

Chapter 1

Introduction

1.1 Introduction

Since the dawn of time, roads have been the foundation of the Pacific transport industry. Before the development of sea, plane, and train travel, roads were used to move people and things from one location to another. Roads continue to be one of the most popular modes of transportation even in today's modern world. In India, 60% of all goods and 87% of all passengers are transported via roadways (Press Information Bureau Government of India, 2013). Furthermore, roads are important contributors to more inclusive economic growth. Because of the crucial role roads play in our everyday lives, knowledge of their locations is crucial. This knowledge improves efficiency in the selection of routes for moving people and things.

Modern applications like urban planning and vehicle routing systems regularly require updated road information. In the past, manually updating road network required a lot of effort and time. However, the flexibility of digital representation allows the use of road data by several Geographic Information Systems (GIS). Road maps updates are commonly obtained using two techniques: ground surveying and image-based road updating(Couloigner, 2004) .

Image-based road updating is based on feature extraction from remotely-sensed (RS) imagery, which has become greater significant recently due to the accessibility of satellite photos with a high spatial resolution (1-4 meters). Road extraction, which is the process of separating roadways from remotely sensed imagery, can be fully automated, semi-automated, or manual (Saravanan, 2016)(Wang N. Y., 2016)(M. Patel, 2022). While semi-automated extraction requires some human input to direct a group of automated processes, manual extraction involves a human operator drawing roadways from remotely sensed data. Finally, there is no need for human input in the completely automated process. The completely automated algorithms show intelligence in their capacity to reason about the composition and distribution of road networks in a way that is comparable to that of humans. This will be possible through

artificial intelligence, image processing, and deep learning-based algorithms for the automatic extraction of road networks from remotely sensed images(Wang N. Y., 2016)(Gao, 2018)(Wang L. Q., 2005)(Xiang, 2017). Moreover, it is a practical and cost-effective method for getting road network information quickly in emergencies.

The basis of convolutional neural networks (CNN) finding served by (Wiesel, 1968) .An advanced framework of CNN called LeNet-5 proposed by (Y. Lecun, November 1998) that is used to recognize handwritten digits. Training by back propagation (D. E. Rumelhart, 1986) algorithm helped LeNet-5 in recognizing visual patterns from raw images directly without using any separate feature engineering. But despite its many advantages at the time, CNN struggled to perform well in complicated tasks due to a lack of training data, poor algorithm, and insufficient computer power. But nowadays large labeled datasets, novel algorithms, and potent GPU machines become available easily.

Convolutional Neural Network is a particular kind of multilayer neural network with respect to deep learning (I. Goodfellow, 2016) architecture and it is particularly useful for spatial data (or CNN or ConvNet.). CNN's design was influenced by how live things perceive their environment. Although it gained popularity as a result of suggested method by(A. Krizhevsky, 2012) ,AlexNet's record-breaking performance in 2012 even though it was really started in 1980. After 2012, CNN picked up the pace and began to dominate several different computer visions, natural language processing, and related domains.

A basic convolutional neural network consists of a 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 fundamental component of a CNN. This layer seeks to learn input feature representations. In order to create various feature maps, the convolutional layer consists of a number of learnable convolution kernels or filters. Each feature map unit is linked to a receptive field in the previous layer. The new feature map is created by convolving the input with the kernels and the result is then processed with an element wise nonlinear activation function. Convolutional layer parameter sharing property minimizes the complexity of the model. A pooling or sub-sampling layer uses a small area of the convolutional output

as an input and down-samples it to create a single output. There are various sub-sampling methods, including maximum pooling, minimum pooling, average pooling, and others. Pooling makes the network translation-invariant and cuts down on the amount of parameters that must be computed. One or more FC layers, which are generally found in feed forward neural networks (FFNN), make up the final portion of CNN. The FC layer creates the final output of CNN using data from the final pooling or convolutional layer.

However, In the image classification, a CNN may be seen as combining two elements: the feature extraction part and the classification part. Convolutional and pooling layers both extract features. Consider a dog image as an example, many convolution layers at various levels identify traits like the dog's two eyes, long ears, four legs, etc.. for further in-depth recognition, the FC layers are added as a classifier on top of these features, and a probability is given that the input image is of a dog. In addition to layer design, a number of other factors, such as activation function, normalization method, loss function, regularization, optimization, and processing speed affect how well CNN performs.

Following AlexNet's success, CNN gained enormous popularity in three key areas: image classification(F. Sultana A. S., NOV,2018) , object detection (F. Sultana A. S., ,2019), and segmentation(Aqel, 2015) . In these areas, numerous CNN advance models have been presented over the course of several years. Image classification, object tracking, object detection, segmentation, human pose estimation, text detection, action recognition, scene labeling, speech and natural language processing and many more use CNN to achieve state-of-the-art performance. Even though the current CNN models are quite effective for a variety of applications, its fundamental workings remain a mystery. Therefore, additional research is needed to understand the core concepts behind CNNs. The chapter will help to gain a deeper grasp of CNN while also facilitating new developments in CNN-based architectures used for semantic segmentation, including U-Net, SegNet, FCN, DeepLab, and many others.

1.2 Concepts of Convolutional Neural Network

Artificial Neural Networks (ANNs) are a subset of convolutional neural networks (CNN), also known as ConvNet. In comparison to other networks with FC layers, it has a deep feed-forward architecture, which has extraordinary generalizing abilities. It can learn highly abstracted aspects of objects, particularly geographical data, and can identify them more effectively. A deep CNN model is made up of finite number of processing layers that can learn different input data features at different levels of abstraction. The deeper layers learn and extract the low level features, whereas the initiatory layers learn and extract the high level features. Figure 1.1 depicts the fundamental conceptual model of CNN, with succeeding sections describing the various types of layers.

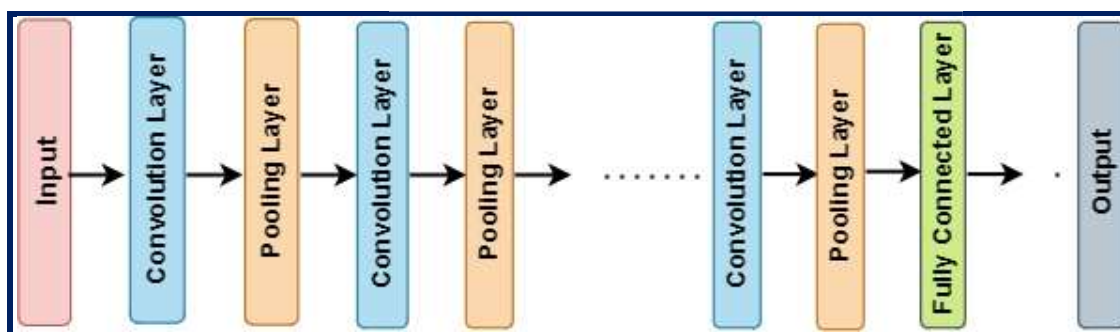


Figure 1.1: Convolutional Neural Network

CNN is widely used in deep learning based architecture due to the weight sharing feature of it reduce the trainable parameters of the model that avoid over fitting as well as improved generalization. Nowadays CNN has been emerged as a mechanism for achieving promising result in various deep learning based algorithms. Different component of the CNN is described as follows.

1.3 Components of CNN

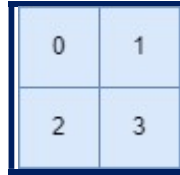
As was previously said, a CNN is made up of several building blocks sometimes referred to as layers of the architecture. In this subsection covers detail about a few of these building elements that fit into the CNN architecture.

1.3.1 Convolutional Layer

The most crucial element of any CNN architecture is the convolutional layer. It has a number of convolutional kernels, also known as filters, which when combined with the input image's N-dimensional metrics yield a feature map as the result.

1.3.1.1 Kernel

A kernel can be thought of as a grid of discrete values or numbers, each of which represents the kernel's weight. All of a kernel's weights are randomly allocated at the beginning of a CNN model's training process. After that, the weights are adjusted and the kernel learnt to extract useful features with each training period as shown in Figure 1.2.



0	1
2	3

Figure 1.2 : Example of a 2x2 Kernel

1.3.1.2 Convolution Operation

In this one or more filter or kernel will be shifted on to whole image and evaluate the convolution operation between input image and filters. If the size of input image the at the q^{th} layer, $(L_q * B_q * d_q)$ and dimension of filter in q^{th} layer is $(F_q * F_q * d_q)$ then, the resultant matrix size is,

$$L_{q+1}(Height) = L_q - F_q + 1 \quad (1.1)$$

$$B_{q+1}(Width) = L_q - F_q + 1 \quad (1.2)$$

$$d_q(Depth) = No. of Filters \quad (1.3)$$

Figure 1.3 shows the convolution operation between the input image and the kernel. It also shows the resultant matrix after the convolution operation labeled as output.

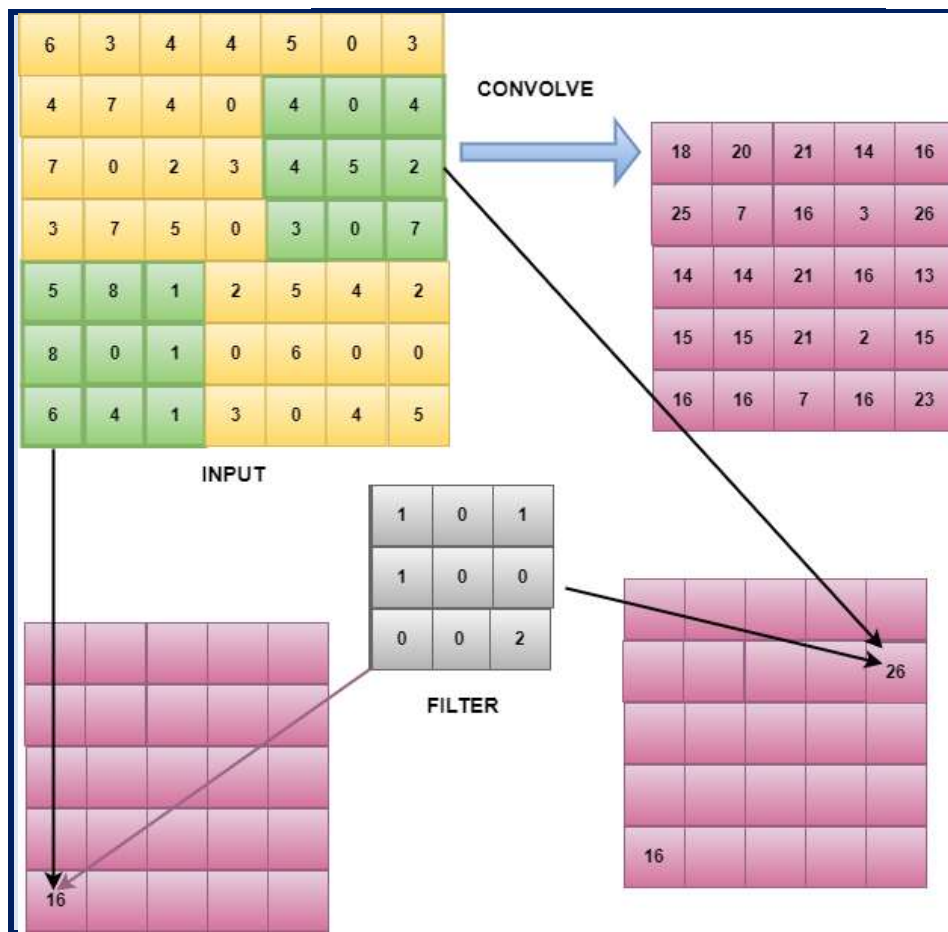


Figure1.3: Understanding of Convolution Operation

1.3.1.3 Padding

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. Three padding types are described as follows.

Same padding

In this type of padding output is same size as input image.

To keep the feature map's spatial footprint, one must add $\frac{f_q-1}{2}$ pixels all around its edges.

$$[(n + 2p) \times (n + 2p)]_{image} * [(F \times F)]_{filter} \rightarrow image\ size\ of\ [n \times n]_{image} \quad (1.4)$$

$$\text{Where, } p = \frac{F_q-1}{2}$$

Valid padding

padding is not used in this case.

$$[n \times n]_{image} * [F \times F]_{filter} \rightarrow image\ size\ of\ [(n - F + 1) \times (n - F + 1)]_{image} \quad (1.5)$$

Full Padding

Filters visit each pixel the same number of times, increasing the output image size.

1.3.1.4 Stride

Stride means taken the step size along the horizontal or vertical position of the image. In Figure 1.3 stride value is consider to 1 for the kernel. But for other stride value (rather than 1) in convolution operation is resulted in lower-dimensional feature map.

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

$$L = \lfloor \frac{n-F+2p}{s} + 1 \rfloor \quad (1.6)$$

$$B = \lfloor \frac{n-F+2p}{s} + 1 \rfloor \quad (1.7)$$

Where, input image size of $(n \times n)$, filter dimension of $(F \times F)$, padding is p and stride is s .

1.3.2 Pooling layer

The Features maps (produced after convolution operations) are sub-sampled using the pooling layers, which means that it takes the larger size feature maps and reduces them to smaller size feature maps. The most significant features (or information) in each pool step are always preserved when the feature maps are shrunk. Similar to the convolution operation, the pooling operation is carried out by specifying the pooled recognize and the operation stride. Different pooling techniques, including max pooling, min pooling, average pooling, gated pooling and tree pooling are used in various pooling layers. The most well liked and frequently employed pool a technique is max pooling. The Figure 1.4 describe the concepts of the pooling layer.

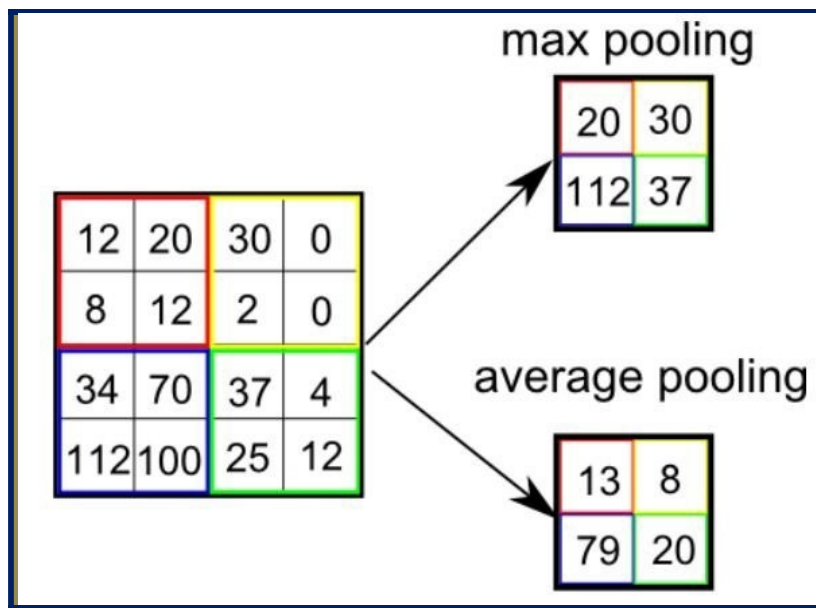


Figure 1.4: Concept of the Pooling Layer (tutorialpoint)

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

$$L' = \lfloor \frac{L-f}{s} \rfloor \quad (1.8)$$

$$B' = \lfloor \frac{B-f}{s} \rfloor \quad (1.9)$$

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

1.3.3 Activation Function

In CNN architecture, after convolution and FC layers, non-linear activation functions are used. These layers manage to map the inputs to the output nonlinearly. The most commonly used activation functions in deep neural networks are described below. Various activation functions is represented in the Figure 1.5.

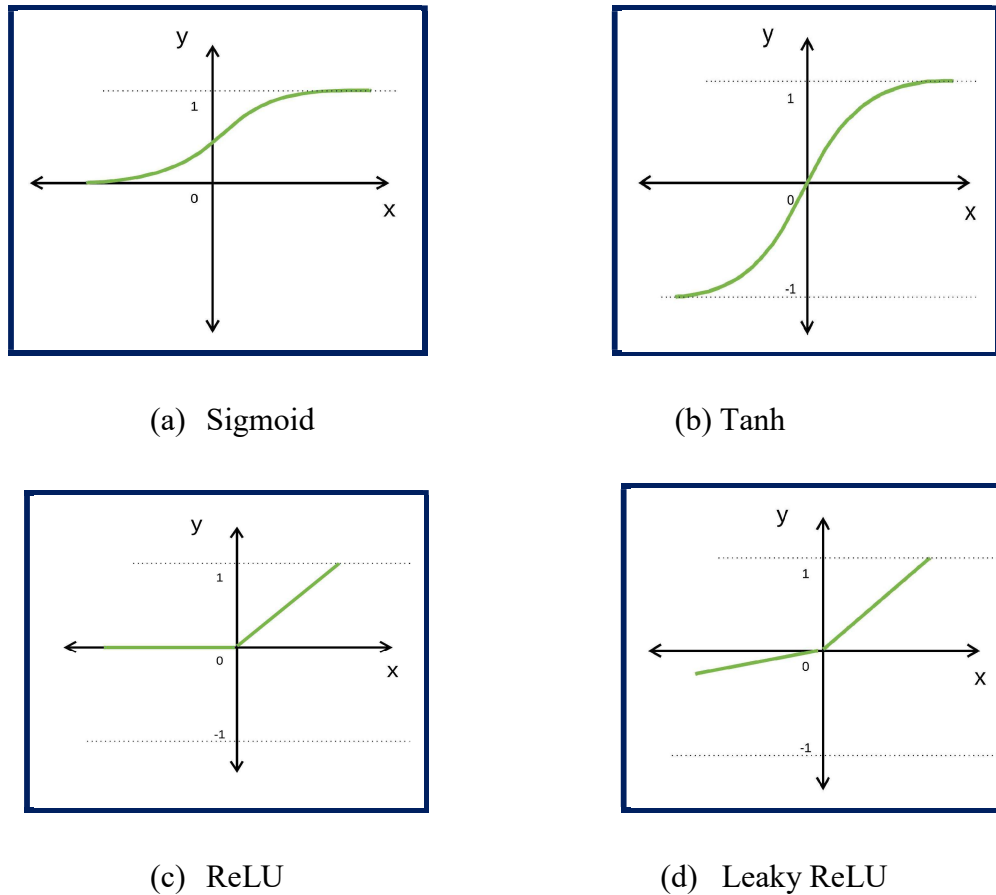


Figure 1.5: Activation Functions (Anirudha Ghosh, January,2020)

1.3.3.1 Sigmoid

The input for the sigmoid activation function is a real number, and the output is bound to the range $[0, 1]$. The sigmoid function's curve has an "S" shape.

The sigmoid is represented mathematically by:

$$f(x)_{sigmoid} = \frac{1}{(1+e^{-x})} \quad (1.10)$$

1.3.3.2 Tanh

The Tanh activation function is used to confine the input real number valued to the interval $[-1, 1]$.

Tanh is represented mathematically as

$$f(x)_{tanh} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (1.11)$$

1.3.3.3 ReLu

The most frequently utilized activation function in the convolutional neural network is the rectifier linear unit (ReLU) (Hidenori Ide, May ,2017). It is used to change every value from the input to a positive number. ReLu has the benefit of requiring a much lower computational load than other methods. ReLu is mathematically represented as follows,

$$f(x)_{ReLU} = \max(0, x) \quad (1.12)$$

1.3.3.4 Leaky ReLu

In contrast to Relu, a Leaky ReLu activation function down-scales negative inputs rather than completely ignoring them. Dying ReLu is solved by a Leaky ReLu.

Leaky ReLu is represented mathematically as follows

$$f(x)_{LeakyReLU} = \begin{cases} x, & \text{if } x > 0 \\ mx, & \text{if } x \leq 0 \end{cases} \quad (1.13)$$

where, m is a constant, called leak factor

1.3.4 Fully Connected (FC) Layer

The architecture of the FC layer as shown in the Figure 1.6.

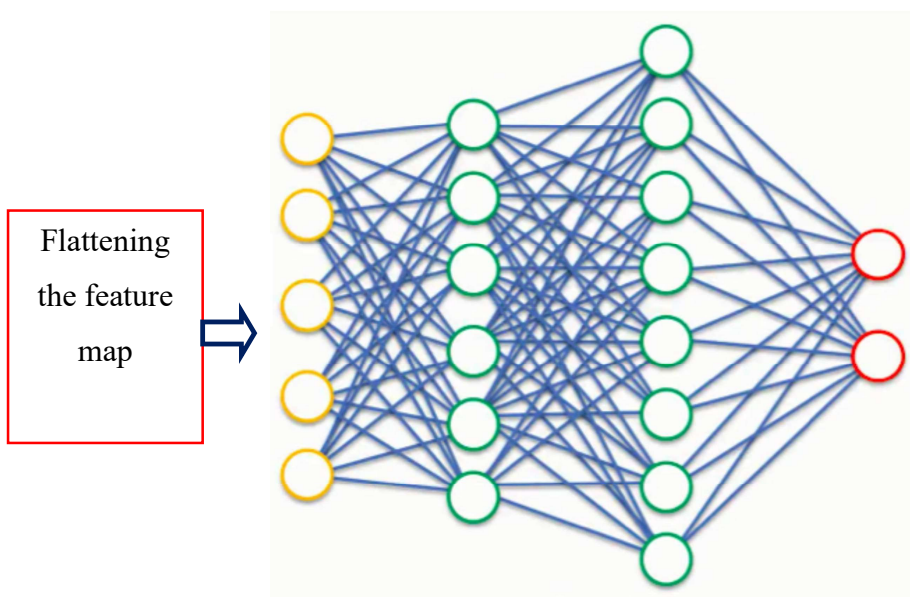


Figure 1.6: The Architecture of Fully Connected Layers (tutorialpoint)

The Fully-Connected Layers are a type of feed-forward artificial neural network (ANN) and follow the principle of traditional multi-layer perceptron neural network. The FC layer takes input from the convolutional or pooling layer, which is in the form of a set of metrics or feature maps and those metrics are fed into the FC layer to generate the output.

1.3.5 Loss Function

The last layer of CNN based architecture is the output layer. In this layer, calculation of the prediction error generated by the CNN model over the training samples is done by some Loss Function. Some of the most used loss functions are softmax loss and squared error loss.

1.3.5.1 Softmax Loss or Cross Entropy Loss Function

Generally it is used with softmax activations at the output layer to generate the output within a probability distribution; It is expressed by the following equation.

$$L_{CE} = -\frac{1}{n} \sum_{i=1}^n \sum_{c=1}^C y_i^c \log(p_i^c) \quad (1.14)$$

Where, probabilistic output p_i^c , C= no of the class, and target segmentation map y_i^c

1.3.5.2 Mean Squared Error

It also known as the Euclidean loss and it is widely used in regression problems.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y})^2 \quad (1.15)$$

Where, y_i is the vector of target values
 \hat{y} is the vector of the predicted values
 n is the number of data

1.4 Neural Network

The neural networks are inspired from the natural neural network of human nervous system as shown in Figure 1.7. The inventor of the first neurocomputer, Dr. Robert Hecht-Nielsen, defines a neural network as – "A computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

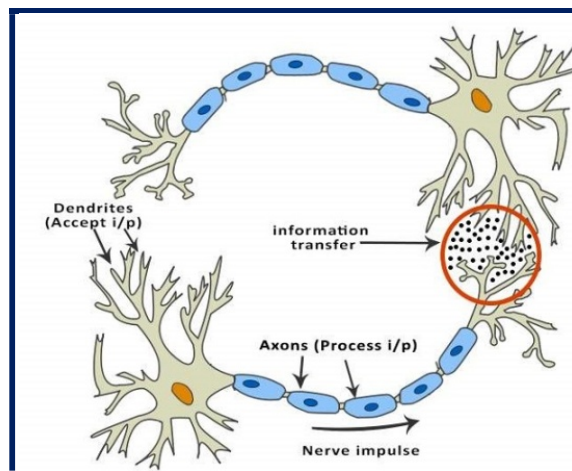


Figure 1.7: Human's Brain Working (tutorialpoint)

1.4.1 Basic Structure of ANNs

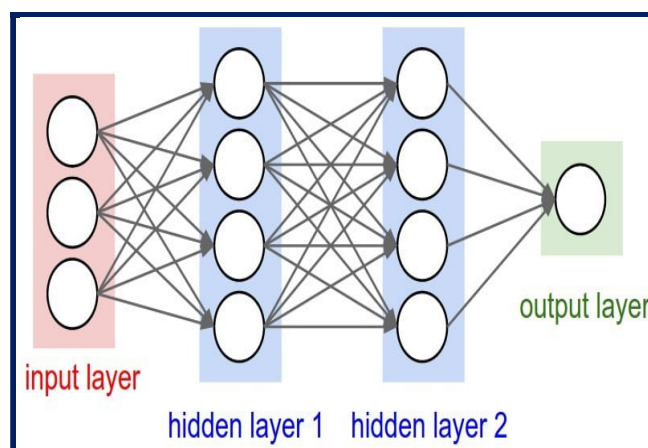


Figure 1.8: Simple ANN (Anirudha Ghosh, January,2020)

ANNs are composed of multiple 'nodes', which imitate nerve cells in the brain as shown in the Figure 1.8. They can take input data and perform simple operations on it. The result of these operations is passed to other neurons. The output at each node is called its activation or node value. Each link is associated with a weight. The following illustration shows a simple ANN in diagrammatic form in Figure 1.9.

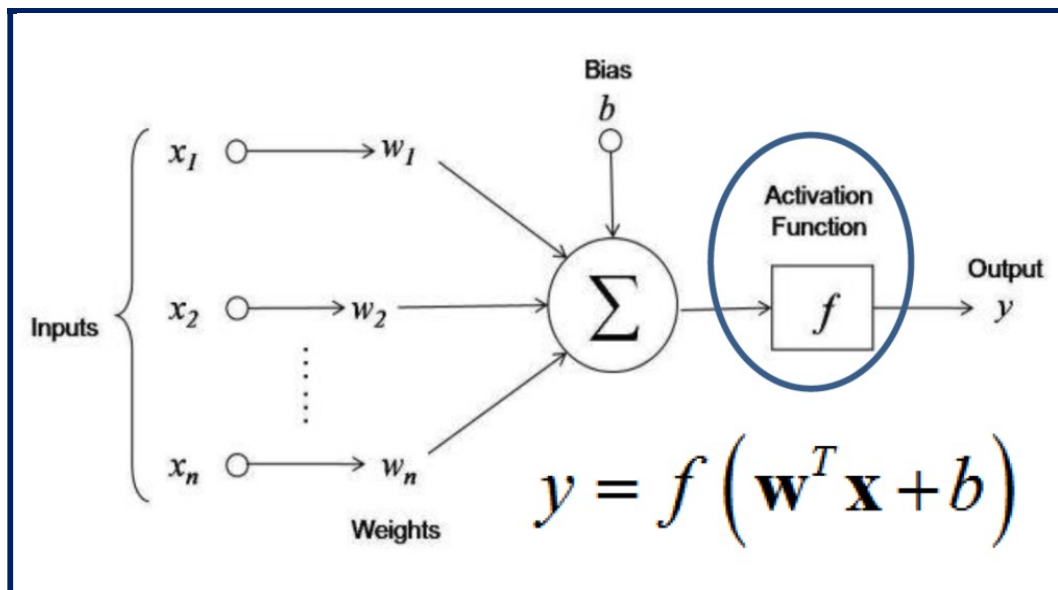


Figure 1.9: Neuron Model (tutorialpoint)

An artificial neural network computation has two phases: forward pass and backward propagation. In a forward pass values for every neuron are calculated sequentially layer by layer starting from the input layer. By numbering the layers in increasing order starting from the input layer the neuron values of a fully connected FFNN (Feed Forward Neural Network) are calculated according to

$$\hat{y} = f(W^T X_i + b) \quad (1.16)$$

$$E = \frac{1}{n} \sum_{i=1}^n [f(W^T X_i) - y_i]^2 \quad (1.17)$$

Backward propagation of error is also known as back-propagation in neural networks. It is a technique for adjusting a neural network's weights based on the error rate recorded during the preceding epoch. By increasing the model's generalization, it can lower error rates and make it more reliable. Figure 1.10 shows the back propagation in the neural network.

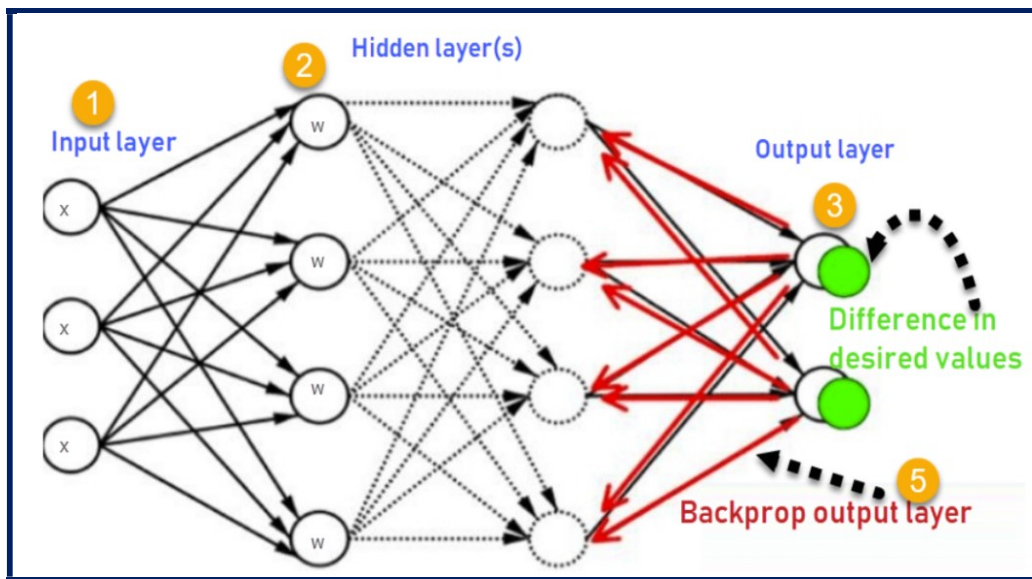


Figure 1.10: Back propagation in ANN(Anirudha Ghosh, January,2020)

Following are five steps for back-propagation of an algorithm.

- X enters as input.
Real weights W are used to model the input. Typically, weights are chosen at random.
- Determine each neuron's output from the input layer through the hidden layers to the output layer.
- Make a calculation of the output error.
- Go back from the output layer to the hidden layer to adjust the weight values such that the error is reduced.

The back propagation algorithm in a neural network computes the gradient of the loss function for a single weight by the chain rule. It efficiently computes one layer at a time.

Taking partial derivatives with respect to the W

$$\nabla_w E = \hat{y}_i(1 - \hat{y}_i)(\hat{y}_i - y_i)X_i \quad (1.18)$$

So weight updating rule for single output layer is ,

$$w_{new} = W_{old} - \eta \hat{y}_i(1 - \hat{y}_i)(\hat{y}_i - y_i)X_i \quad (1.19)$$

If output node is considered as σ_j then squared error equation becomes

$$E = \frac{1}{2} \sum_{j=1}^{M_k} (o_j^k - t_j)^2 \quad (1.20)$$

Considered as k nodes in the network and multiple output layer then weight updating for i^{th} neuron to j^{th} neuron by using the chain rule for,

$$\frac{\partial E}{\partial W_{ij}^k} = \frac{\partial E}{\partial o_j^k} \cdot \frac{\partial o_j^k}{\partial \theta_j^k} \cdot \frac{\partial \theta_j^k}{\partial W_{ij}^k} \quad (1.21)$$

$$\text{Where, } o_j^k = 1/(1 + e^{-\theta_j^k}) \text{ and } \theta_j^k = \sum_{i=1}^{M_{(k-1)}} W_{ij}^k X_i^{k-1}$$

$$= (o_j^k - t_j) o_j^k (1 - o_j^k) o_i^{k-1} \quad (1.22)$$

$$\text{Let } \delta_j^k = (o_j^k - t_j) o_j^k (1 - o_j^k) \quad (1.23)$$

So weight updating rule at the output layer is,

$$W_{ijnew} = W_{ijold} - \eta \delta_j^k o_i^{k-1} \quad (1.24)$$

Considered weight updating for hidden layer between $(k - 2)$ and $(k - 1)$ layer for q^{th} neuron to i^{th} neuron W_{qi}^{k-1}

$$\frac{\partial E}{\partial W_{qi}^{k-1}} = \frac{\partial E}{\partial o_i^{k-1}} \cdot \frac{\partial o_i^{k-1}}{\partial W_{qi}^{k-1}} \quad (1.25)$$

$$= \frac{\partial E}{\partial o_i^{k-1}} \cdot \frac{\partial o_i^{k-1}}{\partial \theta_1^{k-1}} \cdot \frac{\partial \theta_1^{k-1}}{\partial W_{qi}^{k-1}} \quad (1.26)$$

$$= \frac{\partial E}{\partial o_i^{k-1}} o_i^{k-1} (1 - o_i^{k-1}) o_p^{k-2} \quad (1.27)$$

Now, using chain rule

$$\frac{\partial E}{\partial o_i^{k-1}} = \frac{\partial E}{\partial o_j^k} \cdot \frac{\partial o_j^k}{\partial \theta_j^k} \cdot \frac{\partial \theta_j^k}{\partial W_{ij}^k} \quad (1.28)$$

$$= \sum_{j=1}^{M_k} W_{ij}^k (o_j^k - t_j) o_j^k (1 - o_j^k) \quad (1.29)$$

Put (1.23) into (1.29)

$$= \sum_{j=1}^{M_k} W_{ij}^k \delta_j^k \quad (1.30)$$

Put (1.30) into (1.27),

$$\frac{\partial E}{\partial W_{qi}^{k-1}} = o_i^{k-1}(1 - o_i^{k-1})o_p^{k-2} \sum_{j=1}^{M_k} W_{ij}^k \delta_j^k \quad (1.31)$$

Let

$$\delta_i^{k-1} = o_i^{k-1}(1 - o_i^{k-1}) \sum_{j=1}^{M_k} W_{ij}^k \delta_j^k \quad (1.32)$$

Weight updating rule between hidden layer (k-1) and (k-2)

$$W_{pi}^{k-1}{}_{new} = W_{pi}^{k-1}{}_{old} - \eta \delta_i^{k-1} o_p^{k-2} \quad (1.33)$$

For any hidden layer weight W_{ij}^k , weight updating rule is

$$W_{ij}^k{}_{new} = W_{ij}^k{}_{old} - \eta \delta_j^k o_p^{k-1} \quad (1.34)$$

Neural network weight updating at the output layer get by eq. (1.24) and at any hidden layer it is done by eq. (1.34).

1.5 Optimizer Selection

The learning process includes two major things, first one is the selection of the learning algorithm (Optimizer) and the next one is to use several improvements (such as momentum, Adagrad, Adam) to that learning algorithm in order to improve the result.

The learning rate η is a parameter that regulates how quickly model learns features. There exist a number of variants of the gradient-based learning algorithm; out of them the most widely used are Batch Gradient Descent, Mini Batch Gradient Descent and Stochastic Gradient Descent. Another widely used learning algorithm or optimization technique is Adam optimization, which uses second-order derivative represented by Hessian Matrix. These are described as follows.

1.5.1 Batch Gradient Descent

Suggested by (Ruder, 2016) the parameters of the network are updated only once after passing the whole training dataset through the network. It calculates the gradient on the entire training set and then updates the parameters. It produces more stable gradient and also converges faster for small-sized datasets. It needs fewer resources but it's not suitable for larger training dataset.

1.5.2 Stochastic Gradient Descent (SGD)

It update the parameters for each training sample separately proposed by (Bottou, 2010). It converges much faster for large training dataset and it is also memory efficient. But it updates weight parameter frequently that makes noisy steps towards the solution and convergence behavior very unstable.

1.5.3 Mini Batch Gradient Descent

It divides the training samples into a number of mini-batches in non-overlapping manner. The parameters updating is computing by the gradients of each mini-batch. It is more memory efficient, more computationally efficient and also has a stable convergence.

1.5.4 Momentum

It improves 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 the previous techniques is, it easily stuck in a local minima instead of global minimum. It can easily expressed in mathematically as

$$\Delta W_{ij}^e = \eta \cdot \frac{\partial E}{\partial W_{ij}} + \lambda \Delta W_{ij}^{e-1} \quad (1.35)$$

Where, ΔW_{ij}^k is weight increment in e^{th} training epochs, η is learning rate, ΔW_{ij}^{e-1} is weight increment in previous $(e-1)^{\text{th}}$ training epoch, and λ is momentum factor $\in [0,1]$.

1.5.5 Adaptive Moment Estimation (Adam)

Adaptive Moment Estimation (Adam (Ba., 2014)) is another optimizer, which calculates both exponential moving average of the gradient and as well as the exponential moving average of the squared gradients. So the formulas for those estimators are as:

$$W_{ij}^e = \Delta W_{ij}^{e-1} - \frac{\eta}{[\sqrt{E[\delta^2]^e} + \epsilon]} \cdot [E[\delta]^e] \quad (1.36)$$

Where, $E[\delta]^e$ is the estimate of the first moment, $E[\delta^2]^e$ is the estimate of the second moment of the gradient and ϵ is constant.

1.6 Semantic Segmentation

With advancement in the CNN architecture, it plays vital role to achieve excellent result in the image classification, image detection and semantic segmentation. In image classification, input image contains a single object and then classify the image into one of the pre-selected target classes by using various CNN models such as LeNet-5(Y. Lecun, November 1998), AlexNet(A. Krizhevsky, 2012), VGGNet(Zisserman, 2014), GoogLeNet(C Szegedy, 2014) and many more. In the image detection contains more than one object and try to identified targeted image from those input image with help of R-CNN(R. B. Girshick, 2013), SPP-Net(K. He, 2014), YOLO(J. Redmon, June 2016) and many more. However image segmentation is the process of associating each pixel of an image with a class label.. After the record-breaking performance of AlexNet in 2012, many state-of-the-art models of semantic segmentation are proposed by various researchers as described below.

1.6.1 Fully Convolutional Network (FCN)

The fully connected layer of AlexNet, VggNet, and GoogLeNet (all three networks are pretrained on ILSVRC dataset) has been replaced by 1x1 convolutional layers by the inventor of FCN(E. Shelhamer, 2017) to make them dense FCN. The final layer consists of a 1 X1 convolution with a channel dimension of twenty one (including background) predict the scores for each PASCAL VOC[20] class,. The authors include bilinear interpolation and skip connection to produce fine-grained segmentation. Figure 1.11 depicts the end-to-end model of FCN.

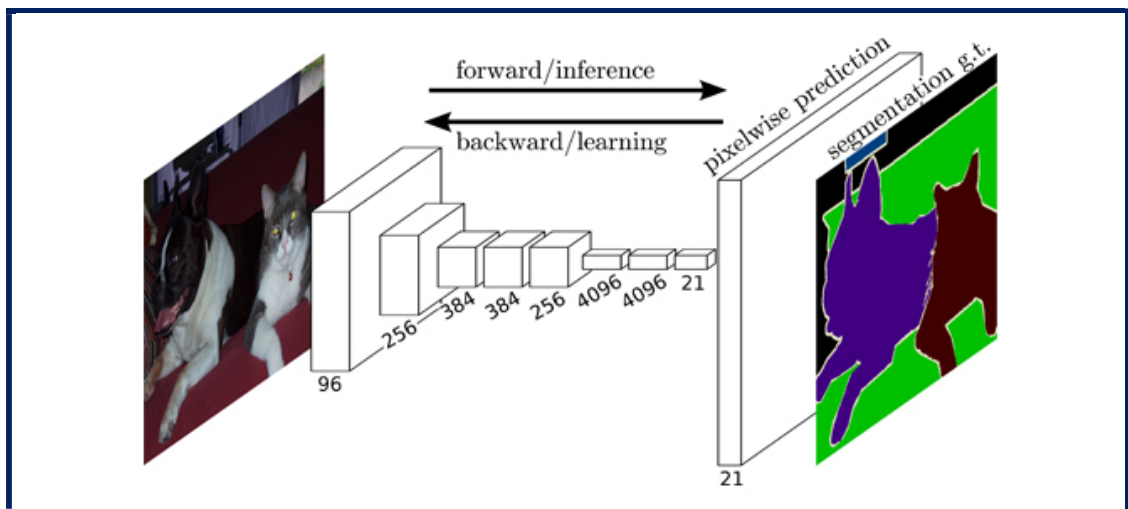


Figure 1.11: End-to-End Model of FCN (E. Shelhamer, 2017)

1.6.2 SegNet

For semantic segmentation, the author has employed encoder-decoder architecture [21]. 13 layers of the VGG16 network are used in the encoder and decoder parts. A pixel-by-pixel classification layer is the final layer. Figure 1.12 displays the SegNet end-to-end model.

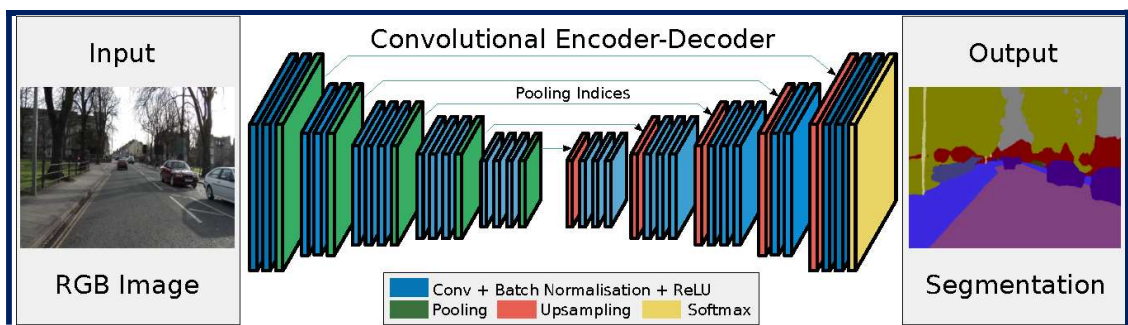


Figure 1.12: End-to-end model of SegNet (A. Krizhevsky, 2012)

1.6.3 U-Net

This structure (Ronneberger O, 2015) has a u-shaped route that alternates between expanding and contracting as present in Figure 1.13. The contracting path has two 3X3 convolutions, ReLu, and 2X2 max-pooling in each step. Expansive path, on the other hand, consists of 3X3 convolutions, 2X2 up-convolutions, and ReLu. The

Develop an Automatic Road Network Extraction System from Remote Sensing Images

feature map is concatenated with the cropped feature map from contracting path from the relevant layer in between up-convolution and convolution in expanding path.

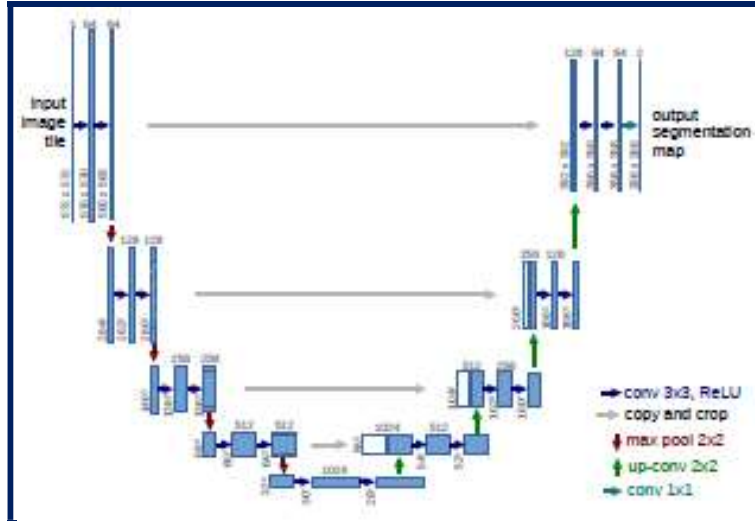


Figure 1.13: End-to-end model of U-Net (Ronneberger O, 2015)