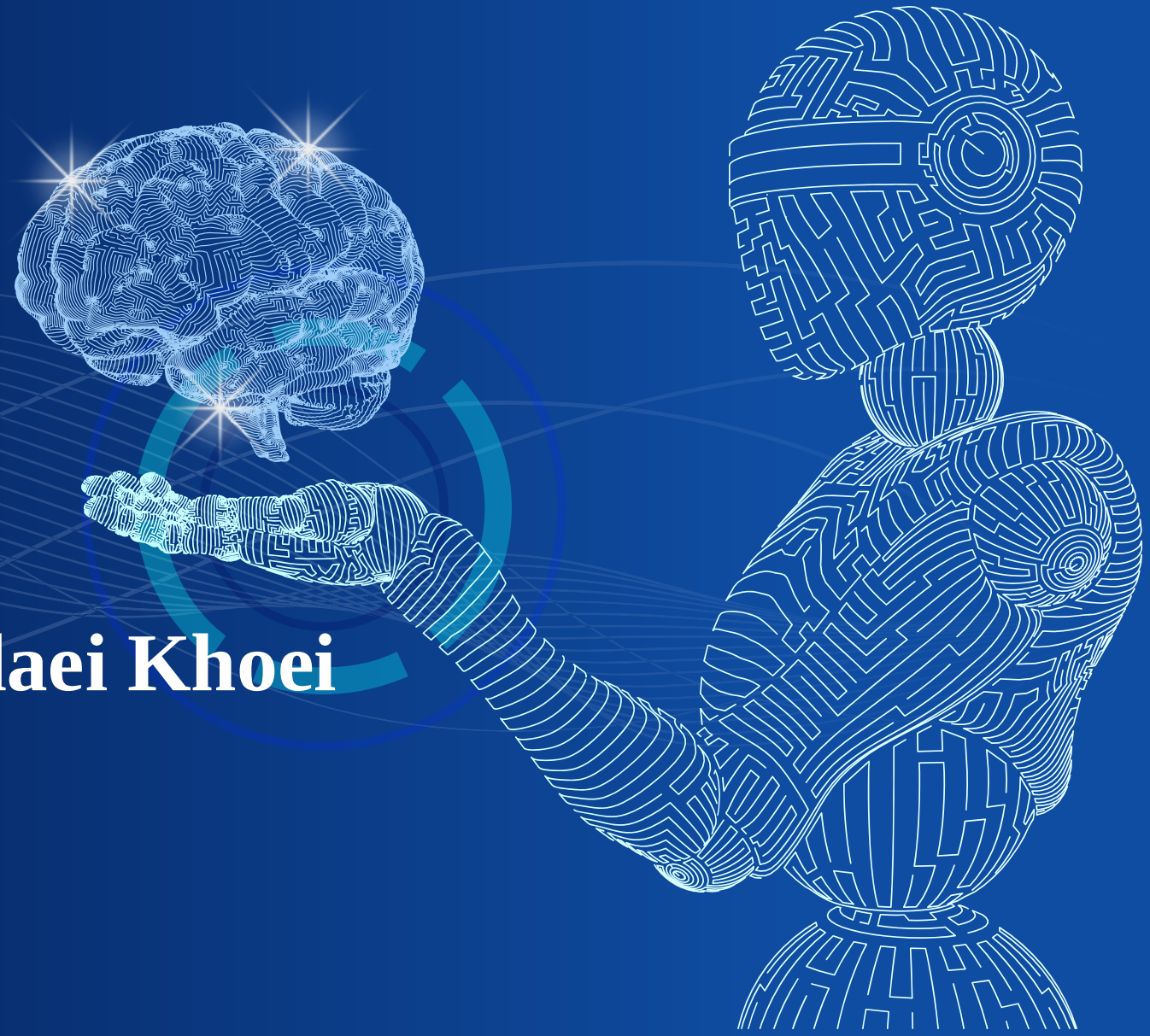
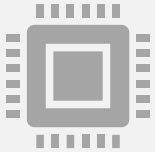
CS 7150: Deep Learning



Presented by: Dr. Tala Talaei Khoei

Convolutional Neural Network

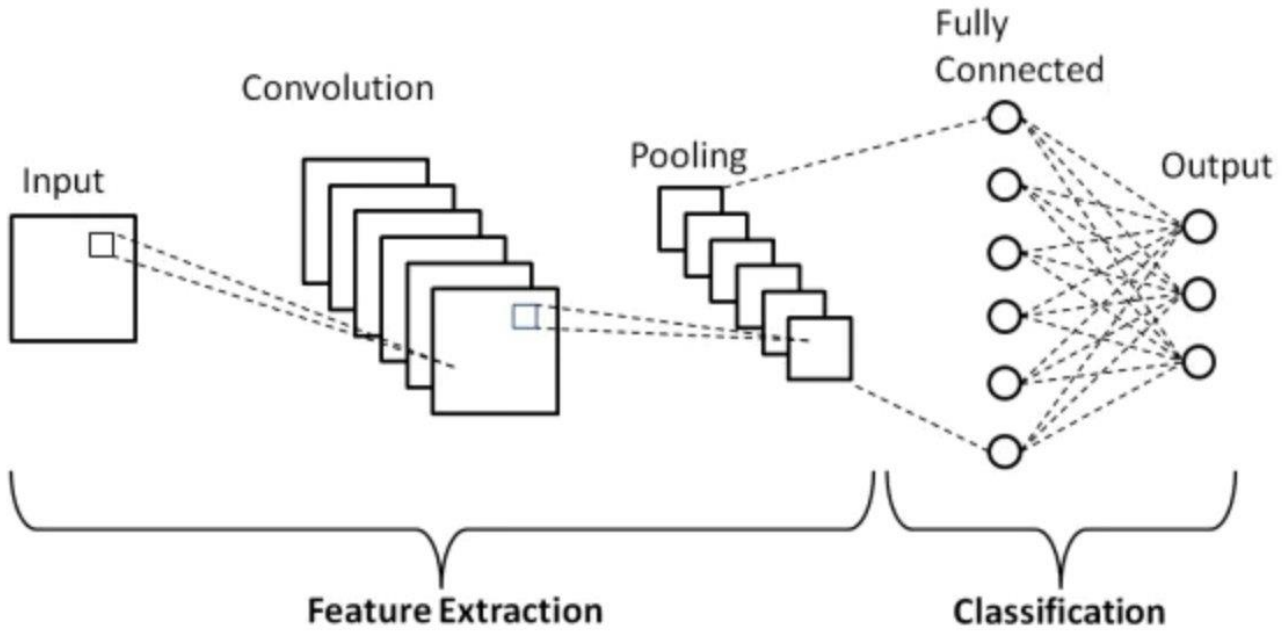
Around the 1980s, CNNs were developed and deployed for the first time. A CNN could only detect handwritten digits at the time. CNN was primarily used in various areas to read zip and pin codes etc.



The most common aspect of any A.I. model is that it requires a massive amount of data to train. This was one of the biggest problems that CNN faced at the time, and due to this, they were only used in the postal industry. Yann LeCun was the first to introduce convolutional neural networks.

Kunihiko Fukushima, a renowned Japanese scientist, who even invented recognition, which was a very simple Neural Network used for image identification, had developed on the work done earlier by LeCun

What is CNN?

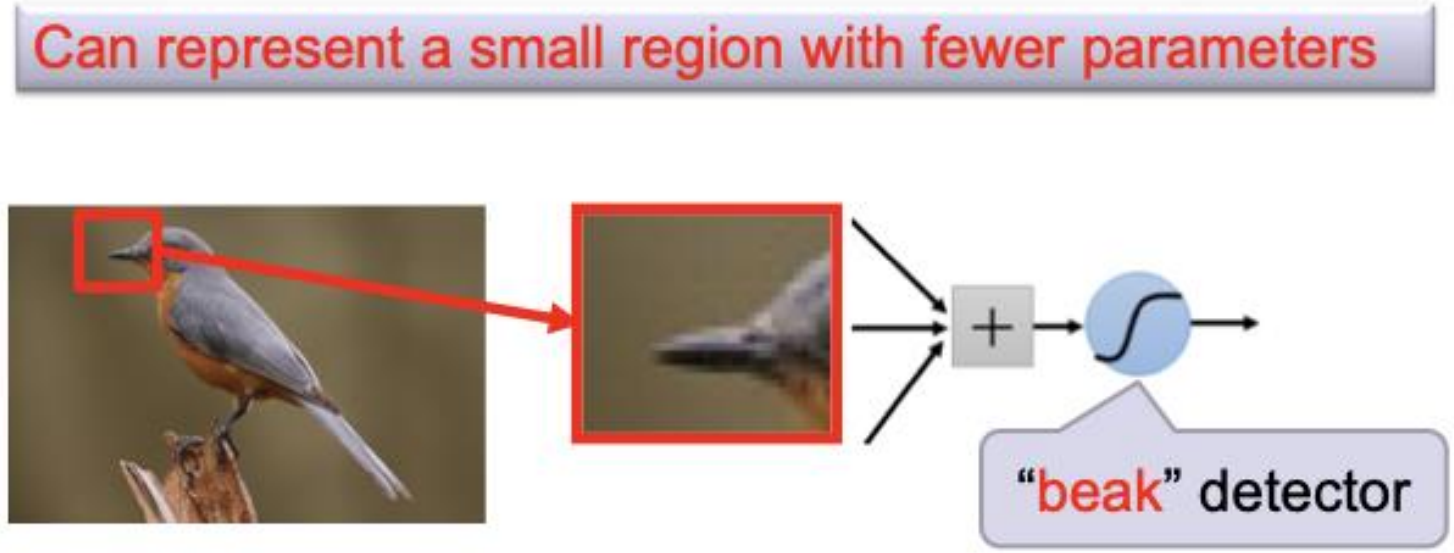


- CNN is a type of artificial neural network, mainly used in the processing of data with grid-like topology, such as image recognition and classification. Compared to other algorithms for image classification, the main advantage of convolutional neural networks is that it is a form of unsupervised learning and it doesn't require labeled samples to learn features from data. Due to this advantage, the convolutional neural network has become one of the most popular deep learning networks. In recent years, the convolutional neural network has also been proved to be valuable tools in other fields of data science including big data analysis.

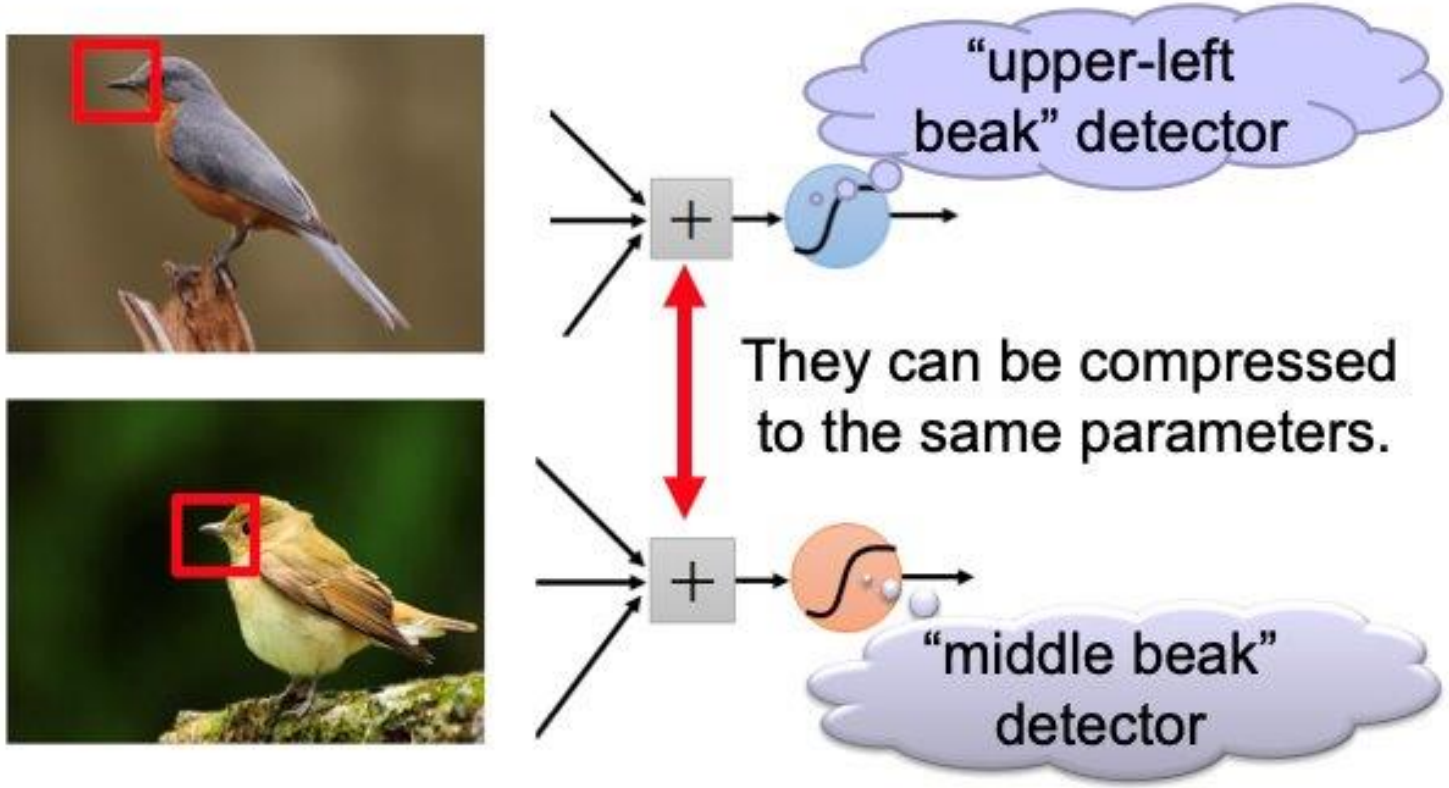
Consider learning an image:

- Some patterns are much smaller than the whole image

Can represent a small region with fewer parameters



Same pattern appears in different places:
They can be compressed!
What about training a lot of such “small” detectors and each detector must “move around”.



The Importance of CNN

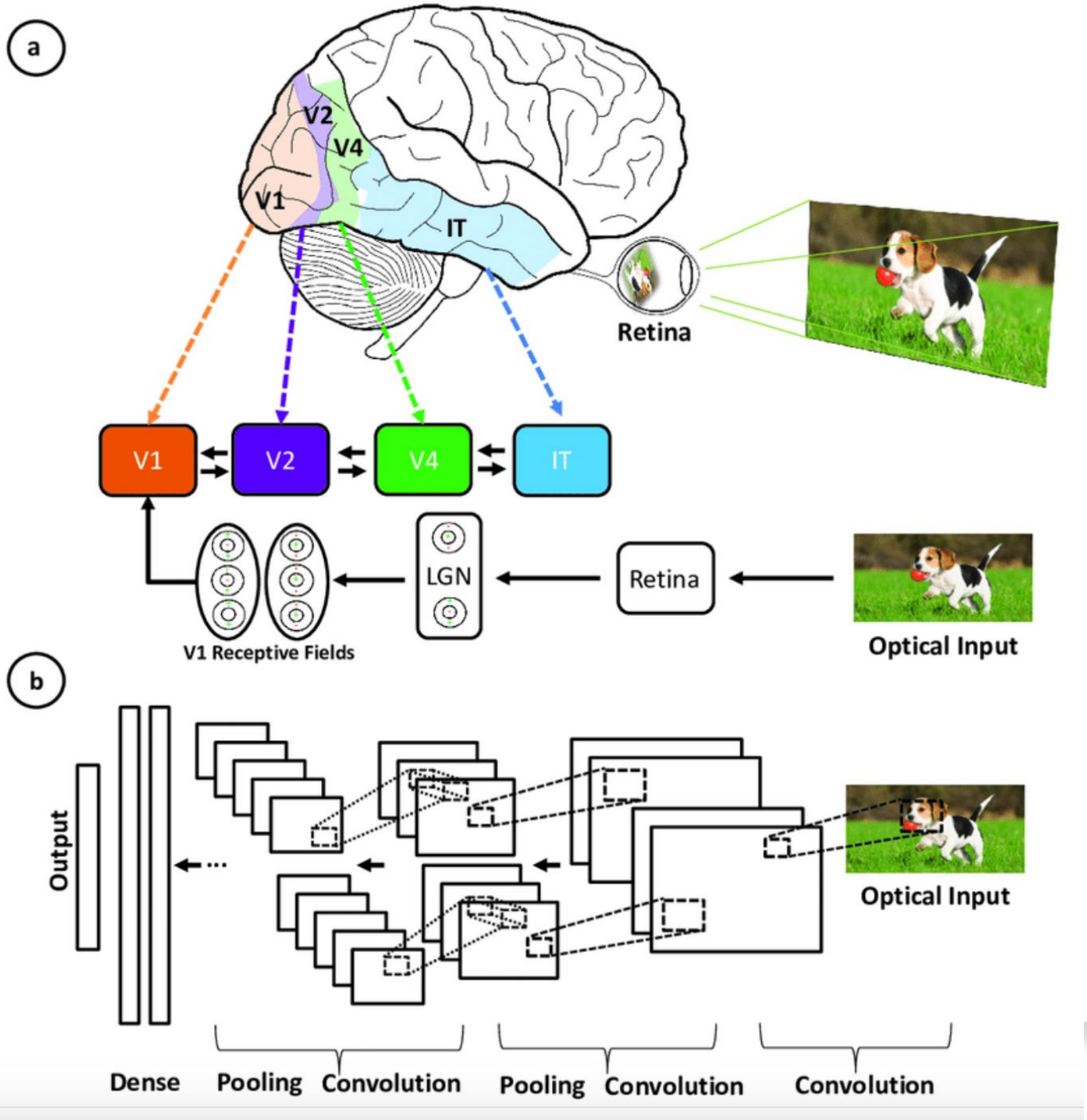
- CNNs are distinguished from classic machine learning algorithms such as SVMs and decision trees by their ability to autonomously extract features at a large scale, bypassing the need for manual feature engineering and thereby enhancing efficiency.

- The convolutional layers grant CNNs their translation-invariant characteristics, empowering them to identify and extract patterns and features from data irrespective of variations in position, orientation, scale, or translation.

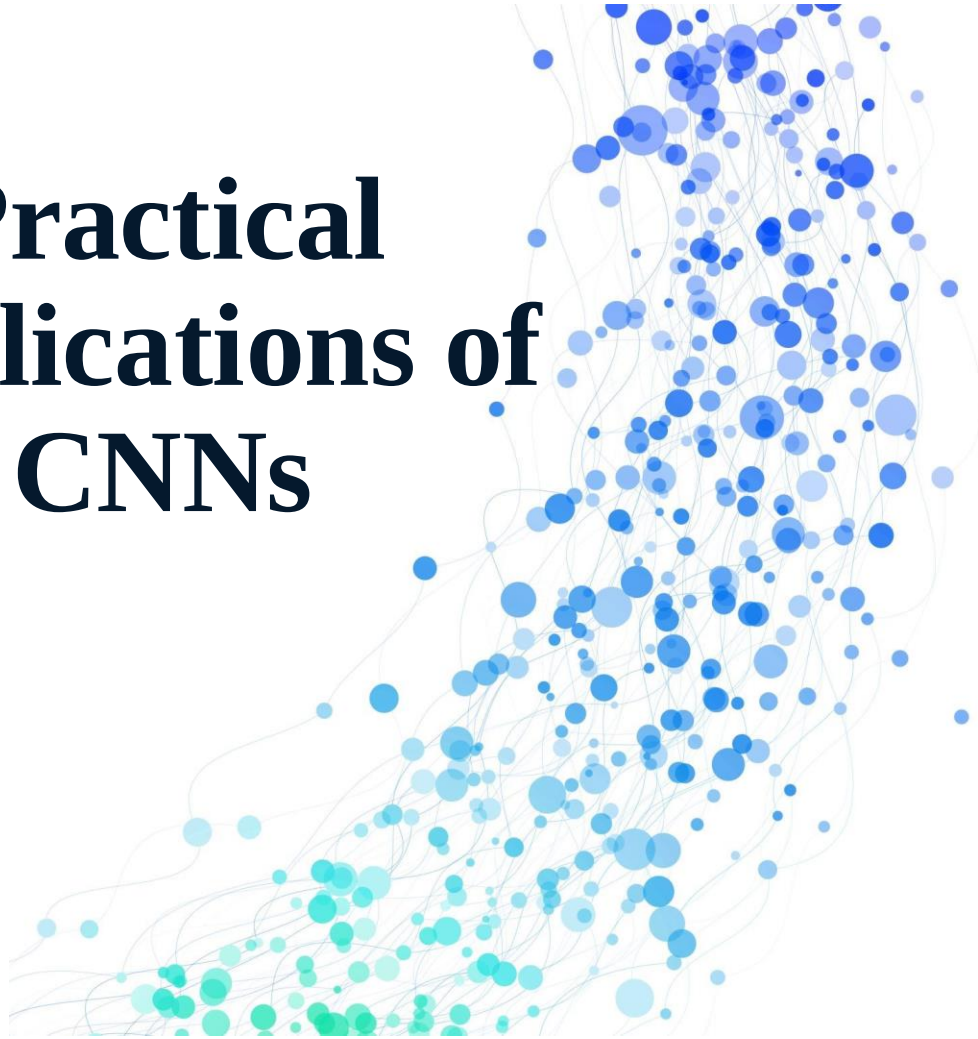
- A variety of pre-trained CNN architectures, including VGG-16, ResNet50, Inceptionv3, and EfficientNet, have demonstrated top-tier performance. These models can be adapted to new tasks with relatively little data through a process known as fine-tuning.

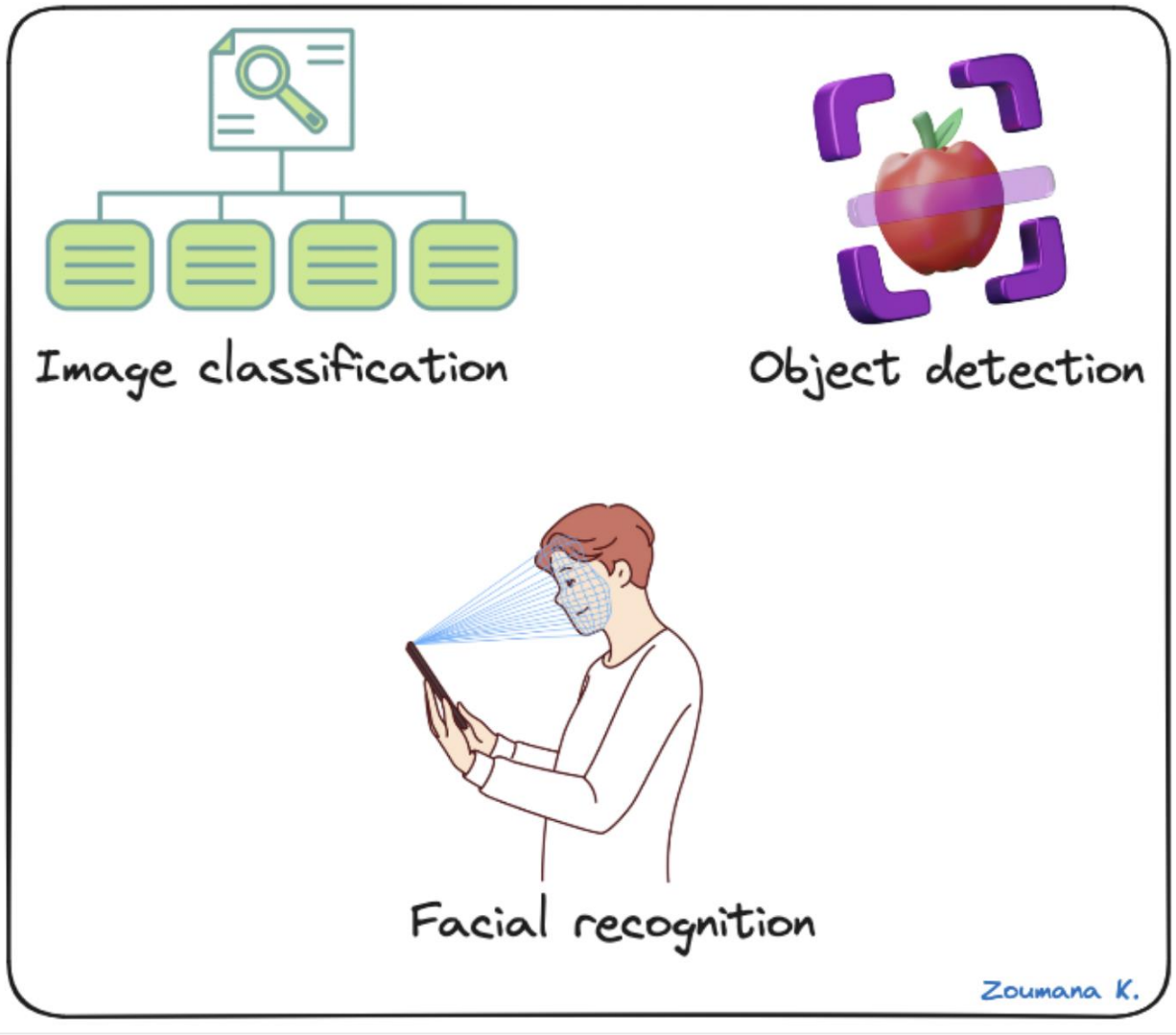
- Beyond image classification tasks, CNNs are versatile and can be applied to a range of other domains, such as natural language processing, time series analysis, and speech recognition.

Inspiration Behind CNN

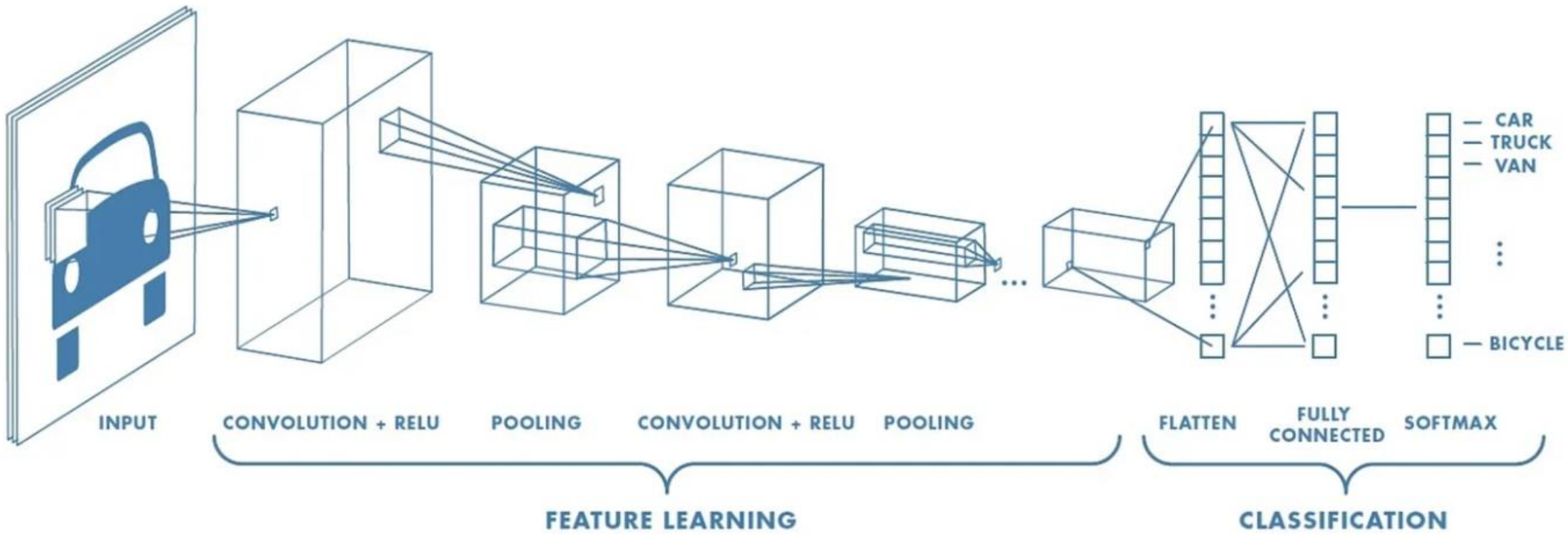


Practical Applications of CNNs

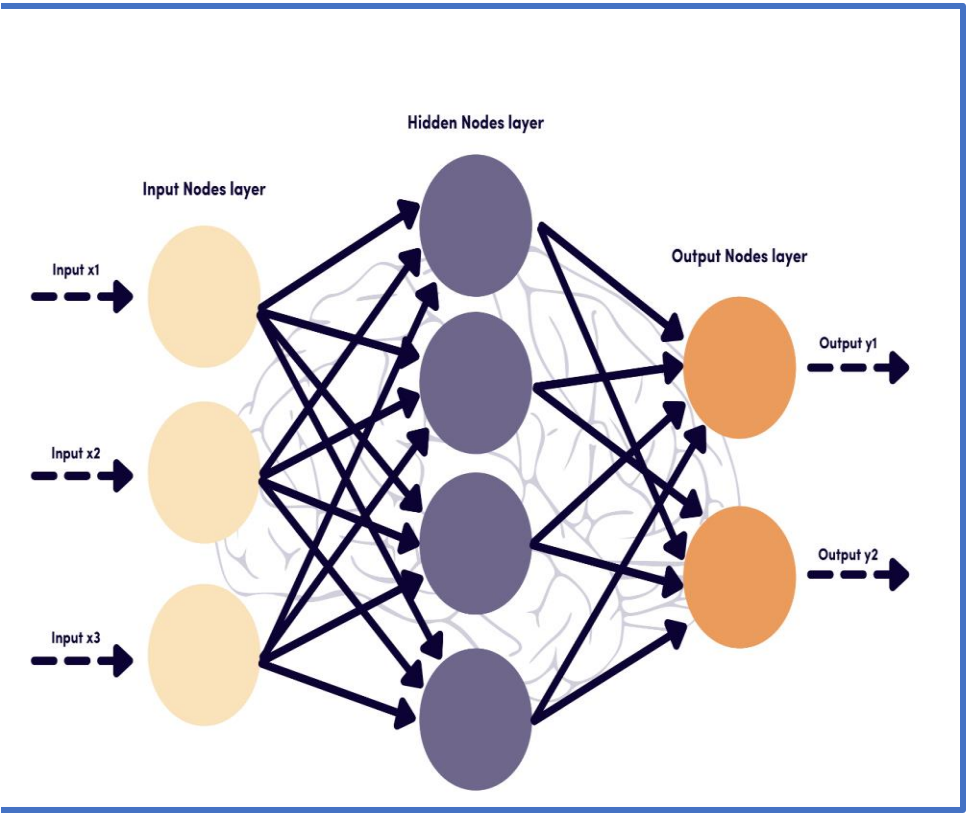




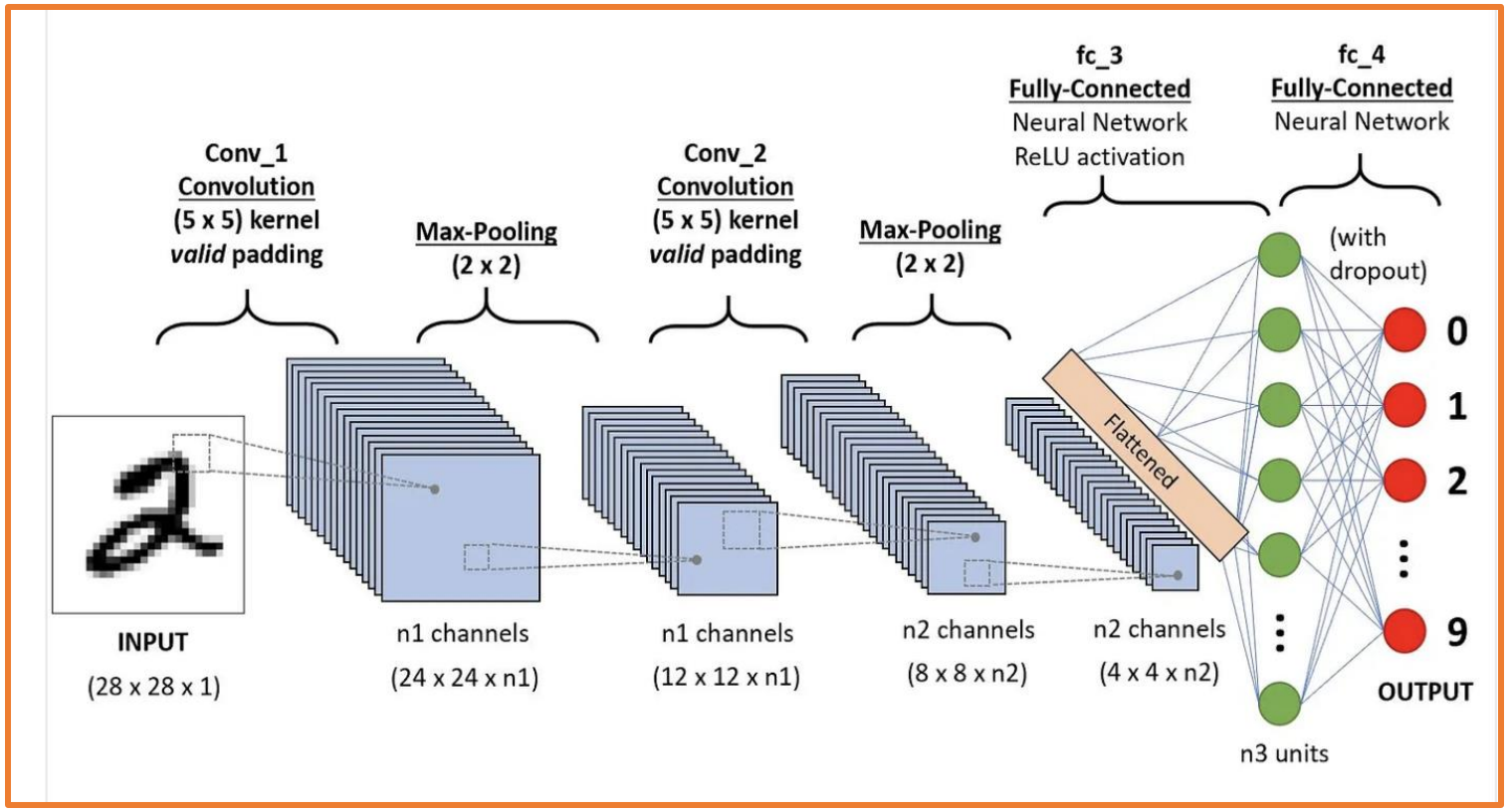
Architecture of CNN



ANN vs CNN

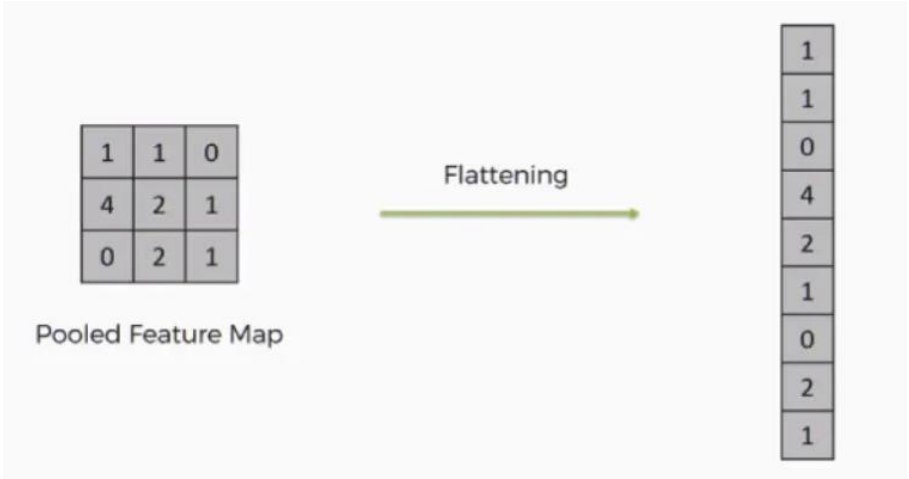


Architecture of ANN



Architecture of CNN

Motivation for CNNs



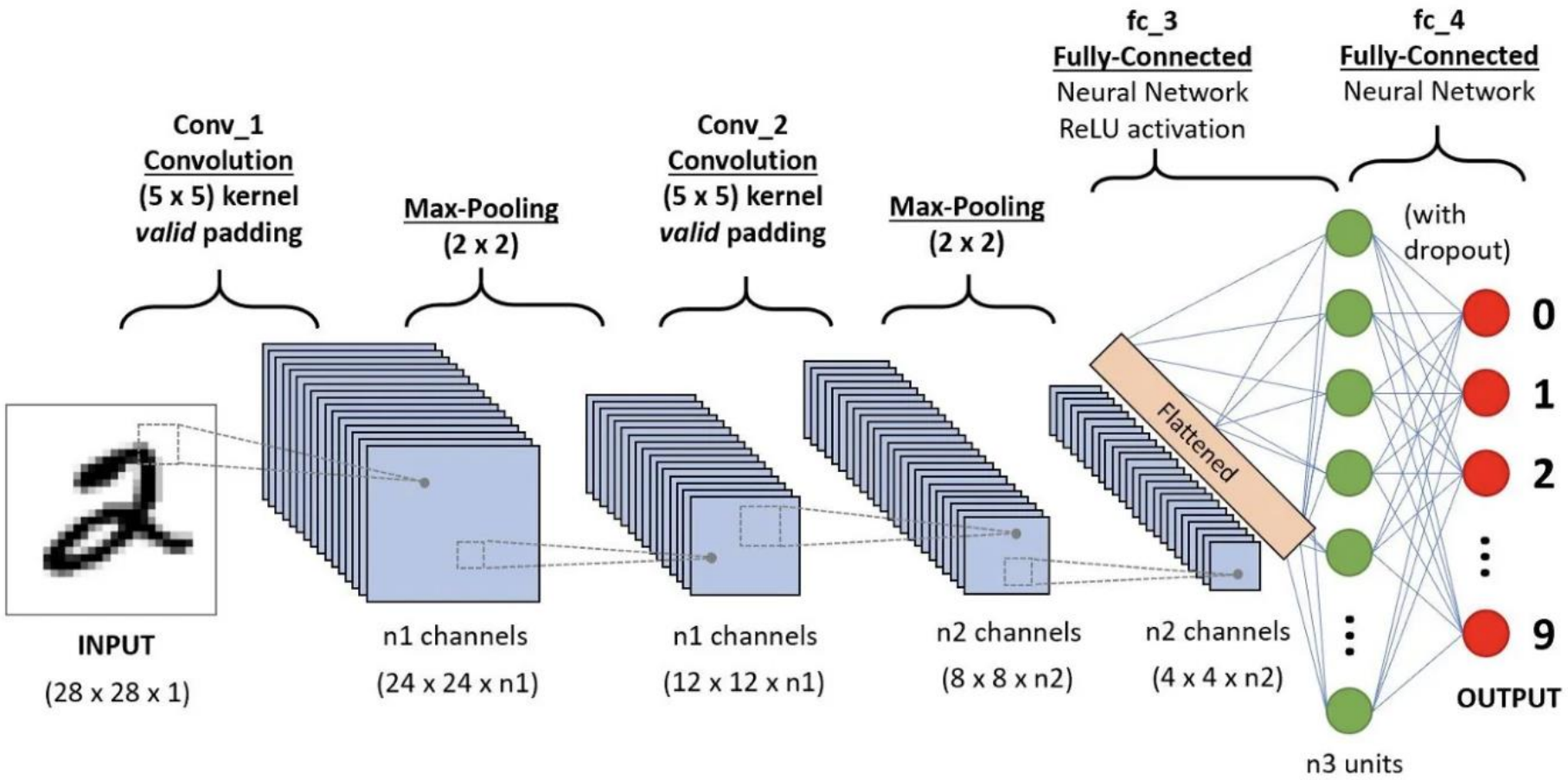
- To use ANNs with images, we need to flatten the image. If we do so, spatial information (relationships between the nearby pixels) will be lost. So, accuracy will be reduced significantly. CNNs can retain spatial information as they take the images in the original format.
- CNNs can reduce the number of parameters in the network significantly. So, CNNs are parameter efficient.

Definitions:

- **Flatten:** Flattening is a technique that is used to convert multi-dimensional arrays into a 1-D array, it is generally used in Deep Learning while feeding the 1-D array information to the classification model.
- **Spatial Information:** Relationships between the nearby pixels

Layers used to build CNN model:

- Input Layer
- Convolutional Layer
- Pooling Layer
- Fully Connected Layer
- Output Layer



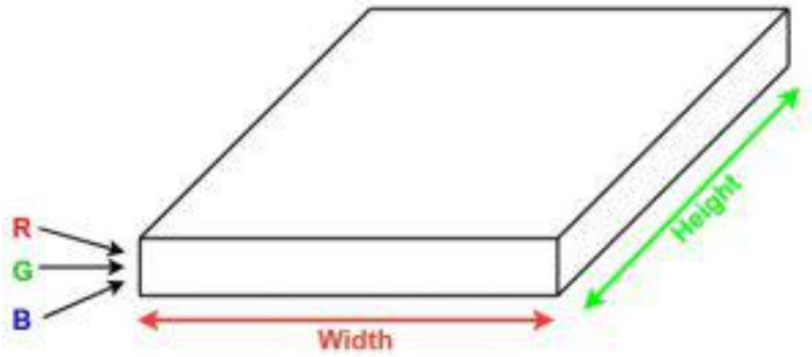
Input Layer

- It is the layer in which we give input to the model. In CNN, Generally, the input will be an image or a sequence of images, Non-image data (audio, time series, and signal data .

- Input layer in CNN should contain image data. Image data is represented by three-dimensional matrix, Width, Height, and Depth. You need to reshape it into a single column.

Example:

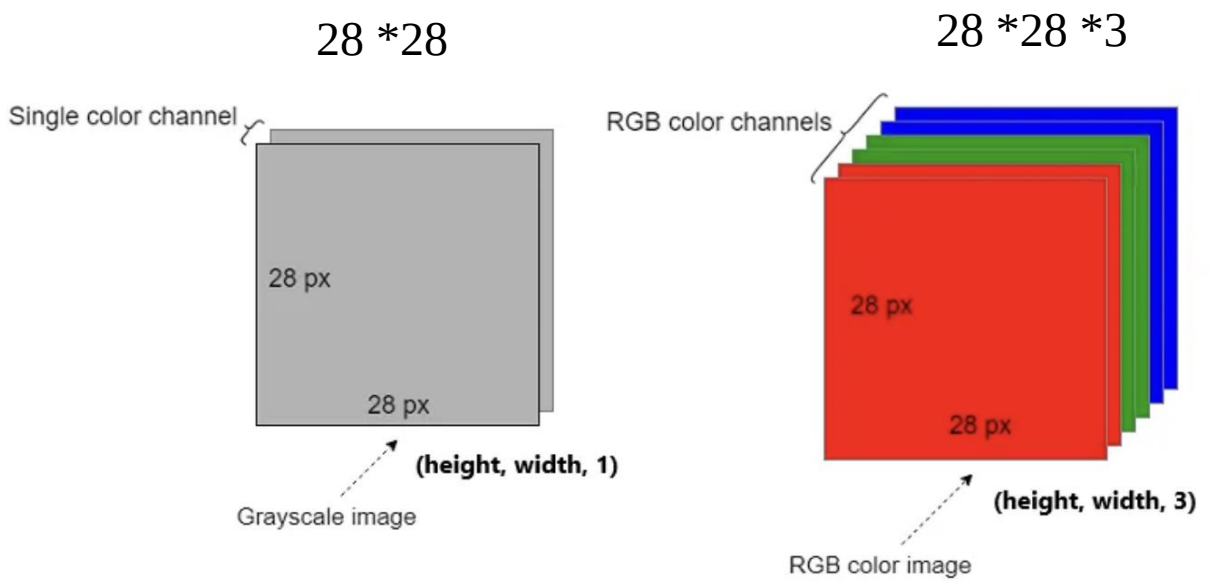
Imagine you have an image. It can be represented as a cuboid having its length, width (dimension of the image), and height (i.e the channel as images generally have red, green, and blue channels).



Image

- An image consists of pixels. In deep learning, images are represented as arrays of pixel values. There is only one-color channel in a grayscale image. So, a grayscale image is represented as (height, width). Therefore, a grayscale image is often represented as a 2D array (tensor).

- There are three color channels (**R**ed, **G**reen and **B**lue) in an RGB image. So, an RGB image is represented as (height, width, 3). The third dimension denotes the number of **color channels in the image.** An RGB image is represented as a 3D array (tensor).

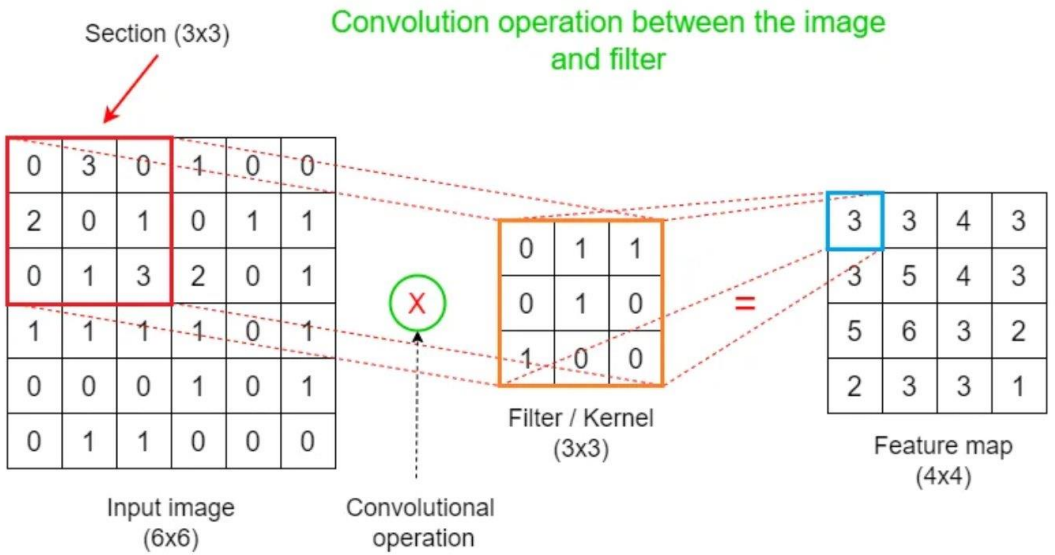


Convolutional Layer

- The first layer in a CNN is a convolutional layer. There can be multiple convolutional layers in a CNN. The first convolutional layer takes the images as the input and begins to process.
- There are three elements in the convolutional layer: **Input image**, **Filters** and **Feature map**. The ***convolution operation*** occurs in each convolutional layer.

Objectives:

• Extract a set of features from the image while maintaining relationships between the nearby pixels.



- **Filter:** This is also called Kernel or Feature Detector. This is a small matrix. There can be multiple filters in a single convolutional layer. The same-sized filters are used within a convolutional layer. Each filter has a specific function. Multiple filters are used to identify a different set of features in the image. The size of the filter and the number of filters should be specified by the user as hyperparameters. The size should be smaller than the size of the input image. The elements inside the filter define the filter configuration. These elements are a type of parameters in the CNN and are learned during the training.
- **Image section:** The size of the image section should be equal to the size of the filter(s) we choose. We can move the filter(s) vertically and horizontally on the input image to create different image sections. The number of image sections depends on the ***Stride*** we use.
- **Feature map:** The feature map stores the outputs of different convolution operations between different image sections and the filter(s). This will be the input for the next pooling layer. The number of elements in the feature map is equal to the number of different image sections that we obtained by moving the filter(s) on the image.

The convolution operation happens between a section of the image and the filter. It outputs the feature map (reduced image).

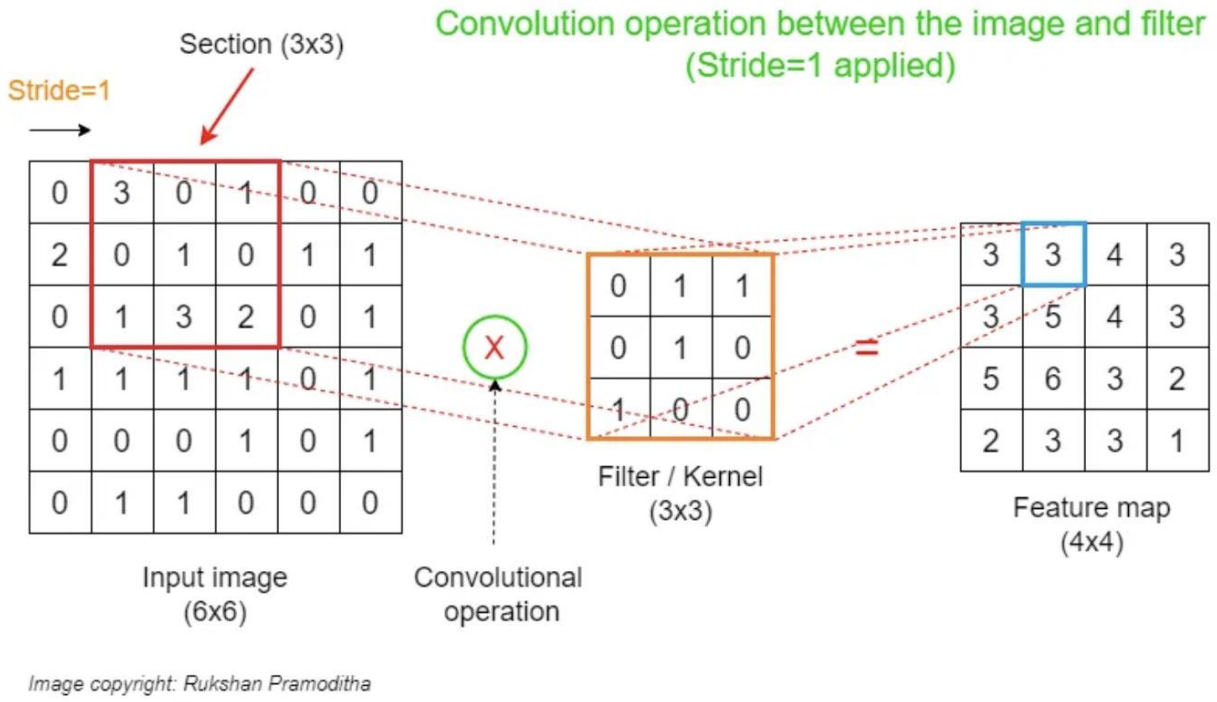
Convolution calculation

The diagram shows a convolution operation between an image section and a single filter. You can get row-wise or column-wise element multiplications and then summation.

```
# Row-wise
(0*0 + 3*1 + 0*1) + (2*0 + 0*1 + 1*0) + (0*1 + 1*0 + 3*0) = 3
```

Then we make another calculation by moving the filter on the image horizontally by one step to the right. The number of steps (pixels) that we shift the filter over the input image is called **Stride**. The shift can be done both horizontally and vertically. Here, we use Stride=1. Stride is also a hyperparameter that should be specified by the user.

```
# Row-wise
(3*0 + 0*1 + 1*1) + (0*0 + 1*1 + 0*0) + (1*1 + 3*0 + 2*0) = 3
```



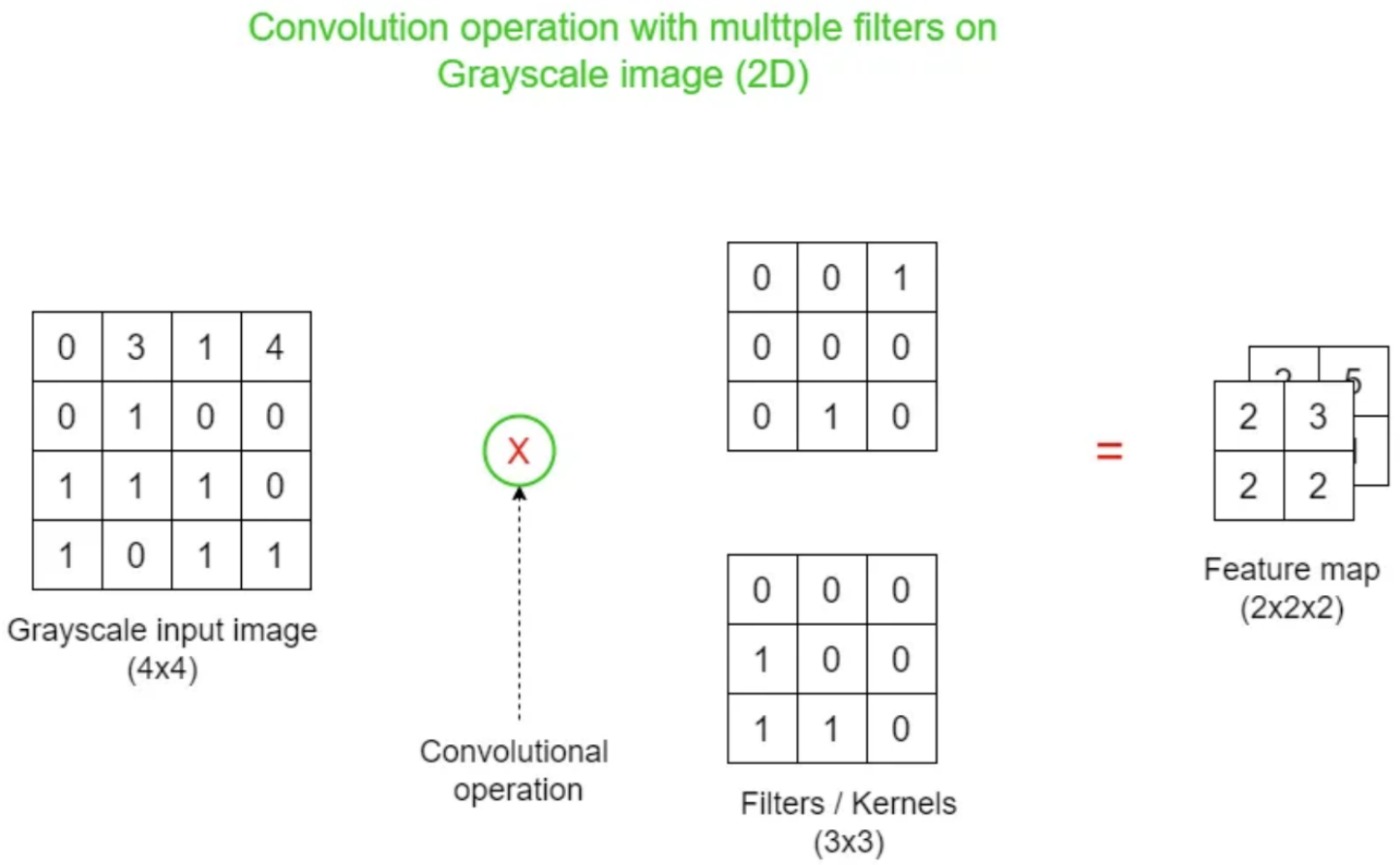
- The result of this calculation is placed in the corresponding area in the feature map.
- Likewise, we can do similar type of calculations by moving the filter on the image horizontally and vertically by one step (with Stride=1).

The size of the feature map is small than the size of the input image. The size of the feature map also depends on the Stride. If we use Stride=2, the size will be further reduced. If there are several convolutional layers in the CNN, the size of the feature map will be further reduced at the end so that we cannot do other operations on the feature map. To avoid this, we use apply **Padding** to the input image. Padding is a hyperparameter that we need to configure in the convolutional layer. It adds additional pixels with zero values to each side of the image. That helps to get the feature map of the same size as the input.

After applying Padding, the new size of the input image is (8, 8). If we do the convolution operation now with Stride=1, we get a feature map of size (6x6) that is equal to the size of the original image before applying Padding.

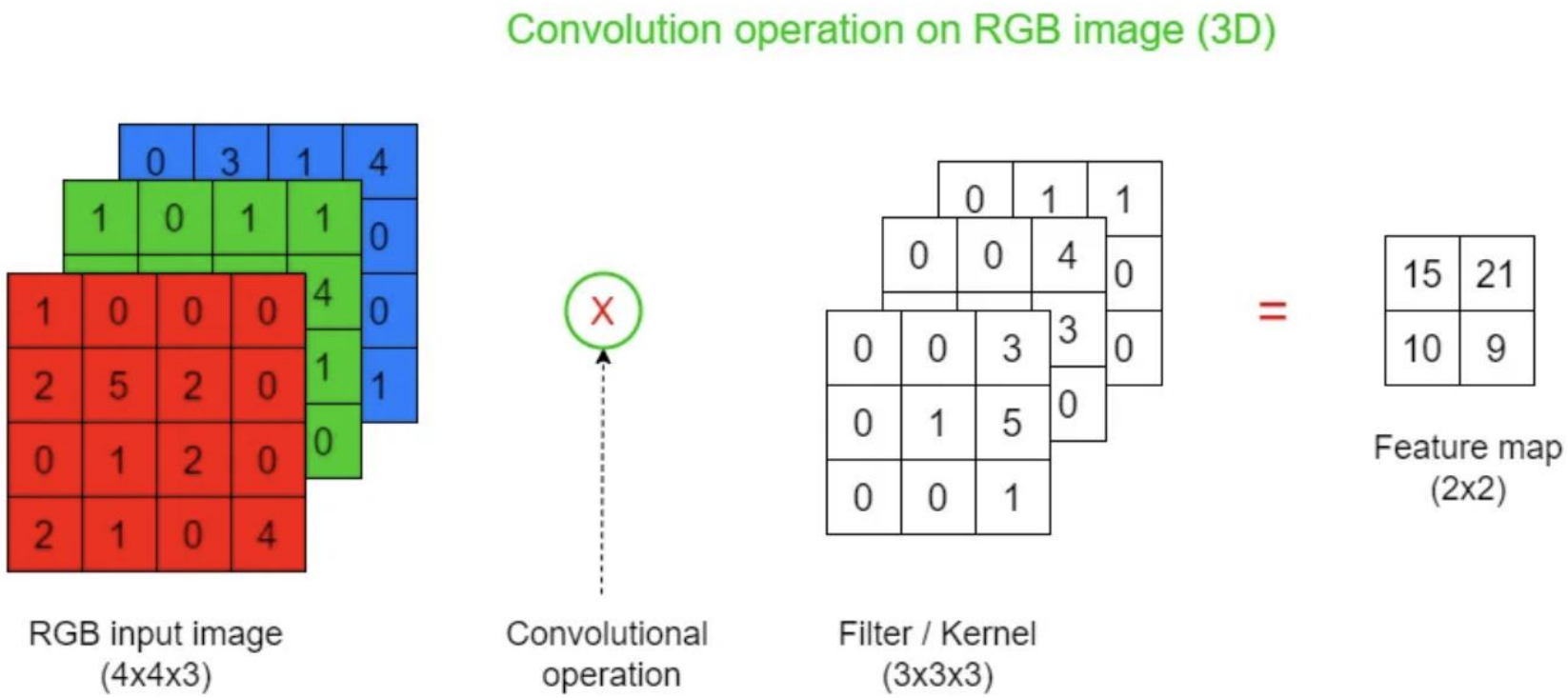
Convolution operation with multiple filters

The only difference is that another dimension is added to the feature map. It is the third dimension which denotes the number of filters.



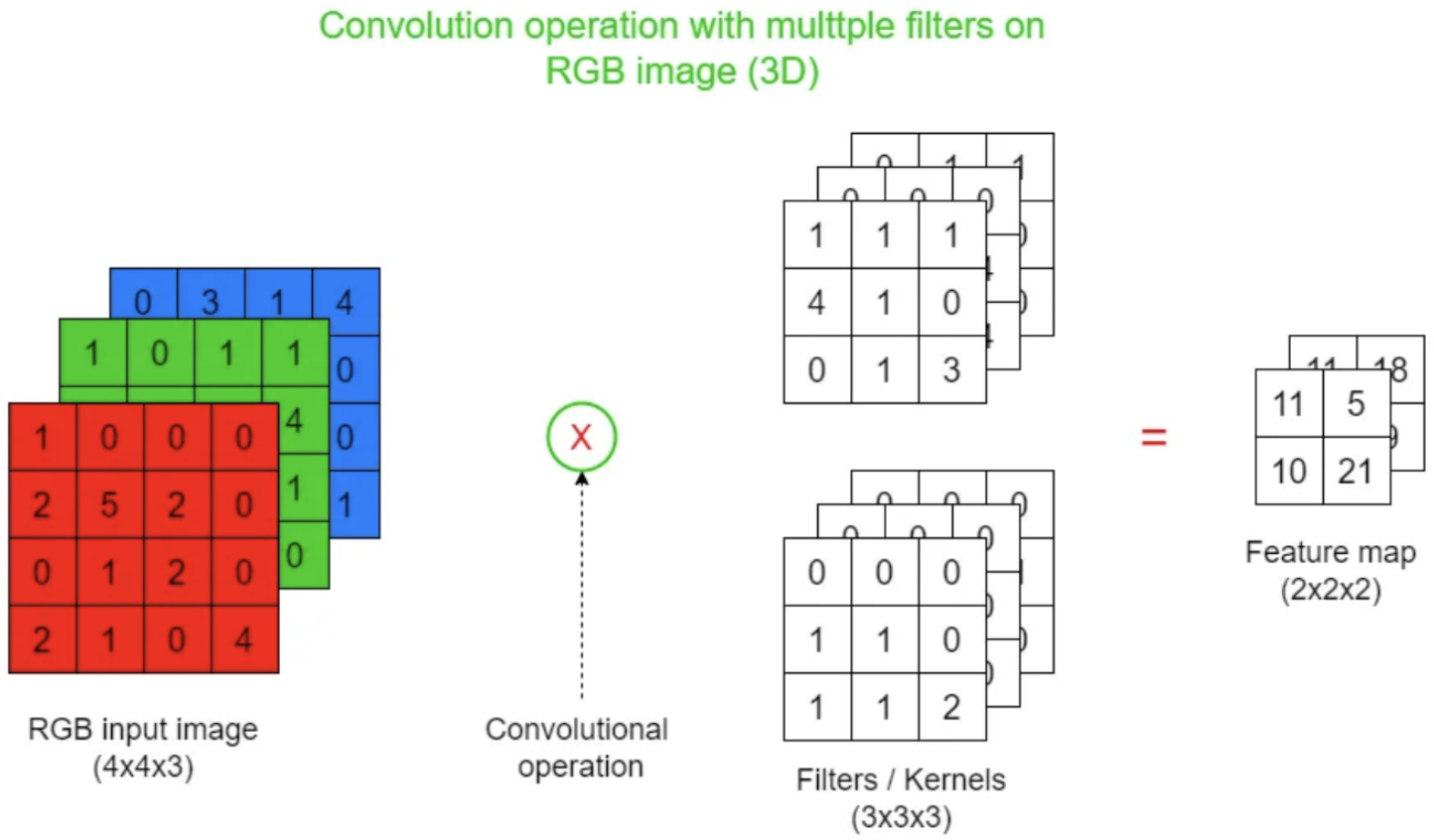
Convolution operation on an RGB image

When the image is RGB, the filter should have 3 channels. This is because an RGB image has 3 color channels and 3-channel filters are needed to do the calculations.
Here, the calculation happens on each corresponding channel between the image section and the filter as previously. The result is obtained by adding all outputs of each channel's calculations. That’s why the feature map does not have a third dimension.



Convolution operation on an RGB image with multiple filters

Another dimension is added to the feature map. It is the third dimension which denotes the number of filters.



Pooling Layer

Pooling layers are the second type of layer used in a CNN. There can be multiple pooling layers in a CNN. Each convolutional layer is followed by a pooling layer. So, convolution and pooling layers are used together as pairs.

Objectives:

1. Extract the most important (relevant) features by getting the maximum number or averaging the numbers.
2. Reduce the dimensionality (number of pixels) of the output returned from previous convolutional layers.
3. Reduce the number of parameters in the network.
4. Remove any noise present in the features extracted by previous convolutional layers.
5. Increase the accuracy of CNNs.

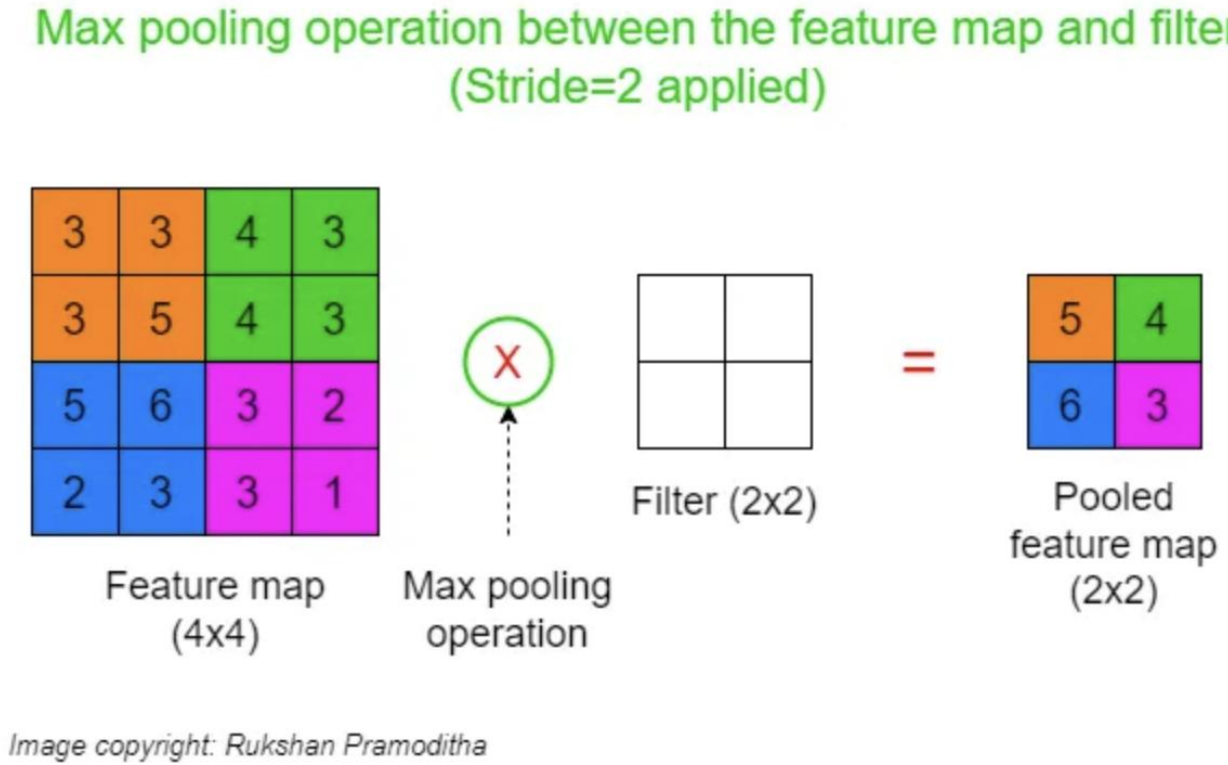
There are three elements in the pooling layer: **Feature map**, **Filter** and **Pooled feature map**. The ***pooling operation*** occurs in each pooling layer.

There are two types of pooling operations.

- **Max pooling:** Get the maximum value in the area where the filter is applied.
- **Average pooling:** Get the average of the values in the area where the filter is applied.

The following diagram shows the max pooling operation applied on the feature map obtained from the previous convolution operation.

- **Filter:** This time, the filter is just a window as there are no elements inside it. So, there are no parameters to learn in the pooling layer. The filter is just used to specify a section in the feature map. The size of the filter should be specified by the user as a hyperparameter. The size should be smaller than the size of the feature map. If the feature map has multiple channels, we should use a filter with the same number of channels. The pooling operations will be done on each channel independently.
- **Feature map section:** The size of the feature map section should be equal to the size of the filter we choose. We can move the filter vertically and horizontally on the feature map to create different sections. The number of sections depends on the Stride we use.
- **Pooled feature map:** The pooled feature map stores the outputs of different pooling operations between different feature map sections and the filter. This will be the input for the next convolution layer (if any) or for the flatten operation.



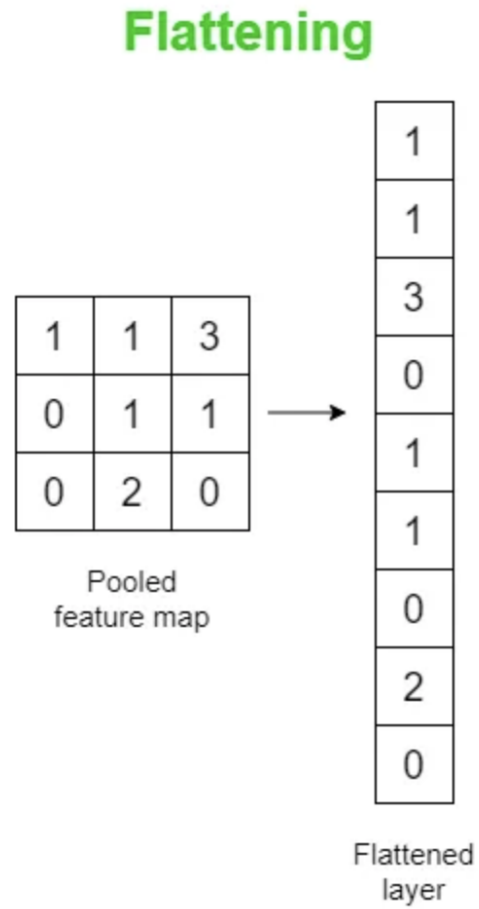
Flatten operation

In a CNN, the output returned from the final pooling layer (i.e. the final pooled feature map) is fed to a Multilayer Perceptron (MLP) that can classify the final pooled feature map into a class label.

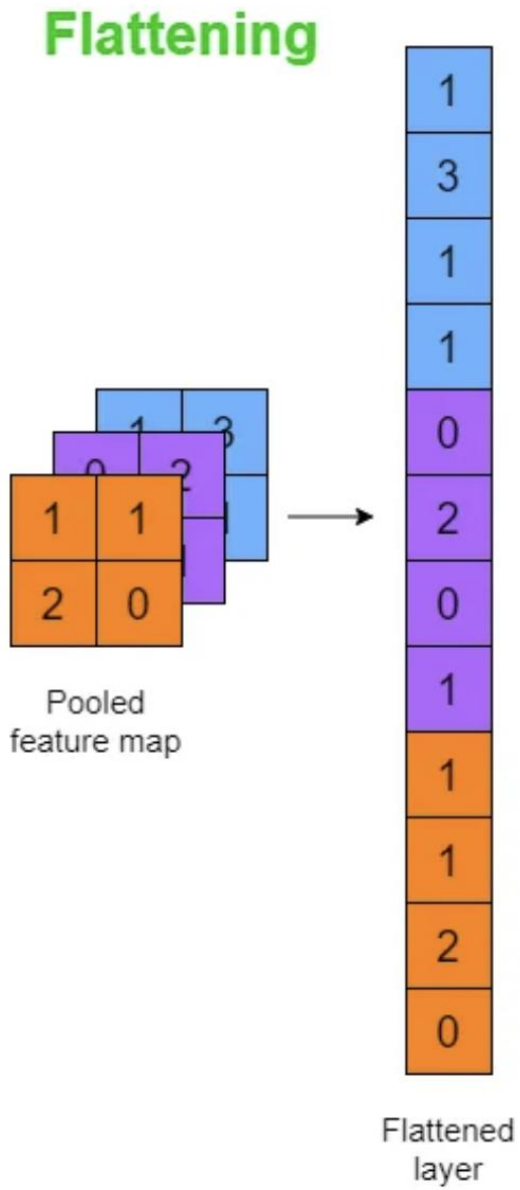
An MLP only accepts one-dimensional data. So, we need to flatten the final pooled feature map into a single column that holds the input data for the MLP.

Unlike flattening the original image, important pixel dependencies are retained when pooled maps are flattened.

The following diagram shows how we can flatten a pooled feature map that contains only one channel.



The following diagram shows how we can flatten a pooled feature map that contains multiple channels.



Fully Connected Layer

- These are the final layers in a CNN. The input is the previous flattened layer. There can be multiple fully connected layers. The final layer does the classification (or other relevant) task. An activation function is used in each fully connected layer.

Objectives:

- Classify the detected features in the image into a class label.

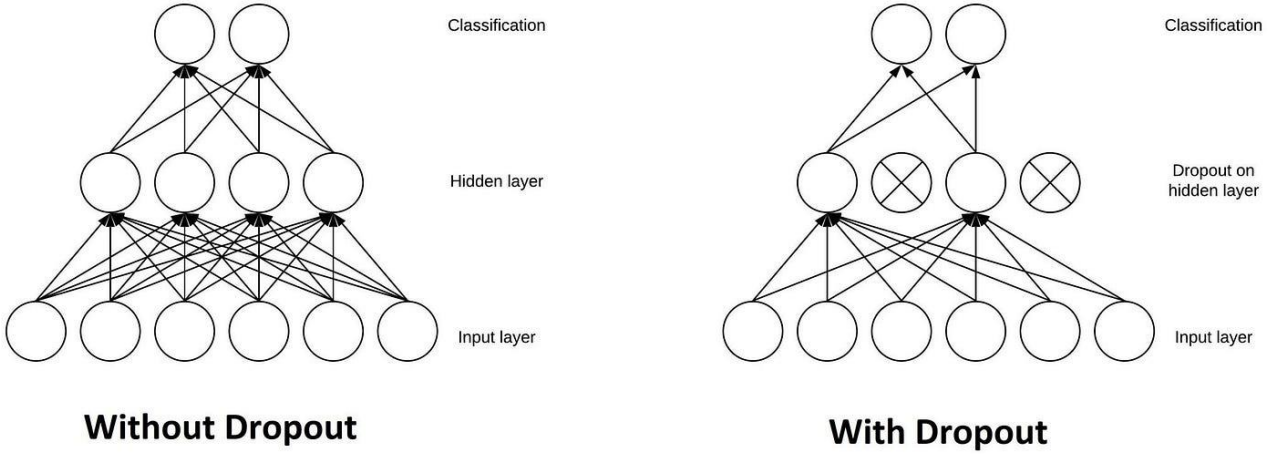
- The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture.

- Neurons in this layer have full connectivity with all neurons in the preceding and succeeding layer as seen in regular FCNN. This is why it can be computed as usual by a matrix multiplication followed by a bias effect.
- The FC layer helps to map the representation between the input and the output.

- The input image from the previous layers are flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place. The reason two layers are connected is that two fully connected layers will perform better than a single connected layer. These layers in CNN reduce the human supervision.

Dropout

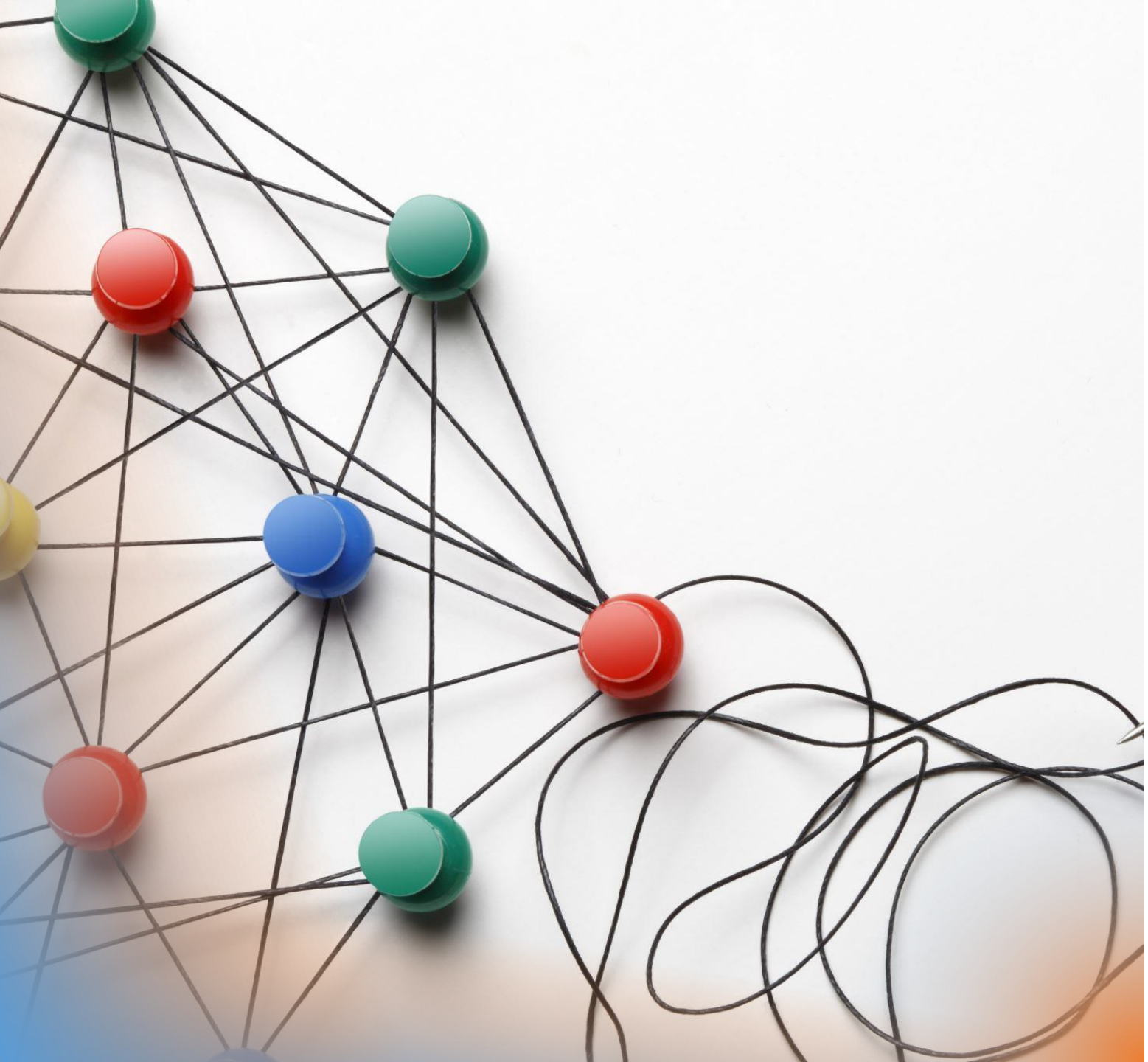
Another typical characteristic of CNNs is a Dropout layer. The Dropout layer is a mask that nullifies the contribution of some neurons towards the next layer and leaves unmodified all others.



Activation Function

An Activation Function decides whether a neuron should be activated or not. This means that it will decide whether the neuron's input to the network is important or not in the process of prediction. There are several commonly used activation functions such as the ReLU, Softmax, tanH, and the Sigmoid functions. Each of these functions has a specific usage.

- **Sigmoid:** For a binary classification in the CNN model
- **TanH:** he tanh function is very similar to the sigmoid function. The only difference is that it is symmetric around the origin. The range of values, in this case, is from -1 to 1.
- S**oftmax:** t is used in multinomial logistic regression and is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes.
- **ReLU:**the main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time.

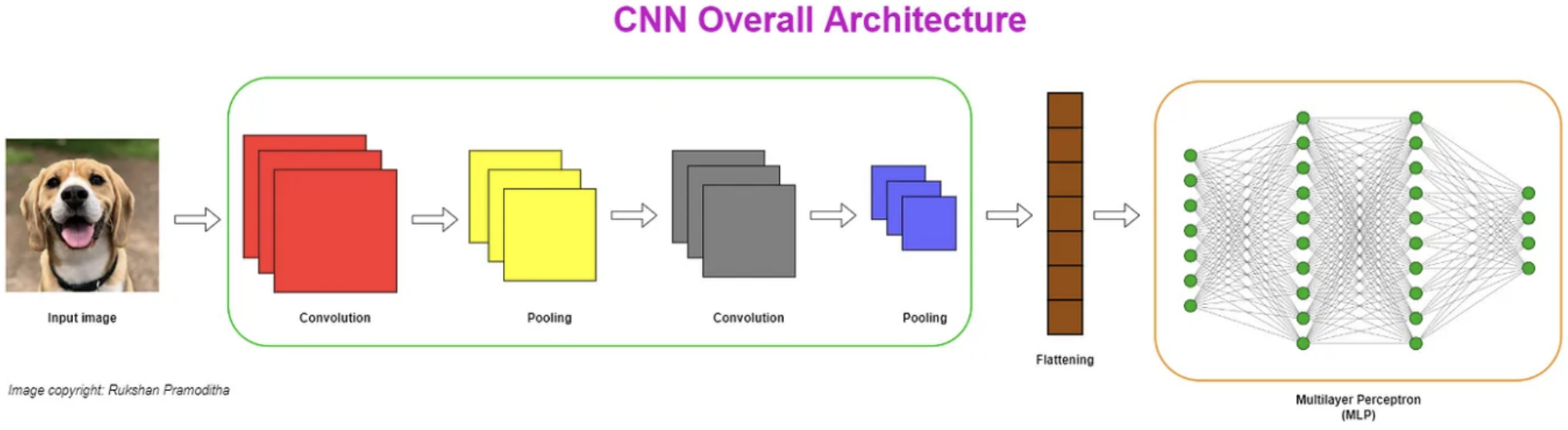


Output Layer

• The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or SoftMax which converts the output of each class into the probability score of each class.

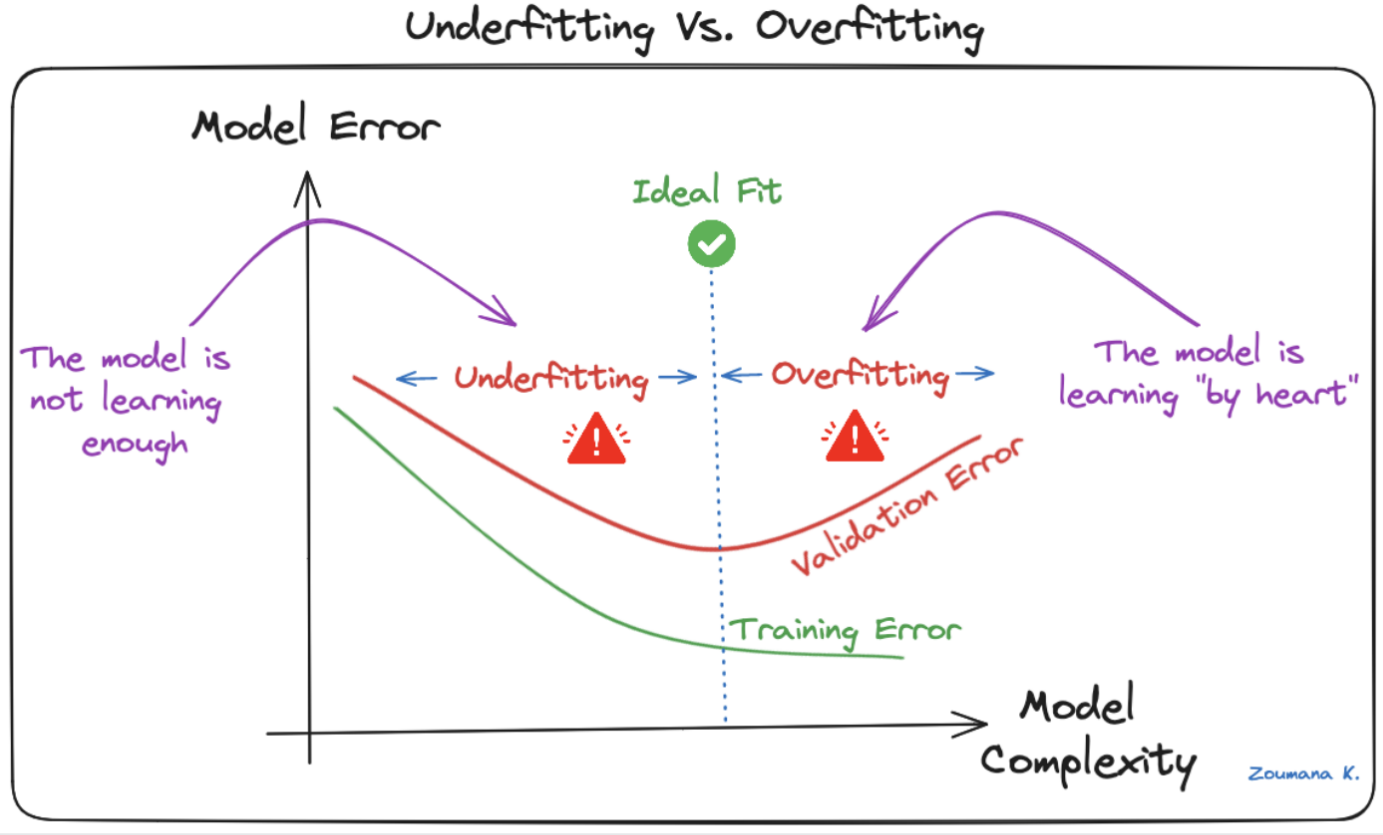
Example:

1. A CNN input takes the image as it is. The input image goes through a series of layers and operations.
2. Convolutional and pooling layers are needed to extract the features from the image while maintaining the important pixel dependencies. They also reduce the dimensionality (number of pixels) in the original image. These layers are used together as pairs.
3. The ReLU activation is used in each convolutional layer.
4. The number of filters increases in each convolutional layer. For example, if we use 16 filters in the first convolutional layer, we usually use 32 filters in the next convolutional layer, and so on.
5. The first few layers focus on less important patterns (such as edges) in the image data. The end layers find more complex patterns (e.g. nose, eyes in a face image). The final layer makes the classification task.
6. The ReLU activation is used in each fully connected layer except in the final layer in which we use the SoftMax activation for multiclass classification.



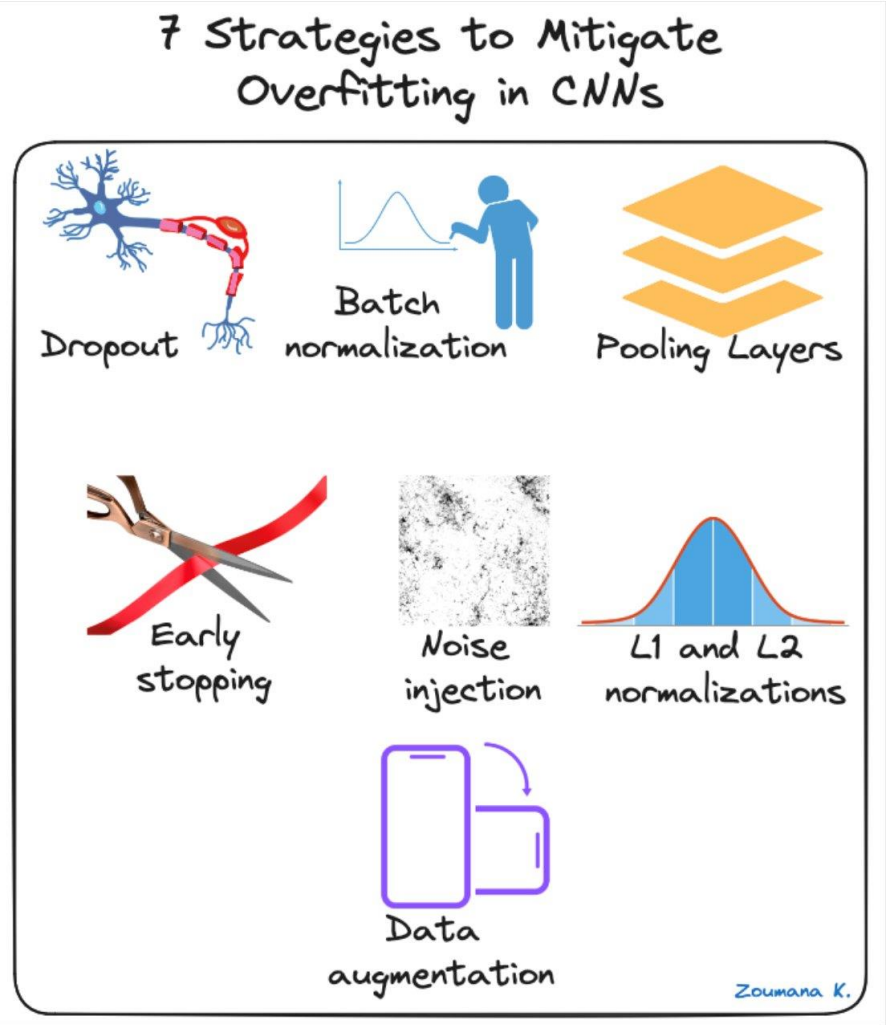
Overfitting and Regularization in CNNs

Overfitting is a common challenge in machine learning models and CNN deep learning projects. It happens when the model learns the training data too well (“learning by heart”), including its noise and outliers. Such a learning leads to a model that performs well on the training data but badly on new, unseen data. This can be observed when the performance on training data is too low compared to the performance on validation or testing data.



Underfitting Vs. Overfitting

Deep learning models, especially Convolutional Neural Networks (CNNs), are particularly susceptible to overfitting due to their capacity for high complexity and their ability to learn detailed patterns in large-scale data. Several regularization techniques can be applied to mitigate overfitting in CNNs.



7 strategies to mitigate overfitting in CNNs

Definitions

- **Dropout:** This consists of randomly dropping some neurons during the training process, which forces the remaining neurons to learn new features from the input data.
- **Batch normalization:** The overfitting is reduced at some extent by normalizing the input layer by adjusting and scaling the activations. This approach is also used to speed up and stabilize the training process.
- **Pooling Layers:** This can be used to reduce the spatial dimensions of the input image to provide the model with an abstracted form of representation, hence reducing the chance of overfitting.
- **Early stopping:** This consists of consistently monitoring the model's performance on validation data during the training process and stopping the training whenever the validation error does not improve anymore.
- **Noise injection:** This process consists of adding noise to the inputs or the outputs of hidden layers during the training to make the model more robust and prevent it from a weak generalization.
- **L1 and L2 normalizations:** Both L1 and L2 are used to add a penalty to the loss function based on the size of weights. More specifically, L1 encourages the weights to be spare, leading to better feature selection. On the other hand, L2 (also called weight decay) encourages the weights to be small, preventing them from having too much influence on the predictions.
- **Data augmentation:** This is the process of artificially increasing the size and diversity of the training dataset by applying random transformations like rotation, scaling, flipping, or cropping to the input images./.

Different Types of CNN

- ***LeNet:*** First CNN Architecture
- ***AlexNet*:** Deep Learning Architecture that popularized CNN
- ***GoogLeNet***: CNN Architecture used by Google
- ***VGGNet:*** CNN Architecture with Large Filters
- ***ResNet:*** CNN architecture that also got used for NLP tasks apart from Image Classification
- ***MobileNet****:* CNN Architecture for Mobile Devices
- ***ZFNet*:** Improved Version of AlexNet CNN Architecture
- ***GoogLeNet_DeepDream***: Generate images based on CNN features

Architecture	Year	Key Features	Use Case
LeNet	1998	First successful applications of CNNs, 5 layers (alternating between convolutional and pooling), Used tanh/sigmoid activation functions	Recognizing handwritten and machine-printed characters
AlexNet	2012	Deeper and wider than LeNet, Used ReLU activation function, Implemented dropout layers, Used GPUs for training	Large-scale image recognition tasks
ZFNet	2013	Similar architecture to AlexNet, but with different filter sizes and numbers of filters, Visualization techniques for understanding the network	ImageNet classification
VGGNet	2014	Deeper networks with smaller filters (3×3), All convolutional layers have the same depth, Multiple configurations (VGG16, VGG19)	Large-scale image recognition
ResNet	2015	Introduced “skip connections” or “shortcuts” to enable training of deeper networks, Multiple configurations (ResNet-50, ResNet-101, ResNet-152)	Large-scale image recognition, won 1st place in the ILSVRC 2015
GoogleLeNet	2014	Introduced Inception module, which allows for more efficient computation and deeper networks, multiple versions (Inception v1, v2, v3, v4)	Large-scale image recognition, won 1st place in the ILSVRC 2014
MobileNets	2017	Designed for mobile and embedded vision applications, Uses depthwise separable convolutions to reduce the model size and complexity	Mobile and embedded vision applications, real-time object detection

Please Refer to Canvas for Coding!


THANK YOU
Any Questions?