# Module 4

## Deep Learning Models

## 📌 Shallow vs Deep Neural Networks

### 🔹 Differences Between Shallow and Deep Networks

Although there is no strict cutoff, the following is a commonly accepted rule of thumb:

* A **shallow neural network** has **one or two hidden layers**.
* A **deep neural network** has **three or more hidden layers**, typically with a **large number of neurons per layer**.

The increase in depth allows deep networks to model much more **abstract**, **hierarchical**, and **non-linear** representations of the input data.

| **Feature** | **Shallow Neural Networks** | **Deep Neural Networks** |
| --- | --- | --- |
| Depth | 1–2 hidden layers | 3 or more hidden layers |
| Input | Typically vectorized features | Capable of handling raw data (images, text, etc.) |
| Feature Engineering | Often requires manual feature extraction | Automatically extracts features |
| Representation Power | Limited | High (can learn hierarchical representations) |
| Use Cases | Simple, structured data | Complex data like images, audio, NLP, etc. |

Shallow networks are easier to understand and train, which makes them useful for **conceptual learning**, **prototyping**, and **simple tasks**.

Deep networks, on the other hand, are preferred when dealing with **complex input data** and problems that require **high abstraction**.

### 🔹 Why Deep Learning Took Off

Although neural networks have existed for decades, the explosive growth and adoption of **deep learning** in recent years can be attributed to **three major breakthroughs**:

1. **Advancements in the Field:**

The development of **better activation functions**, especially **ReLU (Rectified Linear Unit)**, helped address one of the most critical limitations in deep learning: the **vanishing gradient problem**. This allowed deeper networks to train more effectively and enabled the creation of models with dozens or even hundreds of layers.

✅ ReLU maintains stronger gradients during backpropagation, which makes deep networks trainable at scale.

1. **Availability of Large Datasets**

Deep neural networks have an incredible capacity to **memorize and model large volumes of data**. However, this ability can lead to overfitting if the data is limited. The widespread availability of **large, labeled datasets** has unlocked the full potential of deep architectures.

✅ Unlike traditional machine learning algorithms, deep networks continue to improve with more data.

1. **Computational Power (GPUs)**

Training deep networks is computationally intensive. The rise of **powerful GPUs** (and later, TPUs and distributed computing) has made it feasible to:

* Train deep models on massive datasets.
* Perform experiments rapidly.
* Deploy models to production with manageable costs.

✅ What once took days or weeks can now be trained in hours using modern hardware.

### ✅ Takeaways

✅ **Shallow neural networks** are defined by **1–2 hidden layers**, and typically require **vectorized inputs** and **manual feature extraction**.

✅ **Deep neural networks** consist of **multiple hidden layers**, and are capable of **automatically extracting features** from **raw input data** such as images and text.

✅ The modern success of deep learning is attributed to:

* **Better training techniques and architectures** (e.g., ReLU)
* **Widespread access to large datasets**
* **Powerful hardware that accelerates training**

## 📌 Convolutional Neural Networks

**Convolutional Neural Networks (CNNs)** are a specialized type of deep learning architecture designed specifically for handling **image data** and other data with a grid-like structure (e.g., audio spectrograms).

Although CNNs are composed of neurons, weights, biases, and activation functions like standard dense neural networks, their design incorporates unique **architectural assumptions** that allow them to learn from raw image data more efficiently and with fewer parameters.

### 🔹 Why Deep Learning Took Off

CNNs make the **explicit assumption** that inputs are **images**, and they process them using filters that take advantage of the **spatial structure** of the data.

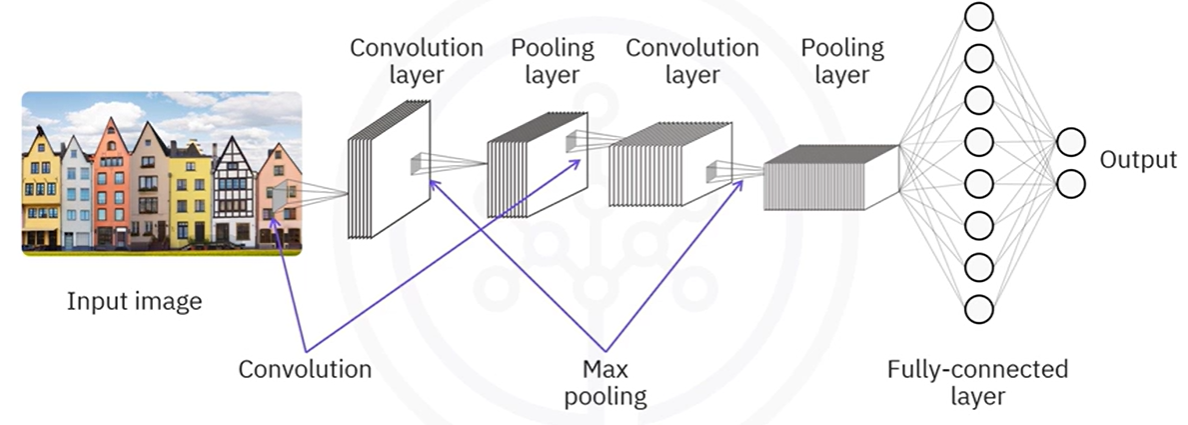
Input shapes:

* Grayscale images: n x m x 1
* Color images: n x m x 3 (where 3 channels represent RGB)

CNNs are particularly powerful for:

* Image classification
* Object detection
* Image segmentation
* Face recognition
* Medical imaging analysis

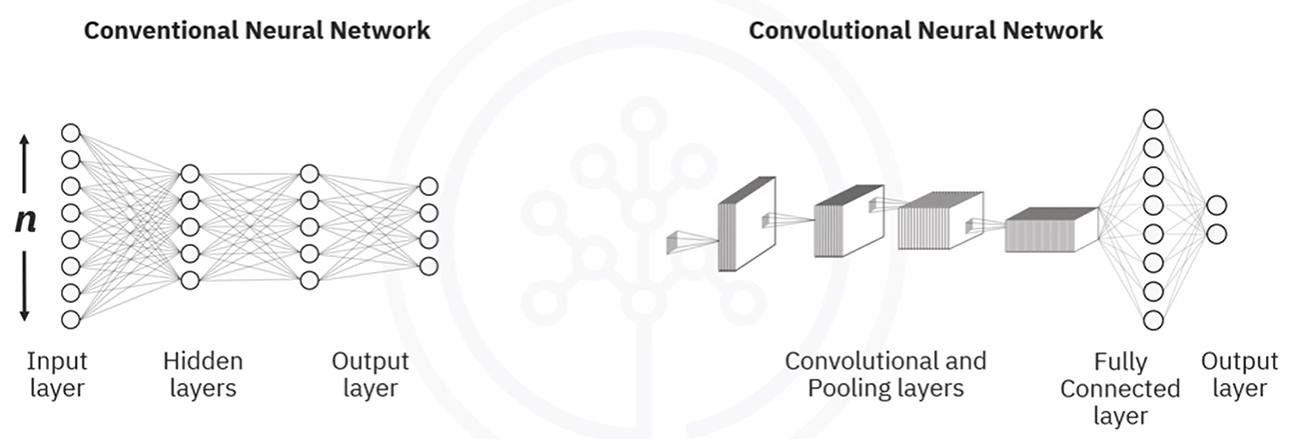
### 🔹 CNN Architecture Breakdown



A typical **Convolutional Neural Network (CNN)** is composed of several key building blocks organized in a specific order to progressively extract spatial features from input images and make accurate predictions.

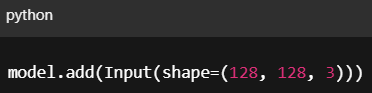
CNNs differ from the fully connected networks introduced earlier in that they assume **images as input**, maintaining their spatial structure. Instead of flattening the data up front, CNNs process **3D input tensors** using specialized layers that are better suited for handling **pixel-level patterns**.

1. **Input Layer:**



Unlike shallow networks that accept a **1D vector**, CNNs accept **3D inputs**:

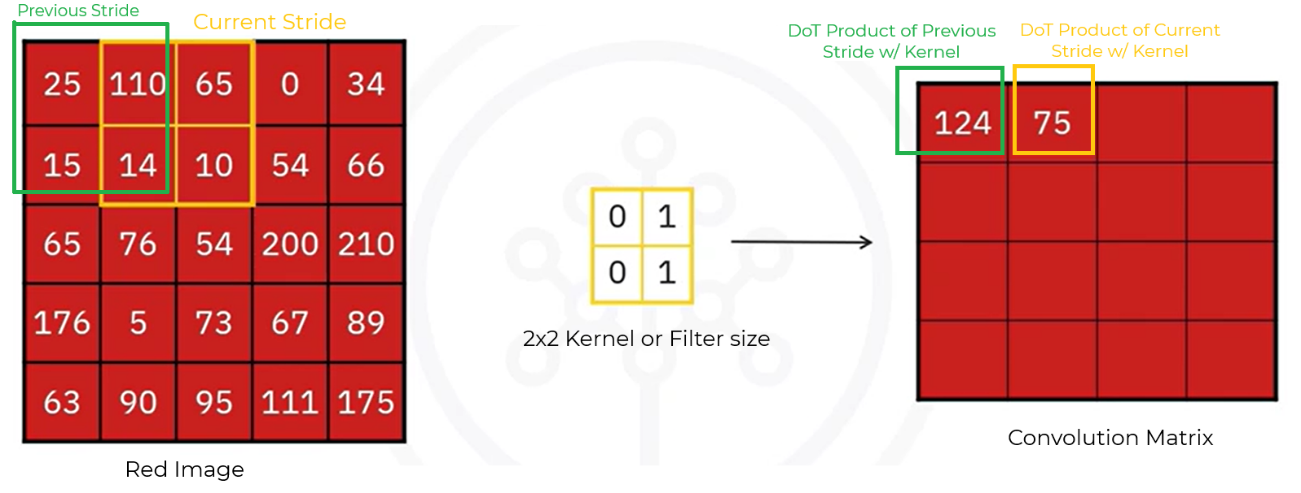
* For grayscale images: height × width × 1
* For color images: height × width × 3 (RGB channels)

For example, a 128×128 RGB image would have an input shape of (128, 128, 3).

This structure allows the model to **preserve spatial relationships** between pixels — crucial for tasks like object detection and image classification.

1. **Convolutional Layer:**

This is the **core feature-extracting layer** in a CNN. It applies a set of **filters** (also called kernels) that scan across the image in small patches.



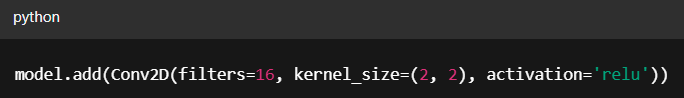
In the convolutional layer, we basically define filters and we compute the convolution between the defined filters and each of the three images (RGB).

This is the **core feature-extracting layer** in a CNN. It applies a set of **filters** (also called kernels) that scan across the image in small patches.

This structure allows the model to **preserve spatial relationships** between pixels — crucial for tasks like object detection and image classification.

Each filter performs a **dot product** between its values and a local patch of the image and stores the result in a new **feature map**.

* Typical filter sizes: 2×2, 3×3, 5×5
* Stride: how far the filter moves per step (usually 1)
* Depth: number of filters determines number of feature maps



**⚠️ Why not flatten the input image into an [ (n x m) x 1 ] vector and pass it into a dense network?**

* Flattening large images would result in **huge parameter counts**, increasing computational cost and overfitting risk.
* Convolution drastically reduces parameters by reusing filters across regions of the image, while maintaining spatial locality.

ℹ️ **ReLU Activation (within Conv2D)**

Each convolutional layer is followed by a **ReLU (Rectified Linear Unit)** activation function. This layer:

* Keeps only **positive values**
* Sets all **negative values to zero**
* Adds **non-linearity** to the model

ReLU ensures the model can learn complex, non-linear representations efficiently, and avoids the vanishing gradient issues of older activations (like sigmoid).

1. **Pooling Layer (MaxPooling2D / AveragePooling2D):**

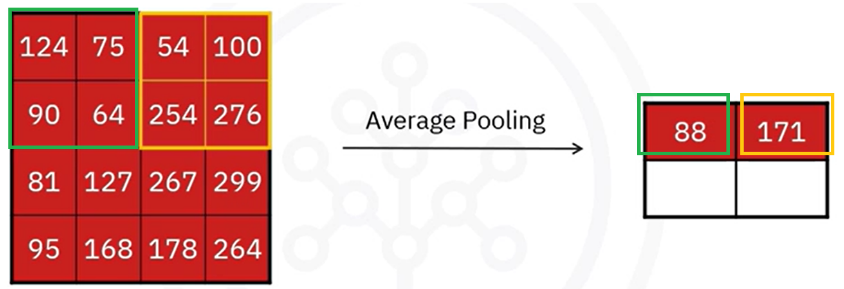
The pooling layer **reduces the spatial dimensions** of the feature maps. It helps the network become more invariant to small shifts or distortions in the input image.

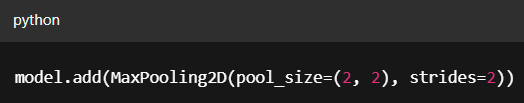
There are two common types:

* **Max Pooling**: keeps the largest value in each region.



* **Average Pooling**: computes the average of each region



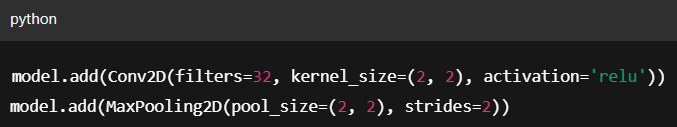
Example: with a 2×2 filter and a stride of 2, the pooling operation slides over the image in blocks, shrinking the dimensions while retaining key features.

✅ Pooling offers:

* **Dimensionality reduction**
* **Noise suppression**
* **Spatial invariance**

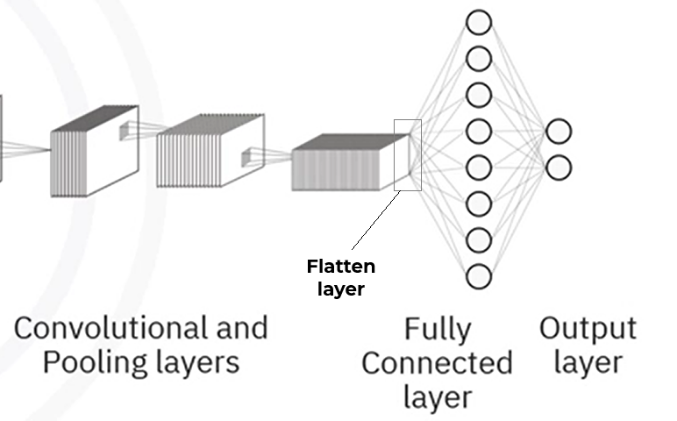
#### 🔁 Repeat: Additional Convolution + Pooling Blocks

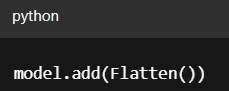
Deeper CNNs stack multiple convolution and pooling layers. Typically:



* Deeper layers use **more filters** to capture more abstract and complex features.
* Each additional block allows the network to **build a hierarchy of features**, from edges and textures to shapes and object parts.

1. **Flatten Layer:**



Before passing data to a fully connected layer, the 3D output of the final convolution/pooling layer is **flattened** into a 1D vector.

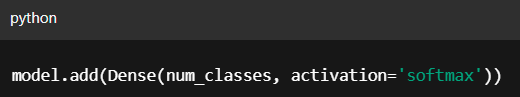
⚠️ This conversion is **necessary** because **dense layers expect flat inputs**. The flattened vector represents all the high-level features extracted from the image.

1. **Fully-Connected Layer:**

These layers operate like the traditional dense layers introduced earlier:

* Each node in one layer is connected to **every node** in the next.
* Often used to **combine features** and make the final prediction.

1. **Fully-Connected Layer:**

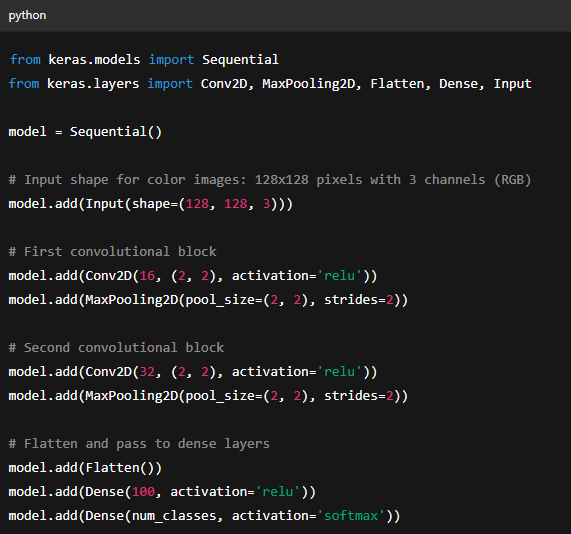
****The final output layer produces predictions. Its size depends on the number of classes:

* Use **softmax activation** to convert output values into probabilities that sum to 1.
* Each neuron corresponds to one possible class.

### ✅ Complete Flow of CNN Layers

| **Layer** | **Purpose** |
| --- | --- |
| Input | Accepts image in 3D shape (e.g., 128×128×3) |
| Convolution (Conv2D) | Detects features by applying filters |
| ReLU | Adds non-linearity, filters out negative values |
| Pooling | Reduces spatial size and introduces spatial invariance |
| (Repeat Conv + Pool) | Extracts deeper and more abstract patterns |
| Flatten | Converts 3D data into 1D vector for Dense layers |
| Dense (Hidden) | Learns global patterns, combines features |
| Dense (Output) | Outputs class probabilities (softmax) |

### 🔹 Building a CNN in Keras

****

**Conv2D:** Applies filters to extract features from images.

**MaxPooling2D:** Reduces spatial size and adds spatial invariance.

**Flatten:** Converts 3D feature maps to 1D vector.

**Dense**: Learns class-specific patterns and outputs final predictions.

### ✅ Takeaways

✅ CNNs are optimized for processing **image data** using a **layered structure** of convolutions, activations, pooling, and dense layers.

✅ The architecture:

* Preserves spatial structure
* Reduces the number of parameters
* Extracts hierarchical features

✅ CNNs are the go-to architecture for solving complex **computer vision problems** like classification, object detection, and segmentation.

✅ Keras provides a clean and modular way to define CNNs using Conv2D, MaxPooling2D, Flatten, and Dense layers.

📌 💡 ✅ ❌ 🚫 ⚠️ **🔍** ℹ️ **🧠** ⚙️ 🔧 **🧰** 🛠 🧩