CONVOLUTIONAL NEURAL NETWORK

- A Convolutional Neural Network (CNN) is a specialized type of deep neural network primarily used for analyzing visual imagery.
- Unlike a standard multi-layer perceptron, a CNN is specifically designed to process data with a known grid-like topology, such as images (2D grids of pixels) or time series (1D grids).
- Its name comes from the use of the **convolution** mathematical operation in place of general matrix multiplication in at least one of its layers.

Typical Architecture

A standard CNN architecture consists of a sequence of layers, typically including:

- **Convolutional Layer:** Applies a set of learnable filters to the input, producing a set of feature maps.
- **Activation Layer:** Applies a non-linear activation function (e.g., ReLU) element-wise to the output of the convolutional layer.
- **Pooling Layer:** Downsamples the spatial dimensions of the feature maps (e.g., max-pooling or average-pooling), reducing computation and increasing translational invariance.
- **Fully Connected Layer:** At the end of the network, one or more fully connected layers are often used to classify the features extracted by the preceding layers. This part of the network resembles a traditional multi-layer perceptron.

The combination of these layers allows the CNN to learn a hierarchy of features, from simple edges and textures in the initial layers to complex object parts and entire objects in deeper layers.

INPUT TO CNN

*The input to a Convolutional Neural Network (CNN) is a multi-dimensional array or **tensor** that represents an image. Each cell in this tensor corresponds to a **pixel** in the image.*

***For a grayscale image**, the input is a 2D matrix where each cell holds a single value representing the intensity of the pixel, typically ranging from 0 (black) to 255 (white).*

***For a color image**, the input is a 3D tensor, which can be thought of as a stack of three matrices. Each matrix corresponds to a color channel: Red (R), Green (G), and Blue (B). The values in each cell of these matrices represent the intensity of that specific color component for the corresponding pixel.*

The dimensions of the input tensor are typically expressed as (H,W,C), where:

- *H is the Height of the image in pixels.*
- *W is the Width of the image in pixels.*
- *C is the number of Channels, which is 1 for grayscale and 3 for RGB.*

CONVOLUTION OPERATION

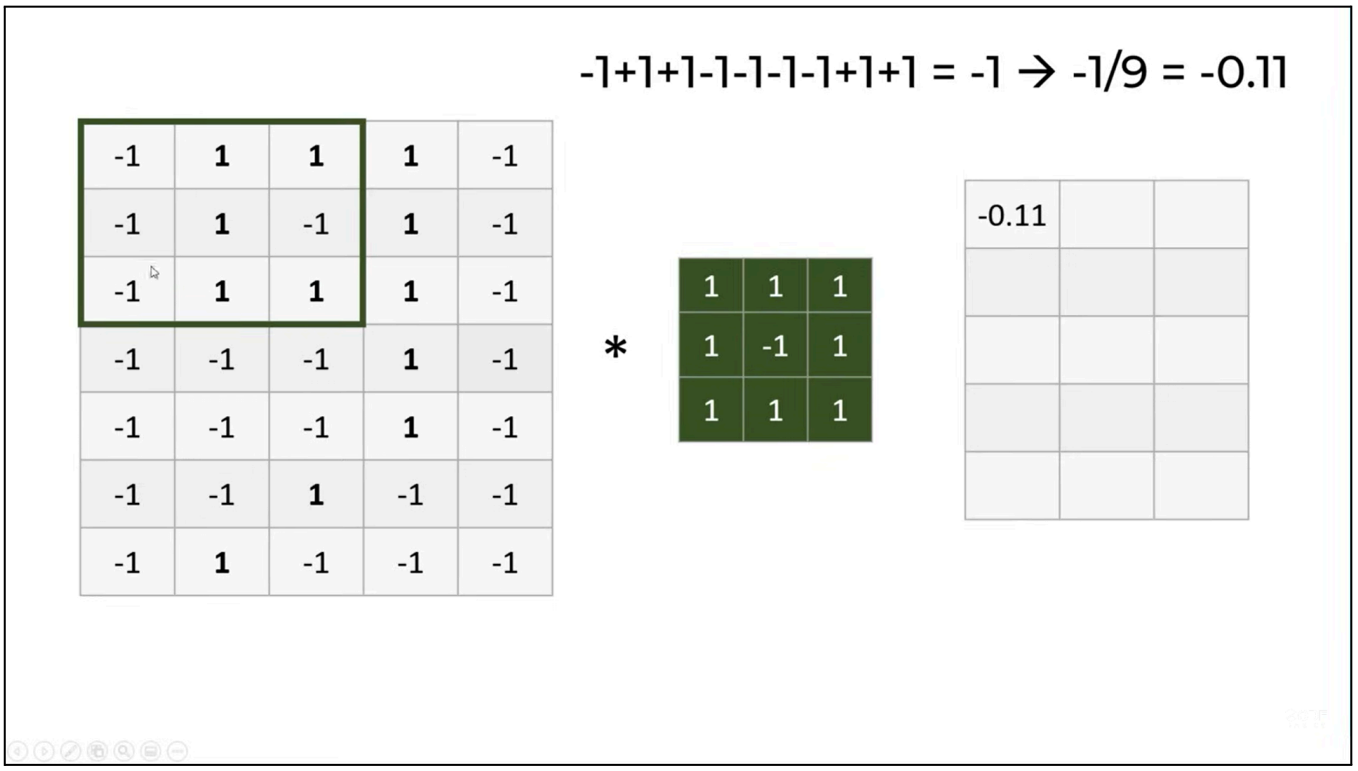
The **convolution operation** is the fundamental building block of a CNN It is a mathematical process that involves sliding a small matrix of numbers, called a **filter** or **kernel**, over an input volume (e.g., an image). This operation produces a new matrix, called a **feature map** or **activation map**, which highlights specific features from the input, such as edges, textures, or shapes

How it Works

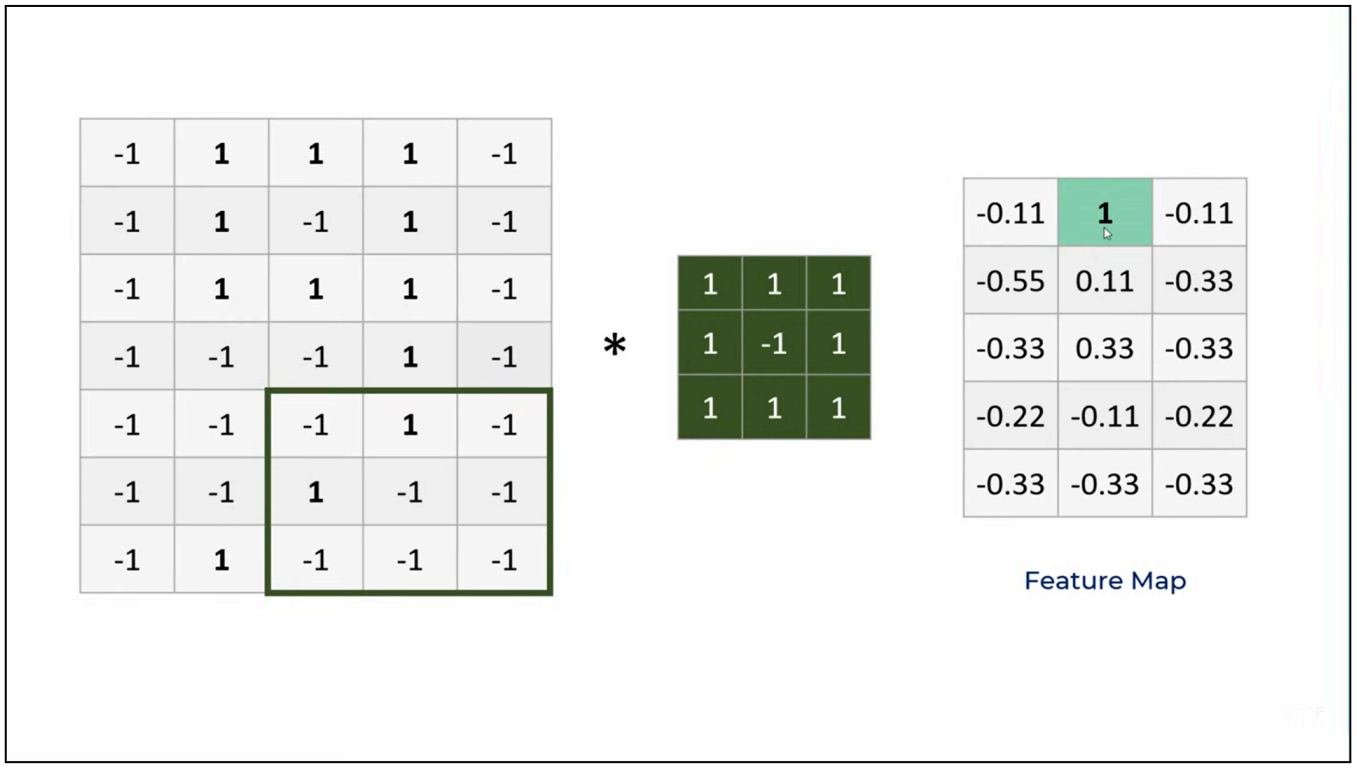
1. **Filter Application**: A filter (a small matrix of weights) is placed over a section of the input data.
2. **Element-wise Multiplication**: The filter's values are multiplied element-wise with the corresponding values in the section of the input it's covering.
3. **Summation**: The products from the element-wise multiplication are summed up to produce a single value. This value becomes a single pixel in the output feature map.
4. **Sliding Window**: The filter then slides to the next position (defined by the **stride**), and the process is repeated across the entire input volume.

Hyperparameters

- **Filter Size**: The dimensions of the kernel (e.g., 3x3, 5x5). Smaller filters are common as they capture more local patterns.
- **Stride**: The number of pixels the filter shifts at each step. A larger stride reduces the size of the output feature map.
- **Padding**: The practice of adding zeros around the border of the input volume. This is often used to ensure that the output feature map has the same spatial dimensions as the input, preventing information loss at the edges.



The example uses a **5*7 input grid**, a **3*3 filter** (green) with **stride = 1** and no padding. Results in a **3*5 Feature map**.



POOLING OPERATION

- A **pooling function**, also known as a **downsampling** or **subsampling** layer, is a key component of CNNs that reduces the spatial dimensions (width and height) of the feature maps.
- Its primary purpose is to decrease computational complexity and memory usage, as well as to increase the model's **translational invariance**.
- This means the network becomes more robust to minor shifts or distortions in the input image.

How it Works

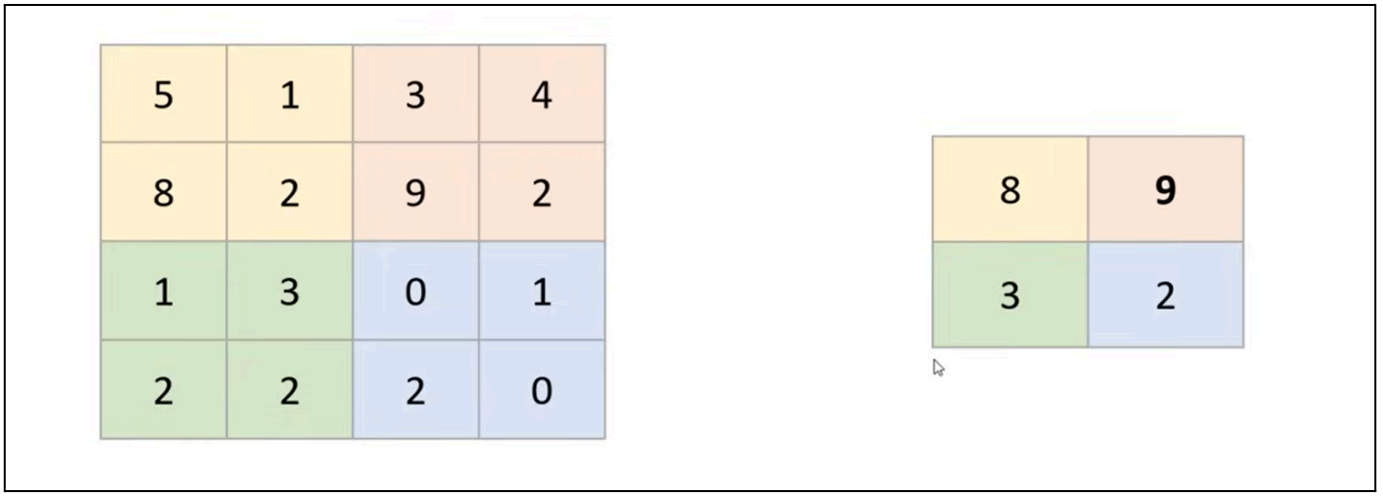
The pooling layer operates by sliding a small window (typically 2x2 or 3x3) over the input feature map and applying an aggregation function to the values within that window. The most common types of pooling are:

- **Max Pooling**: This is the most popular type. It takes the **maximum** value from each window. This has the effect of retaining the most prominent feature response in each region while discarding the less important ones. It's an effective way to downsample while preserving the most salient information.
- **Average Pooling**: This takes the **average** value from each window. It provides a less aggressive form of downsampling and is sometimes used in later stages of a network.
- **Min Pooling**: This takes the **minimum** value. It's rarely used but could be relevant in specific applications where low values are more significant.

Hyperparameters

The key hyperparameters for a pooling layer are similar to those of a convolutional layer:

- **Filter Size** (F): The dimensions of the pooling window (e.g., 2x2).
- **Stride** (S): The step size the window moves across the input. A stride of 2 is common, which halves the spatial dimensions of the feature map.



This example uses a **4*4 feature map**, a **2*2 pooling filter** with **stride = 2**. Results in a **2*2 Feature map**.

WEIGHT/PARAMETER SHARING

Weight sharing, or **parameter sharing**, is a fundamental concept in Convolutional Neural Networks (CNNs). It is the practice of using the same set of weights (a single filter or kernel) across all spatial locations of the input volume. Instead of each neuron in a convolutional layer having its own unique set of weights, they all share the same weights.

For example, a single 3x3 filter, which has 9 learnable parameters (weights), is slid across the entire input image. Each time it is applied to a new receptive field, it performs the same computation using the exact same 9 weights. This means that if a filter learns to detect a vertical edge in the top-left corner of an image, it will use the same weights to detect a vertical edge anywhere else in the image.

Advantages of Weight Sharing

1. **Reduced Number of Parameters**:
   This is the most significant advantage. In a traditional fully connected network, each neuron in a hidden layer would be connected to every pixel in the input image. For a 200x200 pixel image, a single neuron would require 40,000 weights. In contrast, a CNN with weight sharing uses a small filter (e.g., 5x5, with 25 weights) that is applied across the entire image. This drastically reduces the total number of parameters, making the network more memory-efficient and computationally faster to train.
2. **Translational Equivariance**:
   *By applying the same filter across the entire image, the CNN becomes **equivariant** to translation. This means that if an object or feature (like an eye or a nose) shifts its position in the input image, its representation in the output feature map will shift by the same amount. The network can detect the feature regardless of its location. This is a crucial property for tasks like object detection and recognition.*
3. **Improved Generalization and Regularization**:
   *With fewer parameters to learn, the model is less prone to **overfitting**. Weight sharing acts as a form of regularization because it constrains the model by forcing it to learn a single set of feature detectors that work everywhere. This allows the network to generalize better to new, unseen images.*
4. **Feature Hierarchy**:
   Weight sharing enables the network to learn a hierarchy of features. Initial layers with shared weights can learn to detect simple, low-level features (e.g., edges and corners). Subsequent layers can then combine these simple features to form more complex representations (e.g., textures, shapes), and so on, building up to a high-level understanding of the image content.

ARCHITECTURE OF A CNN

A basic Convolutional Neural Network (CNN) architecture is a sequence of layers designed to automatically learn hierarchical features from an input image.

1. **Input Layer:** This layer holds the raw pixel values of the input image.

Function: Feeds the image data into the network, typically represented as a 3D tensor (width, height, and color channels).

2. **Convolutional Layer:** This is the core building block of a CNN. It applies a set of learnable **filters** to the input volume, producing a set of **feature maps**.

Function: Detects specific features like edges, corners, and textures by performing the convolution operation.

3. **Activation Layer (ReLU):** This layer applies a non-linear activation function to the output of the convolutional layer, often the Rectified Linear Unit (ReLU).

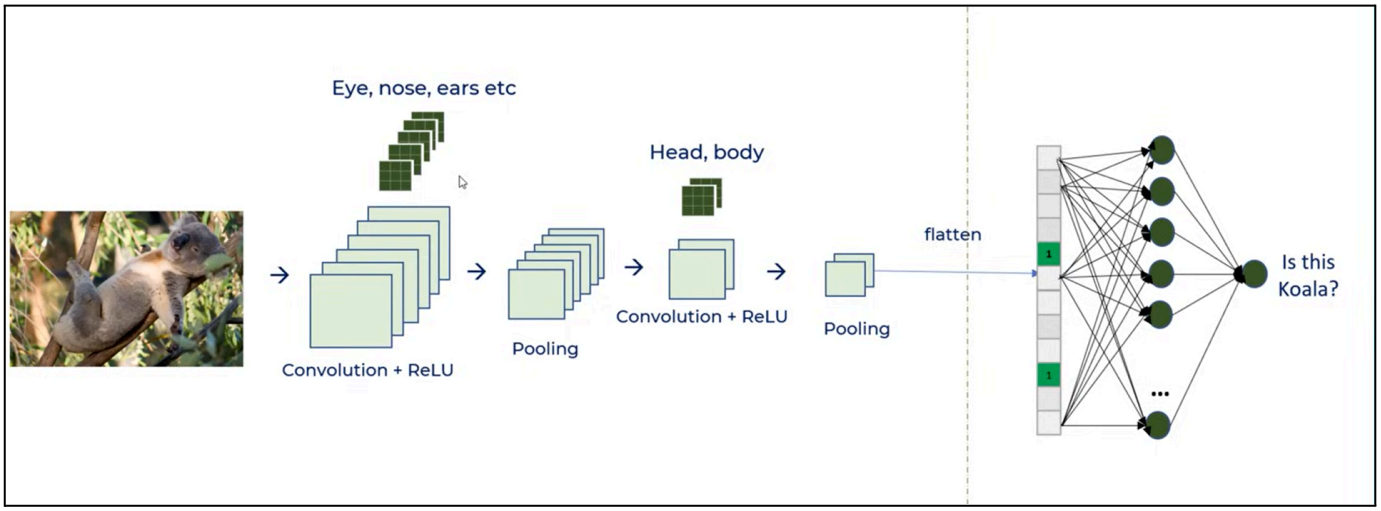
Function: Introduces non-linearity into the model, allowing it to learn more complex patterns than a linear model could.

4. **Pooling Layer:** This layer downsamples the spatial dimensions of the feature maps.

Function: Reduces computational load, memory usage, and makes the model more robust to minor shifts in the input data (**translational invariance**). Max pooling is the most common type.

5. **Fully Connected Layer:** At the end of the network, one or more fully connected layers connect every neuron to every neuron in the previous layer.

Function: Learns high-level features and performs the final classification or regression based on the features extracted by the preceding layers.



Koala image → **input layer** (represented as a matrix, each cell = each pixel)
ReLU → **Activation Layer** (This layer always comes after Convolution, often merged with it)
After flattening → **Fully connected Layer** (Can be more than one)
Dark Green grids → Filters/Kernels

APPLICATIONS OF CNN

1. **Image Classification**:
   CNNs are used to categorize an entire image into a single class. For example, a CNN can identify whether a picture contains a "cat," "dog," or "car."
2. **Object Detection**:
   This involves both classifying objects and localizing them with bounding boxes. Self-driving cars use this to detect pedestrians, other vehicles, and traffic signs in real-time.
3. **Medical Imaging**:
   CNNs are highly effective for analyzing medical scans like X-rays, MRIs, and CT scans. They can help with tasks like tumor detection, disease diagnosis, and image segmentation of organs.
4. **Natural Language Processing (NLP)**:
   CNNs can be applied to text by treating a sentence as a 1D grid. This is used in tasks like sentiment analysis and document classification to identify key features in sequences of words.
5. **Video Analysis**:
   By processing video frames sequentially, CNNs can track objects, recognize actions, and monitor surveillance footage. This is essential for applications like automated surveillance and crowd monitoring.

---

1. Filter Size
The filter size (F) is the spatial dimension of the filter or kernel. Common filter sizes are 3x3, 5x5, or 7x7. A smaller filter size captures more localized features, while a larger one can capture more global or coarse features. A 3x3 filter is very common as it's the smallest size that can capture all four neighbors of a pixel, providing a good balance between feature capture and computational cost.

2. Stride
The stride (S) is the number of pixels the filter shifts across the input volume at each step. A stride of 1 means the filter moves one pixel at a time, resulting in a large output volume. A stride of 2 means the filter skips one pixel, effectively downsampling the output and reducing the spatial dimensions. Larger strides lead to smaller output volumes but can cause some information to be lost.

3. Zero-padding
Zero-padding (P) involves adding a border of zero-value pixels around the input volume. This is often done to control the spatial size of the output volume. Without padding, the output size would be smaller than the input, leading to a loss of information, especially at the edges. Padding can be used to ensure the output volume's spatial dimensions match the input's, which simplifies the network design.

4. Output Depth

The output depth is the number of filters used in a convolutional layer. Each filter learns to detect a unique feature (e.g., a vertical edge, a specific texture). The number of filters directly determines the depth of the resulting output feature map. For instance, if you apply 64 different filters to an input, the output feature map will have a depth of 64. A deeper output allows the network to learn a greater variety of features.

EQUIVARIANT REPRESENTATION

Equivariant Representation *is a crucial property of CNNs that means if the input changes in a specific way, the output changes in the same way. Specifically, a CNN is* ***equivariant to translation****. This means that if you translate an object (e.g., a face) in an input image, the feature map of that object will translate by the same amount. The network recognizes the same feature, just in a different location.*

This property is a direct result of ***weight sharing*** *and the* ***convolution operation****. Since the same filter is applied across the entire image, it will detect the same feature regardless of its position. For example, a filter trained to detect a vertical edge will find that edge whether it is on the left or the right side of the image. This ability to detect a feature anywhere in the input without needing to learn separate weights for each possible location is what makes CNNs so effective and efficient for computer vision.*

ALEXNET

AlexNet is a pioneering Convolutional Neural Network (CNN) architecture that won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012.

Its architecture consists of eight layers with learnable parameters: five convolutional layers and three fully connected layers. Pooling layers are not learnable layes hence are not included.

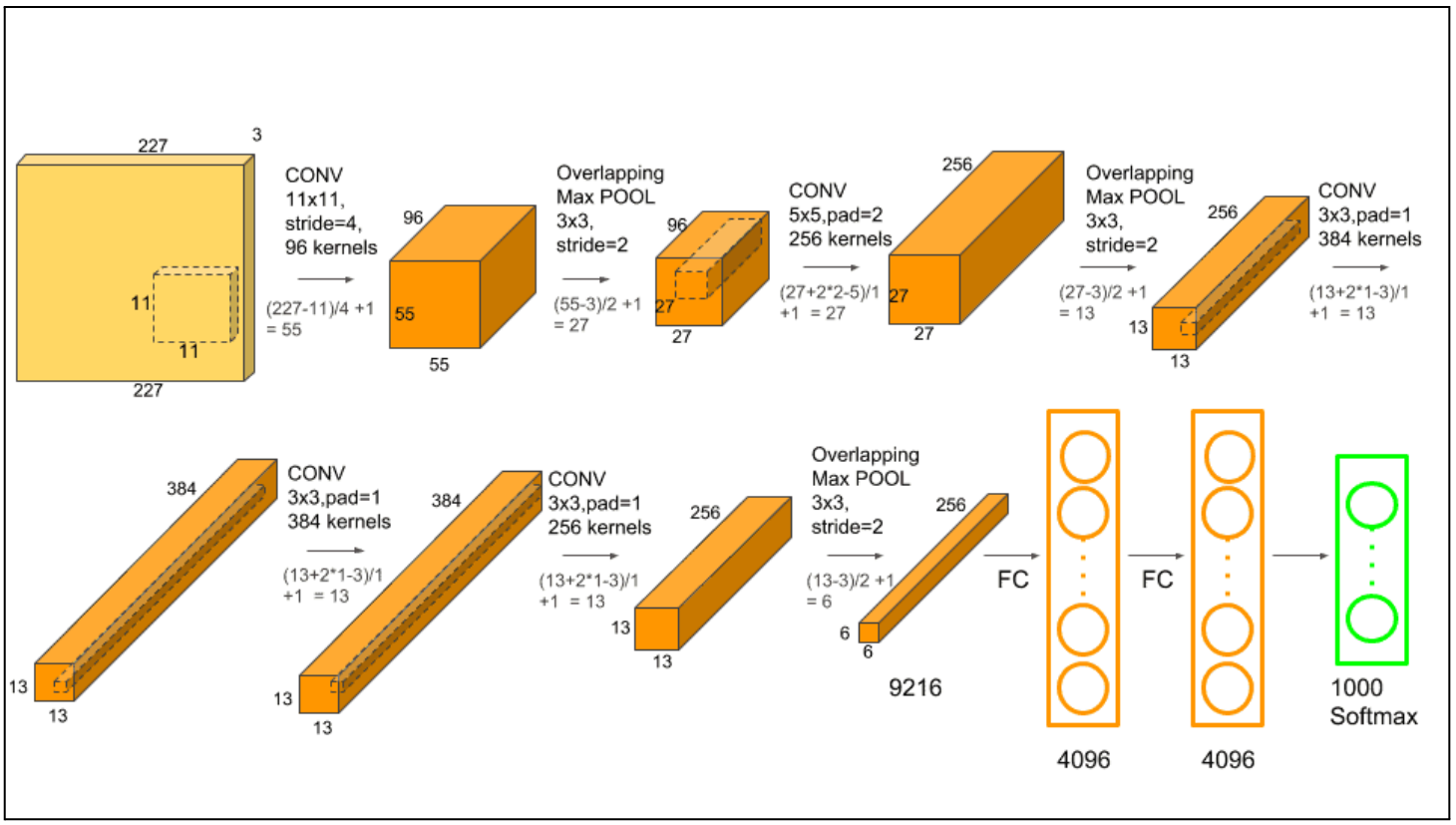
Layer-by-Layer Architecture

The network processes an input image of size **227x227x3** (227 pixels in width and height, and 3 color channels for RGB).

1. **First Convolutional Layer (Conv1)**:
    - **Input**: 227x227x3 image.
    - **Operation**: Applies 96 filters of size **11x11x3** with a **stride of 4**.
    - **Output**: A 55x55x96 feature map.
    - **Followed by**: **Local Response Normalization (LRN)** and **Max Pooling** (3x3 filter, stride 2).
2. **Second Convolutional Layer (Conv2)**:
    - **Input**: The output from the previous pooling layer (27x27x96).
    - **Operation**: Applies 256 filters of size **5x5** with a **stride of 1** and **padding of 2**.
    - **Output**: A 27x27x256 feature map.
    - **Followed by**: **LRN** and **Max Pooling** (3x3 filter, stride 2).
3. **Third Convolutional Layer (Conv3)**:
    - **Input**: The output from the previous pooling layer (13x13x256).
    - **Operation**: Applies 384 filters of size **3x3** with a **stride of 1** and **padding of 1**.
    - **Output**: A 13x13x384 feature map.
    - **Followed by**: No pooling or normalization.
4. **Fourth Convolutional Layer (Conv4)**:
    - **Input**: The output from Conv3 (13x13x384).
    - **Operation**: Applies 384 filters of size **3x3** with a **stride of 1** and **padding of 1**.
    - **Output**: A 13x13x384 feature map.
    - **Followed by**: No pooling or normalization.
5. **Fifth Convolutional Layer (Conv5)**:
    - **Input**: The output from Conv4 (13x13x384).
    - **Operation**: Applies 256 filters of size **3x3** with a **stride of 1** and **padding of 1**.
    - **Output**: A 13x13x256 feature map.
    - **Followed by**: **Max Pooling** (3x3 filter, stride 2). The output of this layer is a 6x6x256 feature map.
6. **First Fully Connected Layer (FC6)**:
    - **Input**: The flattened output of the final pooling layer (6x6x256 = 9216 neurons).
    - **Operation**: Connects all 9216 inputs to **4096 neurons**.
    - **Followed by**: **ReLU** activation and **Dropout** (with a rate of 0.5).
7. **Second Fully Connected Layer (FC7)**:
    - **Input**: The output from the previous fully connected layer (4096 neurons).
    - **Operation**: Connects the 4096 inputs to **4096 neurons**.

- **Followed by**: **ReLU** activation and **Dropout** (with a rate of 0.5).

8. **Output Layer (FC8)**:
   - **Input**: The output from the previous fully connected layer (4096 neurons).
   - **Operation**: Connects the 4096 inputs to **1000 neurons**, corresponding to the 1000 classes of the ImageNet dataset.
   - **Followed by**: **Softmax** activation to produce a probability distribution over the classes.



Formula for Spatial Dimensions

The height (H_{out}) and width (W_{out}) of the output feature map are calculated as follows:

$$H_{out} = \lfloor \frac{H_{in} + 2P - F}{S} \rfloor + 1$$

$$W_{out} = \lfloor \frac{W_{in} + 2P - F}{S} \rfloor + 1$$

Where:

- H_{in} and W_{in} are the height and width of the input volume.
- P is the padding size.
- F is the filter (kernel) size.
- S is the stride.
- $\lfloor \cdot \rfloor$ is the floor function, which rounds the result down to the nearest integer.

You can see this formula being used in above image before every new layer to calculate its size.