Convolution Neural Networks (CNN)

What is Computer Vision?

Computer Vision involves developing algorithms and techniques to process and analyze images and videos. The goal is to enable computers to perform tasks such as object recognition, image classification, and scene reconstruction[1][2].

Key Components

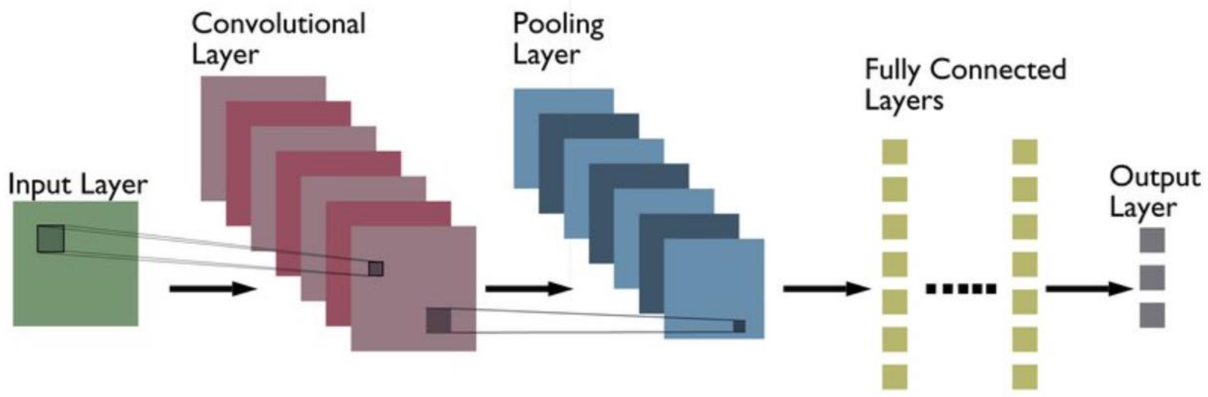
1. **Image Acquisition: Capturing images or video frames using cameras or sensors.**
2. **Preprocessing: Enhancing image quality and preparing data for analysis.**
3. **Feature Extraction: Identifying important features or patterns within the images.**
4. **Model Training: Using machine learning models, particularly deep learning and convolutional neural networks (CNNs), to learn from the data.**
5. **Inference: Applying the trained model to new images to make predictions or decisions[2]**

Applications

- **Healthcare**: Medical imaging for diagnosis and treatment planning.
- **Automotive**: Autonomous vehicles and driver assistance systems.
- **Retail**: Automated checkout systems and inventory management.
- **Security**: Surveillance and facial recognition systems

CNN architecture overview (e.g., LeNet-5, AlexNet, VGG, ResNet, Inception, etc.)

Convolutional Neural Networks, commonly referred to as CNNs, are a specialized kind of neural network architecture that is designed to process data with a grid-like topology. This makes them particularly well-suited for **dealing with spatial and temporal data**, like **images** and **videos**, that maintain a high degree of correlation between adjacent elements.



Convolutional layers perform a **mathematical operation** called **convolution**, a sort of **specialized matrix multiplication**, on the input data. The convolution operation helps to preserve the spatial relationship between pixels by learning image features using small squares of input data.

Convolutional layers operate by sliding a set of ‘filters’ or ‘kernels’ across the input data. Each filter is designed to detect a specific feature or pattern, such as edges, corners, or more complex shapes in the case of deeper layers. As these filters move across the image, they generate a map that signifies the areas where those features were found. **The output of the convolutional layer is a feature map**, which is a representation of the input image with the filters applied. Convolutional layers can be stacked to create more complex models, which can learn more intricate features from images. Simply speaking, convolutional layers are responsible for extracting features from the input images. These features might include edges, corners, textures, or more complex patterns.

Pooling Layers

Pooling layers follow the convolutional layers and are used to **reduce the spatial dimension of the input**, making it easier to process and requiring less memory. In the context of images, “spatial dimensions” refer to the width and height of the image. An image is made up of pixels, and you can think of it like a grid, with rows and columns of tiny squares (pixels). By reducing the spatial dimensions, pooling layers **help reduce the number of parameters or weights** in the network. This helps to **combat overfitting** and help train the model in a fast manner. Max pooling helps in reducing computational complexity owing to reduction in size of feature map, and, making the model invariant to small transitions. Without max pooling, the network would not gain the ability to recognize features irrespective of small shifts or rotations. This would make the model less robust to variations in object positioning within the image, possibly affecting accuracy.

There are two main types of pooling: **max pooling** and **average pooling**. Max pooling takes the maximum value from each feature map. For example, if the pooling window size is 2×2, it will pick the pixel with the highest value in that 2×2 region. Max pooling effectively captures the most prominent feature or characteristic within the pooling window. Average pooling calculates the average of all values within the pooling window. It provides a smooth, average feature representation.

Fully Connected Layers

Fully-connected layers are one of the most basic types of layers in a convolutional neural network (CNN). As the name suggests, each neuron in a fully-connected layer is Fully connected- to every other neuron in the previous layer. Fully connected layers are typically used towards the end of a CNN- when the goal is to take the features learned by the convolutional and max pooling layers and use them to make predictions such as classifying the input to a label. For example, if we were using a CNN to classify images of animals, the final Fully connected layer might take the features learned by the previous layers and use them to classify an image as containing a dog, cat, bird, etc.

Fully connected layers take the high-dimensional output from the previous convolutional and pooling layers and flatten it into a one-dimensional vector. This allows the network to combine and integrate all the extracted features across the entire image, rather than considering localized features. It helps in understanding the global context of the image. The fully connected layers are responsible for mapping the integrated features to the desired output, such as class labels in classification tasks. They act as the final decision-making part of the network, determining what the extracted features mean in the context of the specific problem (e.g., recognizing a cat or a dog).

The combination of Convolution layer followed by max-pooling layer and then similar sets creates a hierarchy of features. The first layer detects simple patterns, and subsequent layers build on those to detect more complex patterns.

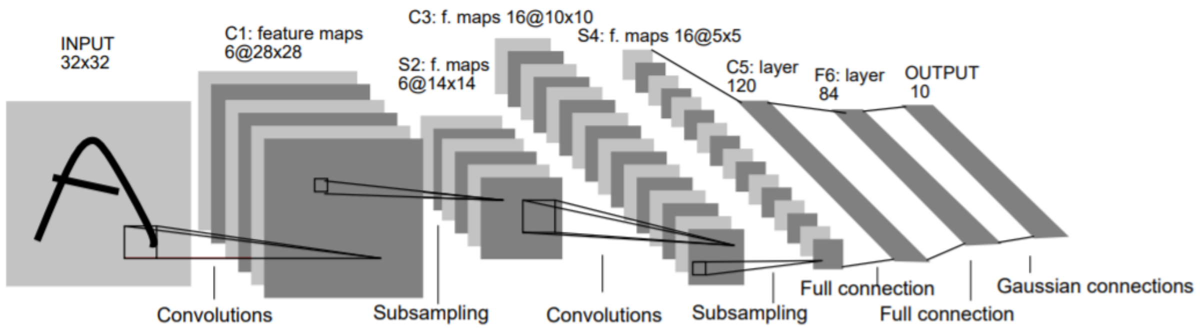
The output layer in a Convolutional Neural Network (CNN) plays a critical role as it's the final layer that produces the actual output of the network, typically in the form of a classification or regression result. Its importance can be outlined as follows:

1. **Transformation of Features to Final Output**: The earlier layers of the CNN (convolutional, pooling, and fully connected layers) are responsible for extracting and transforming features from the input data. The output layer takes these high-level, abstracted features and transforms them into a final output form, which is directly interpretable in the context of the problem being solved.

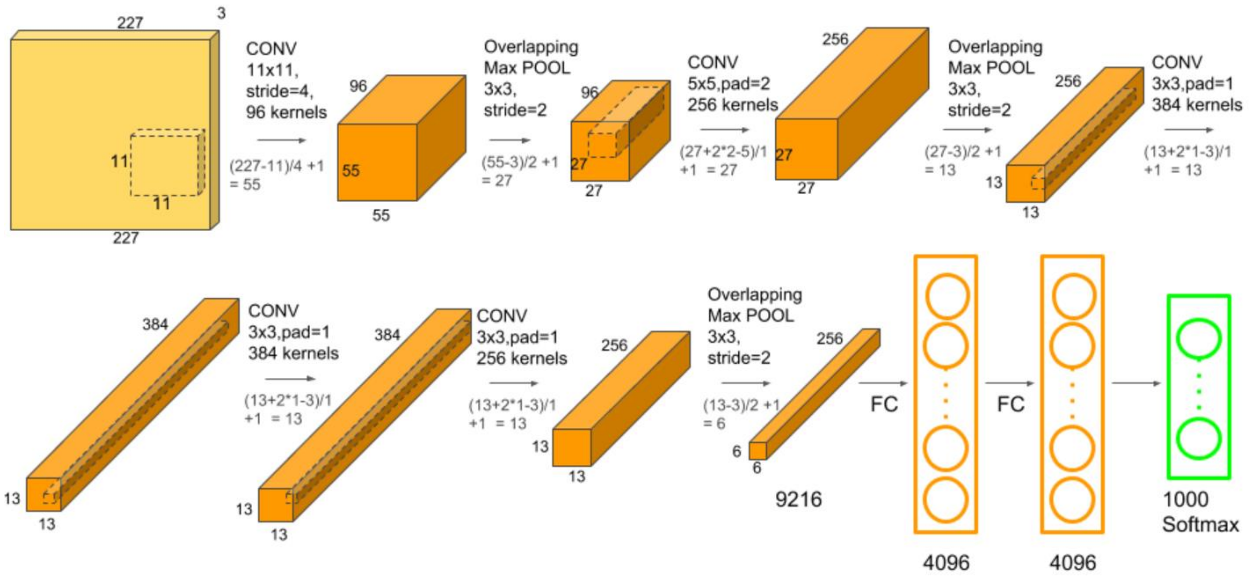
2. **Task-Specific Formulation**:

- For classification tasks, the output layer typically uses a softmax activation function, which converts the input from the previous layers into a probability distribution over the predefined classes. The softmax function ensures that the output probabilities sum to 1, making them directly interpretable as class probabilities.
- For regression tasks, the output layer might consist of one or more neurons with linear or no activation function, providing continuous output values.

3. LeNet – First CNN Architecture

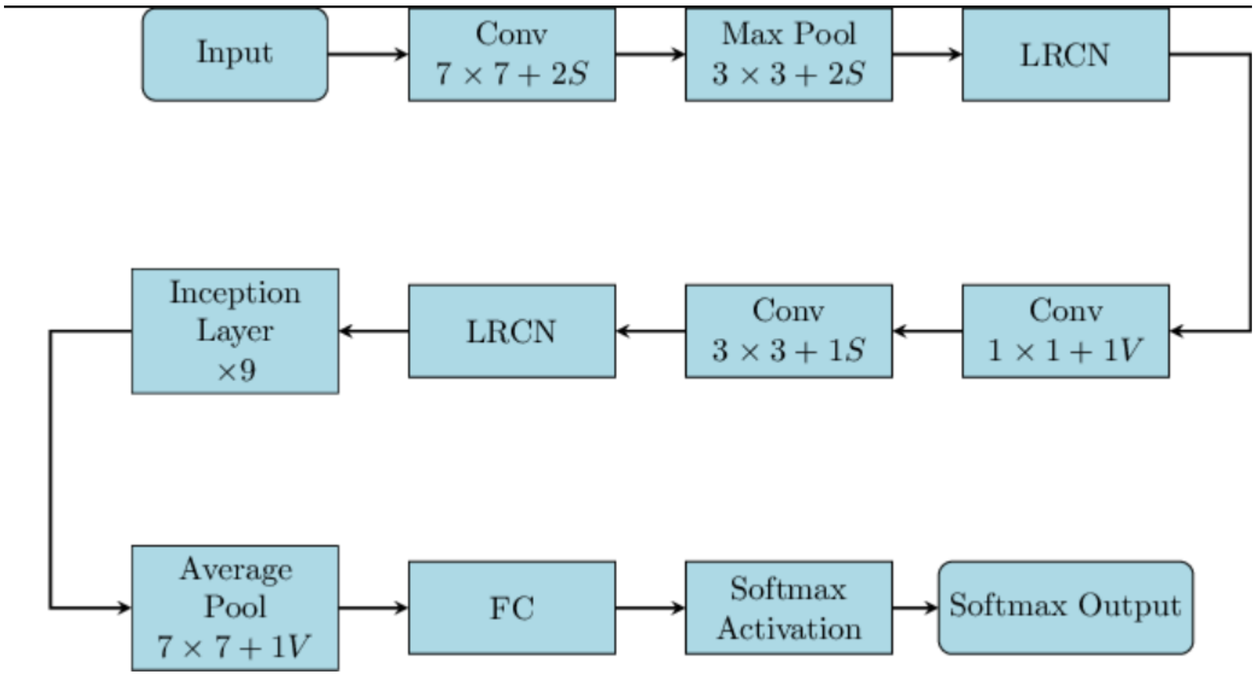


AlexNet



227
3
11
11
227
CONV 11x11, stride=4, 96 kernels
(227-11)/4 +1 = 55
96
55
55
Overlapping Max POOL 3x3, stride=2
(55-3)/2 +1 = 27
96
27
27
CONV 5x5,pad=2 256 kernels
(27+2*2-5)/1 +1 = 27
256
27
27
Overlapping Max POOL 3x3, stride=2
(27-3)/2 +1 = 13
256
13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
13
13
CONV 3x3,pad=1 256 kernels
(13+2*1-3)/1 +1 = 13
256
13
13
Overlapping Max POOL 3x3, stride=2
(13-3)/2 +1 = 6
256
6
6
9216
FC
FC
4096
4096
1000 Softmax

GoogleLeNet



Code

```
from sklearn.datasets import load_digits

from sklearn.model_selection import train_test_split

from tensorflow.keras.utils import to_categorical

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D


# Load the digits dataset

digits = load_digits()

X, y = digits.images, digits.target


# Preprocessing
# Normalizing pixel values

X = X / 16.0


# Reshaping the data to fit the model

# CNN in Keras requires an extra dimension at the end for channels,

# and the digits images are grayscale so it's just 1 channel

X = X.reshape(-1, 8, 8, 1)


# Convert labels to categorical (one-hot encoding)

y = to_categorical(y)


# Splitting into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Building a simple CNN model

model = Sequential()

model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(8, 8, 1)))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(50, activation='relu'))

model.add(Dense(10, activation='softmax'))  # 10 classes for digits 0-9


# Compiling the model

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])


# Training the model
```

```
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10)


# Evaluate the model
accuracy = model.evaluate(X_test, y_test)[1]
print(&amp;quot;Test accuracy:&amp;quot;, accuracy)
```

input, convolution layer, filters, stride and padding, activation function, output feature maps

Convolutional Layer

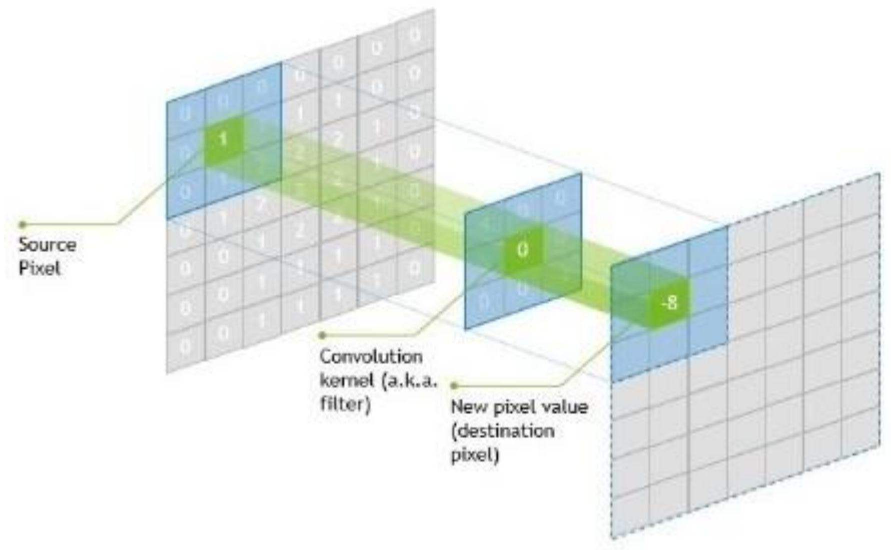
In convolutional neural networks (CNNs), the primary components are convolutional layers. These layers typically involve input vectors, like an image, filters (or feature detectors), and output vectors, which are often referred to as feature maps. As the input, such as an image, traverses through a convolutional layer, it undergoes abstraction into a feature map, also known as an activation map. This process involves the convolution operation, which enables the detection of more complex features within the image. Additionally, rectified linear units (ReLU) are commonly used as activation functions within these layers to introduce non-linearity into the network. Furthermore, CNNs often employ pooling operations to reduce the spatial dimensions of the feature maps, leading to a more manageable output volume. Overall, convolutional layers play a crucial role in extracting meaningful features from the input data, making them fundamental in tasks such as image classification and natural language processing, among others, within the realm of machine learning models.

Feature Map = Input Image x Feature Detector

The input is convolved by convolutional layers, which then pass the output to the next layer. This is analogous to a neuron's response to a single stimulus in the visual cortex. Each convolutional neuron only processes data for the receptive field it is assigned to.

A convolution is a grouping function in mathematics. Convolution occurs in CNNs when two matrices (rectangular arrays of numbers arranged in columns and rows) are combined to generate a third matrix.

In the convolutional layers of a CNN, these convolutions are used to filter input data and find information.


Source Pixel
Convolution kernel (a.k.a. filter)
New pixel value (destination pixel)

Source: Cadalyst.com

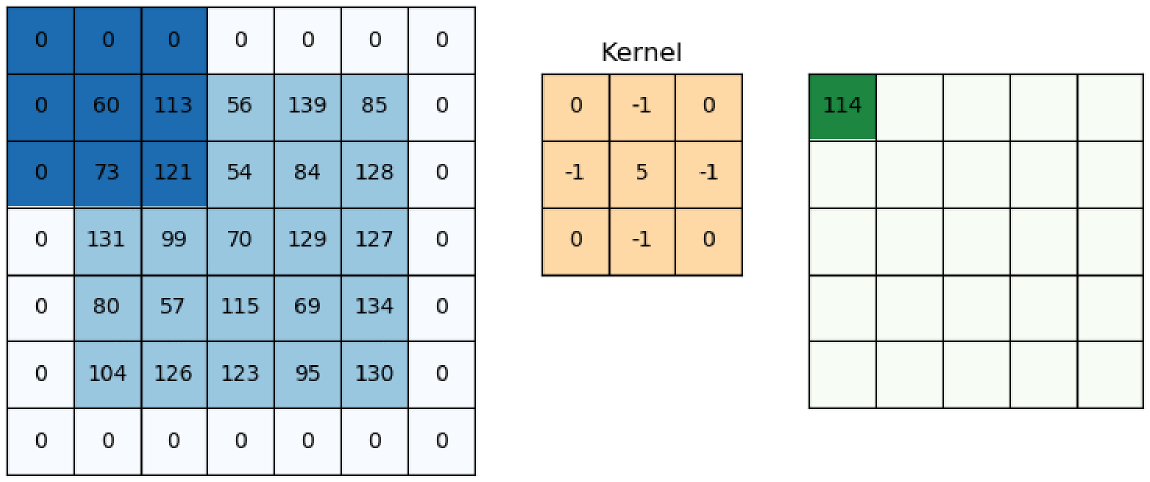
The kernel’s centre element is put above the source pixel. After that, the source pixel is replaced with a weighted sum of itself and neighboring pixels.

Parameter sharing and local connectivity are two principles used in CNNs. All neurons in a feature map share weights, which is known as parameter sharing. Local connection refers to the idea of each neural being connected to only a part of the input image (as opposed to a neural network in which all neurons are fully connected). This reduces the number of parameters in the system and speeds up the calculation.

Padding and Stride

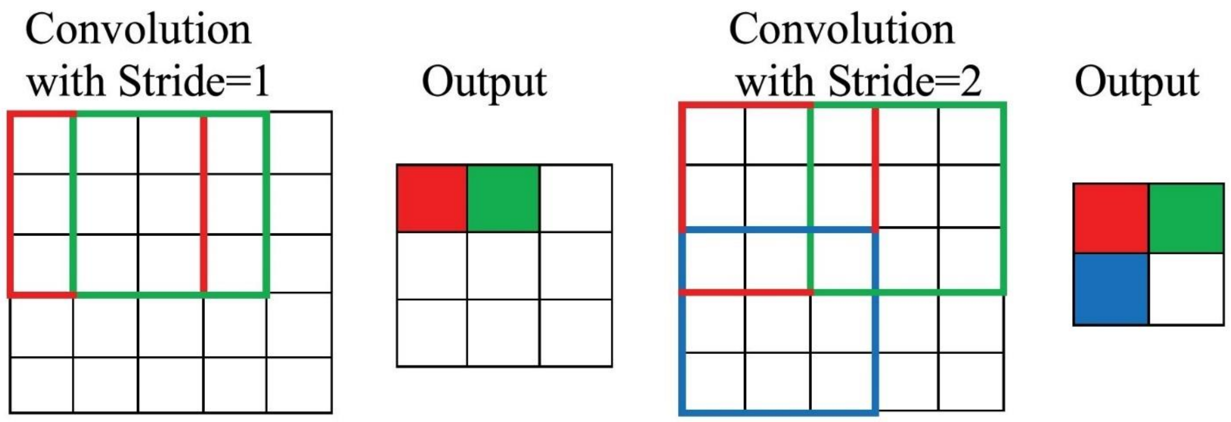
Padding and stride have an impact on how the convolution procedure is carried out. They can be used to increase or decrease the dimensions (height and width) of input/output vectors.

It is a term used in convolutional neural networks to describe how many pixels are added to an image when it is processed by the CNN kernel. If the padding in a CNN is set to zero, for example, every pixel value-added will have the value zero. If the zero padding is set to one, a one-pixel border with a pixel value of zero will be added to the image.



Source: vitalflux.com

Padding works by increasing the processing region of a convolutional neural network. The kernel is a neural network filter that moves through a picture, scanning each pixel and turning the data into a smaller or bigger format. Padding is added to the image frame to aid the kernel in processing the image by providing more room for the kernel to cover the image. Adding padding to a CNN-processed image provides for more accurate image analysis.

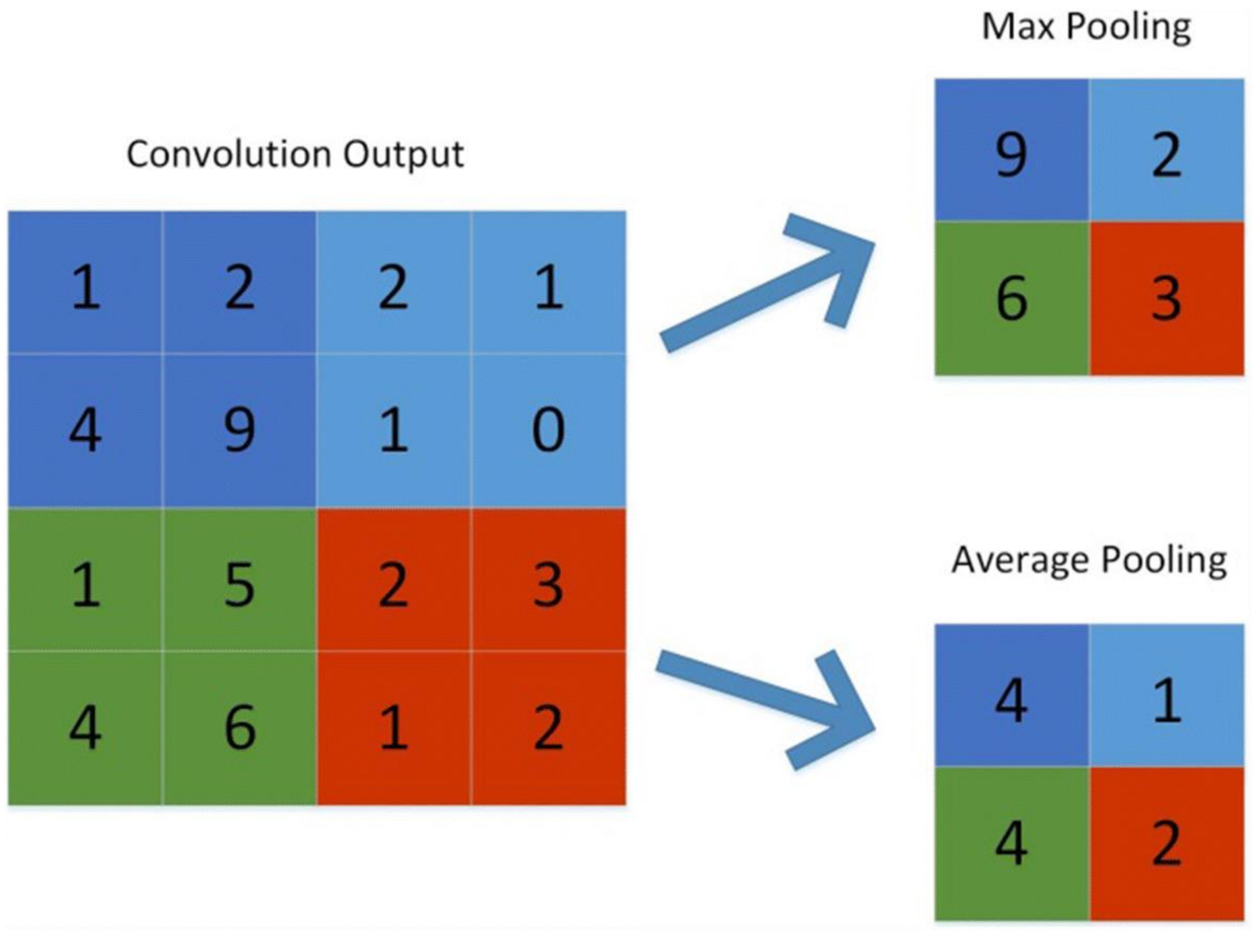


Source: Computer.org

Stride determines how the filter convolves over the input matrix, i.e. how many pixels shift. When stride is set to 1, the filter moves across one pixel at a time, and when the stride is set to 2, the filter moves across two pixels at a time. The smaller the stride value, the smaller the output, and vice versa.

Pooling

Its purpose is to gradually shrink the representation's spatial size to reduce the number of parameters and computations in the network. The pooling layer treats each feature map separately.



Source: Springer.com

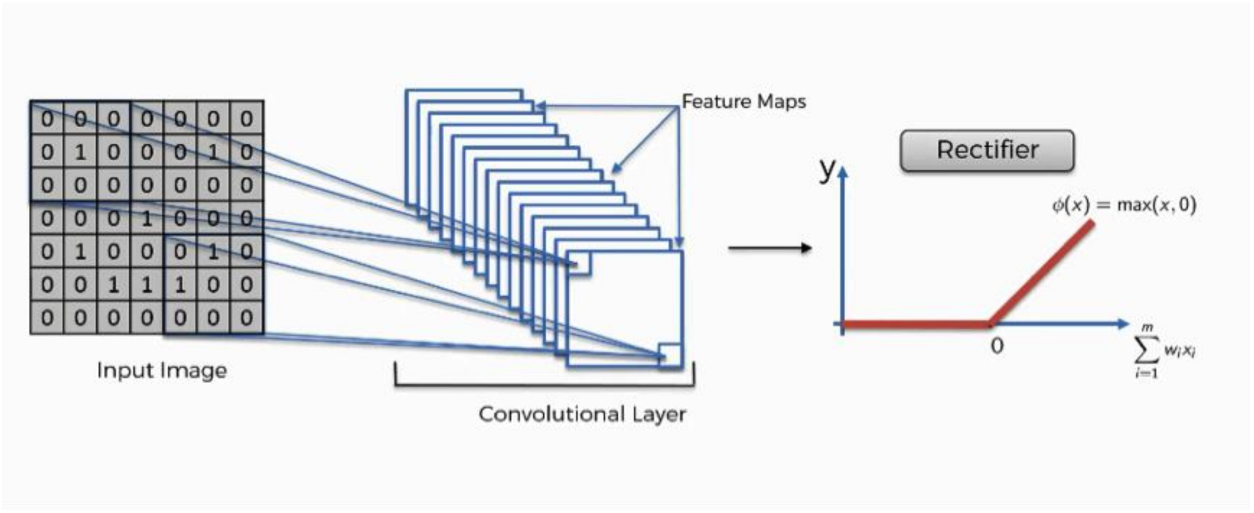
The following are some methods for pooling:

- ***Max-pooling***: It chooses the most significant element from the feature map. The feature map's significant features are stored in the resulting max-pooled layer. It is the most popular method since it produces the best outcomes.
- ***Average pooling***: It entails calculating the average for each region of the feature map.

Pooling gradually reduces the spatial dimension of the representation to reduce the number of parameters and computations in the network, as well as to prevent overfitting. If there is no pooling, the output has the same resolution as the input.

ReLU

The ***rectified linear activation function***, or ReLU for short, is a piecewise linear function that, if the input is positive, outputs the input directly; else, it outputs zero. Because a model that utilizes it is quicker to train and generally produces higher performance, it has become the default activation function for many types of neural networks.



Reference link: Basics of CNN in Deep Learning - Analytics Vidhya

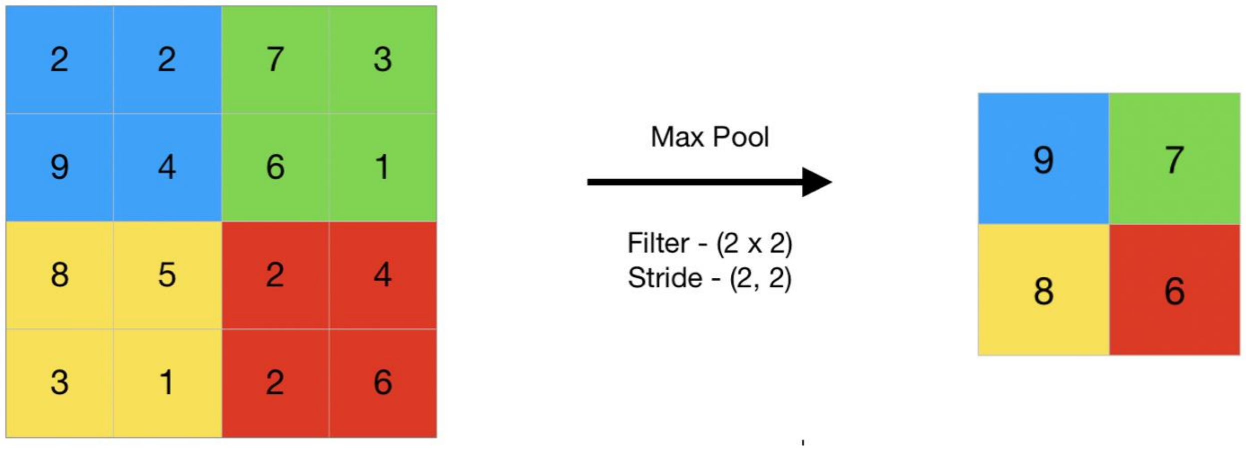
Pooling Layers: Max Pooling and Average Pooling

Why is Pooling Layers used on CNN?

-It reduces the dimensions of the feature maps. Thus, reducing the number of parameters to learn and the amount of computation performed in the network.
-It summarises the features present in a region of the feature map generated by a convolution layer.

Types of Pooling Layers

Max Pooling
Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after the max-pooling layer would be a feature map containing the most prominent features of the previous feature map.



Average Pooling

Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max-pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.

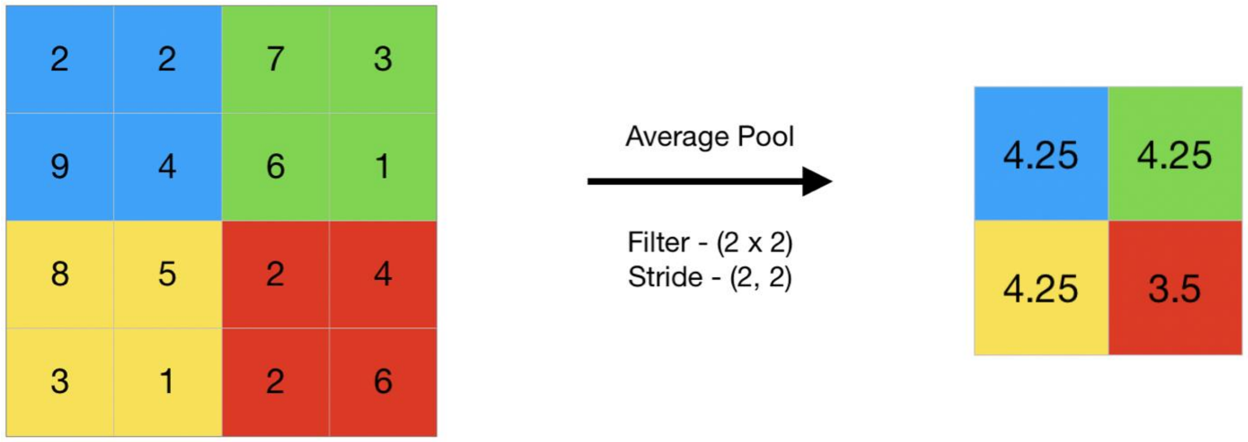
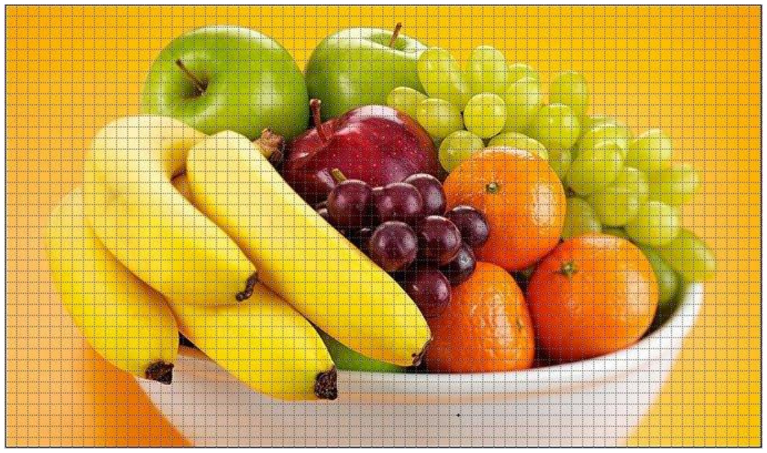


Image Classification



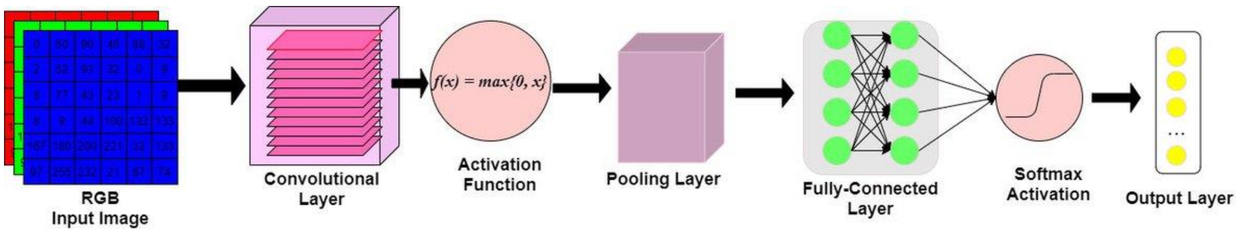
Digital images are composed of a grid of pixels. An image's pixels are valued between 0 and 255 to represent the intensity of light present. Therefore, we can think of the fruit bowl image above as a matrix of numerical values. This matrix has two axes, X and Y (i.e. the *width* and *height*.

Color images are constructed according to the *RGB* model and have a third dimension - *depth*. Color images are a 3-Dimensional matrix of red, green, and blue light-intensity values. These color channels are stacked along the Z-axis.

The dimensions of this fruit bowl image are 400 x 682 x 3. The width and height are 682 and 400 pixels, respectively. The depth is 3, as this is an RGB image.

Suppose we have a 32-pixel image with dimensions [32x32x3]. Flattening this matrix into a single input vector would result in an array of 32×32×3=3,072 nodes and associated weights.

This could be a manageable input for a regular network, but our fruit bowl image may not be so manageable! Flattening its dimensions would result in 682×400×3=818,400 values. Time and computation power simply do not favor this approach for image classification.



Evaluating performance of CNN

Evaluating the performance of a Convolutional Neural Network (CNN) involves several key metrics and techniques to ensure the model is accurate and reliable. Here are some common methods and metrics used:

1. Accuracy

- **Definition**: The ratio of correctly predicted instances to the total instances.
- **Formula**:

$\text{Accuracy}=\text{Total Number of Predictions}/\text{Number of Correct Predictions}$

2. Confusion Matrix

- **Definition**: A table used to describe the performance of a classification model.
- **Components**:
    - **True Positives (TP)**: Correctly predicted positive instances.
    - **True Negatives (TN)**: Correctly predicted negative instances.
    - **False Positives (FP)**: Incorrectly predicted positive instances.
    - **False Negatives (FN)**: Incorrectly predicted negative instances.

3. Precision, Recall, and F1-Score

- **Precision**: The ratio of correctly predicted positive observations to the total predicted positives.
    - **Formula**:

$\text{Precision}=TP/TP+FP$

- **Recall (Sensitivity)**: The ratio of correctly predicted positive observations to all observations in the actual class.
    - **Formula**:

$\text{Recall}=TP/TP+FN$

- **F1-Score**: The harmonic mean of precision and recall.
    - **Formula**:

$\text{F1-Score}=2\times\text{Precision}\times\text{Recall}/\ \text{Precision}+\text{Recall}$

4. ROC Curve and AUC

- **ROC Curve**: A graphical representation of the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.
- **AUC (Area Under the Curve)**: Measures the entire two-dimensional area underneath the ROC curve. A higher AUC indicates a better performing model.

5. Loss Function

- **Definition**: A function that measures how well the model's predictions match the actual data.
- **Common Loss Functions**: Cross-entropy loss for classification tasks.

6. Cross-Validation

- **Definition**: A technique for assessing how the results of a statistical analysis will generalize to an independent dataset.
- **Method**: Split the dataset into k subsets, train the model on k-1 subsets, and validate it on the remaining subset. Repeat this process k times.

7. Learning Curves

- **Definition**: Graphs that show the model's performance on the training and validation datasets over time.
- **Purpose**: To diagnose whether the model is overfitting or underfitting.