Fundamental Concepts of Convolutional Neural Network

Anirudha Ghosh[1], Abu Sufian[1,*], Farhana Sultana[1], Amlan Chakrabarti[2], Debashis De[3]

[1]Department of Computer Science, University of Gour Banga, W.B., India.
[2]A.K. Choudhury School of Information Technology, University of Calcutta, W.B., India.
[3]Department of Computer Science & Engineering, M.A.K.A.U.T., W.B., India.
[*]Corresponding Author (sufian.csa@gmail.com)

Abstract: Convolutional neural network (or CNN) is a special type of multilayer neural network or deep learning architecture inspired by the visual system of living beings. The CNN is very much suitable for different fields of computer vision and natural language processing. The main focus of this chapter is an elaborate discussion of all the basic components of CNN. It also gives a general view of foundation of CNN, recent advancements of CNN and some major application areas.

Keywords:
Computer Vision, Convolutional Neural Network, CNN, Deep Learning, Image Classification, Image Understanding.

Contents

1 Introduction

Among different deep learning[11] 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[20] 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 [16]. 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[7]. 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[21] to recognize handwritten digits. Training by backpropagation[35] 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[20] showed excellent performance on the ILSVRC[36]. The success of AlexNet paved the way to invent different CNN models[41] 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 downsamples 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 featu res, 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[41], object detection[42] and segmentation[45], 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. The rest of the chapter consist of foundation of CNN in section 2, main contributions of the chapter i.e. detailed concepts of CNN in section 3 with different subsections, the learning process of CNN with different aspects for improvements in section 4, recent advancements of CNN in section 5, and some major application areas in section 7. The last section i.e. section 6 will conclude the chapter.

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**"[16], 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 principle of Deep Learning.
Inspired by the work of Hubel and Wiesel, in 1980, Kunihiko Fukushima proposed **Neocognitron**[7], 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 recogniz 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 performs well in various complex problems. But nowadays, in the era of Big Data we have large labeled datasets, more innovative algorithms and especially powerfull GPU machines. With these type of upgradation, in 2012, Krizhevsky et al. designed AexNet, which achieved a fantastic accuracy on the ImageNet Large Scale Visual Recognition Challenge(ILSVRC[36]). The victory of AlexNet paved the way to invent several CNN models[41] as well as to apply those models in different field of computer vision and natural language processing.

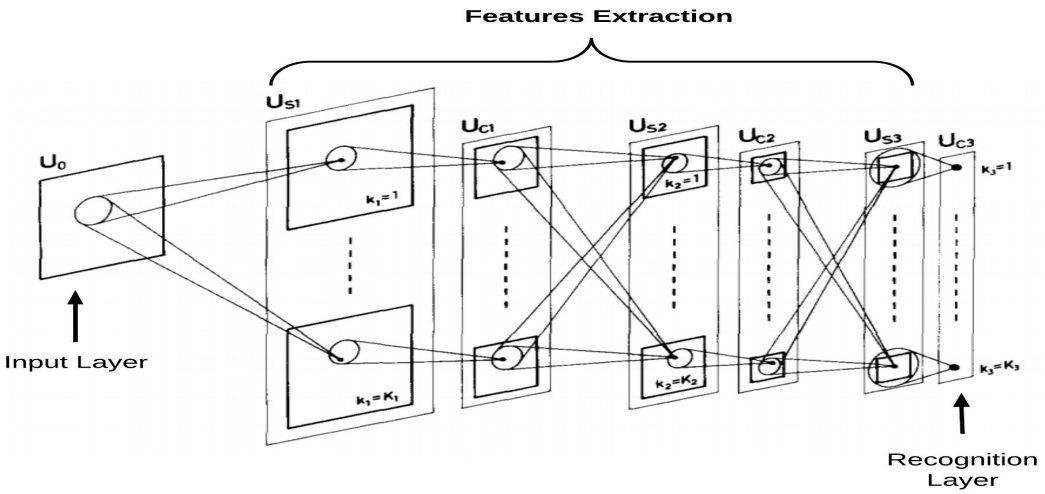

Figure 1: Schematic diagram illustrating the interconnections between layers in the neocognitron, Kunihiko Fukushima [7].

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 indentify 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.

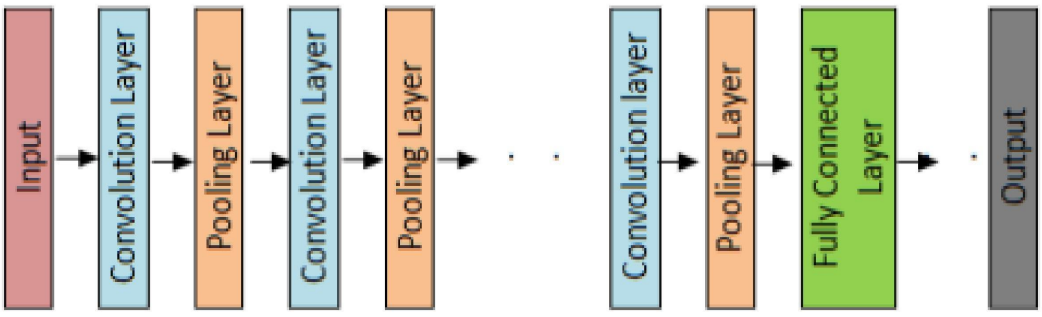

Figure 2: Conceptual model of CNN[41]

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

- One of the main reason 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 roll 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

[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.

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 gray-scale image of 4×4 diamension, shown in Fig.5 and an 2×2 kernel with randomly initialized weights as shown in Fig.6.

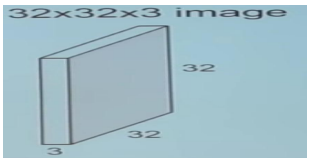

Figure 4: Example of a RGB image

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

Figure 5: An 4×4 Gray-Scale image

0	1
-1	2

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.

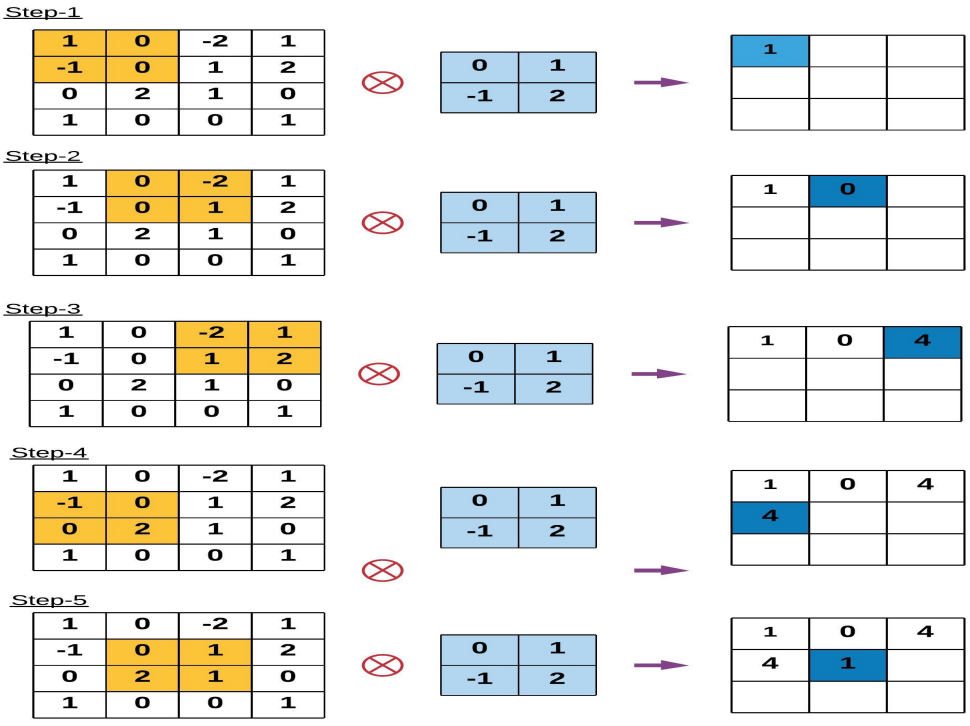

Figure 7: Illustrating the first 5 steps of convolution operation

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. the 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 get 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:

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

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

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.

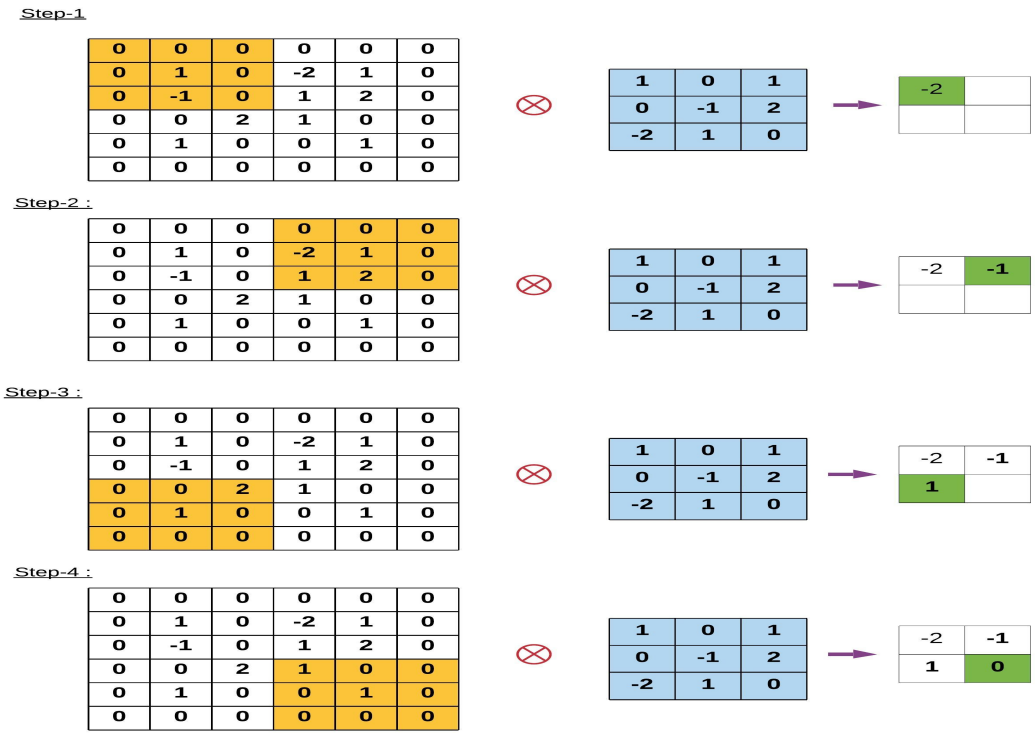

Figure 9: The computations performed at each step, where the 3×3 kernel (shown in light blue color) is multiplied with the same sized region (shown in yellow color) within the 6×6 input image (where we applyed zero-padding to the original input image of 4×4 dimension and it becomes of 6×6 dimensional) 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 preserve 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

[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*

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.

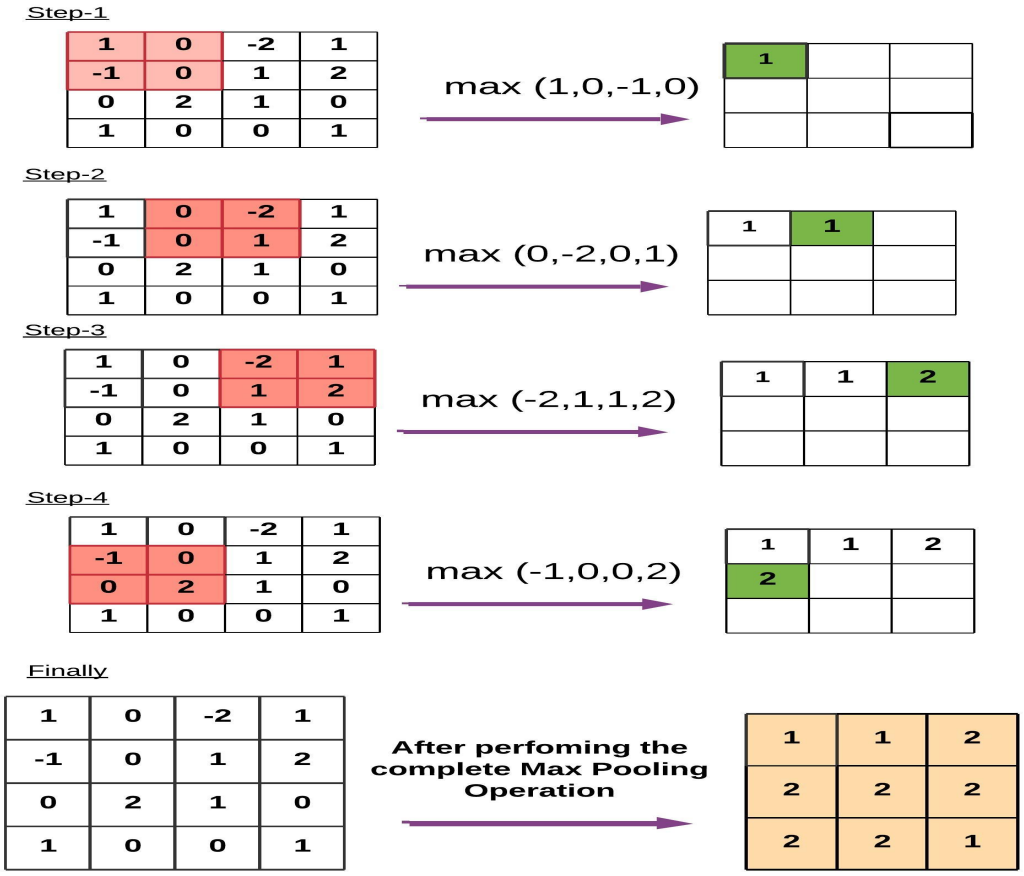

Figure 10: Illustrating an example that shows some initial steps as well as the final output of max-pooling operation, where the size of the pooling region is 2×2 (shown in orange color, in the input feature map) and the stride is 1 and the corresponding computed value in the output feature map (shown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

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 non-linearly. 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:

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

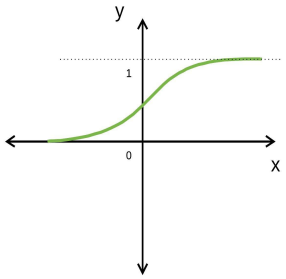

Figure 11: Sigmoid

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:

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

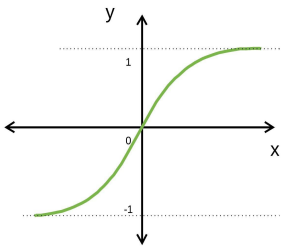

Figure 12: $Tanh$

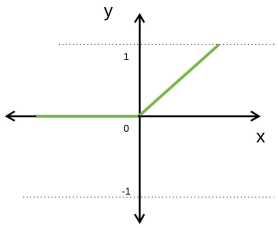

Figure 13: ReLU

3.1.3.3 ReLU:

The Rectifier Linear Unit (ReLU)[25] is the most commonly used activation function in Convolutional Neural Networks. It is used to convert all the input values to posititve 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:

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

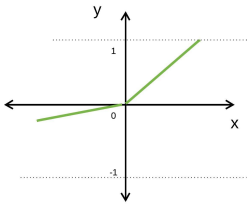

Figure 14: Leaky ReLU

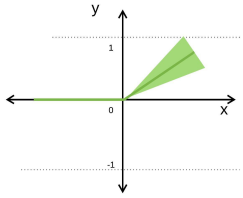

Figure 15: Noisy ReLU

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 it's 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.

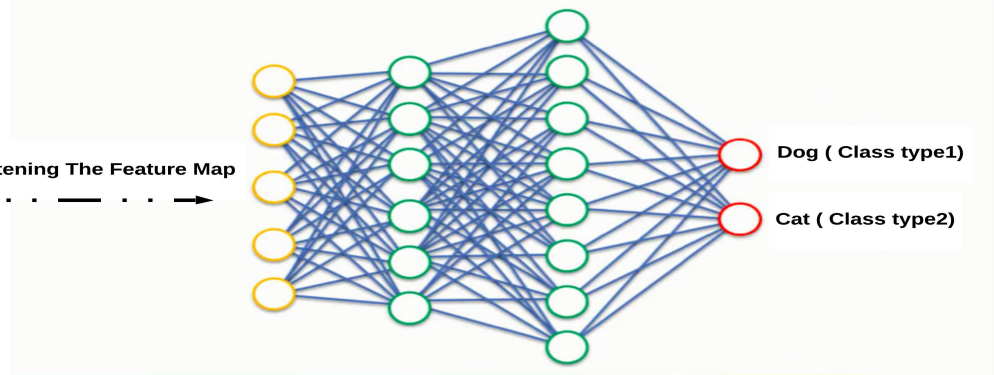

Figure 16: The architecture of Fully Connected Layers

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. $p, y \in \mathbb{R}^N$, 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 :

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

$$\textbf{where } i \in [1, N]$$

3.2.2 Euclidean Loss Function

The Euclidean loss also called mean squared error is widely used in regression problems. The mean squared error between the predicted output $\mathbf{p} \in \mathbb{R}^N$ and the actual output $y \in \mathbb{R}^N$ in

each neuron of the output layer of CNN is defined as $H(p, y) = (p - y)^2$. So, if there are N neurons in the output layer then, the estimate euclidean loss is defined as:

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

3.2.3 Hinge Loss Function

The Hinge loss function is widely used in binary classification problems. It is used in "maximum-margin" based classification problem, most notably for support vector machines (SVMs). Here, the optimizer tries to maximize the margin between two target classes. The hinge loss is defined as:

$$H(p, y) = \sum_{i=1}^{N} \mathbf{max}(0, m - (2y_i - 1)p_i)$$

where m is the margin which is normally set equal to 1, p_i denotes the predicted output and y_i denotes the desired output.

4 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 has described in the next subsections.

4.1 Data pre-processing and Data augmentation

Data pre-processing refers to some artificial transformations to the raw dataset (including training, validation and testing datasets) in order to make the dataset more clean, more feature-full, 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 reduces 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 substruct 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:

$$X^{'} = X - x^{*}$$

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

where N denotes the size of the training datset.

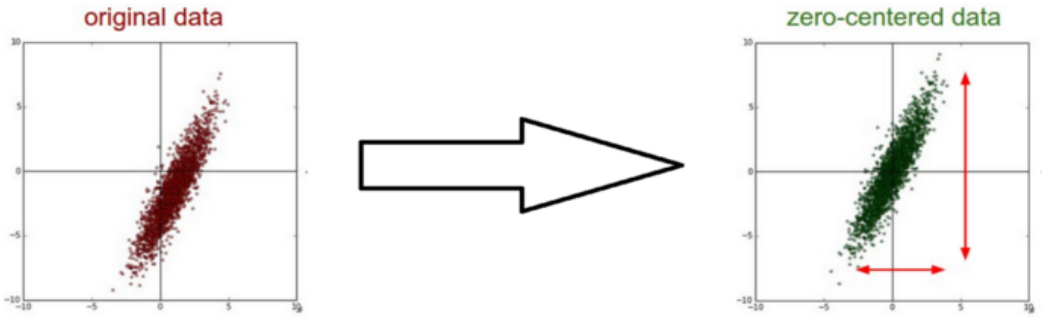

Figure 17: Mean-subtraction (Zero centering the data)(source: http://cs231n.stanford.edu/)

4.1.2 Normalization:

Here we normalize the data sample's (belonging from both train, validation and test dataset) dimension by divideing each dimension by its standard deviationas 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 , $X^{'}$ and X^{*} are the same as discussed in section 4.1.1

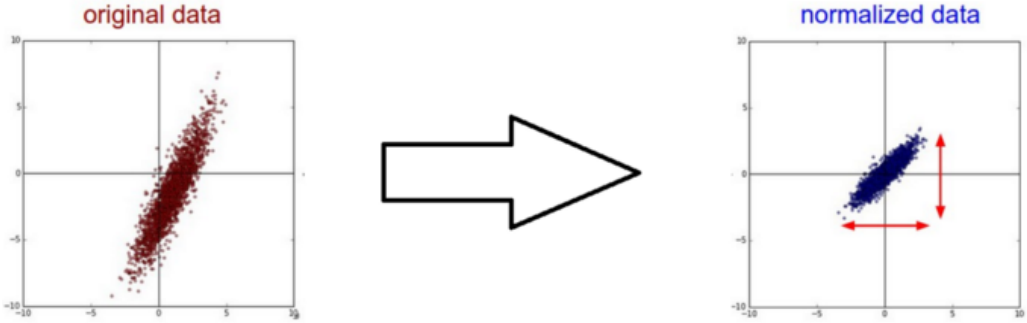

Figure 18: Normalization of data (source: http://cs231n.stanford.edu/)

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

datset 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 most 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:

4.2.1 Random Initialization

As the name suggests, here we initialize the weights (belonging from both convolutional and FC layers) randomly using random matrices, where the elements of that matrices are sampled from some distribution with small standard deviation (e.g., 0.1 and 0.01) and with zero mean. But the key problem of random initialization is that it may potentially lead to **vanishing gradients** or **exploding gradients** problems. Some popular random initialization methods are:

- **Gaussian Random Initialization:** Here weights are randomly initialized using a random matrices, where the elements of that matrices are sampled from an Gaussian distribution .
- **Uniform Random Initialization:** Here weights are randomly initialized using a random Uniform matrices, where the elements of that matrices are sampled from an uniform distribution.
- **Orthogonal Random Initialization:** Here weights are randomly initialized using random orthogonal matrices, where the elements of that matrices are sampled from an orthogonal distribution.

4.2.2 Xavier Initialization

This technique is proposed by Xavier Glorot and Yoshua Bengio in 2010, it tries to make the variance of output connections and input connections to be equal for each layer in the network. The main idea is to balancing of the variance the activation functions. This technique is turned out to be very useful at that time and since Xavier Glorot and Yoshua Bengio designed this tecnique for logistic sigmoid activation function, thus it does not perform well with ReLU (mostly used in CNN architecture nowadays) activation function. Later on, He initialization technique proposed by Kaiming He at al is used to work with ReLU activation on the same idea.

4.2.3 Unsupervised Pre-training Initialization

Here in this technique, we initialize a convolutional neural network with another convolutional neural network (that was trained using an unsupervised technique), for example, a deep autoencoder or a deep belief network. This method can sometimes works very well by helping to handle both the optimization and overfitting issues.

- **Vanishing Gradient Problem:**
  During Back-propagation over a deep convolutional neural network (CNN with many layers e.g.,1000), we need to calculate gradients of loss (Error) with respect to corresponding weights in each layer's neurons for updating those weights. So, here we used the derivative operation to perform this task and it leads the gradients to gets smaller and smaller as we keep on moving backward in the network. Because of that, the earlier layer's neurons receive very small gradients (sometimes gradients may become almost zero), as a result, the earlier layer's weights get very mild update and those layers learn very slowly and inefficiently. This problem is known as Vanishing Gradient Problem.

  Few Solutions:
  1. We can use ReLU/ leaky ReLU instead of others (tanh,sigmoid) as the activation function in CNN architecture, that helps to avoid this problem.
  2. Residual networks provide another solution for this type of problem. Fig.23 shows basic block of the Residual network
  3. Batch normalization layers can also resolve the issue.

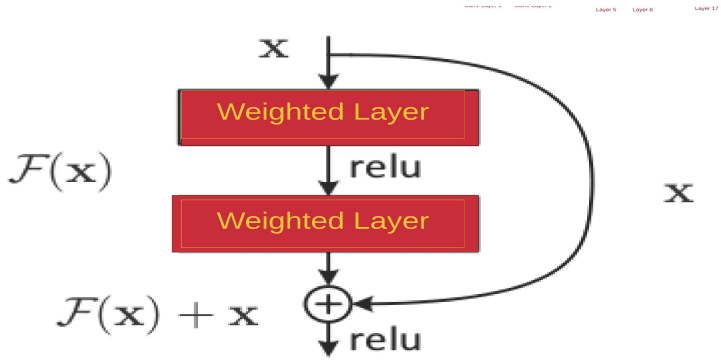

Figure 23: The basic block of the Residual network.

- **Exploding Gradient Problem:**
  This is the exact opposite of vanishing gradient problem, here large error gradients accumulate during back-propagation and it resulted in very large updates to network's weights and make the model unstable and then this unstable model can not able to learns efficiently.
  The explosion in weight update occurs through the exponential growth of gradient by repeatedly multiplying gradients through the network layers during back-propagation when the gradients move backward in the network. At the extreme point, the values of weights (after updated with large gradients) may become so large as to overflow and resulted in NaN values.

  Few Solutions:
  1. We can use different Weight Regularization techniques to avoid this problem.
  2. We can re-design the Network Model architecture to resolve the issue.

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.

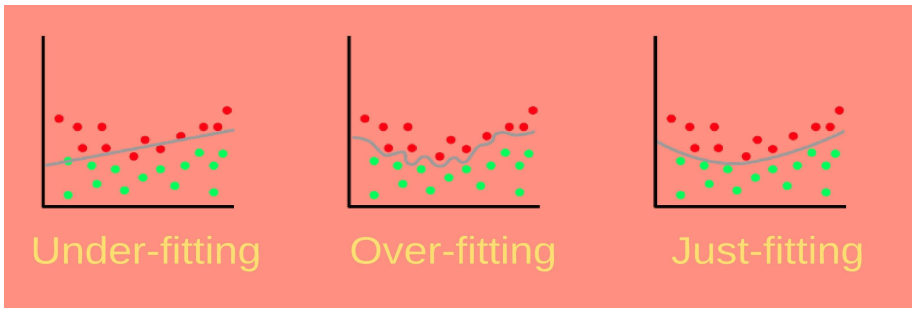

Figure 24: The Examples of over-fitting, under-fitting and just-fitting model's hypothesis with respect to binary classification

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[39] is one of the most used approach 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.

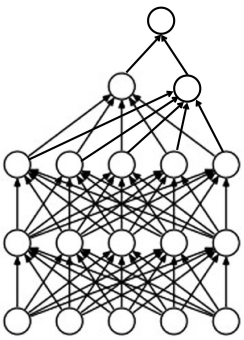

Figure 25: An normal Neural network

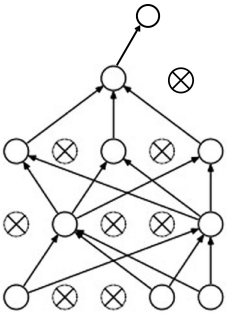

Figure 26: After applying dropout

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[26] 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 $\|w\|$ 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:

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

4.3.4 The $\imath^1$ Regularization

The $\imath^1$ regularization[26] 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 $\imath^1$ 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.

4.3.5 Data Augmentation

The easiest way to avoid over-fitting is to train the model on a large amount of data with several varieties. This can be achived by using data augmentation, where we use several techniques to artificially expand the training dataset size. Section 4.1 have described the data augmentation techniques in more detail.

4.3.6 Early Stopping

In early stoping, we keep a small part (maybe 20% to 30%) of the train dataset as the validation set which is then used for Cross-Validation purposes, where we evaluate the performance of the trained model over this validation set in each training epochs.
Here we use a strategy that stops the training process when the performance on the validation set is getting worse with the next subsequent epoch as shown in Fig.27. As the validation error gets increased the generalization ability of the learned model also gets decreased.

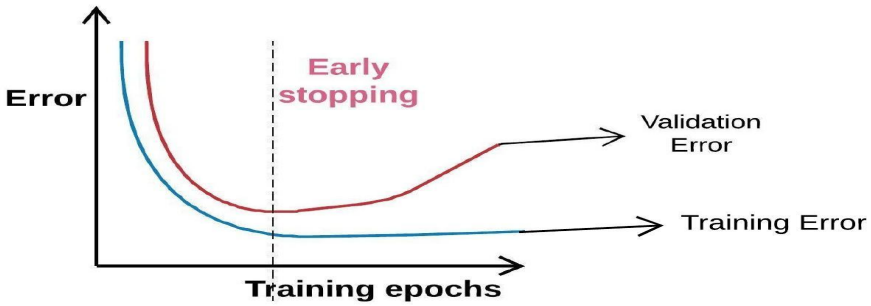

Figure 27: A tipical illustration of early stopping approach during network training.

4.3.7 Batch Normalization

Batch Normalization[17] explicitly makes sure that the output activations of a network will follow a unit Gaussian distribution by normalizing the output at each layer by subtracting the mean and dividing by the standard deviation. It can be imagined as the preprocessing step at every layer of the network, but it can be differentiable and integrated with the networks. Batch Normalization is used to reduce the "internal covariance shift" of the activation layers. The internal covariance shift can be explained as the change in the distribution of activations in each layer. Due to continuous weight updation during training, the "internal covariance shift" can may become very high (it may happen when the training data samples are taken from several different distribution e.g., day-light images and night-vision images) and with this high "internal covariance shift" the model takes more time to converge and training time will increase. To solve this problem Batch Normalization operation is implemented as a layer in a CNN architecture.
The benefits of using batch normalization are given below:

1. It also avoids the vanishing gradient problem.
2. It can handle bad weight initialization very efficiently.
3. It greatly improves the network convergence time (it becomes very helpful in case of large-scale dataset).
4. It tries to reduce training dependency over hyper-parameters.
5. It reduces the chances of overfitting because it has a slight regularization effect.

4.4 Optimizer selection

After successfully discussing the pre-processing steps with the data samples and enforcing regulization techniques to the CNN model, here we are going to discuss the learning process of the CNN model. The learning process includes two major things, first one is the selection of the learning alorithm (Optimizer) and the next one is to use several improvements (such as momentum, Adagrad, AdaDelta) to that learning algorithm in order to improve the result.
The main objective of any supervised learning algorithm is to minimize the **Error** (difference between the predicted output and the actual output) or we can say the loss functions, based on several learnable parameters like weights, biases, etc. In case of learning to a CNN model the **gradient-based learning** methods come as a natural choice. To reduces the error the model parameters are being continuously updated during each training epoch and the model iteratively search for the locally optimal solution in each training epoch.

The size of parameter updating steps is called the "**learning rate**" and a complete iteration of parameter update which includes the whole training dataset once is called a "**training epoch**". Although the learning rate is a hyper-parameter, but we need to choose it so carefully that, it does not affect the learning process badly as shown in Fig.28.

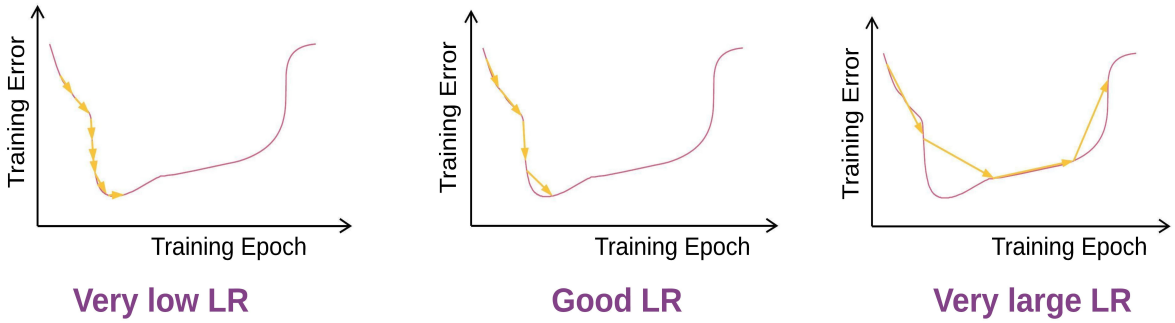

Figure 28: The effect of different learning rate (LR) value on the training process, where we can see, a very low LR value need more training epochs to get the optimal solution and a very large LR value can overshoot the optimal solution.

Gradient Descent or Gradient-Based Learning Algorithm: As we discussed, the gradient descent algorithm update the model parameters continuously during each training epoch in order to reduce the training error. For updating those parameters in correct way, it first calculate the slope (gradient) of the objective function by using first-order derivative with respect to the model's parameters, and then to minimize the error it update the parameter in the opposite direction of the slope (gradient) as shown in Fig.29. This parameter updation process is done during back-propagation of the model, where the gradient at each neuron is back propagate to all the neurons belonging form it's previous layer. This operation can be mathematically represented as:

$$w_{ij}^{t} = w_{ij}^{t-1} - \Delta w_{ij}^{t}$$

$$\Delta w_{ij}^{t} = \eta * \frac{\partial E}{\partial w_{ij}}$$

where w_{ij}^{t} denotes the final weight in current t'th training epoch, w_{ij}^{t-1} denotes the weight in previous $(t-1)$'th training epoch, η is the learning rate, E is the prediction Error.

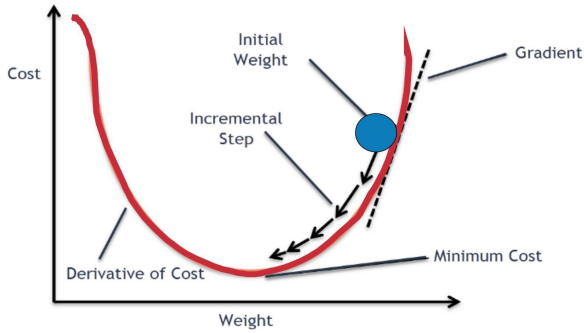

Figure 29: The working principle of Gradient-Based Learning Algorithm.

There exist a number of variants of the gradient-based learning algorithm, out of them the most widely used are:

1. Batch Gradient Descent
2. Mini Batch Gradient Descent
3. Stochastic Gradient Descent

These are described very briefly in the next subsections.

Another widely used learning algorithm or optimization technique is Adam optimization, which uses second-order derivative represented by Hessian Matrix. It has described very briefly in next subsection.

4.4.1 Batch Gradient Descent

In Batch Gradient descent[34] the parameters of the network are updated only once after passing the whole training dataset through the network. That is, it computes the gradient on the entire training set and then updates the parameters using this gradient.
With Batch gradient descent the CNN model produces more stable gradient and also converges faster for small-sized datasets. It also needs fewer resources, because the parameters are updated only once for each training epoch. But if the training dataset becomes large, then it takes more time to converge and it may converge in local optimal solution (in case of non-convex problems).

4.4.2 Stochastic Gradient Descent (SGD)

Unlike Batch Gradient descent, here the parameters are updated for each training sample separately[3]. Here it is recommended to randomly shuffle the training samples in each epoch before training. The benefit of using it over Batch Gradient descent is that it converges much faster in case of large training dataset and it is also memory efficient. But the problem is, due to frequent updates it takes very noisy steps towards the solution that make the convergence behavior very unstable.

4.4.3 Mini Batch Gradient Descent

Here we divide the training examples into a number of mini-batches in non-overlapping manner, where each mini-batch can be imagined as the small set of samples and then update the parameters by computing the gradients on each mini-batch. It carries the benefit of both Stochastic Gradient Descent and Mini Batch Gradient Descent by mixing them. It was more memory efficient, more computationally efficient and also has a stable convergence.
Next are descriptions about the different improvement techniques in Gradient-Based learning algorithms (typically, in SGD) which improves the training process of the CNN model more efficiently.

4.4.4 Momentum

Momentum is a technique used in the objective function of neural networks, it improves both training speed and accuracy by adding the gradient calculated at the previous training step weighted by a parameter λ called the momentum factor. The major problem of Gradient-Based learning algorithm is that it easily stuck in a local minima instead of global minimum, this mostly happens when the problem has non-convex solution space (or surface) as shown in Fig.30. The following figure illustrates this problem.

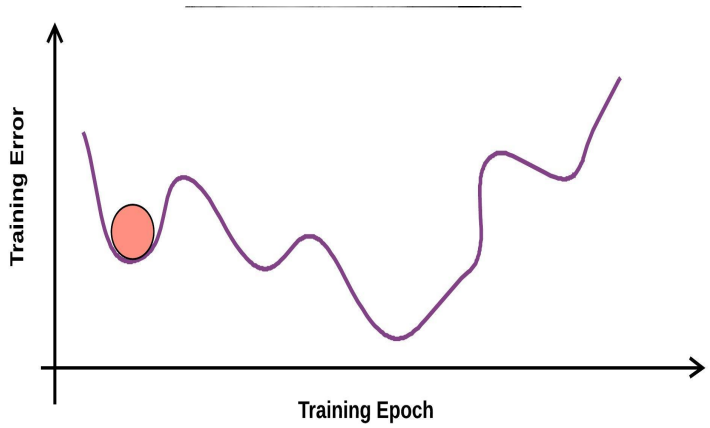

Figure 30: illustration the examples of optimizer stuck in a local minima problem

To solve this issue we used the momentum along with the learning algorithm. It can easily mathematically expressed as :

$$\Delta w_{ij}^{t} = (\eta * \frac{\partial E}{\partial w_{ij}}) + (\lambda * \Delta w_{ij}^{t-1})$$

where Δw_{ij}^{t} is weight increment in current t'th training epoch, η is the learning rate and λ is the momentum factor and (Δw_{ij}^{t-1}) is weight incrementation in previous $(t-1)$'th training epoch.

The value of momentum factor is staying in between 0 and 1 that increases the step size of weight update towards minima, for minimizing the error. The large value of momentum factor helps the model to converge faster and the very lower value of momentum factor can not able to avoid local minima. But if we use both LR and momentum factor value as high, it may also missed global minima by jumping over it.
In case, if the gradient keeps changing its driection continuously during training the good momentum factor value makes smoothing out the variations of weight update. The momentum factor is a hyper-parameter.

4.4.5 AdaGrad

AdaGrad or adaptive learning rate method update each network parameter differently, based on their significance for the problem, that is, here we perform larger updates for infrequent parameters (by using larger learning rate value) and smaller updates for frequent parameters (by using smaller learning rate value). It is done by dividing the learning rate of each parameter with the sum of square of all the past gradients for each parameter w_{ij} in each training epoch t. In practice AdaGrad is very useful, especially in case of sparse gradients or when we have sparse training data for a large scale neural networks. The update operation can easily mathematically expressed as:

$$w_{ij}^{t} = w_{ij}^{t-1} - \frac{\eta}{\sqrt{\sum_{k=1}^{t} {\delta_{ij}^{k}}^{2} + \in}} * \delta_{ij}^{t}$$

where w_{ij}^{t} is the weight in current t'th training epoch for parameter w_{ij} , w_{ij}^{t-1} is the weight in previous $(t-1)$ 'th training epoch for parameter w_{ij} , δ_{ij}^{t} is the local gradient of parameter w_{ij} in t 'th epoch , δ_{ij}^{t-1} is local gradient of parameter w_{ij} in $(t-1)$'th epoch, η is the learning rate and $\in$ is a term contains very small value to avoid dividing by zero.

4.4.6 AdaDelta

AdaDelta can be imagined as the extension of AdaGrad. The problem with AdaGrad is that, if we train the network with many large training epochs (t), then the sum of square of all the past gradients ($\sum_{m=1}^{t} \delta_{ij}^{m2}$) become large, as a result, it almost vanishes the learning rate.
To solve this issue the adaptive delta (AdaDelta) method divide the learning rate of each parameter with the sum of square of past k gradients (instead of using all the past gradients, which is done in the case of AdaGrad) for each parameter w_{ij} in each training epoch t. The update operation can easily mathematically expressed as:

$$w_{ij}^{t} = w_{ij}^{t-1} - \frac{\eta}{\sqrt{\sum_{m=(t-k+1)}^{t} \delta_{ij}^{m2} + \in}} * \delta_{ij}^{t}$$

where w_{ij}^{t} is the weight in current t'th training epoch for parameter w_{ij}, w_{ij}^{t-1} is the weight in previous $(t-1)$'th training epoch for parameter w_{ij}, δ_{ij}^{t} is local gradient of parameter w_{ij} in t'th epoch , δ_{ij}^{t-1} is local gradient of parameter w_{ij} in $(t-1)$'th epoch, η is the learning rate and $\in$ is a term contains very small value to avoid dividing by zero.

4.4.7 RMSProp

Root Mean Square Propagation (RMSProp) is also designed to solve the Adagrad's radically diminishing learning rates problem as discussed in the previous section. It was developed by Geoffrey Hinton's group, it tries to resolve Adagrad's issue by using a moving average over past squared gradient E[δ^2]. The update operation can easily mathematically expressed as:

$$w_{ij}^{t} = w_{ij}^{t-1} - \frac{\eta}{\sqrt{\mathbf{E}[\delta^2]^t}} * \delta_{ij}^{t}$$

And,

$$\mathbf{E}[\delta^2]^t = \gamma \mathbf{E}[\delta^2]^{t-1} + (1-\gamma)(\delta^t)^2$$

where for an efficient learning rate adjustment during training. Hinton suggests γ to be set to 0.9, with a good default initial learning rate value like 0.001.

4.4.8 Adaptive Moment Estimation (Adam)

Adaptive Moment Estimation (Adam[19]) is another learning strategy, which calculates adaptive LR for each parameter in the network and it combines the advantages of both Momentum and RMSprop by maintaining the both exponential moving average of the gradients (as like Momentum) and as well as the exponential moving average of the squared gradients (as like RMSprop). So the formulas for those estimators are as:

$$\mathbf{E}[\delta]^t = \gamma_1 \mathbf{E}[\delta^2]^{t-1} + (1-\gamma_1)[\delta^t]$$
$$\mathbf{E}[\delta^2]^t = \gamma_2 \mathbf{E}[\delta^2]^{t-1} + (1-\gamma_2)(\delta^t)^2$$

$\mathbf{E}[\delta]^t$ is the estimate of the first moment (the mean) and $\mathbf{E}[\delta^2]^t$ is the estimate of the second moment (the uncentered variance) of the gradients. Since at initial training epoch the both estimates are set to zero, they can remain biased towards zero even after many iterations,

especially when γ_1, γ_2 are very small. To counter this issue the estimates are calculated after bias-correction and the final formulas for those estimators become as:

$$\widehat{\mathbf{E}[\delta]^t} = \frac{\mathbf{E}[\delta]^t}{(1-(\gamma_1)^t)}$$

$$\widehat{\mathbf{E}[\delta^2]^t} = \frac{\mathbf{E}[\delta^2]^t}{(1-(\gamma_2)^t)}$$

So the parameter update operation in Adam finally can easily mathematically expressed as:

$$w_{ij}^t = w_{ij}^{t-1} - \frac{\eta}{\sqrt{\widehat{\mathbf{E}[\delta^2]^t}} + \in} * \widehat{\mathbf{E}[\delta]^t}$$

- Adam is more memory efficient than others and also needs less computational power.

5 Recent advancement in CNN Architectures

Till now it has covered the basic concepts of CNN along with different basic components or building blocks of CNN in section 3. Then in section 4, we discuss the learning process of CNN with several learning algorithms with guidelines in order to improve efficiency (including pre-processing, parameter initialization and regularization to CNN). In this section we try to explain some example of successful CNN architecture that shows the recent major advancements in CNN architecture in the computer vision field. Computer vision has three major subdomain where several CNN architectures (models) contribute a vital role to achieve excellent result, we are going to discuss those subdomains with related CNN models as follows.

5.1 Image Classification

In image classification, we assume that the input image contains a single object and then we have to classify the image into one of the pre-selected target classes by using CNN models. Some of the major CNN architectures (models) designed for image classification are briefly described as follows:

5.1.1 LeNet-5

The LeNet-5[21] is one of the earliest CNN architecture, which was designed for classifying the handwritten digits. It was introduced by LeCun et al. in 1998. The LeNet-5 has 5 weighted (trainable) layers, that is, three convolutional layer and two FC layers. Among them, each of first two convolution layer is followed by a max-pooling layer (to sub-sample the feature maps) and afterward, the last convolution layer is followed by two fully connected layers. The last layer of those fully connected layers is used as the classifier, which can classify 10 digits. The architecture of LeNet-5 is shown in Fig.31.

Key Note:

- The LeNet-5 was trained on the MNIST digit dataset.
- It used sigmoid non-linearity as the activation function.
- It used stochastic gradient descent (SGD) learning algorithm with 20 training epoch.
- It used 0.02 as the momentum factor value.
- It reduced test error rate to 0.95% on MNIST data set.

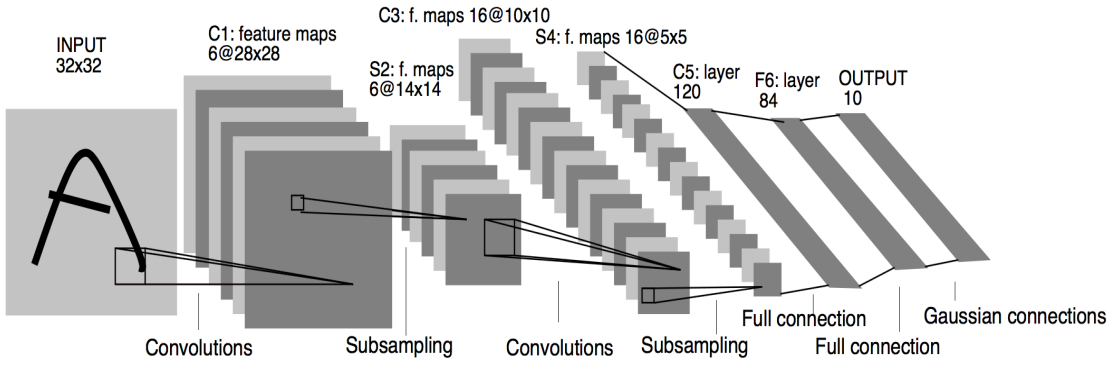

Figure 31: The Architecture of LeNet-5[21].

5.1.2 AlexNet:

Inspired from LeNet, Krizhevky et al. designed first large-scale CNN model, called AlexNet[20] in 2012, which is designed to classify ImageNet data. It consists of eight weighted (learnable) layers among which the first five layers are convolutional layers, and afterward, the last three layers are fully connected layers. Since it was designed for ImageNet data, so the last output layer classify the input images into one of the thousand classes of the ImageNet dataset with the help of 1,000 units. The architecture of AlexNet is shown in Fig.32.

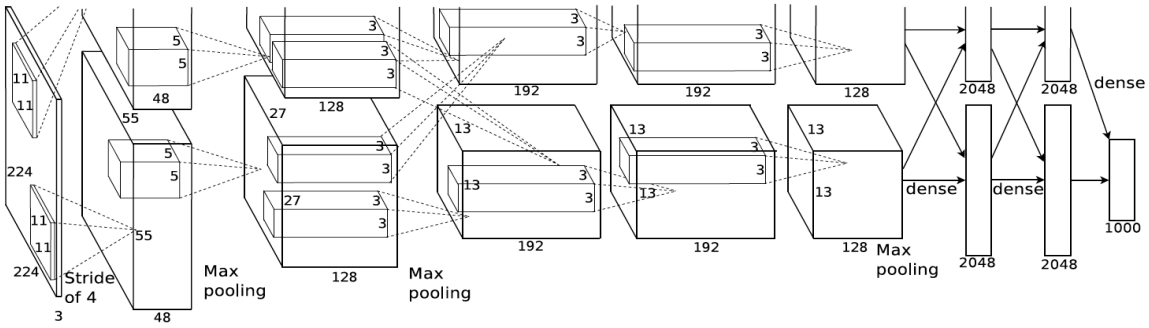

Figure 32: The Architecture of AlexNet[20].

Key Note:

- The AlexNet used rectified linear unit (ReLU) non-linearity activation function after each convolutional and fully connected layer.
- It used max-pooling layer after each LRN layer and the last convolutional layer.
- Since it has a larger number of weights (learnable), so to avoid over-fitting it uses several regularization tricks like dropout and data augmentation.
- The AlexNet was trained using stochastic gradient descent(SGD) learning algorithm with min-batch size 128, weight decay 0.0005 and momentum factor value 0.9.
- The AlexNet was trained (on the ImageNet dataset) in two NVIDIA GTX 580 (with 3 GB memory) using cross-GPU parallelization and it takes around six days to complete.
- AlexNet was the winner of ILSVRC-2012.

5.1.3 ZFNet:

ZFNet[46] was presented by Zeiler and Fergus in ECCV-2014, it has almost similar architecture as AlexNet except that here they used 7×7 filter with stride 2 in 1'st convolutional layer. In case of AlexNet, Krizhevky et al. use 11×11 filter with stride 4 in 1'st convolutional layer. As a result ZFNet becomes more efficient than AlexNet and become the winner of ILSVRC-2013. The architecture of ZFNet is shown in Fig.33.

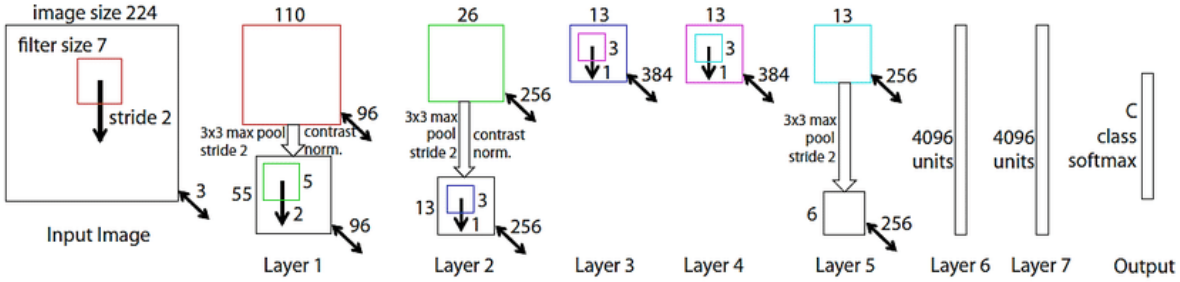

Figure 33: The Architecture of ZFNet[46].

5.1.4 VGGNet:

VGGNet[38] is one of the most popular CNN architecture, which is introduced by Simonyan and Zisserman in 2014. The authors introduced a total 6 different CNN configurations, among them the VGGNet-16 (configuration D) and VGGNet-19 (configuration E) are the most successful ones. The architecture of VGGNet-16 is shown in Fig.3.1.1.2.

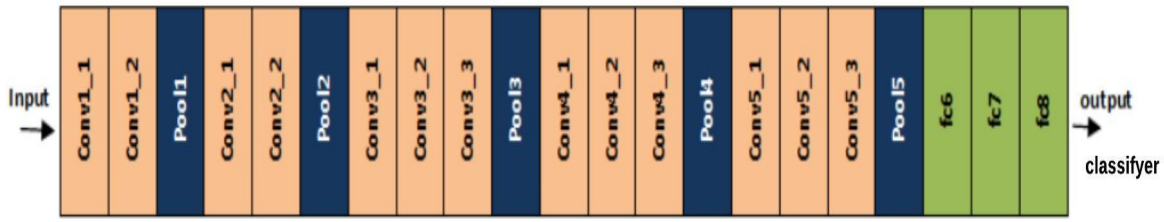

Figure 34: The Architecture of VGGNet[38] (VGGNet-16).

Key Note:

- The reason behind the popularity of VGGNet is its architectural simplicity and the use of small-sized filters for convolutional operation.
- It shows, a stack of 3×3 sized filters has same effective receptive field as the larger sized filters in convolution operation (e.g., two layers of 3×3 sized filters has same effective receptive field as the 5×5 filters in convolution operation, 7×7 filters with three layer of 3×3 sized filters, and so on). Most importantly, use of small sized filters decreases the number of parameters of the network.
- The VGGNet was designed and trained on the ILSVRC dataset.
- It is very deeper network compare to AlexNet.

- It also used rectified linear unit (ReLU) non-linearity activation function after each convolutional and fully connected layer.
- It also uses several regularization tricks like dropout and data augmentation to avoid over-fitting.

5.1.5 GoogLeNet:

The GoogleNet[43] architecture is different from all the previously discussed conventional CNN models, It uses network branches instead of using single line sequential architecture. The GoogleNet was proposed by Szegedy et al. in 2014. The GoogleNet has 22 weighted (learnable) layers, it used "Inception Module" as the basic building block of the network. The processing of this module happens in parallel in the network, and each (a simple basic) module consist of 1×1, 3×3 and 5×5 filtered convolution layers in parallel and then it combines their output feature maps, that can resulted in very high-dimensional feature output. To solve this issue they used inception module with dimensionality reduction (as shown in Fig.35-2) in their network architecture instead of the naive (basic) version of inception module (as shown in Fig.35-1).

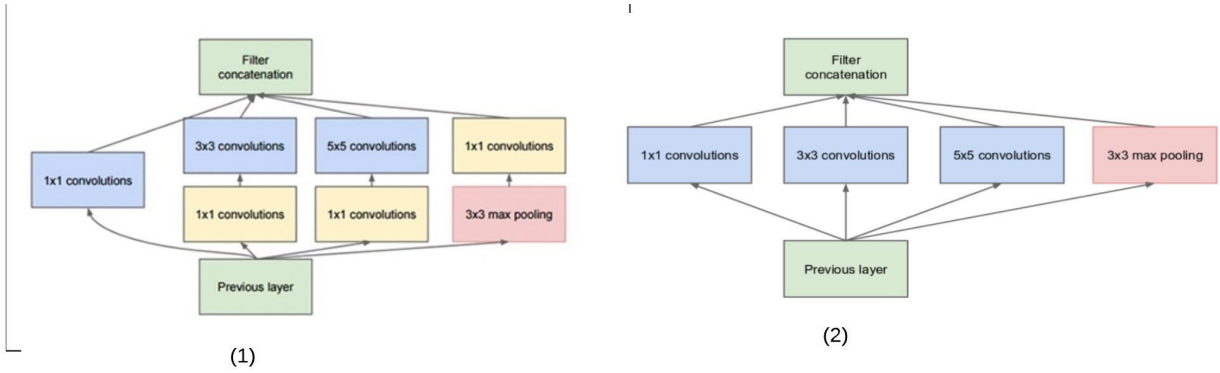

Figure 35: (1) Simple Inception Module[43], (2) Inception Module with dimensionality reduction[43].

Key Note:

- Although the GoogLeNet has 22 layers, but it has 12 times lesser parameters than AlexNet.
- It has auxiliary classifiers, that is use to combat vanishing gradient problem.
- It also used rectified linear unit (ReLU) non-linearity activation function.
- It used an average pooling layer instead of the fully connected layers.
- The GoogLeNet used SGD learning algorithm with a fixed learning rate and with 0.9 as momentum factor.
- The GoogLeNet was the winner of ILSVRC-2014

5.1.6 ResNet:

Since a deep CNN model suffers from vanishing gradient problems as we discussed earlier, He et al. from Microsoft, introduced the idea of "identity skip connection" to solve vanishing gradient problem by proposing the ResNet[14] model. The ResNet's architecture use residual mapping ($H(x) = F(x) + x$) instead of learning a direct mapping ($H(x) = F(x)$) and these bolcks are called residual bocks. The complete ResNet architecture is consist of many residual bocks with 3×3 convolution layers. Fig.36 illustrates the difference between the direct mapping and the residual mapping.

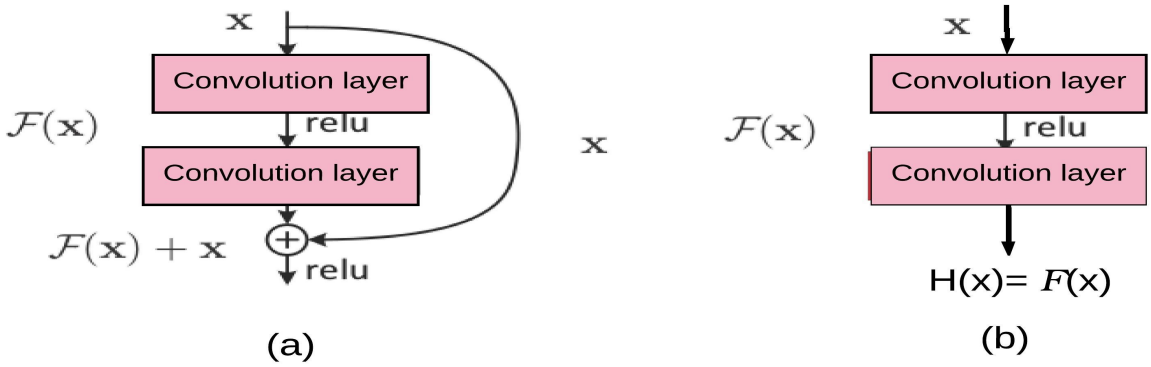

Figure 36: (a) Mapping inside Residual block, (b) Simple direct mappings.

Key Note:

- The authors propose several version of ResNet with different depth, and they also used 'bottleneck' layer for dimensionality reduction in each ResNet architecture that has depth more than 50.
- Although the ResNet (with 152 Layer) is 8 times deeper than VGGNets (22 layers), it has complexity lower than VGGNets (16/19).
- The ResNet used SGD learning algorithm with the min-batch size of 128, weight decay of 0.0001 and momentum factor of 0.9.
- The ResNet was the winner of ILSVRC-2015 with a big leap in performance, it reduces the top-5 error rate to 3.6% (the previous year's winner GoogleNet has the top-5 error rate to 6.7%).

5.1.7 DenseNet:

DenseNet[15] extends the idea of residual mapping by propagating the output of each block to all subsequent blocks inside each dense block in the network as shown in Figure 35. By propagating the information in both forward and backward directions during the training of the model it strengthens feature propagation ability and solve the vanishing gradient problem. DenseNet was introduced by Huang et al. in 2016 and it becomes the winner of ILSVRC-2016. Fig.37 shows a DenseNet based model.

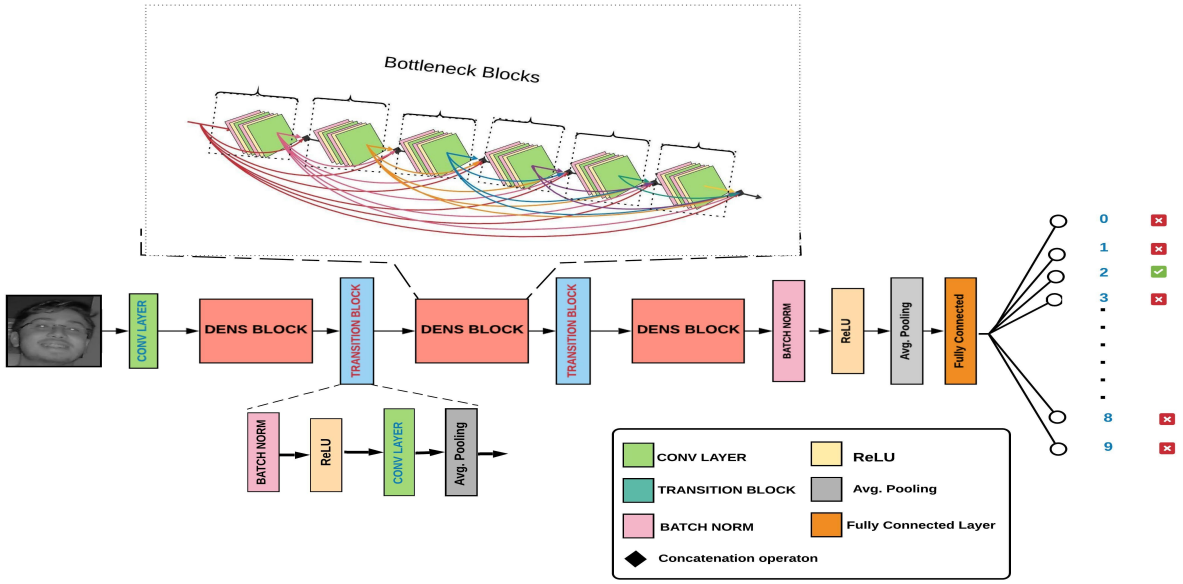

Figure 37: A DenseNet based model with three dense blocks. [40].

5.2 Object Detection

Unlike image classification, here an input image can contain more than one object. We try to detect those objects inside the input image with proper identification of each object along with their correct location inside that image by using CNN models. Some of the major CNN architectures(models) designed for object detection are briefly described as follows:

5.2.1 R-CNN:

The first CNN model designed for object detection is Region-based CNN (R-CNN[10]), which uses sliding window based approaches for successfully detect the objects. Here the authors divides the complete task into three modules. The first module extracts the region (window) proposals from each input images that may likely contain some object (by using traditional selective search technique) and then in the second module, the authors used affine image warping to make all the extracted region proposals of fixed sized (or fixed aspect ratio) and then fed those warped region proposals through the AlexNet CNN model to extract the final features (fixed sized feature vectors). Finally, in the third module, the objects are classified from each region proposals by using category-specific linear support vector machine (SVM) classifier. The architecture of R-CNN is shown in Fig.38.

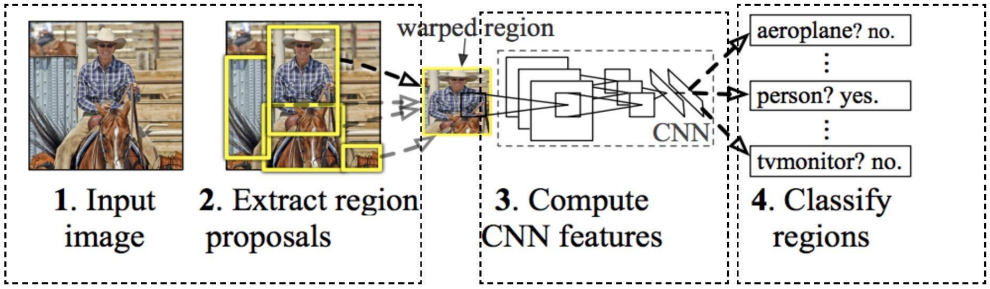

Figure 38: Architecture of R-CNN[10]

5.2.2 SPP-Net:

SPP-Net is almost similar to R-CNN, but SPP-Net[13] uses spatial pyramid pooling (SPP) on each variable length region (window) proposals to make them fixed sized output before feeding to the fully connected layers. SPP-Net removes the constraint of fixed sized input images from R-CNN and make itself more effective.

5.2.3 Fast R-CNN:

Although R-CNN and SPP-Net works well, but they had some major problems like taking huge time to extract region proposals because of the traditional Selective Search technique, multi-stage pipeline process, etc. To solve those issues the Fast R-CNN[9], an CNN based detector uses convolution layer before the extraction of region proposals. This method substantiality reduce the number of region proposals, as the result Fast R-CNN became more efficient than R-CNN. The Fast R-CNN was designed by Ross Girshick, here both the objectives, identification of each object along with their correct location was performed in a single-stage process by using two sibling branch in the output layer. Fast R-CNN uses region of interests (RoI) pooling layer for reshape the variable length region (window) proposals into fixed sized output before feeding them to the fully connected layers. The architecture of Fast R-CNN is shown in Fig.40.

5.2.4 Faster R-CNN:

Faster R-CNN[32] is almost similar to Fast R-CNN, but, here the authors replaced previous traditional selective search technique with a region proposal network (RPN). The RPN is a fully convolutional network used to produce the high-quality region proposals. Faster R-CNN (combining RPN with Fast R-CNN) can be trained in an end-to-end fashion. By using RPN, Faster R-CNN achieves further speed-up in order to detect the objects inside the input image. The architecture of Faster R-CNN is shown in Fig.39.

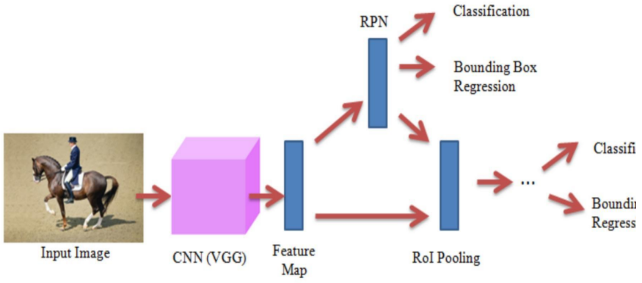

Figure 39: A architecture of Faster R-CNN [42]

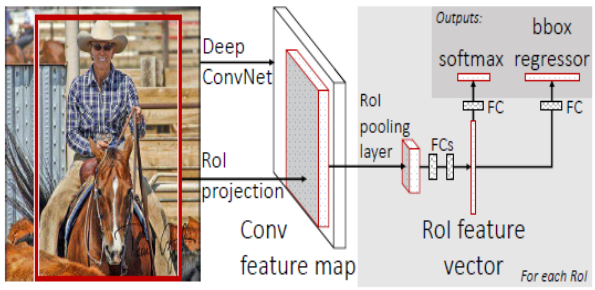

Figure 40: Architecture of Fast R-CNN[9]

5.2.5 Mask R-CNN:

Mask R-CNN[12] extend the concept of Faster R-CNN by locating exact pixels of each object (combinedly called object's mask), whereas Faster R-CNN just predicts the bounding boxes of each object. Mask R-CNN use a RoIAlign layer instead of RoI pooling layer in same Faster R-CNN architecture. RoIAlign layer is used to align the extracted features of an object from region proposals to the location of that object in the final output. So, the Mask R-CNN generate three outputs: predicted class, the location of the object and a binary object mask. The architecture of Mask R-CNN is shown in Fig.41.

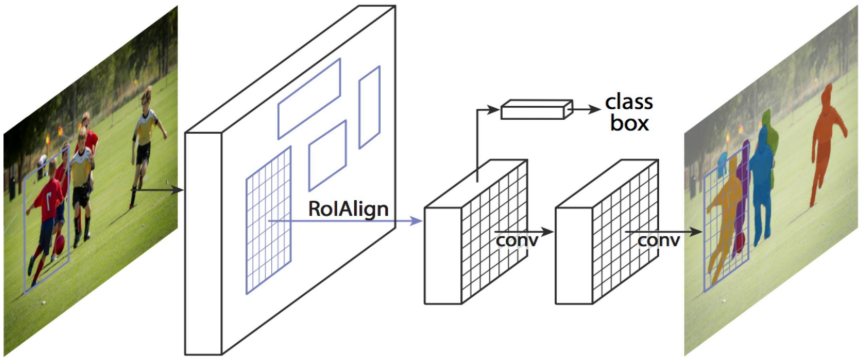

Figure 41: Architecture of Mask R-CNN[12].

5.2.6 YOLO:

YOLO[31] (You Only Look Once) is a single pipeline based detection model, that can directly detect the objects as well as their location using an end-to-end trained CNN model.
YOLO splits the input image into a set of grides (fixed number) and then from each grides the network generate a fixed number of bounding box locations with a class probability. Then it uses a threshold value to select and locate the object within the image or not(when the class probability is less than the threshold value).

5.3 Image Segmentation

CNN has shown excellence in the segmentation task also. After the record-breaking performance of AlexNet in 2012, we have got many state-of-the-art models of semantic segmentation and instance segmentation. Some of them are highlighted below.

5.3.1 Semantic Segmentation:

Unlike Classification and object detection, semantic segmentation is a low level vision task. It is the process of associating each pixel of an image with a class label i. e. It detects all the objects present in an image.

5.3.1.1 Fully Convolutional Network (FCN):

The author of FCN[37] have substituted the fully connected layer of AlexNet, VggNet and GoogLeNet (all three networks are pretrained on ILSVRC dataset) by 1x1 convolutional layers to make them dense FCN. The last layer is made up with a 1×1 convolution with channel dimension tweenty one to predict scores for each of the PASCAL VOC[6] class (including background). To produce fine-grained segmentation the authors have used bilinear interpolation and skip connection in their architecture. The end-to-end model of FCN is shown in Fig.42.

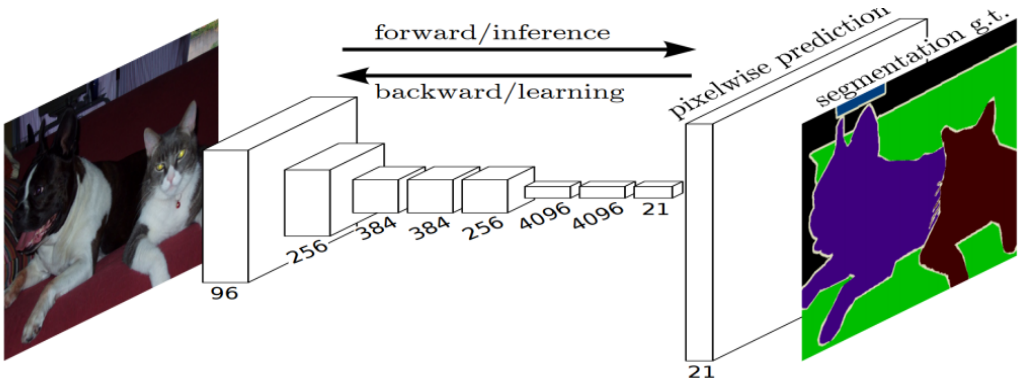

Figure 42: End-to-end model of FCN[37].

5.3.1.2 DeepLab:

Application of deep CNN in semantic segmentation has encountered two drawbacks: down-sampling and spatial invariance. To handle the first problem the authors have used 'atrous' (with holes) algorithm. To get rid of the second problem they have used conditional random field (CRF) to capture fine details. DeepLab[4] achieved 71.6% Intersection over Union (IoU) accuracy in the test set of the PASCAL VOC 2012 semantic segmentation task. DeepLabv2 has one additional technology called Atrous Spatial Pyramid Pooling (ASPP), which is the main difference from DeepLabv1.

5.3.1.3 SegNet:

The author has used encoder-decoder architecture[2] for semantic segmentation. The encoder part uses 13 layers of VGG16 network and the decoder part uses those 13 layers in reverse order. The last layer is a pixel-wise classification layer. The end-to-end model of SegNet is shown in Fig.43.

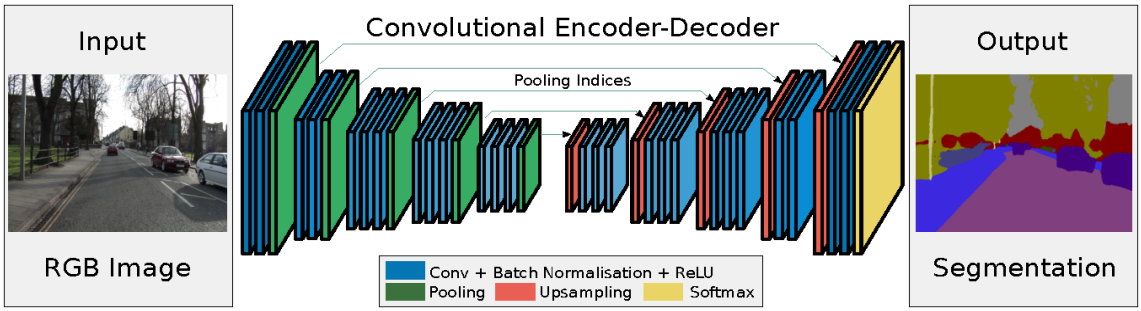

Figure 43: End-to-end model of SegNet[2].

5.3.1.4 Deconvnet:

This network[27] consists of 13 convolutional layers and 2 fully connected layers from VGG16 as convolutional network and those 15 layers in hierarchically opposite order are used in deconvolutional network. Convolutional layer extracts feature maps using convolution and pooling layers whereas the deconvolutional network uses deconvolution and unspooling to reconstruct the original size of activation. The end-to-end model of Deconvnet is shown in Fig.44.

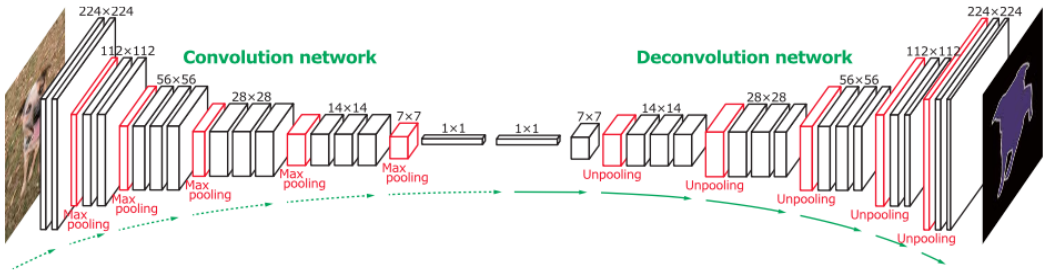

Figure 44: End-to-end model of Deconvnet[27].

5.3.1.5 U-Net:

This architecture[33] consists of a u-shaped contracting and expansive path. Each step of the contracting path consists of two 3×3 convolutions, ReLU and 2×2 max-pooling. In contrast, expansive path consists of 2×2 upconvolution, 3×3 convolution and ReLU. In between upconvolution and convolution in expansive path, the feature map is concatenated with the cropped feature map from contracting path from corresponding layer.

5.3.2 Instance Segmentation:

Instance segmentation takes the semantic segmentation one step ahead. It detects as well as distinguishes all the instances of an object present in an image.

5.3.2.1 DeepMask:

The author used VGGNet in their architecture[28] but removed last max-pooling layer and fully connected layers. After VGGNet, the extracted feature map enters into two paths for class agnostic semantic mask and assigning a score corresponding to how likely the patch is to contain an object. The first path contains 1×1 convolution layer, a nonlinear layer, two FC layers followed by a bilinear interpolation to mask single object. The second path contains 2×2 max pool layer followed by two FC layers and a prediction layer. The end-to-end model of DeepMask is shown in Fig .45.

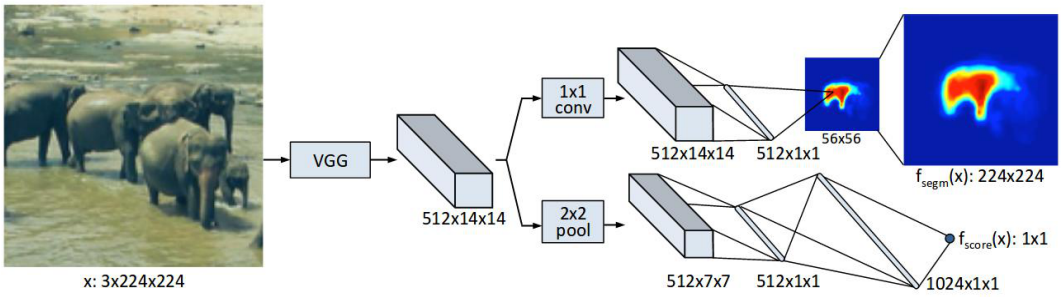

Figure 45: End-to-end model of DeepMask[28]

5.3.2.2 SharpMask:

The architecture of SharpMask[29] is similar to DeepMask except it produces sharper, pixel-accurate object mask. This network consists of a series of convolution –pooling layer followed by fully connection layer to generate the object Mask.

5.3.2.3 PANet:

This model[23] is based on the framework of Mask R-CNN and the Feature Pyramid Network. The main idea of PANet is to enhance information propagation through the network. The

authors have used FPN based feature extractor associated with a new augmented bottom up pathway to improve the propagation of low layer features. To extract proposals from all level features , the feature maps are subsampled with a RoIAlign pooling layer. In each stage, an adaptative feature pooling layer processes the features maps with a fully connected layer . Then the network concatenates all the outputs. The output feature pooling layer goes to three branch for prediction of bounding box, prediction of object class and prediction of binary pixel mask.

5.3.2.4 TensorMask:

This architecture uses dense sliding window approach instead of detecting object in a bounding box. The main concept of TensorMask[5] architecture is the use of structured high-dimensional tensors to present image content such as masks in a set of densely sliding windows. These models have two heads : One for generating mask in sliding window another for prediction of object categories.

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 recognization, 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[41] 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[36] 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[22] dataset with a good accuracy. After that in recent years, with several improvements, CNN contributes a vital role[44] 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[18] 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 state-of-the-art foundation, by achieving enhanced performances in the diagnosis of diseases by processing medical images[1] like MRI, X-rays, etc. Nowadays, CNN based models can successfully diagnose the various health problems like breast cancer, pneumonia, brain tumour, 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[30] .

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[47]. 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 concepts of representing an image in the style of other image, for that a new artificial image could be generated. This style transfer could be efficiently done using convolutional neural networks[8].

6.8 Satellite Imagery

Nowadays, CNN contribute a vital role to detect different natural hazards[24] 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 components 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.

References

[1] S. M. Anwar, M. Majid, A. Qayyum, M. Awais, M. Alnowami, and M. K. Khan. Medical image analysis using convolutional neural networks: A review. *J. Med. Syst.*, 42(11):1–13, Nov. 2018.

[2] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, abs/1511.00561, 2015.

[3] L. Bottou. Large-scale machine learning with stochastic gradient descent. In Y. Lechevallier and G. Saporta, editors, *Proceedings of COMPSTAT'2010*, pages 177–186, Heidelberg, 2010. Physica-Verlag HD.

[4] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[5] X. Chen, R. B. Girshick, K. He, and P. Doll&39;ar. Tensormask: A foundation for dense object segmentation. *CoRR*, abs/1903.12174, 2019.

[6] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vision*, 88(2):303–338, June 2010.

[7] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, Apr 1980.

[8] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[9] R. Girshick. Fast r-cnn. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1440–1448, Washington, DC, USA, 2015. IEEE Computer Society.

[10] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[11] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[12] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, Oct 2017.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729, 2014.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[15] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[16] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *Journal of Physiology (London)*, 195:215–243, 1968.

[17] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[18] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[19] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[21] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[22] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[23] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. *CoRR*, abs/1803.01534, 2018.

[24] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez. Convolutional neural networks for large-scale remote-sensing image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 55(2):645–657, Feb 2017.

[25] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[26] A. Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 78–, New York, NY, USA, 2004. ACM.

[27] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. *CoRR*, abs/1505.04366, 2015.

[28] P. H. O. Pinheiro, R. Collobert, and P. Doll&39;ar. Learning to segment object candidates. *CoRR*, abs/1506.06204, 2015.

[29] P. H. O. Pinheiro, T. Lin, R. Collobert, and P. Doll&39;ar. Learning to refine object segments. *CoRR*, abs/1603.08695, 2016.

[30] P. Rasti, T. Uiboupin, S. Escalera, and G. Anbarjafari. Convolutional neural network super resolution for face recognition in surveillance monitoring. volume 9756, pages 175–184, 07 2016.

[31] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, June 2016.

[32] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[33] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. (available on arXiv:1505.04597 [cs.CV]).

[34] S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[35] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. Mcclelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[36] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[37] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):640–651, 2017.

[38] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[39] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014.

[40] A. Sufian, A. Ghosh, A. Naskar, and F. Sultana. Bdnet: Bengali handwritten numeral digit recognition based on densely connected convolutional neural networks. *CoRR*, abs/1906.03786, 2019.

[41] F. Sultana, A. Sufian, and P. Dutta. Advancements in image classification using convolutional neural network. In *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, pages 122–129, Nov 2018.

[42] F. Sultana, A. Sufian, and P. Dutta. A review of object detection models based on convolutional neural network. *CoRR*, abs/1905.01614, 2019.

[43] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[44] T. Wang, D. J. Wu, A. Coates, and A. Y. Ng. End-to-end text recognition with convolutional neural networks. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 3304–3308, Nov 2012.

[45] N. M. Zaitoun and M. J. Aqel. Survey on image segmentation techniques. *Procedia Computer Science*, 65:797 – 806, 2015. International Conference on Communications, management, and Information technology (ICCMIT'2015).

[46] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 818–833, Cham, 2014. Springer International Publishing.

[47] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. *CoRR*, abs/1603.08511, 2016.