Table of Contents

Contents				
1	**Introduction**			
2	**Foundation of Convolutional Neural Network**			
3	**Concepts of Convolutional Neural Network**			
	3.1	Network Layers		
		3.1.1	Convolutional Layer	
			3.1.1.1	What is a kernel?
			3.1.1.2	What is Convolution Operation?
		3.1.2	Pooling Layer	
		3.1.3	Activation Functions (Non-Linearity)	
			3.1.3.1	Sigmoid
			3.1.3.2	Tanh
			3.1.3.3	ReLU
			3.1.3.4	Leaky ReLU
			3.1.3.5	Noisy ReLU
			3.1.3.6	Parametric Linear Units
		3.1.4	Fully Connected (FC) Layer	
	3.2	Loss Functions		
		3.2.1	Cross-Entropy or Soft-Max Loss Function	
4	**Training process of Convolutional Neural Network**			
	4.1	Data pre-processing and Data augmentation		
		4.1.1	Mean-subtraction (Zero centering)	
		4.1.2	Normalization	
		4.1.3	Data Augmentation	

	4.2	Parameter Initialization	
	4.3	Regularization to CNN	
		4.3.1	Dropout
		4.3.2	Drop-Weights
		4.3.3	The l^2 Regularization
		4.3.4	The l^1 Regularization
5	**Fresh Scratch Leaf Image Classification Using CNN**		
6	**Applications Areas of CNNs**		
	6.1	Image Classification	
	6.2	Text Recognition	
	6.3	Action Recognition	
	6.4	Image Caption Generation	
	6.5	Medical Image Analysis	
	6.6	Security and Surveillance	
	6.7	Automatic colorization of image and style transfer	
	6.8	Satellite Imagery	
7	**Conclusion**		

1 Introduction

Among different deep learning architecture, a special type of multilayer neural network for spatial data is Convolutional Neural Network (or CNN or ConvNet.). The architecture of CNN is inspired by the visual perception of living beings. Though it is become popular after the record-breaking performance of AlexNet in 2012 but it is actually initiated in 1980. After 2012, the CNN got the pace to take over different fields of computer vision, natural language processing and many more.

The foundation of convolutional neural network started from the discovery of Hubel and Wisel in 1959. According to them, cells of animal visual cortex recognize light in the small receptive field. In 1980, inspired by this work, Kunihiko Fukusima proposed neocognitron. This network is considered as the first theoretical model for CNN. In 1990, LeCun et al. developed the modern framework of CNN called LeNet-5 to recognize handwritten digits. Training by backpropagation algorithm helped LeNet-5 in recognizing visual patterns from raw images directly without using any separate feature engineering. But in those days' despite of several merits, the performance of CNN in complex problems was lacked by the limited training data, lack of innovation in algorithm and insufficient computing power. Recently we have large labeled datasets, innovative algorithms and powerful GPU machines. In 2012, with these up-gradation, a large deep CNN, called AlexNet, designed by Krizhevsky et al showed excellent performance on the ILSVRC. The success of AlexNet paved the way to invent different CNN models as well as to apply those models in different fields of computer vision and natural language processing.

A traditional convolutional neural network is made up of single or multiple blocks of convolution and pooling layers, followed by one or multiple fully connected (FC) layers and an output layer. The convolutional layer is the core building block of a CNN. This layer aims to learn feature representations of the input. The convolutional layer is composed of several learnable convolution kernels or filters which are used to compute different feature maps. Each unit of feature map is connected to a receptive field in the previous layer. The new feature map is produced by convolving the input with the kernels and applying elementwise non-linear activation function on the convolved result. The parameter sharing property of convolutional layer reduces the model complexity. Pooling or sub-sampling layer takes a small region of the convolutional output as input and down samples it to produce a single output. There are different sub-sampling techniques as example max pooling, min pooling, average pooling, etc. Pooling reduces the number of parameters to be computed as well as it makes the network translation invariant. Last part of CNN is basically made up of one or more FC layers typically found in feedforward neural network. The FC layer takes input from the final pooling or convolutional layer and generates final output of CNN. In case of image classification, a CNN can be viewed as a combination of two parts: feature extraction part and classification part. Both convolution and pooling layers perform feature extraction. As an example of dog's image, different convolution layers from lower level to higher level detect various features such as two eyes, long ears, four legs, etc. for further recognition. On top of this feature, the FC layers are added as classifier, and a probability is assigned for the input image being a dog. Beside the layer design, the improvement of CNN depends on several different aspects such as activation function, normalization method, loss function, regularization, optimization and processing speed, etc. After the success of AlexNet, CNN got huge popularity in three major fields namely image classification, object detection and segmentation, and many advance models of CNN has been proposed in those areas in successive years. major application areas that apply CNN to achieve state-of-the-art performance includes image classification, object tracking, object detection, segmentation, human pose estimation, text detection, visual saliency detection,

action recognition, scene labelling, visual question answering, speech and natural language processing, etc. Though current CNN models work very well for various applications, it is yet to know why and how it works essentially. So, more efforts on investigating the fundamental principles of CNNs are required.
The chapter will provide a better understanding of CNN as well as facilitates for future research activities and application developments in the field of CNN.

2 Foundation of Convolutional Neural Network

In 1959, two neurophysiologists David Hubel and Torsten Wiesel experimented and later published their paper, entitled "**Receptive fields of single neurons in cat's striate cortex**", described that the neurons inside the brain of a cat are organized in layered form. These layers learn how to recognize visual patterns by first extracting the local features and then combining the extracted features for higher level representation. Later on, this concept is essentially become one of the core principles of Deep Learning.
Inspired by the work of Hubel and Wiesel, in 1980, Kunihiko Fukushima proposed **Neocognitron**, which is a self-organizing Neural Network, containing multiple layers, capable of recognizing visual patterns hierarchically through learning and this architecture became the first theoretical model of CNN as in the figure 1. A major improvement over the architecture of **Neocognitron** was done by LeCun et. in 1989 by developing a modern framework of CNN, called LeNet-5, which successfully recognized the MNIST handwritten digits dataset. LeNet-5 was trained using error back-propagation algorithm and it can be recognizing visual patterns directly from raw input images, without using any separated feature engineering mechanism. After discovering LeNet-5, because of several limitation like lack of large training data, lack of innovation in algorithm and inadequate computing power, CNN did not perform well in various complex problems. But nowadays, in the era of Big Data we have large labeled datasets, more innovative algorithms and especially powerful GPU machines. With this type of upgradation, in 2012, Krizhevsky et al. designed AexNet, which achieved a fantastic accuracy on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). The victory of AlexNet paved the way to invent several CNN models as well as to apply those models in different field of computer vision and natural language processing.

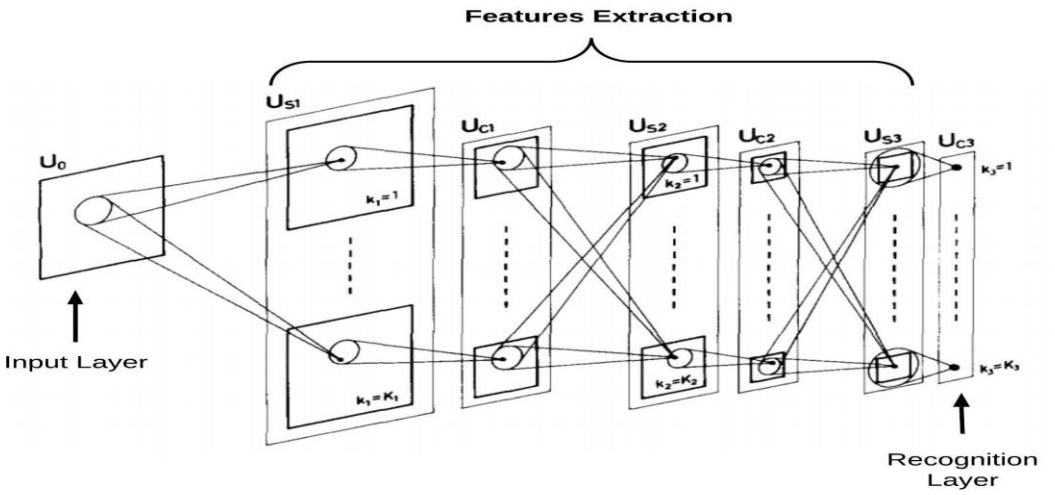


Figure 1: Schematic diagram illustrating the interconnections between layers in the neocognitron, Kunihiko Fukushima.

3 Concepts of Convolutional Neural Network

Convolutional Neural Network (CNN), also called ConvNet, is a type of Artificial Neural Network (ANN), which has deep feed-forward architecture and has amazing generalizing ability as compared to other networks with FC layers, it can learn highly abstracted features of objects especially spatial data and can identify them more efficiently.
A deep CNN model consists of a finite set of processing layers that can learn various features of input data (e.g., image) with multiple level of abstraction. The initiatory layers learn and extract the high-level features (with lower abstraction), and the deeper layers learns and extracts the low-level features (with higher abstraction). The basic conceptual model of CNN was shown in figure 2, different types of layers described in subsequent sections.

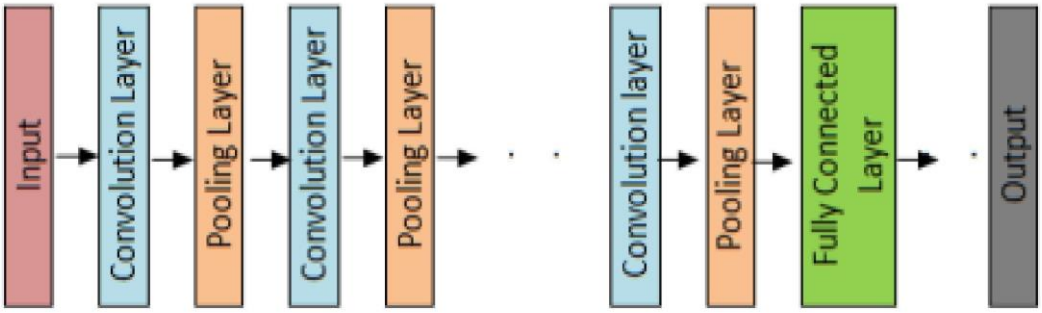


Figure 2: Conceptual model of CNN

Why Convolutional Neural Networks is more considerable over other classical neural networks in the context of computer vision?

- One of the main reasons for considering CNN in such case is the weight sharing feature of CNN, that reduce the number of trainable parameters in the network, which helped the model to avoid overfitting and as well as to improved generalization.
- In CNN, the classification layer and the feature extraction layers learn together, that makes the output of the model more organized and makes the output more dependent to the extracted features.
- The implementation of a large network is more difficult by using other types of neural networks rather than using Convolutional Neural Networks.

Nowadays CNN has been emerged as a mechanism for achieving promising result in various computer vision-based applications like image classification, object detection, face detection, speech recognition, vehicle recognition, facial expression recognition, text recognition and many more.

Now description of different components or basic building blocks of CNN briefly as follows.

3.1 Network Layers

As we mentioned earlier, that a CNN is composed of multiple building blocks (known as layers of the architecture), in this subsection, we described some of these building blocks in detail with their role in the CNN architecture.

3.1.1 Convolutional Layer

Convolutional layer[1] is the most important component of any CNN architecture. It contains a set of convolutional kernels (also called filters), which gets convolved with the input image (N-dimensional metrics) to generate an output feature map.

3.1.1.1 What is a kernel?

A kernel can be described as a grid of discrete values or numbers, where each value is known as the weight of this kernel. During the starting of training process of an CNN model, all the weights of a kernel are assigned with random numbers (different approaches are also available there for initializing the weights). Then, with each training epoch, the weights are tuned and the kernel learned to extract meaningful features. In Fig.3, we have shown a 2D filter.

0	1
-1	2

Figure 3: Example of a 2 × 2 kernel

3.1.1.2 What is Convolution Operation?

Before we go any deeper, let us first understand the input format to CNN. Unlike other classical neural networks (where the input is in a vector format), in CNN the input is a multi-channeled image (e.g., for RGB image as in figure 4, it is 3 channeled and for Gray-Scale image, it is single channeled).
Now, to understand the convolution operation, if we take a grayscale image of 4 × 4 dimension, shown in Fig.5 and a 2 × 2 kernel with randomly initialized weights as shown in Fig.6.

1	0	-2	1
-1	0	1	2
0	2	1	0
1	0	0	1

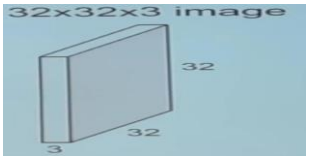


Figure 4: Example of a RGB image

0	1
-1	2

[1] **Notable thing**: CNN will uses a set of multiple filters in each convolutional layers so that each filter can extract the different types of features.

Figure 5: A 4 × 4 Gray-Scale image Figure 6: A kernel of size 2 × 2

Now, in convolution operation, we take the 2 × 2 kernel and slide it over all the complete 4×4 image horizontally as well as vertically and along the way we take the dot product between kernel and input image by multiplying the corresponding values of them and sum up all values to generate one scaler value in the output feature map. This process continues until the kernel can no longer slide further.

To understand the thing more clearly, let's do some initial computations performed at each step graphically as shown in Fig. 7, where the 2 × 2 kernel (shown in light blue color) is multiplied with the same sized region (shown in yellow color) within the 4×4 input image and the resulting values are summed up to obtain a corresponding entry (shown in deep blue) in the output feature map at each convolution step.

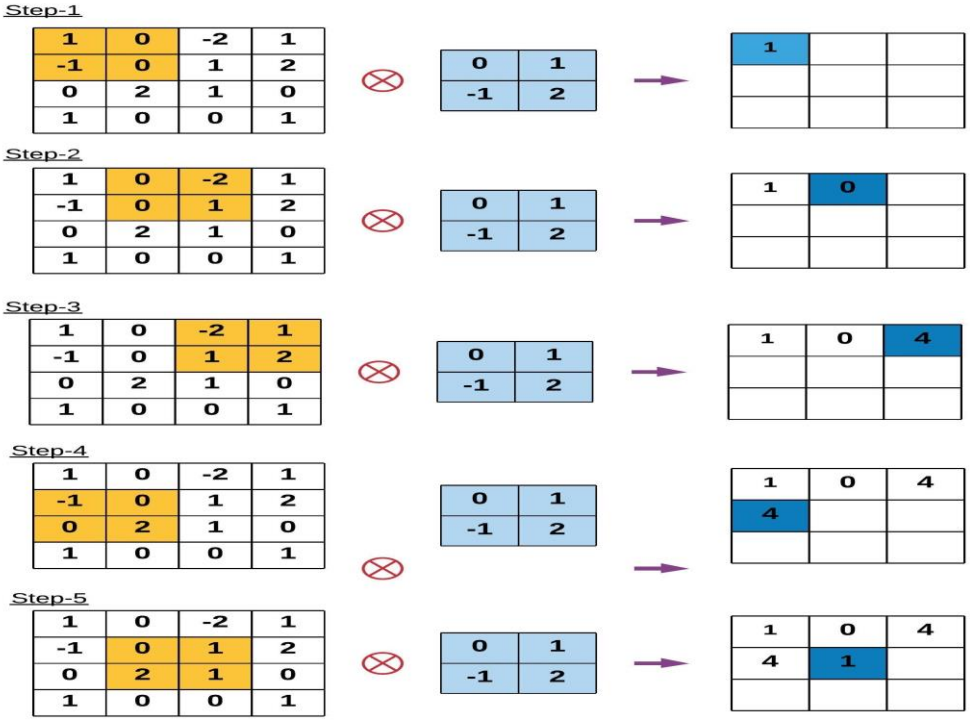


Figure 7: Illustrating the first 5 steps of convolution operation

After performing the complete convolution operation, the final output feature map is shown in Fig.8 as follows.

1	0	4
4	1	1
1	1	2

Figure 8: The final feature map after the complete convolution operation

In the above example, we apply the convolution operation with no **padding** to the input image and with **stride** (i.e., then taken step size along the horizontal or vertical position) of 1 to the kernel. But we can use other stride value (rather than 1) in convolution operation. The noticeable thing is if we increase the stride of the convolution operation, it resulted in lower-dimensional feature map.

The **padding** is important to give border size information of the input image more importance, otherwise without using any padding the border side features are gets **washed away** too quickly. The padding is also used to increase the input image size, as a result the output feature map size also gets increased. The Fig.9 gives an example by showing the convolution operation with **Zero-padding** and **3 stride value**.

The formula to find the output feature map size after convolution operation as below:

$$h' = \left\lfloor \frac{h - f + p}{s} + 1 \right\rfloor$$

$$w' = \left\lfloor \frac{w - f + p}{s} + 1 \right\rfloor$$

0 0

Where *h* denotes the height of the output feature map, *w* denotes the width of the output feature map, *h* denotes the height of the input image, *w* denotes the width of the input image, *f* is the filter size, *p* denotes the padding of convolution operation and *s* denotes the stride of convolution operation.

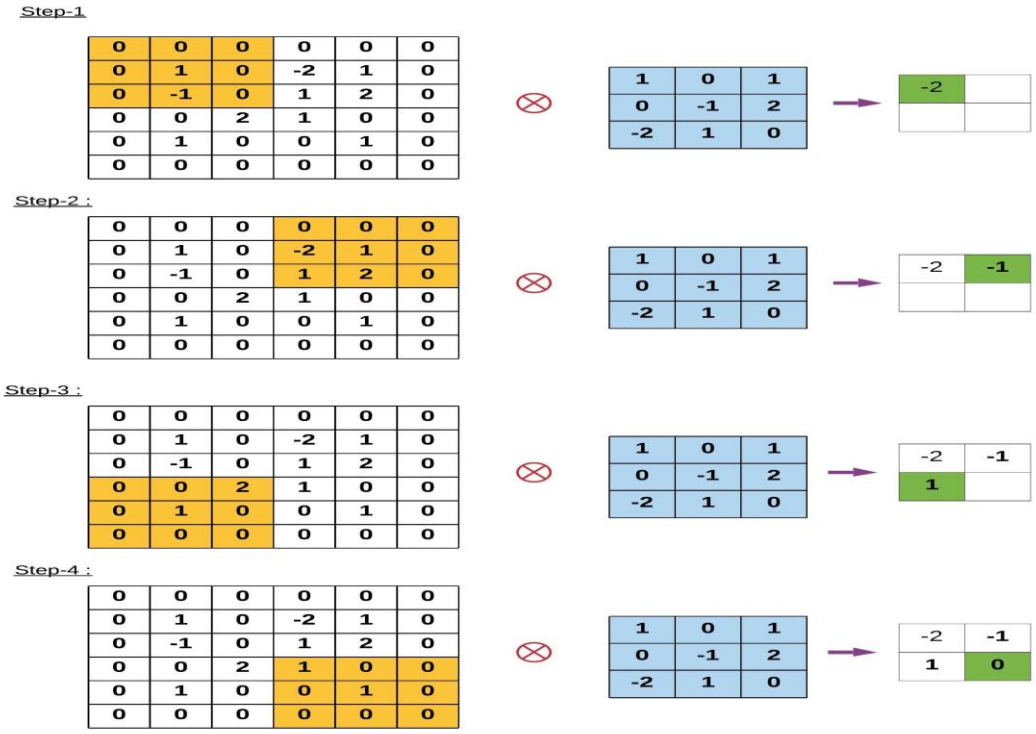


Figure 9: The computations performed at each step, where the 3 × 3 kernel (shown in light blue color) is multiplied with the same sized region (shown in yellow color) within the 6 × 6 input image (where we applied zero-padding to the original input image of 4 × 4 dimension and it becomes of 6 × 6 dimensional) and values are summed up to obtain a corresponding entry (shown in deep green) in the output feature map at each convolution step.

Main advantages of convolution layers are:

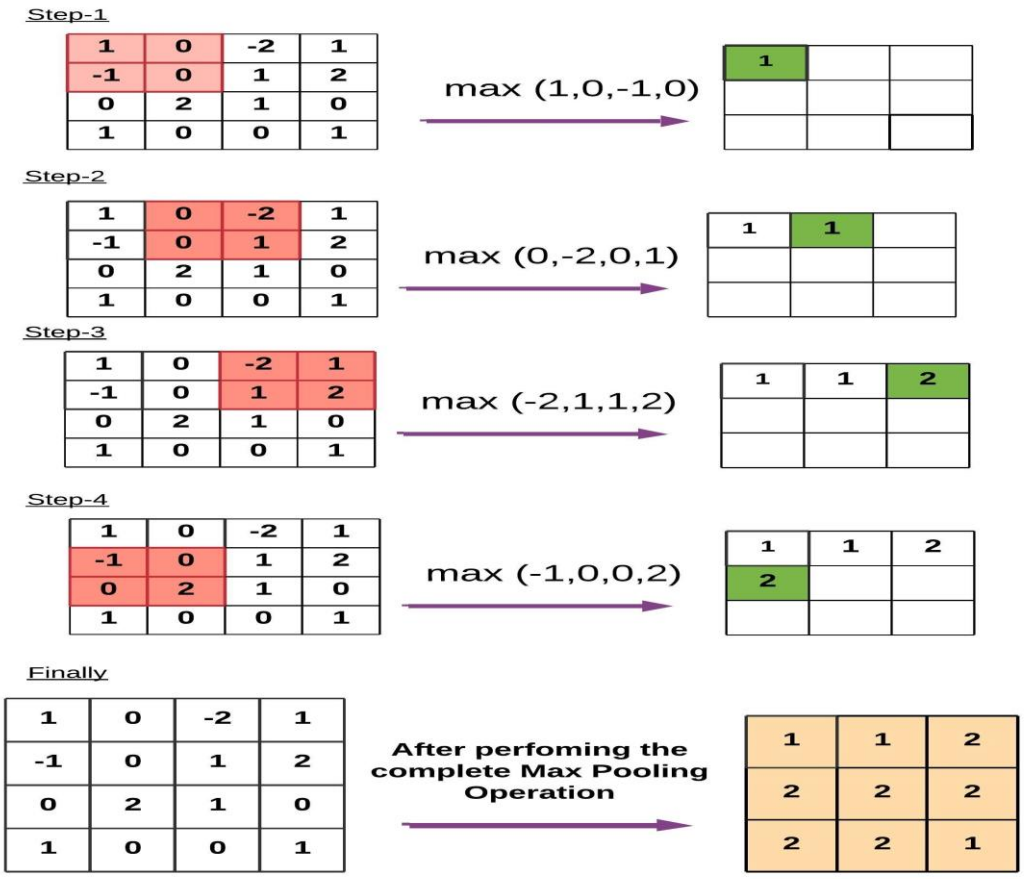
- **Sparse Connectivity:** In a fully connected neural network each neuron of one layer connects with each neuron of the next layer but in CNN small number of weights are present between two layers. As a result, the number of connection or weights we need is

small, and the amount of memory to store those weights is also small, so it is memory efficient. Also, the dot(.) operation is computationally cheaper than matrix multiplication.

- **Weight Sharing:** In CNN, no dedicated weights are present between two neurons of adjacent layers instead of all weights works with each and every pixel of the input matrix. Instead of learning new weights for every neuron we can learn one set of weights for all inputs and this drastically reduces the training time as well as the other costs.

3.1.2 Pooling Layer

The pooling[2] layers are used to sub-sample the feature maps (produced after convolution operations), i.e., it takes the larger size feature maps and shrinks them to lower sized feature maps. While shrinking the feature maps it always preserves the most dominant features (or information) in each pool steps. The pooling operation is performed by specifying the pooled region size and the stride of the operation, similar to convolution operation. There are different types of pooling techniques are used in different pooling layers such as max pooling, min pooling, average pooling, gated pooling, tree pooling, etc. Max Pooling is the most popular and mostly used pooling technique.
The main **drawback** of pooling layer is that it sometimes decreases the overall performance of CNN. The reason behind this is that pooling layer helps CNN to find whether a specific feature is present in the given input image or not without caring about the correct position of that feature.



[2] *"The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster." —Geoffrey Hinton*

Figure 10: Illustrating an example that shows some initial steps as well as the final output of max-pooling operation, where the size of the pooling region is 2 × 2 (shown in orange color, in the input feature map) and the stride is 1 and the corresponding computed value in the output feature map (shown in green)

The formula to find the output feature map size after pooling operation as below:

$$h' = \left\lfloor \frac{h - f}{s} \right\rfloor$$

$$w' = \left\lfloor \frac{w - f}{s} \right\rfloor$$

Where h' denotes the height of the output feature map, w' denotes the width of the output feature map, h denotes the height of the input feature map, w denotes the width of the input feature map, f is the pooling region size and s denotes the stride of the pooling operation.

3.1.3 Activation Functions (Non-Linearity)

The main task of any activation function in any neural network-based model is to map the input to the output, where the input value is obtained by calculating the weighted sum of neuron's input and further adding bias with it (if there is a bias). In other words, the activation function decides whether a neuron will fire or not for a given input by producing the corresponding output.

In CNN architecture, after each learnable layers (layers with weights, i.e., convolutional and FC layers) non-linear activation layers are used. The non-linearity behavior of those layers enables the CNN model to learn more complex things and manage to map the inputs to outputs nonlinearly. The important feature of an activation function is that it should be differentiable in order to enable error backpropagation to train the model. The most commonly used activation functions in deep neural networks (including CNN) are described below.

3.1.3.1 Sigmoid:

The sigmoid activation function takes real numbers as its input and bind the output in the range of [0,1]. The curve of the sigmoid function is of '*S*' shaped. The mathematical representation of sigmoid is:

$$f(x)_{sigm} = \frac{1}{1 + e^{-x}}$$

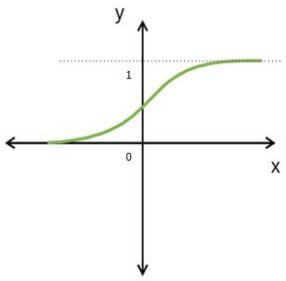


Figure 11: Sigmoid

3.1.3.2 Tanh:

the *Tanh* activation function is used to bind the input values (real numbers) within the range of [-1, 1]. The mathematical representation of *Tanh* is:

$$f(x)_{tanh} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

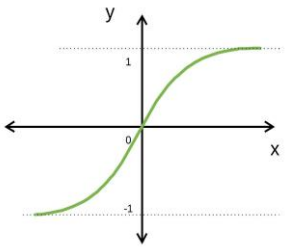


Figure 12: *Tanh*

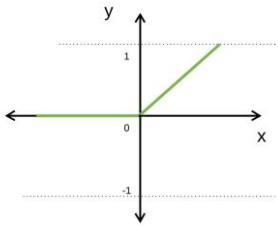


Figure 13: ReLU

3.1.3.3 ReLU:

The Rectifier Linear Unit (ReLU) is the most commonly used activation function in Convolutional Neural Networks. It is used to convert all the input values to positive numbers. The advantage of ReLU is that it requires very minimal computation load compared to others. The mathematical representation of ReLU is:

$$f(x)_{ReLU} = \mathbf{max}\ (0, x)$$

But sometimes there may occur some major problems in using ReLU activation function. For example, consider a larger gradient is flowing during error back-propagation algorithm, and when this larger gradient is passed through a ReLU function it may cause the weights to be updated in such a way that the neuron never gets activated again. This problem is known as the Dying ReLU problem. To solve these types of problems there are some variants of ReLU is available, some of them are discussed below.

3.1.3.4 Leaky ReLU

Unlike ReLU, a Leaky ReLU activation function does not ignore the negative inputs completely, rather than it down-scaled those negative inputs. Leaky ReLU is used to solve Dying ReLU problem. The mathematical representation of Leaky ReLU is:

$$f(x)_{LeakyReLU} = \begin{cases} x, & \text{if } x > 0 \\ mx, & x \leq 0 \end{cases}$$

where *m* is a **constant**, called leak factor and generally it set to a small value (like 0.001).

3.1.3.5 Noisy ReLU:

Noisy ReLU is used Gaussian distribution to make ReLU noisy. The mathematical representation of Noisy ReLU is:

$$f(x)_{NoisyReLU} = \mathbf{max}\ (x + Y), \mathbf{with}\ Y \sim N\ (0, \sigma(x))$$

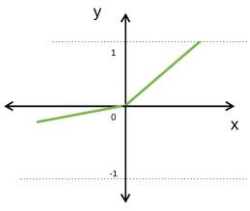


Figure 14: Leaky ReLU

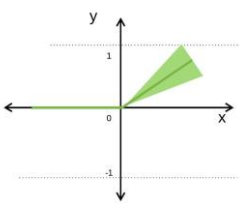


Figure 15: Noisy ReLU

3.1.3.6 Parametric Linear Units

It is almost similar to Leaky ReLU, but here the leak factor is tuned during the model training process. The mathematical representation of Parametric Linear Units is:

$$f(x)_{ParametricLinearUnits} = \begin{cases} x, & \text{if } x > 0 \\ ax, & x \leq 0 \end{cases}$$

where *a* is a learnable weight.

3.1.4 Fully Connected (FC) Layer

Usually the last part (or layers) of every CNN architecture (used for classification) is consist of fully-connected layers, where each neuron inside a layer is connected with each neuron from its previous layer. The last layer of Fully-Connected layers is used as the output layer (classifier) of the CNN architecture.

The Fully-Connected Layers are type of feed-forward artificial neural network (ANN) and it follows the principle of traditional multi-layer perceptron neural network (MLP). The FC layers take input from the final convolutional or pooling layer, which is in the form of a set of metrics (feature maps) and those metrics are flattened to create a vector and this vector is then fed into the FC layer to generate the final output of CNN as shown in Fig.16.

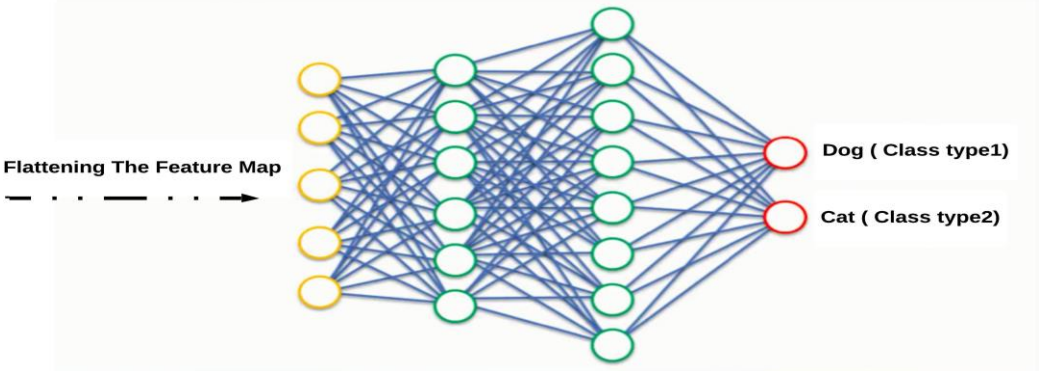


Figure 16: The architecture of Fully Connected Layers

3.2 Loss Functions

In section 3.1, has described different types of layers used in CNN architecture. Now, we know that the last layer of every CNN architecture (classification based) is the output layer, where the final classification takes place. In this output layer, we calculate the prediction error generated by the CNN model over the training samples using some **Loss Function**. This prediction error tells the network how off their prediction from the actual output, and then this error will be optimized during the learning process of the CNN model.

The loss function uses two parameters to calculate the error, the first parameter is the estimate output of the CNN model (also called the prediction) and the second one is the actual output (also known as the label). There are different types of loss functions used in different types of problem. Some of the most used loss functions are briefly described in next subsections.

3.2.1 Cross-Entropy or Soft-Max Loss Function

Cross-entropy loss, also called log loss function is widely used to measure the performance of CNN model, whose output is the probability $p \in \{0,1\}$. It is widely used as an alternative of

squared error loss function in the multi-class classification problems. It uses SoftMax activations in the output layer to generate the output within a probability distribution, i.e., *p, y* $\in R^N$, where *p* is the probability for each output category and *y* denotes the desired output and the probability of each output class can be obtained by:

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}$$

where *N* is the number of neurons in the output layer and e^{a_i} denotes each unnormalized output from the previous layer in the network. Now finally, cross-entropy loss can be defined as:

$$H(p, y) = -\sum_i y_i \log (p_i)$$

$$H(p, y) = \frac{1}{2N} \sum_{i=1}^{N} (p_i - y_i)^2$$

Training process of Convolutional Neural Network

In the previous section 3, we have described the basic concepts of convolutional neural network (CNN) as well as the different key components of CNN architecture. Here in this section, we try to discuss the training or learning process of a CNN model with certain guidelines in order to reduce the required training time and to improve model accuracy. The training process mainly includes the following steps:

- Data pre-processing and Data augmentation.
- Parameter initialization.
- Regularization of CNN.
- Optimizer selection.

Those steps have described in the next subsections.

4 Data pre-processing and Data augmentation

4.1 Data pre-processing refers to some artificial transformations to the raw dataset (including training, validation and testing datasets) in order to make the dataset cleaner, more featureful, more learnable and in a uniform format. The data pre-processing is done before feeding the data to the CNN model. In a convolutional neural, network it is a fact that the performance of CNN is directly proportional to the amount of data used to train it, i.e. good pre-processing, always increases accuracy of the model. But on the other side, a bad pre-processing can also reduce the performance of the model.
The general pre-processing techniques that are mostly used are given in the following subsections.

4.1.1 Mean-subtraction (Zero centering):

Here we sub struct the mean from every individual data point (or feature) to make it zero centered as shown in Fig.17. The operation can be implemented mathematically as:

$$X_0 = X - x^*$$

And,
$$x^* = \frac{1}{N}\sum_{i=1}^{N} x_i$$

where *N* denotes the size of the training dataset.

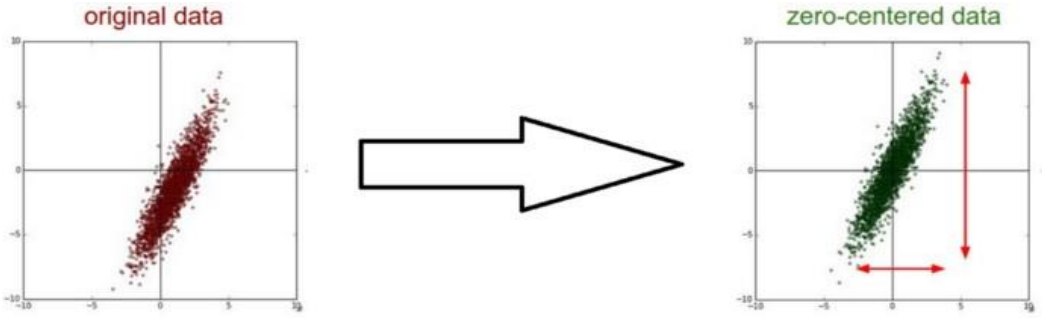


Figure 17: Mean-subtraction (Zero centering the data) (source: http://cs231n.stanford.edu/)

4.1.2 Normalization:

Here we normalize the data sample's (belonging from both train, validation and test dataset) dimension by dividing each dimension by its standard deviations shown in Fig.18. The operation would be implemented mathematically as:

$$X'' = \frac{X'}{\sqrt{\frac{\sum_{i=1}^{N}(x_i - x^*)^2}{N-1}}}$$

where *N*, X_0 and X^* are the same as discussed in section 4.1.1

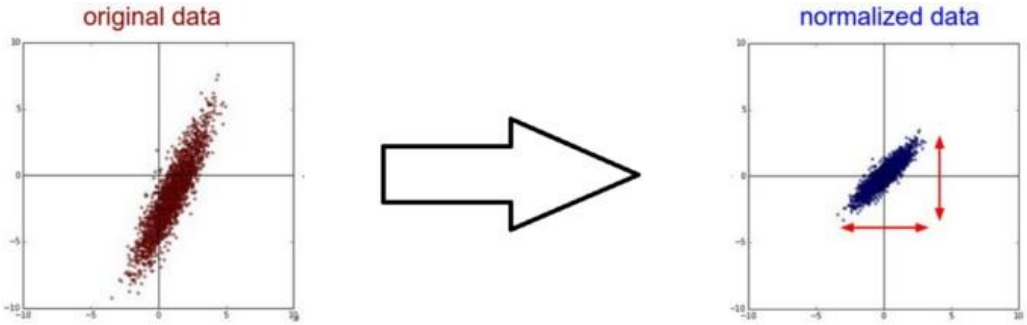


Figure 18: Normalization of data (source: http://cs231n.stanford.edu/)

4.1.3 Data Augmentation is a technique used to artificially increase or expand the size of the training dataset. Here we apply different operations to the data samples (belonging to training dataset only) and artificially transform it to one or many new data samples (new version), which is then used in training process. Data Augmentation is important because, sometimes there is a very limited sized training data set is available for most of the real-life complex problems (e.g., medical datasets) and the true fact is that the more training data samples can resulted in a more skillful CNN model.

There are several data augmentation operations are available such as cropping, rotations, flipping, translations, contrast adjustment, scaling, etc. We can apply those operations separately or in combination to make several new versions from a single data sample. Another reason to use it is that the data augmentation is also able to enforce regularization in the CNN model by avoiding over-fitting problem as shown in figures Fig.19, Fig.20, Fig.21, and Fig.22.

Figure 19: An example of a raw training data sample

Figure 20: Three new data samples, which are created by applying random cropping augmentation technique.

Figure 21: Three new data samples, which are created by applying random flipping augmentation technique.

Figure 22: Three new data samples, which are created by applying random cropping and flipping augmentation technique in combination.

4.2 Parameter Initialization

A deep CNN consists of millions or billions number of parameters. So, it must be well initialized at the begin of the training process, because weight initialization directly determines how fast the CNN model would converge and how accurately it might end up. Here in this section, we discuss some mostly used parameter initialization techniques used in CNN as follows:

The easiest way to doing it is by initializing all the weights with zero. However, this turns out to be a mistake, because if we initialize weights of all layer to zero, the output as well as the gradients (during backpropagation) calculated by every neuron in the network will be the same. Hence the update to all the weights would also be the same. As a result, there is no disparity between neurons and the network will not learn any useful features. To break this disparity between neurons, we do not initialize all weights with the same value, rather than, we use different techniques to initialize the weights randomly as follows:

4.3 Regularization to CNN

The core challenge of deep learning algorithms is to adapt properly to new or previously unseen input, drawn from the same distribution as training data, the ability to do so is called generalization. The main problem for a CNN model to achieve good generalization is **over-fitting**.

When a model performs exceptionally well on training data but it fails on test data (unseen data), then this type of model is called over-fitted. The opposite is an under-fitted model, that happen when the model has not learned enough from the training data and when the model performs well on both train and test data, then these types of models are called just-fitted model. Fig.24 try to show the examples of over-fitted, under-fitted and just-fitted models.

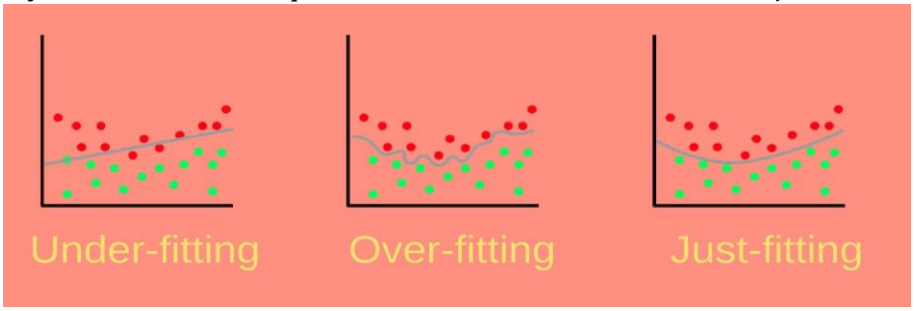
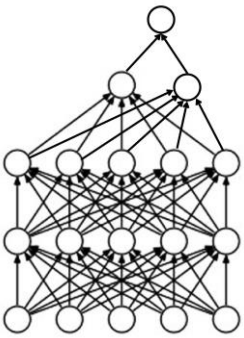


Figure 23: The Examples of over-fitting, under-fitting and just-fitting model's hypothesis with respect to binary classification

Regularization helps to avoid over-fitting by using several intuitive ideas, some of which are discussed in the next subsections.

4.3.1 Dropout

Dropout is one of the most used approaches for regularization. Here we randomly drop neurons from the network during each training epoch. By dropping the units (neurons) we try to distribute the feature selection power to all the neurons equally and we forced the model to learn several independent features. Droping a unit or neuron means; the dropped unit would not take part in both forward propagation or backward propagation during the training process. But in the case of testing process, the full-scale network is used to perform prediction. With the help of FIg.25 and Fig.26, we try to show the effect of dropout in network's architecture during training.



24: A normal Neural network

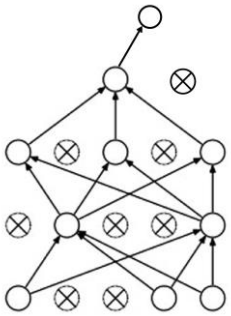


Figure 25: After applying dropout Figure

4.3.2 Drop-Weights

It is very much similar to dropout (discuss in section 4.3.1). The only difference is instead of dropping the neurons, here we randomly drop the weights (or connections between neurons) in each training epoch.

4.3.3 The $\imath^2$ Regularization

The $\imath^2$ regularization or "weight decay" is one of the most common forms of regularization. It forces the network's weights to decay towards zero (but not equal to zero) by adding a penalty term equal to the "squared magnitude" of the coefficient to the loss function. It regularizes the weights by heavily penalize the larger weight vectors. This is done by adding $\frac{1}{2}\lambda\|w\|^2$ to the objective function, where λ is a hyper-parameter, which decides the strength of penalization and kwk denotes the matrix norm of network weights.

Consider a network with only a single hidden layer and with parameters w. If there are N neurons in the output layer and the prediction output and the actual output are denoted by y_n and p_n where $n \in [0, N]$. Then the objective function:

$$CostFunction = loss + \frac{1}{2}\lambda\|w\|^2$$

In the case of Euclidean objective function:

$$CostFunction = \sum_{m=1}^{M}\sum_{n=1}^{N}(p_n - y_n)^2 + \frac{1}{2}\lambda\|w\|^2$$

Where M is number of training examples. Now the weight incrementation rule with the $\imath^2$ regularization will be as:

$$WeightIncrement = \underset{w}{\operatorname{argmin}} \sum_{m=1}^{M}\sum_{n=1}^{N}(p_n - y_n)^2 + \lambda k w k$$

4.3.4 The $\imath^1$ Regularization

The $\imath^1$ regularization is almost similar to the $\imath^2$ regularization and also widely used in practice, but the only difference is, instead of using "squared magnitude" of coefficient as a penalty, here

we used the absolute value of the magnitude of coefficients as a penalty to the loss function. So, the objective function with ι^1 regularization as:

$$CostFunction = loss + \lambda \| w \|$$

where λ is a hyper-parameter, which decides the strength of penalization and $\| w \|$ denotes the matrix norm of network weights.

5 Fresh Scratch Leaf Image Classification Using CNN

Source Code and Output:

```
import numpy as np
import cv2
import os
import random
import matplotlib.pyplot as plt
```

```
from google.colab import drive
drive.mount('/content/drive')

DIRECTORY = r"/content/drive/MyDrive/leaf/leaf"
CATAGORIES = ['Strawberry_fresh','Strawberry_scrotch']
```

```
data = []

for categories in CATAGORIES:
    folder=os.path.join(DIRECTORY,categories)
    label=CATAGORIES.index(categories)


    for img in os.listdir(folder):
        img=os.path.join(folder,img)
        img_arr=cv2.imread(img)
        img_arr=cv2.resize(img_arr,(100,100))
        data.append([img_arr,label])
```

```
random.shuffle(data)
x=[]
y=[]


for features,label in data:
    x.append(features)
    y.append(label)

X= np.array(x)
Y=np.array(y)

X = X/255

X.shape
```

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import
Conv2D,MaxPooling2D,Flatten,Dense,Activation
```

```
#Model

model=Sequential()
model.add( Conv2D(64,(3,3),input_shape=X.shape[1:],activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add( Conv2D(32,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add( Conv2D(32,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))


model.add(Flatten())

model.add(Dense(2,activation='softmax'))
```

```
#compile
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metri
cs=['accuracy'])

#Train
model.fit(X,Y,epochs=15,validation_split=0.1)
```

```
Epoch 1/15
3/3 [==============================] - 12s 307ms/step - loss: 0.7017 - accuracy: 0.5667 - val_loss:
0.6708 - val_accuracy: 0.8000
Epoch 2/15
3/3 [==============================] - 0s 22ms/step - loss: 0.6651 - accuracy: 0.6444 - val_loss: 0.6341
- val_accuracy: 1.0000
Epoch 3/15
3/3 [==============================] - 0s 31ms/step - loss: 0.6136 - accuracy: 0.8889 - val_loss: 0.5496
- val_accuracy: 1.0000
Epoch 4/15
3/3 [==============================] - 0s 30ms/step - loss: 0.5212 - accuracy: 0.8778 - val_loss: 0.4060
- val_accuracy: 1.0000
Epoch 5/15
3/3 [==============================] - 0s 28ms/step - loss: 0.4205 - accuracy: 0.8556 - val_loss: 0.4179
- val_accuracy: 0.8000
Epoch 6/15
3/3 [==============================] - 0s 30ms/step - loss: 0.3449 - accuracy: 0.8556 - val_loss: 0.2230
- val_accuracy: 1.0000
Epoch 7/15
3/3 [==============================] - 0s 28ms/step - loss: 0.2564 - accuracy: 0.8889 - val_loss: 0.1732
- val_accuracy: 1.0000
Epoch 8/15
3/3 [==============================] - 0s 27ms/step - loss: 0.1871 - accuracy: 0.9444 - val_loss: 0.1394
- val_accuracy: 1.0000
Epoch 9/15
3/3 [==============================] - 0s 27ms/step - loss: 0.1429 - accuracy: 0.9667 - val_loss: 0.1014
- val_accuracy: 1.0000
Epoch 10/15
3/3 [==============================] - 0s 34ms/step - loss: 0.1583 - accuracy: 0.9333 - val_loss: 0.2589
- val_accuracy: 0.8000
Epoch 11/15
```

```
3/3 [==============================] - 0s 27ms/step - loss: 0.1878 - accuracy: 0.9000 - val_loss: 0.7413
- val_accuracy: 0.8000
Epoch 12/15
3/3 [==============================] - 0s 26ms/step - loss: 0.2696 - accuracy: 0.9000 - val_loss: 0.3676
- val_accuracy: 0.7000
Epoch 13/15
3/3 [==============================] - 0s 27ms/step - loss: 0.2855 - accuracy: 0.8889 - val_loss: 0.8432
- val_accuracy: 0.8000
Epoch 14/15
3/3 [==============================] - 0s 31ms/step - loss: 0.3864 - accuracy: 0.8667 - val_loss: 0.0735
- val_accuracy: 1.0000
Epoch 15/15
3/3 [==============================] - 0s 26ms/step - loss: 0.1999 - accuracy: 0.9111 - val_loss: 0.0585
- val_accuracy: 1.0000
<tensorflow.python.keras.callbacks.History at 0x7f3a48308bb0>
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 98, 98, 64)        1792
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 49, 49, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 47, 47, 32)        18464
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 21, 21, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 10, 10, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 3200)              0
_________________________________________________________________
dense (Dense)                (None, 2)                 6402
=================================================================
Total params: 35,906
Trainable params: 35,906
Non-trainable params: 0
```

```
import keras.utils as image
import numpy as np

img_pred=tf.keras.utils.load_img(r"/content/drive/MyDrive/leaf/leaf/Strawbe
rry_fresh/00e9a277-ca5e-4350-95ce-8b2918b69fb9___RS_HL
4667.JPG",target_size=(100,100))

img_pred=tf.keras.utils.img_to_array(img_pred)
img_pred=np.expand_dims(img_pred, axis=0)

#Predict
rslt= model.predict(img_pred)
```

```
print(rslt)
if rslt[0][0]>rslt[0][1]:
    prediction="Strawberry_fresh"


else:
    prediction="Strawberry_scrotch"
print(prediction)
```

Output:

```
[[1. 0.]]
Strawberry_fresh
```

6 Applications Areas of CNNs

In this section, we discuss some of the major application areas that apply CNN to achieve state of-the-art performance including image classification, text recognition, action recognition, object detection, human pose estimation, image captioning, etc.

6.1 Image Classification

Because of several capabilities like weight sharing, different level of feature extraction like classifiers, etc., the CNN have been achieving better classification accuracy compared to other methods especially in the case of large-scale datasets. The first breakthrough in image classification is comes with the development of AlexNet in 2012, which won the ILSVRC challenge in that same year. After that, several improvements in CNN model have made by the researchers over the times, and that makes CNN as the first choice for image classification problem.

6.2 Text Recognition

The text detection and text recognition inside an image has been widely studied for a long time. The first breakthrough contribution of CNN in this field begins with LeNet-5, which recognized the data in MNIST dataset with a good accuracy. After that in recent years, with several improvements, CNN contributes a vital role to recognize the text (digits, alphabet and symbols belonging from several languages) inside the image.

6.3 Action Recognition

Based on the visual appearance and motion dynamics of any human body, various effective CNN base methods are now able to predict the action or behavior of human subjects with a notable accuracy. This leads the CNN to the next level in the context of AI. It includes recognition of action from a video sequence or from the still images.

6.4 Image Caption Generation

It means to obtaining a description about the target image, which includes detection and recognition of different objects inside that image with their status description. Here we used CNN to perform the first task and we used several Natural Language Processing (NLP) techniques for a textual status description.

6.5 Medical Image Analysis

With the advancement in CNN-based image analysis, CNN is rapidly proved to be a stateof-the-art foundation, by achieving enhanced performances in the diagnosis of diseases by processing medical images like MRI, X-rays, etc. Nowadays, CNN based models can successfully diagnose the various health problems like breast cancer, pneumonia, brain tumor, diabetes, Parkinson's diseases and many others.

6.6 Security and Surveillance

Nowadays, Security system with Computer Vision capabilities provides constant surveillance to houses, metro stations, roads, schools, hospitals, and many other places, that gives the ability to find or identify the criminals even in crowded areas.

6.7 Automatic colorization of image and style transfer

In the last few years, with the deep learning revolution, some popular CNN models give an automation way to convert black and white images or gray images to equivalent colorful RGB images. As a result, now we can see the old black and white movies in color format. On the other hand, image style transfer is a concept of representing an image in the style of another image, for that a new artificial image could be generated. This style transfer could be efficiently done using convolutional neural networks.

6.8 Satellite Imagery

Nowadays, CNN contribute a vital role to detect different natural hazards like tsunamis, hurricanes, floods, and landslides. By satellite image analysis we can do smart city plan, roadway and river extraction, land classification, crop pattern classification, prevention of deforestation and many more.

7 Conclusion

Convolutional Neural Networks (CNN) has become state-of-the-art algorithm for computer vision, natural language processing, and pattern recognition problems. This CNN has been using to build many use cases models from simply digit recognition to complex medical image analysis. This chapter tried to explain each component of a CNN, how it works to image analysis, and other relevant things. This chapter also gives a review from foundation of CNN to latest models and mentioned some applications areas.