If the image were flattened into a single list of numbers, local patterns and relationship cannot be preserved.

An image is a structured arrangement of pixels, where the position of each pixel relative to its neighbours carries vital information. When an image is given as input to a CNN, its pixel values are usually scaled to a smaller range, such as 0 to 1. This gives the network a more stable input during training. It also reduces the effect of very large pixel values. In color images, the red, green, and blue channels are kept together, so the original image structure remains intact. Flattening image pixels for an ANN ignores this spatial structure, leading to inefficient learning and poor generalization.

Example 1:

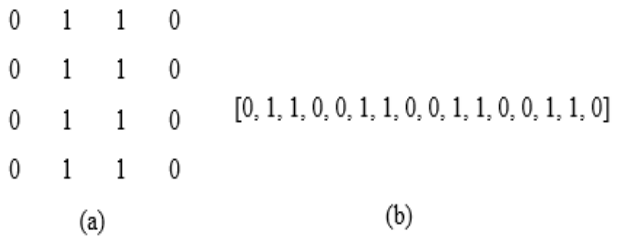

Fig. 2: - Image A (4×4 grid) and Fig. 2(b) - Flattened vector for ANN

Image B (shifted right by 1 pixel):

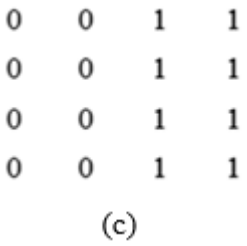

Fig. 2: - Image B (Image A shifted right by 1 pixel)

Figure 2 shows a tiny grayscale image with a vertical line in the centre. The meaning of each pixel depends on its arrangement with neighbours. However, when the image is flattened as in Figure 2, the ANN receives sixteen numbers with no hint of their spatial arrangement. Two adjacent pixels in the image are treated the same as two pixels far apart. Figure 2 shows an Image B, which contains the same vertical line shifted one pixel to the right. Although Images A and B represent the same pattern, their flattened vectors differ completely. An ANN would need to learn two separate representations, making learning inefficient and sensitive to small spatial changes.

This example highlights a fundamental limitation that ANNs do not naturally understand spatial structure. Images, however, require a model that recognizes local patterns and treats the same pattern similarly, regardless of where it appears. CNNs are particularly designed to address this need by preserving and exploiting spatial relationships within image data.

3 Seeing Small Parts Instead of the Whole Image

Humans do not understand an image by looking at all its pixels at the same time. In practice, visual perception begins with small and simple details such as edges, corners, or short lines. These local details are then combined in the brain to form a complete understanding of the image.

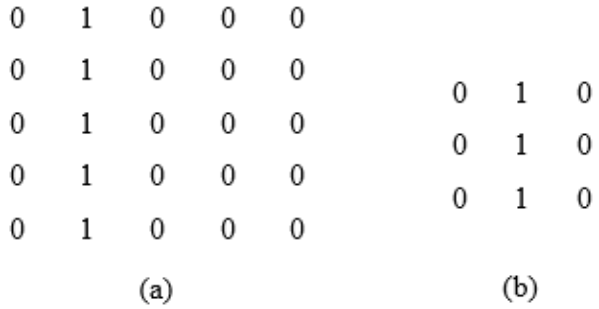

Fig. 3: – Image C Figure 3 – Local 3×3 region from Image C

For this reason, a learning system that works with images should also focus on local information rather than processing the entire image at once. Although the vertical line occupies only a small part of the image, it can be easily identified by a human observer. This recognition does not require analysing all 25 pixels simultaneously. Instead, the pattern becomes obvious when attention is given to small groups of neighbouring pixels. Figure 3 highlights a 3×3 region extracted from the image.

This 3×3 patch alone contains enough information to indicate the presence of a vertical line. Similar local patches can be found at different positions in the image, and each of them provides the same local evidence. By checking these small regions one after another, the network can begin to understand the overall structure of the image. This example shows that images can be understood more clearly by looking at local neighbourhoods and finding simple patterns within them. A learning system can therefore focus on local regions to detect

meaningful features, instead of processing the entire image in a single step. This idea forms the basis of convolution, where the same local pattern detector is applied across different parts of the image to gradually build a higher-level understanding.

4 Convolution as Simple Pattern Checking

The previous section discussed how important image patterns can be understood by focusing on small local regions. The next question is how a learning system can automatically search for such patterns across the entire image. Convolution offers a simple and effective solution to this problem. At its core, convolution involves applying the same small pattern check repeatedly at different locations within an image. For simplicity, bias terms and padding are ignored in this example.

Example 2

Figure 4 shows a 5×5 grayscale image containing a vertical line slightly to the left of the centre. Suppose the task is to determine whether a small region of the image contains a vertical line. One simple approach is to examine neighbouring pixels within a 3×3 region. Figure 4 shows such a pattern, commonly referred to as a filter or kernel.

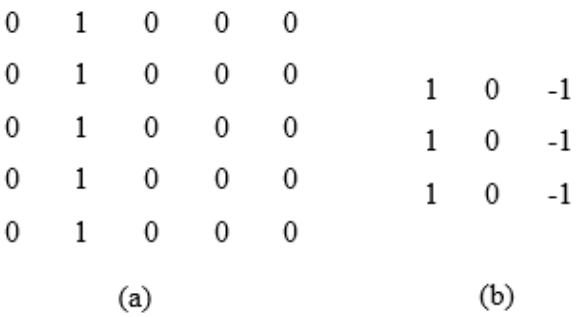

Fig. 4: – Image D and Figure 4 – Pattern Checker (Filter)

This pattern checker is designed to respond strongly when the pixel values on the left side of the region differ from those on the right side, which is characteristic of a vertical line. The same 3×3 checker is first placed over the top-left part of the image, where it produces a numerical response, as illustrated in Figure 4. The checker is then shifted one position to the right and applied again. This shifting process continues until all possible 3×3 regions of the image have been examined. At locations where the image region closely matches the pattern checker, the response value is large. In regions where the pattern is absent, the response is small or close to zero. Figure 4 shows these response values as a new grid. This grid shows where the vertical line pattern appears in the image.

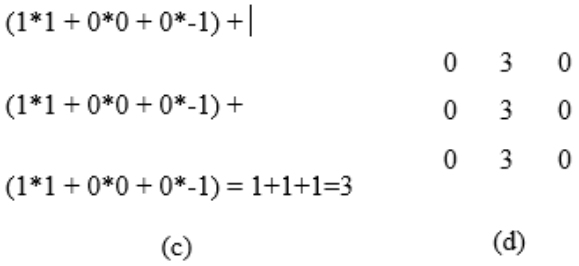

Fig. 4: – illustration of first step in convolution and Figure 4 Output of convolution operation

In this way, convolution can be understood as a repeated check for small local patterns across the image. A small filter moves over the image step by step and responds when it finds a matching pattern. By recording these responses, the network can identify where important visual features are present in the image. This simple mechanism forms the foundation of convolutional neural networks and enables them to analyse images in a structured and efficient manner. It is to be noted that although these pattern checker/filters are shown manually for clarity, in practice CNNs learn such patterns automatically during training.

5 Why the Same Filter Is Reused (Weight Sharing)

The previous section described convolution as a repeated pattern-checking operation. The next question is why a convolutional neural network uses the same pattern checker at every location in the image. The answer lies in the idea of weight sharing, which is central to the effectiveness of CNNs and can be understood through simple reasoning.

Example 3

Both images clearly contain the same visual pattern, namely a vertical line. The only difference between them is the position of that line within the image. The vertical line in both Image E and Image F can be detected using the same pattern checker shown in Figure 5. Figure 5 shows a 5×5 grayscale image containing a vertical line located near the centre. Figure 5 shows another image in which the same vertical line appears closer to the right edge.

0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0

(a)

0	0	0	1	0
0	0	0	1	0
0	0	0	1	0
0	0	0	1	0
0	0	0	1	0

(b)

Fig. 5: – Image E and Figure 5 – Image F

2	0	-2
2	0	-2
2	0	-2

(c)

Fig. 5: – Pattern Checker (Filter)

If different pattern checkers were used at different image locations for the same task, such as detecting vertical edges, the network would have to learn multiple versions of the same concept. This would introduce unnecessary redundancy, because the visual pattern itself does not change with its position in the image. A vertical line remains a vertical line whether it appears at the centre, near an edge, or in a corner.

For this reason, a CNN uses a single pattern checker to detect a specific feature and applies it across the entire image. This approach is known as weight sharing. Reusing the same filter in this way provides two important advantages. First, the network needs to learn far fewer parameters, making learning more efficient. Second, the network can detect the same visual pattern in a similar way, no matter where it appears in the image. This helps CNNs work well on new images and makes them different from traditional artificial neural networks. Second, the network responds to the same visual pattern in a consistent manner regardless of its location. This principle enables CNNs to generalize well and represents a key distinction between convolutional neural networks and traditional artificial neural networks. When a convolutional filter is applied to an image, it does not examine all pixels at once but moves step by step across the image. The number of pixels by which the filter shifts at each step is known as the stride. A smaller stride results in more overlap between neighbouring regions and produces a larger feature map. A larger stride reduces the number of positions examined and leads to a smaller output. At the image boundaries, the filter may not fully fit within the image area. In such cases, the image can be conceptually extended by adding extra pixel values around its edges, a process commonly referred to as padding. Padding allows patterns near the borders of the image to be treated in the same way as those near the centre. Together, stride and padding control how much the spatial size of the image changes during convolution.

6 What the Image Becomes After Convolution (Feature Maps)

In the previous sections, the convolution operation was described as applying a pattern checker (filter) across an image. After this operation, the result is no longer the original image. Instead, it is a new grid of values that indicates where a particular pattern has been detected. This new grid is called a feature map. Understanding feature maps is important because they represent how the network views an image after checking for a specific pattern.

Example 4

Figure 6 shows a 5×5 grayscale image containing a vertical line located near the centre.

0	1	0	0	0
0	1	0	0	0
0	1	0	0	0
0	1	0	0	0
0	1	0	0	0

(a)

0	3	0
0	3	0
0	3	0

(b)

Fig. 6: – Image G and Figure 6 – Image H (Feature Map)

A vertical-line filter, such as the one shown in Figure 4, is applied across this image. At each position, the filter looks at a small local region of the image and produces a response depending on how well that region matches the vertical pattern. As the filter moves across the image, it does not give only one output valueInstead, it creates a new image made of response values, as shown in Figure 6.

In this feature map, higher values appear where the vertical pattern is clearly detected. Lower values appear where the pattern is weak or absent. The feature map is different from the original image. It does not show brightness or color. It shows how strongly a specific pattern is found at each location. A high value means the pattern is strongly present. A low value means the pattern is weak or not present. The feature map is usually smaller than the original image because the filter examines small regions rather than individual pixels.

Each value in the feature map therefore represents information summarized from a small area of the original image. In practice, multiple filters are applied to the same image. Each filter produces its own feature map. For example, one filter may detect vertical lines, another may detect horizontal lines, and a third may respond to corners. Together, these feature maps provide different perspectives of the same image by highlighting different visual properties. By transforming the original image into a collection of feature maps, a convolutional neural network converts raw pixel data into meaningful visual information that can be processed in later layers.

7 From Detection to Decision (Activation Functions)

In the previous sections, convolution was shown to perform the role of a simple pattern checker. Convolution produced feature maps that indicated where certain patterns like edges are present in an image. However, detecting a pattern is not the same as deciding whether it is important or not. This distinction of deciding whether it is important or not is dealt using an activation function.

Consider the scenario of edge detection without an activation. Assume a convolution filter is designed to detect vertical edges. When this filter is applied to different regions of an image, it produces numerical responses that indicate the strength of the detected edge. Let the convoluted output be [-2 0 1 4]. It is to be noted here that large positive values indicate a strong presence of the pattern, while a small or near-zero values indicate weak or no presence. Also, negative values indicate the opposite pattern. A negative response does not mean the feature is useless, but that it represents an inverted contrast. At this stage, the network has only measured pattern strength. All responses, including weak and negative ones, are passed forward. However, if such linear outputs were passed directly to the next layers, weak and noisy detections would mix with strong ones. Also, stacking multiple layers would not increase expressive power and the network would struggle to form meaningful higher-level features. In effect, the network would be aware of everything but selective about nothing. The following Example 5 demonstrates the significance of applying an activation function like ReLU (Rectified Linear Unit). ReLU is defined as: $\mathrm{ReLU}(x) = \max(0, x)$

Example 5

Input:	-2	0	1	4
After ReLU:	0	0	1	4

It can be easily observed from the above, that the activation function has performed a simple decision here. Strong positive responses are preserved while weak or negative responses are suppressed. This transforms the feature map from a measurement into a selection. Convolution can be thought of as “Is this pattern present, and how strongly?” while Activation can be interpreted as “Is this pattern important enough to keep?”. Only after this decision is made does it make sense to apply pooling or combine features into more complex structures.

Activation functions introduce non-linearity into the network. Without this non-linearity, multiple convolution layers would behave like a single linear filter. Without activation functions, the network would find it difficult to combine simple patterns into more complex shapes. In that case, deeper layers would not be able to learn useful hierarchical features. Activation functions help the network keep important signals and reduce the effect of less useful ones before passing information to the next layer. This allows deeper layers to gradually learn more abstract and meaningful representations.

8 Why CNNs Shrink Images on Purpose (Pooling) and putting it all together

In the previous section, we saw that convolution produces feature maps that show where certain patterns appear in an image. After this step, these feature maps are usually reduced in size using a process called pooling. At first, this may seem strange because pooling removes some information from the feature maps. However, this step is very useful because it makes convolutional neural networks more robust and also reduces computational cost.

Example 6

Figure 7 shows a small feature map that could result from detecting a vertical line in an image.

1	2	1	0
2	4	2	1
1	3	1	0
0	1	0	0

(a)

4	2
3	1

(b)

Fig. 7: – Feature Map Before Pooling and Figure 7 – Feature Map After 2×2 Max Pooling

A 2×2 max pooling operation is applied to this feature map, producing the result shown in Figure 7b).

In max pooling, the operation examines each 2×2 region of the feature map and retains only the largest value from that region. Each value in the pooled feature map therefore summarizes the most significant response within its corresponding area of the original feature map.

Pooling serves two main purposes. First, it reduces the spatial size of the feature maps, which decreases the amount of computation required in subsequent layers. Second, pooling makes the network less sensitive to small changes in the position of patterns within the image. For example, if a vertical line shifts slightly, the exact location of the highest response in the feature map may change. After pooling, the strongest response in a small local region is kept. This helps the network detect a pattern even when its position changes slightly in the image. In simple terms, pooling helps the network pay more attention to the presence of a pattern than to its exact location. This is somewhat similar to human perception, where small shifts in an object's position usually do not affect how we recognize the object.

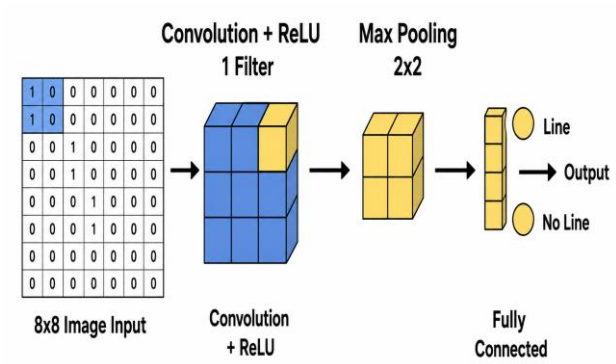

Fig. 8: CNN with a single filter

Although pooling reduces some spatial detail, this controlled reduction often helps CNNs generalize better to new and unseen images. While pooling was an important component in early CNN architectures, some modern networks replace pooling with strided convolutions in order to retain more information. Figure 8 shows a simple CNN with a single filter used to detect an edge in an image, without the need for training.

9 Why CNNs Use Multiple Convolution and Pooling Layers

Convolutional Neural Networks (CNNs) learn to understand images by progressively detecting visual patterns, from simple to complex. This is achieved by stacking multiple convolution and pooling layers, each with multiple filters of the same or different sizes. For example, consider a 6×6 image of a plus symbol ("+") as shown in Figure 9:

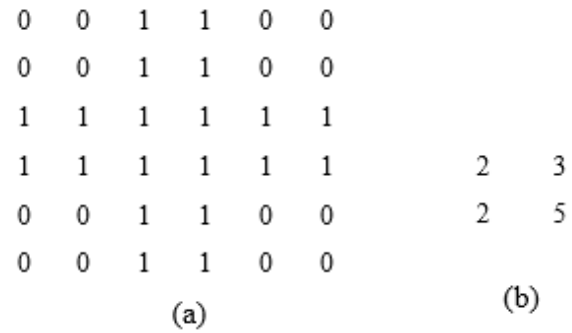

Fig. 9: a) 6×6 image of a plus symbol ("+") and Fig. 9(b) Application of max-pooling to a random feature map

A convolution layer applies several filters simultaneously, each acting as a dedicated pattern detector. One filter may detect vertical lines, another horizontal lines, and yet another corners. Smaller filters capture fine details like the thin vertical stroke of the plus, while larger filters may capture the full cross shape. By passing the output of one convolution layer to the next, the network slowly builds a more detailed understanding of the image. The first layers detect simple patterns. The deeper layers combine these simple patterns to identify larger and more complex structures. In the end, the network can recognize the complete object.

Pooling layers are usually placed between convolution layers. They keep the most important information while reducing the size of the feature maps. They also make the network less affected by small changes in the position of a pattern. For example, when max-pooling is applied to the feature map in Figure 9, it produces a maximum value of 5. This value represents the strongest feature detected in that region.

After several convolution and pooling steps, the network creates a group of feature maps that describe the important patterns in the image. At this stage, the process moves from feature detection to decision making. The feature maps are flattened into a single vector and passed to fully connected layers. These layers combine the detected features and use them to classify the image. The final layer changes the output scores into class probabilities using a function such as softmax. This helps the network express its confidence for each class. In simple terms, convolution layers search for useful features, while fully connected layers use those features to make the final decision.

This layered structure allows CNNs to build understanding gradually. The early layers may respond to edges and lines. The middle layers can join these responses into simple shapes. The deeper layers can then form more meaningful object parts. Because of this, the network can recognize full objects, such as digits or symbols, even though no single layer handles the whole object by itself. Each layer adds something to the representation. By placing convolution and pooling layers one after another, CNNs turn raw images into compact features that are useful for recognition.

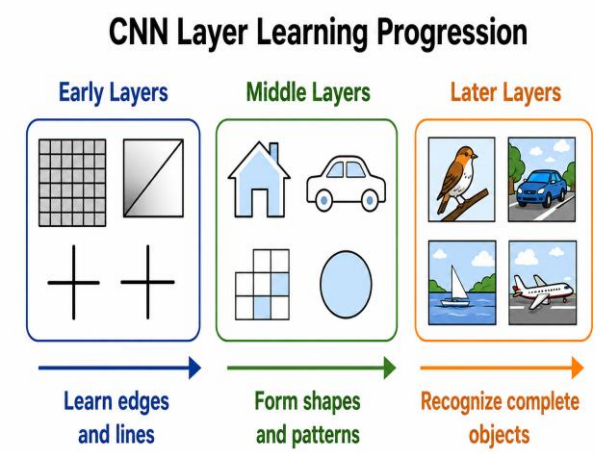

Fig. 10: Progressive Learning by CNNs Layers

10 Mathematical View of Basic CNN Operations

Although this tutorial is mainly written for basic understanding, a few simple formulas are included here. They help connect the visual idea with the formal form used in CNNs.

Convolution Operation

Convolution can be written as a weighted sum between a small part of the image and a filter:

$$S(i,j) = \sum_m \sum_n I(i+m, j+n) \cdot K(m,n)$$

Here, I represents the input image, K represents the convolution filter or kernel, and $S(i,j)$ represents one value in the output feature map.

This formula means that the filter is placed on a small image region. Each filter value is multiplied with the matching pixel value. The multiplied values are then added together. The same process is repeated across the image. In simple terms, convolution is a local weighted sum.

Activation Function (ReLU)

The Rectified Linear Unit, or ReLU, is often written as:

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

This function changes negative values to zero. Positive values are kept as they are. This makes the output easier for the next layer to use.

Output Size After Convolution

The output size after convolution can be calculated as:

$$Output = \frac{N - F + 2P}{S} + 1$$

Here, N is the input size, F is the filter size, P is the padding, and S is the stride.

This formula shows how the size of the feature map is decided. The output may become smaller, stay the same, or change depending on the filter size, padding, and stride.

Learning Through Loss Minimization

During training, the network adjusts filter weights to minimize a loss function:

$$Loss = \sum (y_{true} - y_{pred})^2$$

The weights are updated using gradient descent:

$$w_{new} = w_{old} - \eta \frac{\partial Loss}{\partial w}$$

where η is the learning rate.

These equations provide a mathematical interpretation of the conceptual ideas discussed earlier in the tutorial.

11 How CNNs Learn and Improving Filters Through Feedback

In the earlier sections, convolution filters were illustrated as just simple pattern checkers and detectors for vertical or horizontal lines. Now, an important question that naturally arises is "**how does a CNN learn the right filters automatically, instead of relying on manually designed ones?**" The answer lies in learning through feedback. During training, the network gradually improves its filters based on how well its predictions match the correct labels. At the beginning of training, the filters in a CNN are initialized with random values. When an image is passed through the network, these random filters usually produce weak or meaningless responses, which often leads to incorrect predictions. The prediction is then compared with the true label

using a loss function, which measures how wrong the prediction is. The loss value works like a feedback signal. It tells the network whether the patterns it detected helped to make the correct prediction or not.

This can be understood with the simple vertical-line filter discussed earlier. Before training, the filter is initialized with random values. At this stage, it may respond only weakly to a vertical line. For example, it may produce values such as:

Before training response: 0.3, 0.5, 0.4

These values show that the filter is not yet strongly matched with the vertical pattern. After the network calculates the prediction error, it slightly changes the filter values. The aim is to make the filter respond more strongly when the correct pattern is present and respond less when the pattern is not useful. After many updates using many training images, the same filter may produce stronger responses such as:

After training response: 2.1, 2.4, 2.2

This shows that the filter has slowly learned to match the vertical structure better. In simple terms, filters that help reduce the prediction error become stronger, while filters that do not help become weaker. This adjustment is done through backpropagation. In this process, the network sends the prediction error backward through its layers. The network then uses this error to update its parameters. Each filter is adjusted based on whether a stronger or weaker response would have helped the network make a better prediction.

It is important to note that the network is not explicitly told what a "vertical line" is. Instead, the filter gradually becomes a vertical-line detector because detecting that pattern consistently helps reduce the classification error. Over time, useful patterns are reinforced, and less useful patterns disappear.

As training continues, filters in the early layers begin to detect simple patterns such as edges and lines, similar to the manually illustrated examples discussed earlier. The deeper layers then combine these simple patterns to detect more complex structures. This connects back to the earlier concept-first examples in the tutorial: The pattern checkers discussed earlier are not hand-made rules. They are learned by the network through repeated error correction. During training, the CNN keeps adjusting its filters based on the feedback it receives from the prediction error. Over time, these filters move away from random values and become useful feature detectors. In this way, a CNN slowly learns to move from raw pixel values to a better understanding of visual patterns.

12 Transfer Learning in CNN

Training a CNN from the beginning usually needs a large labeled dataset and strong computing resources. However, many visual patterns such as edges, corners, textures, and simple shapes are common across a wide range of image recognition tasks. Transfer learning exploits this property by reusing feature representations learned from a large source dataset and adapting them to a new target task. In transfer learning, two common strategies are used: feature extraction and fine-tuning. In feature extraction, the convolutional base of a pre-trained network is frozen and only the final classification layers are trained.

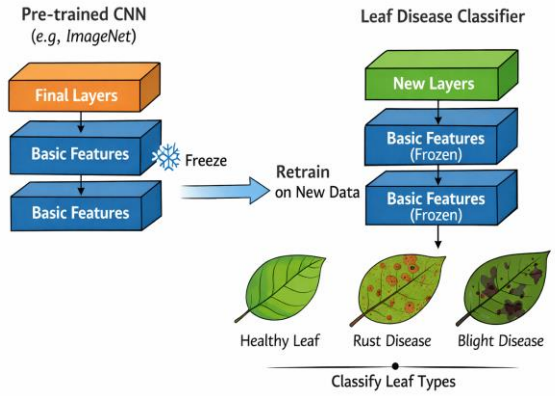

Fig. 11: Illustration of Transfer Learning

In fine-tuning, some of the deeper convolution layers are also unfrozen and retrained with a smaller learning rate so that the network can adapt more specifically to the new dataset. The choice between freezing and fine-tuning depends on dataset size and similarity between source and target tasks. Figure 11 illustrates the scenario of transfer learning. To illustrate this, consider a CNN such as ResNet that has been pre-trained on the ImageNet dataset. During pre-training, the early layers of the network learn to detect generic visual features such as vertical and horizontal edges, color transitions, and simple textures similar to the patterns discussed in earlier sections of this tutorial. These learned features are largely independent of the specific task and are therefore reusable. When this network is adapted to a new problem, such as plant disease classification, the convolutional layers used for feature extraction are usually frozen, which means their learned weights are not changed during training. Only the final layers, which combine the extracted features and make task-specific decisions, are retrained using the

new dataset.For example, in a plant disease classification problem, the frozen layers continue to detect basic visual patterns such as edges, spots, and texture variations on leaves, while the retrained layers learn to connect these features with specific disease categories. Because the early layers already contain useful visual knowledge, the model does not need to learn everything from the beginning. This reduces the amount of training data and training time required, while still maintaining good performance. In simple terms, transfer learning separates the learning of visual patterns from the learning of task-specific decisions.

Table 1. presents some commonly used CNN architectures and explains how they are useful for transfer learning. These models are not the same in depth, parameter size, connection style, or computational cost. Still, they are built on a similar idea. The first convolutional layers usually learn basic visual patterns such as edges, textures, and simple shapes. These patterns are useful in many image tasks. The deeper layers are more closely related to the final task, such as classifying a particular object. This is why transfer learning is useful in many image classification problems. It also shows that CNNs learn in stages, starting from simple patterns and moving toward more complex visual features. For example, AlexNet showed that deeper convolutional models can work well for large-scale image classification. VGG followed a simple design with small filters. ResNet used residual connections to train very deep networks more easily. MobileNet was designed for mobile and embedded devices, where memory and computation are limited. This comparison shows why different CNN models are chosen for different tasks and hardware conditions

Table 1. Salient features of popular CNNs

Year	Architecture / Technique	Main Feature	Key Applications
2015	ResNet	Residual connections in very deep networks	Image classification, object detection
2017	DenseNet	Dense connections for feature reuse and better gradient flow	Classification, medical imaging
2017	Inception-v4 / ResNeXt	Multi-scale and grouped convolutions	Classification, detection
2017	MobileNet	Depthwise separable convolutions for lightweight models	Mobile and embedded vision tasks
2017	Xception	Grouped convolutions for efficient parallel feature extraction	Image classification, detection
2020	EfficientNet	Compound scaling for optimal performance	Classification, transfer learning
2020	Vision Transformer (ViT)	Combines CNN features with self-attention	Large-scale image classification
2021	CNN + Attention / CBAM	Focus on important features via channel and spatial attention	Fine-grained recognition, detection
2022	Hybrid CNN-Transformer	Integrate CNN + global transformer context	Image understanding, segmentation
2023	AutoML / NAS	Automatically designs CNN architectures	General image tasks, industrial applications

13 Limitations

This tutorial is meant to give readers a basic and clear introduction to CNNs. It does not go deeply into mathematical derivations, optimization methods, or coding practice. Those topics are important for advanced study, but they are not the main concern of this paper. The examples are also kept small by design. They are used to explain the core ideas, not to represent the full complexity of modern CNN models. The discussion mainly follows classical CNN designs and does not cover many recent CNN variants in detail. Even with these limits, the tutorial can give beginners a strong starting point before they move on to more technical and mathematical topics.
This tutorial is also presented as a teaching approach, not only as a basic introduction to CNNs. Many students first learn deep learning through equations or programming tools. This can make the topic difficult at the beginning. This paper tries to make the first stage of learning easier by using small numerical examples and simple visual reasoning. This approach may be useful in undergraduate courses, interdisciplinary programs, and self-learning settings. It may be especially useful for learners who do not yet have a strong mathematical background. A concept-first method can support traditional mathematical teaching by helping students build intuition before they study formal analysis.

14 Conclusion

In this tutorial, CNNs were explained using a concept-first approach. This tutorial gave more importance to understanding the idea than to explaining the mathematics. Small examples and simple visual explanations were used to show how a CNN turns raw pixel values into useful features. The discussion covered convolution, activation functions, pooling, and layered feature learning. These parts were explained as connected steps, not as separate technical terms. The main purpose was to help beginners see how CNN layers work, why they are needed, and how they support one another inside the network. Once this basic idea is clear, students can move to formulas and practical implementation with less difficulty. The same teaching style may also be useful for explaining other deep learning models, such as recurrent neural networks and vision transformers. This can make AI concepts easier for students to learn.

References:

[1] P. Purwono, et al., “Understanding of Convolutional Neural Network (CNN): A Review,” International Journal of Robotics

and Control Systems, Vol. 2, No. 4, pp. 739–748, 2022.

[2] J. Gupta, S. Pathak, and G. Kumar, "Deep Learning (CNN) and Transfer Learning: A Review," Journal of Physics: Conference Series, Vol. 2273, No. 1, 2022.

[3] R. Kaur, R. Kumar, and M. Gupta, "Review on Transfer Learning for Convolutional Neural Network," in Proceedings of the 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N), IEEE, 2021, pp. 922-926

[4] J. Ayeni, "Convolutional Neural Network (CNN): The Architecture and Applications," Applied Journal of Physical Sciences, Vol. 4, No. 4, pp. 42–50, 2022.

[5] L. Alzubaidi, et al., "Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions," Journal of Big Data, Vol. 8, No. 1, Art. No. 53, 2021.

[6] Y. Gonzalez Tejeda and H. A. Mayer, "Deep Learning with Convolutional Neural Networks: A Compact Holistic Tutorial with Focus on Supervised Regression," Machine Learning and Knowledge Extraction, Vol. 6, No. 4, pp. 2753–2782, 2024.

[7] A. Upreti, "Convolutional Neural Network (CNN): A Comprehensive Overview," 2022.

[8] V. Jadeja, et al., "Convolutional Neural Networks: A Comprehensive Review of Architectures and Application," in Proceedings of the 6th International Conference on Contemporary Computing and Informatics (IC3I), IEEE, 2023, pp. 460-467.

[9] A. Alsajri and A. V. Hacimahmud, "Review of Deep Learning: Convolutional Neural Network Algorithm," Babylonian Journal of Machine Learning, Vol. 2023, pp. 19–25, 2023.

[10] I. Cacciari and A. Ranfagni, "Hands-On Fundamentals of 1D Convolutional Neural Networks—A Tutorial for Beginner Users," Applied Sciences, Vol. 14, No. 18, 2024.

[11] A. O. Ige and M. Sibiya, "State-of-the-Art in 1D Convolutional Neural Networks: A Survey," IEEE Access, Vol. 12, pp. 144082-144105, 2024.

[12] L. Stanković and D. Mandic, "Convolutional Neural Networks Demystified: A Matched Filtering Perspective-Based Tutorial," IEEE Transactions on Systems, Man, and Cybernetics: Systems, Vol. 53, No. 6, pp. 3614–3628, 2023.

[13] Z. J. Wang, et al., "CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization," IEEE Transactions on Visualization and Computer Graphics, Vol. 27, No. 2, pp. 1396–1406, 2020.