Traffic Sign Recognition

A Project Report submitted
by

shuvankar Naskar
Roll No. 21MA60R35

Under the supervision of
Dr. Prof. debjani chakraborty

in Partial Fulfillment for the award
of the degree of

Master of Technology

in

Computer Science and Data Processing

Department of Mathematics
Indian Institute of Technology Kharagpur
Kharagpur - 721302 , India
November 16, 2022

Department of Mathematics
Indian Institute of Technology,
Kharagpur
India - 721302

CERTIFICATE

This is to certify that we have examined the project report entitled **Traffic Sign Recognition**, submitted by **Shuvankar Naskar**(Roll Number: *21MA60R35*) a postgraduate student of **Department of Mathematics** in partial fulfillment for the award of degree of M.Tech Computer Science and Data Processing. We hereby accord our approval of it as a study carried out and presented in a manner required for its acceptance in partial fulfillment for the Post Graduate Degree for which it has been submitted. The project report has fulfilled all the requirements as per the regulations of the Institute and has reached the standard needed for submission.

Supervisor

Department of Mathematics
Indian Institute of Technology, Kharagpur

Place: Kharagpur
Date:

ABSTRACT

Numerous practical applications, including autonomous driving, traffic surveillance, driver assistance, road network maintenance, and traffic scene analysis, depend on the ability to recognise traffic signs.
In particular, its variant with 35K images of the widely used data set GTSRB (German Traffic Sign Recognition Benchmark) is taken into consideration. Convolutional neural networks built using deep learning are modified for classification problems; the ResNet 55 architecture is taken into account for the training data set; and the model is trained for 45 epochs without the use of transfer learning. Test accuracy is attained at 94.82 percent. A few changes are suggested to improve test accuracy.

Contents

Chapter 1

Introduction

A vehicle can now recognise traffic signs placed on the road, such as Stop or No Entry, thanks to a technology called Traffic Sign Recognition (TSR). This is a characteristic of the ADAS (Advanced driver-assistance systems) system as a whole.

There are a lot of truck and taxi drivers in India, some of whom are quite senior. The ability to see and one's vision deteriorate as one gets older. This system assists in the detection of traffic signs in such situations because driving at night and in bad weather are problems in addition to declining eyesight.

The Government of Jharkhand's Transport Department lists failure to comprehend traffic signs as one of the causes of accidents, so computer-aided recognition of traffic signs can be done more accurately than human recognition of it, preventing some road accidents.

Deep learning-based convolutional neural networks are being looked at as a possible replacement for hand featuring and machine learning algorithms, which excel on smaller data sets but dreadfully underperform on larger ones. With the aid of supplied hyper-parameters, these networks are capable of learning features on their own. Convolutional neural networks are frequently used in applications for computer vision; the chapters that follow will examine these networks.

In appendices analysis of our model is given and suggestion for improvement along with depiction of model architecture incorporated with improvements.

Chapter 2

Convolutional Neural Networks

A **Convolutional Neural Network (CNN)** is a Deep Learning algorithm that can take in an input image, give various aspects and objects in the image importance (learnable weights and biases), and be able to distinguish between them. Comparatively speaking, a CNN requires much less pre-processing than other classification algorithms.
Convolution Neural Network architecture as described in figure 2.1 comprises of two prime modules:

- **Feature Extraction:** In this module, the network performs a number of convolution, ReLU, and pooling operations to extract features that are typically artefacts in an image, such as edges and blobs, as well as some unique features, such as the ear, nose, and so forth. This module includes the following layers: convolutional, reLU, and pooling.
- **Classification:** Fully connected layers use extracted features to perform classification tasks in this module. Layer that is fully connected is included in this module.

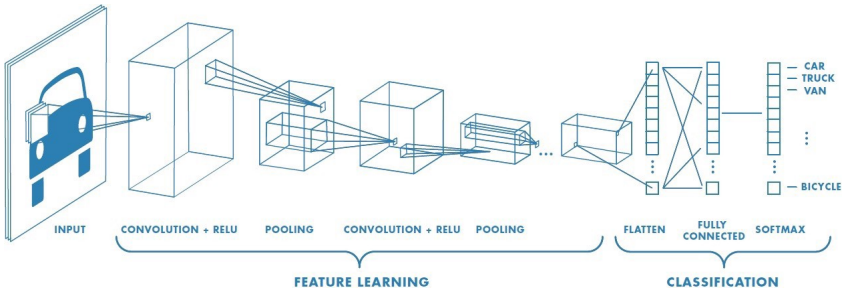


Figure 2.1: Convolutional Neural Network Architecture

2.1 Convolutional Layer

Convolution is a mathematical operation on two functions (X and W) that produces a third function (X*W) that expresses how the shape of one is modified by the other.

$$Y(t) = \int X(a)W(t-a)\,da \tag{2.1}$$

For discrete case equation can be defined as,

$$Y(t) = \sum_{a=-\infty}^{\infty} X(a)W(t-a) \tag{2.2}$$

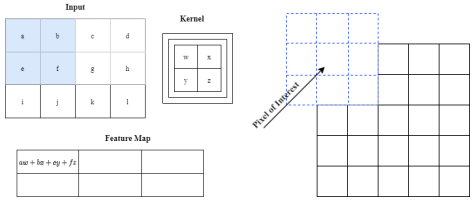
In convolutional network terminology, the function X)is often referred to as the **input**, and the function W) as the **kernel**. The output is sometimes referred to as the **feature map**. Convolutional layer consists of kernels for conducting convolution operation.

2.1.1 Convolution Operation for Images

Convolution operation for a gray scale image ($2D$ matrix) as an input with $2D$ matrices as filters for detecting artifacts is defined as follows, consider image to be of $N1 \times N2$ dimension and filter to be of $K_1 \times K_1$ dimension then resultant feature map Y in the form of matrix can be expressed as:

$$Y[i,j] = \sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} W[a,b]X[i-a,j-b] \tag{2.3}$$

Illustration of convolution operation is defined in figure 2.2(a). But in practice convolution formula is defined using succeeding pixels. It would be $Y[i,j] = \sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} W[a,b]X[i+a,j+b]$



(a) (b)

Figure 2.2: Convolution operations on image

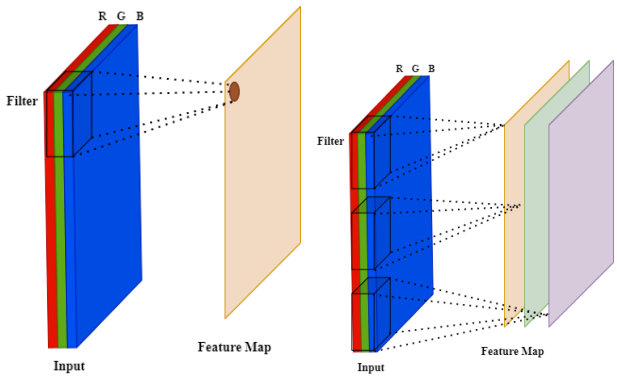
When convolution is defined by allowing kernel to be centered on pixel of interest same formula with $K_1 \times K_2$ filter W with image X can be expressed as:

$$Y[i,j] = \sum_{a=\lfloor -\frac{K_1}{2} \rfloor}^{\lfloor \frac{K_1}{2} \rfloor} \sum_{b=\lfloor -\frac{K_2}{2} \rfloor}^{\lfloor \frac{K_2}{2} \rfloor} W[\frac{K_1}{2} + a, \frac{K_2}{2} + b] X[i-a, j-b] \tag{2.4}$$

The idea of above convolution is depicted in figure 2.2(b).

Convolution of Coloured Image with 3D Filter

Input image is represented in three matrices of the same dimensions, one for each colour, as coloured images are typically represented in RGB space. The filter is also three dimensional with a depth of three, so in this case the filter will slide over the input due to the same depth. The output is $2D$ feature map in the form of matrix as depicted in figure 2.3(a).



(a) (b)

Figure 2.3: Convolution of coloured image and Convolution with multiple filters

Consider input image X which is colourful and $K \times K \times 3$ filter W then resultant image Y in the form of matrix can be expressed as:

$$Y[i,j] = \sum_{m=1}^{3} \sum_{a=0}^{K-1} \sum_{b=0}^{K-1} W_m[a,b] X_m[i-a, j-b] \tag{2.5}$$

Where X_m and W_m where $1 \leq m \leq 3$ are matrices corresponding to red, blue and green colours.

In case of D such filters then we have feature map with D matrices or depth of feature map is D as shown in figure 2.3(b), depth of input and kernel filter is same (multiple filters are denoted by $K \times K \times D$ where D is number of filters, its depth is equal to depth of input is presumed).

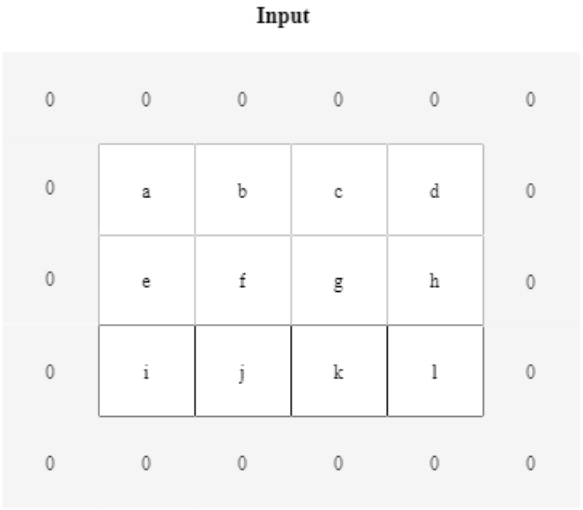
2.1.2 Stride and Padding

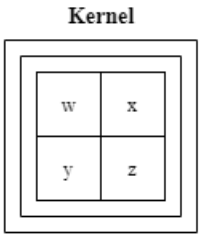
If input image is of dimension $W_1 \times H_1$ and filter $K \times K$ then dimension of feature map is $W_2 \times H_2$ where,

$$\begin{aligned} W_2 &= W_1 - F + 1 \\ H_2 &= H_1 - F + 1 \end{aligned} \tag{2.6}$$

Padding

Because convolution with a kernel reduces the size of the input image, pixels in the middle of the image are used more frequently than those on the corners and edges. As a result, the information on an image's border is lost. So layers of zeroes are added to the input, which is known as padding, in order to preserve this information while also maintaining the size of the input image to some extent as shown in figure 2.4





Feature Map

aw	$ay + bz$	$by + cz$	$cy + dz$	dy
$ax + ez$	$aw + bx + ey + fz$	$bw + cx + fy + gz$	$cw + dx + gy + hz$	$dw + hy$
$ex + iz$	$ew + fx + iy + jz$	$fw + gx + jy + kz$	$gw + hx + ky + lz$	$hw + ly$
ix	$iw + jx$	$jw + kx$	$kw + lx$	lw

Figure 2.4: Padding on image

If input image is of dimension W1 × H1 and filter K × K and adding P layers to input then dimension of feature map is W2 × H2 where

$$W_2 = W_1 - F + 2P + 1$$
$$H_2 = H_1 - F + 2P + 1 \tag{2.7}$$

Stride

It is the number of pixels shifts by filter over the input matrix while applying convolution operation. It is depicted in figure 2.5

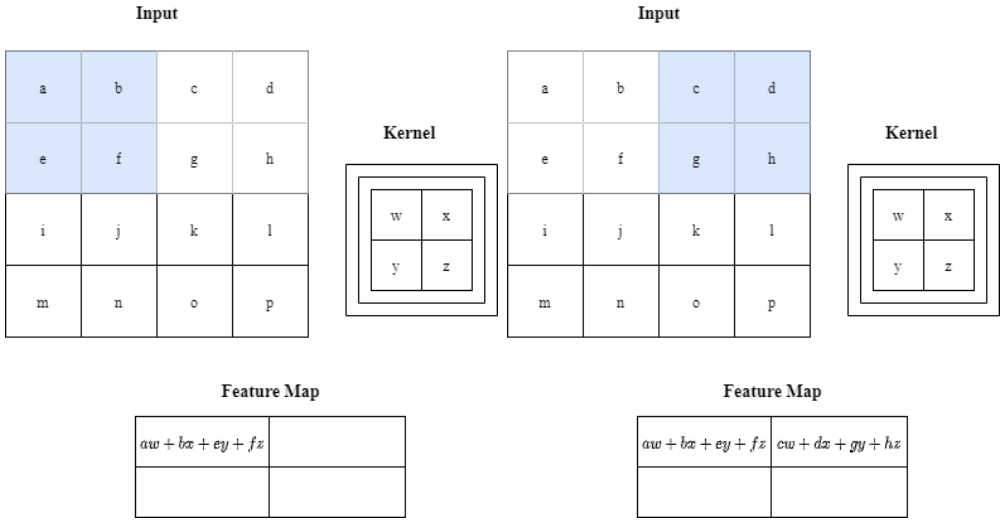


Figure 2.5: Stride in convolution

For input image of height and width H_1 and W_1 respectively with padding P convolved with $F \times F$ filter with stride S then width and height of feature map W_2 and H_2 can given by:

$$W_2 = \left\lfloor \frac{W_1 - F + 2P}{S} + 1 \right\rfloor$$
$$H_2 = \left\lfloor \frac{H_1 - F + 2P}{S} + 1 \right\rfloor \tag{2.8}$$

Final representation of convolution operation with padding and stride is rendered in figure 2.6

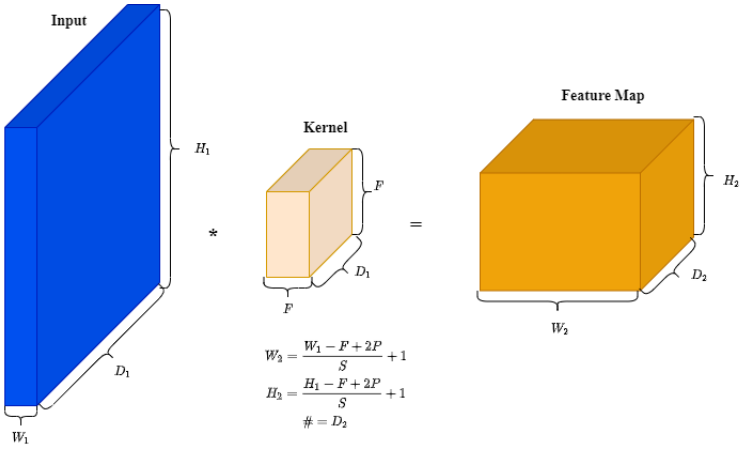
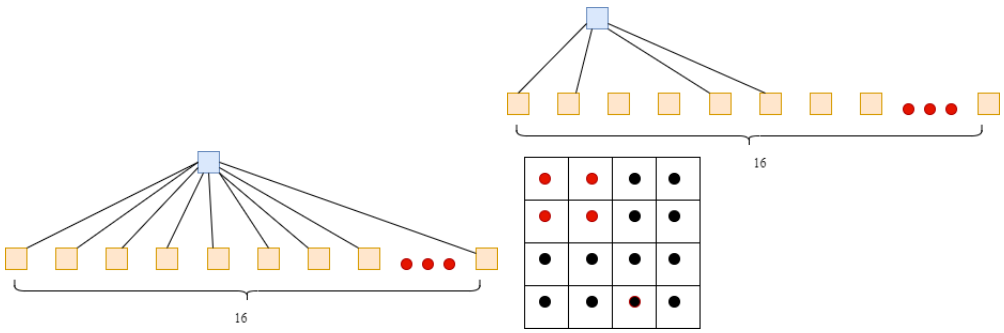


Figure 2.6: Output on convolution with filter

2.1.3 Characteristics of Convolutional Layer

sparse connectivity is possible to achieve by making the kernel smaller than the input. Although the input image may have thousands or even millions of pixels, small, significant features like edges can be found using kernels that only take up a few tens or hundreds of pixels. This implies that fewer parameters must be stored, which lowers the model's memory needs and boosts its statistical effectiveness. Convolutional neural networks have sparser connectivity than artificial neural networks, as shown in Figure 2.7.



(a) (b)

Figure 2.7: Sparse connectivity in CNN's

Parameter sharing is the practise of applying the same parameter to multiple functions in a model. Each component of the weight matrix is used exactly once when calculating the output of a layer in a conventional neural network. It is multiplied by one input element before being left alone.

It is known as **translation-invariance**.when a filter reacts to an object or artefact in an image the same way regardless of where it is located in the image due to parameter sharing. To capture various artefacts, one can employ a variety of kernels, but each one must be designed to respond to every aspect of the image, as shown in figure. 2.8

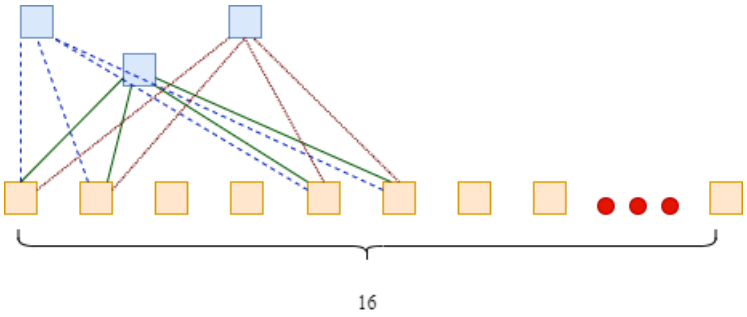


Figure 2.8: Different kernels on input image

2.2 ReLU Layer

ReLU is an acronym of **Rectified Linear Unit**. ReLU layer is non parametric, in this layer ReLU function shown in figure 2.9 is applied to input given to it. ReLU layer comes after convolution layer, feature maps generated from convolution layer are operated by ReLU function, it dosen't change dimensions of input feature map as shown in figure 2.10.

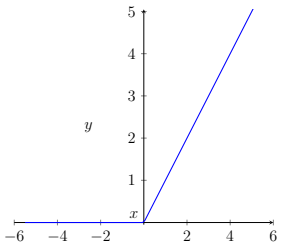


Figure 2.9: ReLU

$$\phi(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

Consider Y to be feature map generated from convolution layer then after applying ReLU activation with bias b one can get feature map Y' of same

dimensions

$$\begin{aligned} Y'[i,j] = 0 \quad & Y[i,j] + b < 0 \\ Y'[i,j] = Y[i,j] \quad & Y[i,j] + b \geq 0 \end{aligned} \tag{2.9}$$

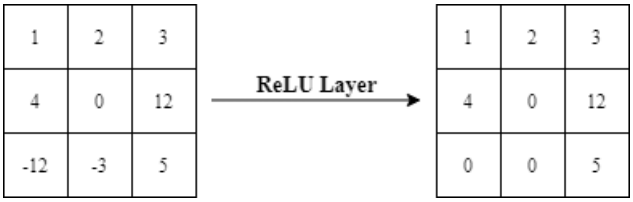


Figure 2.10: ReLU on feature map with zero bias

Advantages of ReLU function

- Convolutional neural network nonlinearity is increased by ReLU. The mapping from the input to the output of a CNN must also be highly nonlinear because the semantic information in an image is obviously a highly nonlinear mapping of pixel values in the input.
- The vanishing gradient issue prevents the use of the sigmoid and hyperbolic tangent activation functions in networks with many layers. The vanishing gradient issue is solved by the rectified linear activation function, which enables models to learn more quickly and perform better.

2.3 Pooling Layer

A two-dimensional filter is slid over each channel of the feature map during the pooling operation, and the features contained within the region covered by the filter are summarised. There are different types of pooling operations, including max pooling, average pooling, and L_2 pooling. Pooling is a parameter-free down sampling operation. In max pooling, the maximum pixel value is taken within the filter's coverage area, whereas in average pooling. the average pixel value is taken into account. Max pooling is described in figure 2.11

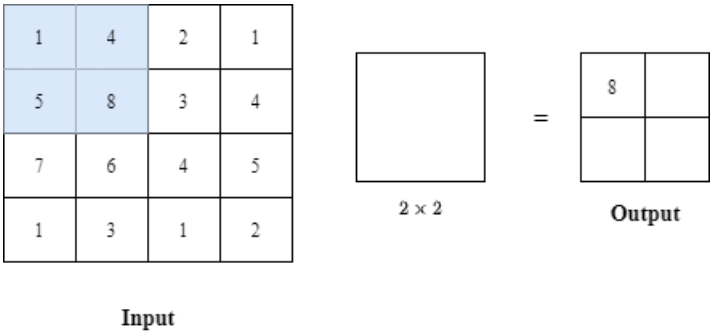


Figure 2.11: Max pooling operation

Max pooling and average pooling can be described as

$$\begin{aligned} Y[i,j] &= \max_{\{(a,b):a\in[0,F-1],b\in[0,F-1]\}} X[i+a,j+b] \text{ (Max pooling)} \\ Y[i,j] &= \frac{1}{F^2}\sum_{a=0}^{F-1}\sum_{b=0}^{F-1} X[i+a,j+b] \text{ (Average pooling)} \end{aligned} \tag{2.10}$$

If input image is of dimension $W_1 \times H_1$ and $F \times F$ with stride S then output dimension W_2 and H_2 are given by

$$\begin{aligned} W_2 &= \left\lfloor \frac{W_1 - F}{S} + 1 \right\rfloor \\ H_2 &= \left\lfloor \frac{H_1 - F}{S} + 1 \right\rfloor \end{aligned} \tag{2.11}$$

Necessity of pooling layer:

- The size of the feature maps is reduced by pooling layers. As a result, it lessens the quantity of network computation and the number of parameters that must be learned.
- The feature map created by a convolution layer's feature pooling layer summarises the features that are present in a particular area. Therefore, instead of precisely positioned features produced by the convolution layer, further operations are performed on summarised features. As a result, the model is more.

2.4 Fully Connected Layer

The final classification is carried out using fully connected layers after a number of convolution and max pooling layers as shown in figure 2.12(b).The

final few layers of the network are fully connected layers. The regular feed forward artificial neural network is made up of these layers (ANN). The final pooling or convolution layer's output, which is flattened from a matrix to a vector, serves as the input to the fully connected layer as shown in figure 2.12(a) and then fed into the fully connected layer.

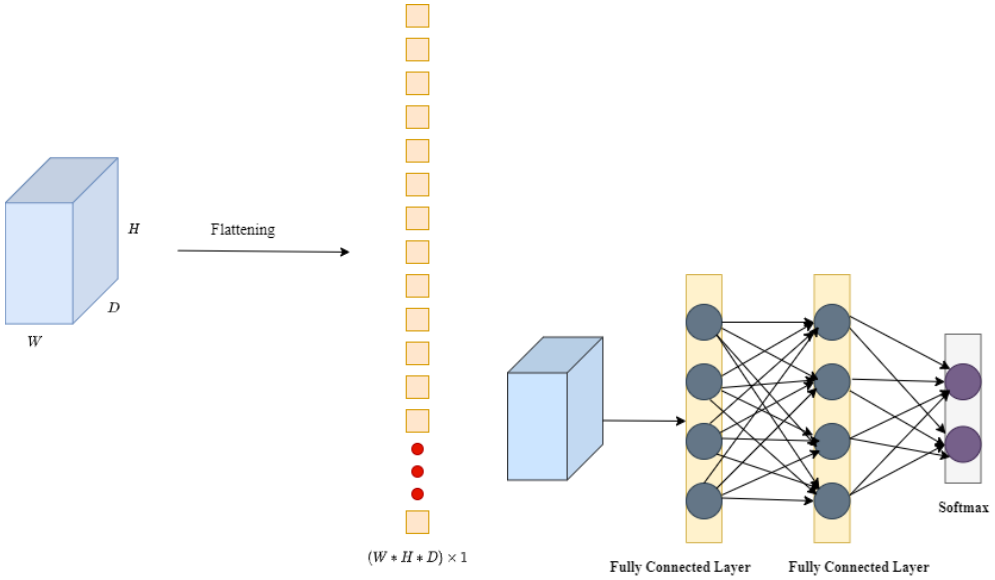


Figure 2.12: Flattening and Fully connencted layers

This ANN works with forward propagation and backward propagation algorithm (shown in figure 2.14) and eThe final layer uses the softmax activation function and activates with the ReLU activation function at the very fully connected layer, as described in equation. 2.13 which is used to get probabilities of the input being in a particular class.

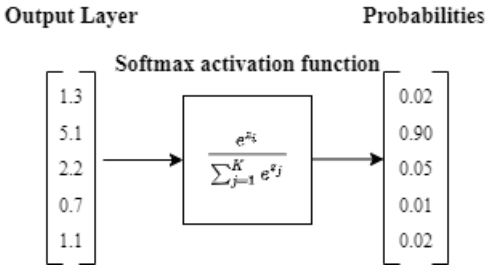


Figure 2.13: Softmax for final classification

2.4.1 Working of ANN's

- **Gradient Descent algorithm**
  It is an optimization algorithm that iteratively moves in the direction of the steepest descent, as indicated by the negative of the gradient, in order to minimise some function. Gradient descent is used in machine learning to update parameters because it is virtually impossible to calculate analytically.

- **Forward Propagation Algorithm**
  In a forward propagation network, every value in the intermediate or hidden layers is serially computed. For the first epoch, weights used in computation are initialised to random values; for subsequent epochs, the weights from the preceding epoch are used.

- **Backward Propagation Algorithm**
  The synaptic weights of the hidden layer nodes and input layer nodes are updated by propagating error backwards in this method, which makes use of the gradient descent algorithm. Error is determined at the output layer with respect to the corresponding outcome.

Forward and backward propagation work together during neural network training to achieve acceptable accuracy or error.

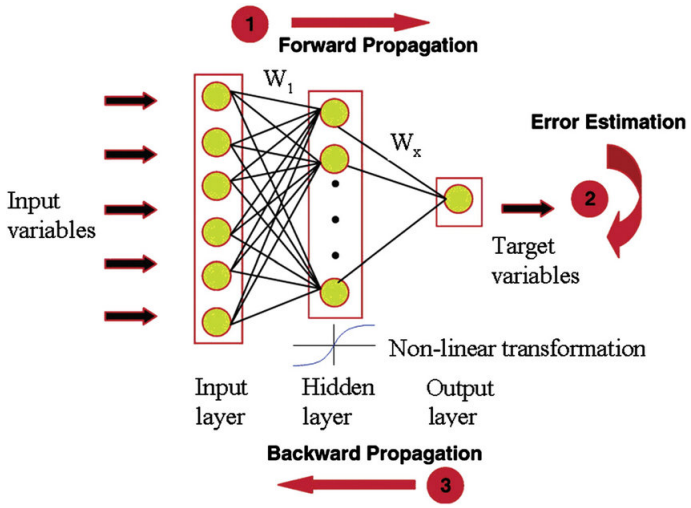


Figure 2.14: Artificial Neural Network

2.5 Back-Propagation in CNN

For simplicity, a grayscale image i.e image with one channel is assumed. Along with this, single convolutional filter is assumed which will yield single channel feature map. A single convolutional filter W $K_1 \times K_2$ applied to an image X of $N_1 \times N_2$ dimensions, resulting an output feature map Y of $M_1 \times M_2$ can be seen from 2.3.
Once gradients are computed parameters can be updated using gradient descent.

2.5.1 Back-propagation in Convolutional Layer

Consider cost function/loss function L used to train the CNN, for convolutional layer calculation of two gradients are required:

- $\frac{\partial L}{\partial W}$ for weight update
- $\frac{\partial L}{\partial X}$ gradient with respect to input feature map to Convolutional layer to back-propagation of errors to previous layer.

Computation of $\frac{\partial L}{\partial W}$
First $\frac{\partial L}{\partial W}$ is to estimate for which one can compute $\frac{\partial L}{\partial W[a,b]}$ i.e gradient with respect to single weight then easily $\frac{\partial L}{\partial W}$ is determined. $W[a,b]$ affects every pixel in Y as each pixel in the output corresponds to one position of the filter overlapping the input moreover, every pixel in the output is a weighted sum of a part of the input image. First hand assumption is $\frac{\partial L}{\partial Y}$ known as computation of gradients is done from backward i.e from last layer.
$\frac{\partial L}{\partial W[a,b]}$ can be written as summation of all gradients coming from each pixel in output.

$$\frac{\partial L}{\partial W[a,b]} = \sum_{i=0}^{M_1-1} \sum_{j=0}^{M_2-1} \frac{\partial L}{\partial Y[i,j]} \frac{\partial Y[i,j]}{\partial W[a,b]} \tag{2.12}$$

Now $\frac{\partial Y[i,j]}{\partial W[a,b]}$ to be determined.
from 2.3

$$Y[i,j] = \sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} W[a,b] X[i-a, j-b]$$

Hence,

$$\begin{aligned} \frac{\partial Y[i,j]}{\partial W[a,b]} &= \frac{\partial (\sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} W[a,b] X[i-a, j-b])}{\partial W[a,b]} \\ &= \frac{\partial (W[a,b] X[i-a, j-b])}{\partial W[a,b]} = X[i-a, j-b] \end{aligned} \tag{2.13}$$

Substituting equation 2.13 in equation 2.12 yields,

$$\frac{\partial L}{\partial W[a,b]} = \sum_{i=0}^{M_1-1} \sum_{j=0}^{M_2-1} \frac{\partial L}{\partial Y[i,j]} X[i-a, j-b] \tag{2.14}$$

Hence one can write the gradient loss function with respect to weights as:

$$\frac{\partial L}{\partial W[a,b]} = \sum_{i=0}^{M_1-1} \sum_{j=0}^{M_2-1} \frac{\partial L}{\partial Y[i,j]} X[i-a, j-b] = X * \frac{\partial L}{\partial Y} \tag{2.15}$$

Computation of $\frac{\partial L}{\partial X}$

Effect of single pixel $X[i,j]$ on output feature map pixels depends on size of filter as shown in figure 2.15(a), the dotted region in the output represents the output pixels affected by $X[i,j]$. Let that region be P.

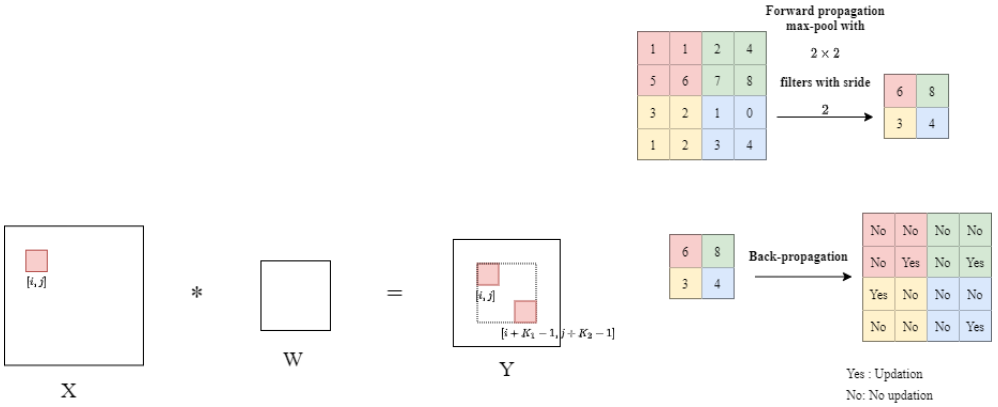


Figure 2.15: Region of effect of input pixel and Back-propagation in pooling

First $\frac{\partial L}{\partial X}$ is to estimate for which one can compute $\frac{\partial L}{\partial X[i,j]}$ i.e gradient with respect to single pixel then easily $\frac{\partial L}{\partial X}$ can be determined. On application of chain rule, $\frac{\partial L}{\partial X[i,j]}$ can be written as

$$\frac{\partial L}{\partial X[i,j]} = \sum_{p \in P} \frac{\partial L}{\partial Y[p]} \frac{\partial Y[p]}{\partial X[i,j]} \tag{2.16}$$

It is possible since derivative of pixels outside region is 0. From figure 2.15(a) region P can be determined

$$P = \{[i+a, j+b] : 0 \leq a \leq K_1 - 1\ , 0 \leq b \leq K_2 - 1\}$$

$$\frac{\partial L}{\partial X[i,j]} = \sum_{a=0}^{K_1-1}\sum_{b=0}^{K_2-1} \frac{\partial L}{\partial Y[i+a,j+b]} \frac{\partial Y[i+a,j+b]}{\partial X[i,j]} \tag{2.17}$$

To calculate $\frac{\partial Y[i+a,j+b]}{\partial X[i,j]}$ we consider 2.3 by which

$$Y[i,j] = \sum_{a=0}^{K_1-1}\sum_{b=0}^{K_2-1} W[a,b]X[i-a,j-b] \tag{2.18}$$

rewriting above equation,

$$Y[i+a,j+b] = \sum_{a=0}^{K_1-1}\sum_{b=0}^{K_2-1} W[a,b]X[i,j] \tag{2.19}$$

Differentiating above equation partially with respect to $X[i,j]$ yields required quantity as shown below

$$\frac{\partial Y[i+a,j+b]}{\partial X[i,j]} = W[a,b] \tag{2.20}$$

Equation 2.17 can be written as

$$\frac{\partial L}{\partial X[i,j]} = \sum_{a=0}^{K_1-1}\sum_{b=0}^{K_2-1} \frac{\partial L}{\partial Y[i+a,j+b]} W[a,b] = \frac{\partial L}{\partial Y} * flip_{180^\circ}(W) \tag{2.21}$$

2.5.2 Back-propagation in ReLU Layer and Pooling Layer

Back-propagtion in ReLU Layer
The only obstacle to back-propagation in the ReLU layer is the non-differentiability of the ReLU activation function(refer figure 2.9) but When using the back-propagation algorithm, one of the two one-sided(either right derivative or left hand derivative).
Back-propagation in Pooling Layer
As pooling layer is non parametric there are no weights to learn.

- In **max-pooling**, Only the winning pixel, the one with the highest value in the pooling block, receives the back-propagated gradient; the forward pass can keep track of this(refer 2.15(b)). Multiple pixels will be assigned with gradient if the winning pixel.
- In **average pooling**, the back-propagated gradient is divided by the area of the pooling block ($K \times K$) and equally assigned to all pixels in the block.

Chapter 3

ResNet 50

ResNet, also known as a residual neural network, is a concept that builds on the traditional convolutional neural network by incorporating residual learning. ResNet addresses the issue of gradient dispersion and accuracy degradation (training set) in deep networks and allows the network to become more and more accurate while maintaining control over its depth. There are fifty convolutional layers in the ResNet 50 variant.
The problem caused by increasing depth

- The issue of gradient explosion or dissipation is the first issue that depth increases bring about. This is due to the network's gradient of backpropagation becoming unstable as the number of layers rises due to constant multiplication. The issue of gradient dissipation frequently arises among them. i.e., the impact of weight loss.
- Another issue with increasing depth is network degradation, which occurs when adding more layers to a sufficiently deep neural network and causes.

3.1 1×1 Convolution

The fact that the number of feature maps frequently rises with the network's depth is a drawback of deep convolutional neural networks. When larger filter sizes are used, this issue may cause a noticeably large increase in the number of parameters and computation needed. such as 5×5 and 7×7. To address this problem, a 1×1 It is possible to use a convolutional layer that provides channel-wise pooling, also known as feature map pooling or a projection layer. This straightforward method can be used to reduce the number of feature maps while keeping the important features.

Working of 1×1 convolution

- For each channel in the input, A 1×1 filter will only have one parameter or weight, which yields one output value.
- With an input from the same position across all of the input feature maps, an 1×1 filter behaves like a single neuron.
- The result is a feature map with the same width and height as the input but a depth equal to the number of filters. This single neuron can then be applied repeatedly with a stride of 1, left to right, and top to bottom as shown in figure 3.1.

3.2 Batch Normalization

The assumption that the input distribution will remain constant over the course of training learning systems is one of the main presumptions. This condition is never not satisfied when dealing with neural networks, which are made up of multiple layers stacked on top of each other, as opposed to linear models, which simply map input data to some suitable outputs.

In such an architecture, the parameters of all earlier layers have an impact on each layer's inputs (small changes to the network parameters amplify as the network becomes deeper). As a result, a small change made during the back-propagation step within one layer can result in a significant variation in the inputs of another layer and, ultimately, alter the distribution of feature maps. Each layer must continuously adjust to the new distribution that it has learned from the previous one during training, which slows convergence.

By reducing the covariance shift within internal layers (change in the distribution of network activations due to the change in network parameters during training) during training and with the benefits of working in batches, batch normalisation solves this problem and makes the training more efficient at the same time.

Algorithm 1: Batch Normalization

Data: x_i's values in mini batch $\mathcal{B}$ where $1 \leq i \leq m$

Result: $y_i = BN_{\gamma,\beta}(x_i)$, $1 \leq i \leq m$

1 $\mu_{\mathcal{B}} \leftarrow \frac{\sum_{i=1}^{m} x_i}{m}$; /* Mini-batch mean */

2 $\sigma^2_{\mathcal{B}} \leftarrow \frac{\sum_{i=1}^{m}(x_i-\mu_{\mathcal{B}})^2}{m}$; /* Mini-batch variance */

3 $\hat{x}_i \leftarrow \frac{x_i-\mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}}+\epsilon}}$; /* Normalize (ϵ to forbid division by zero) */

4 $y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$; /* Scale and shift */

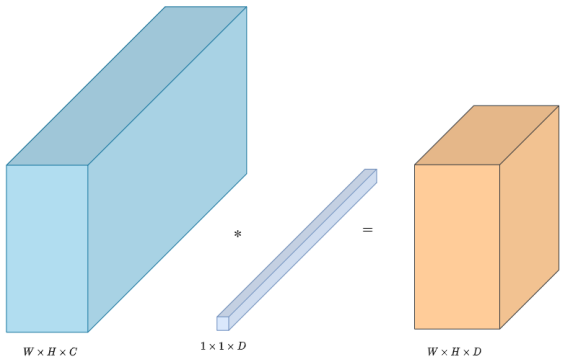


Figure 3.1: 1×1 Convolution

3.3 ResNet 50 Architecture

It starts with a convolutional layer, then moves through max-pooling, four bottleneck/residual blocks, a 2D global average pooling layer, a fully connected layer, and softmax for classification. as shown in figure 3.3. It uses batch normalization after every convolutional layer and for flattening **2D global average pooling** which takes average of pixels of each input channel.

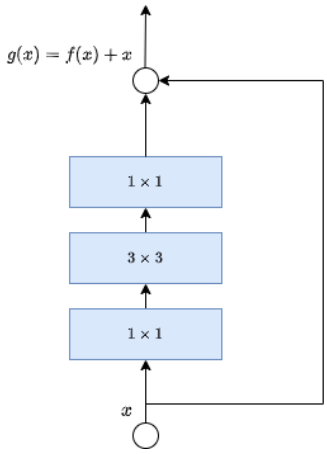
Residual Block / Bottleneck Block

- It is certain that the deeper network will improve the feature finding capabilities of the initial shallow network if deeper neural net function classes contain the simpler and shallower network function classes.
- In order to address the degradation issue, simpler functions (found in shallower networks) should be a subset of complex functions (found in deeper networks).

- Since identity mapping is necessary in these situations and learning identity mapping is difficult and time-consuming for many deep neural networks, deep neural networks struggle to perform well in these situations.

- Consider input x and the desired mapping from input to output is denoted by $g(x)$. Instead of dealing with $g(x)$ one can deal with a simpler function $f(x) = g(x) - x$. The original mapping is then recast to $f(x) + x$. Optimization of residual takes care of learning identity mapping if required by ensuring the weights towards zero of the residual function. This is done by introducing skip connection.

- Each residual block of ResNet 50 consists of 3 layers stacked on each other and shortcut connection skips 3 convolution layers as shown in the figure 3.2(a). Contents of each residual block is shown in 3.2(b).



(a)

	Block Contents	
Residual Block 1	$1 \times 1, 64$ $3 \times 3, 64$ $1 \times 1, 256$	$\times 3$
Residual Block 2	$1 \times 1, 128$ $3 \times 3, 128$ $1 \times 1, 512$	$\times 4$
Residual Block 3	$1 \times 1, 256$ $3 \times 3, 256$ $1 \times 1, 1024$	$\times 6$
Residual Block 4	$1 \times 1, 512$ $3 \times 3, 512$ $1 \times 1, 2048$	$\times 3$

Table 1: Residual Block Contents

(b)

Figure 3.2: Residual Block structure and blocks' contents in ResNet 50

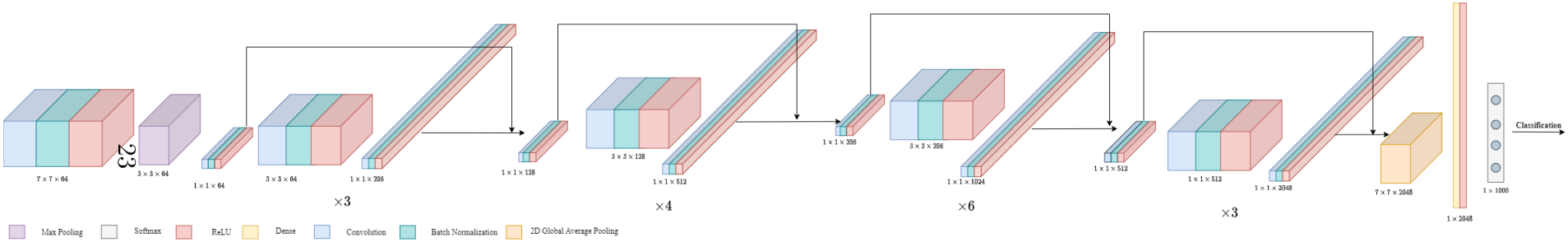


Figure 3.3: ResNet 50 Architecture

Appendix A

Implementation

Work is being done on the benchmark data set for traffic sign recognition GTSRB (German Traffic Sign Recognition Benchmark)with 35K images.

A.1 Data-set Visualization

It includes 34,799 images of traffic signs from 43 different sign types. They are distributed as follows. Classes have an uneven distribution of images.

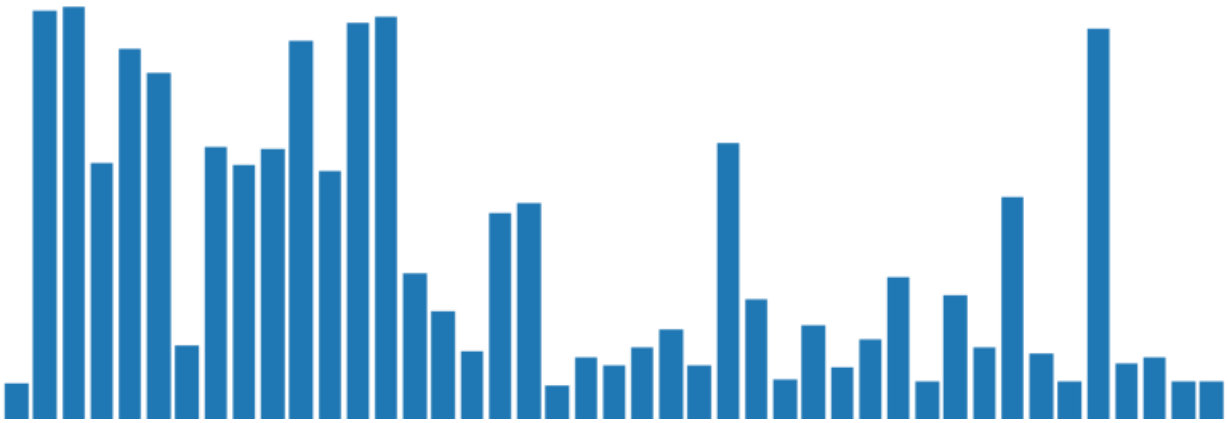


Figure A.1: Traffic Signs Data Set

A.2 Image Pre-processing

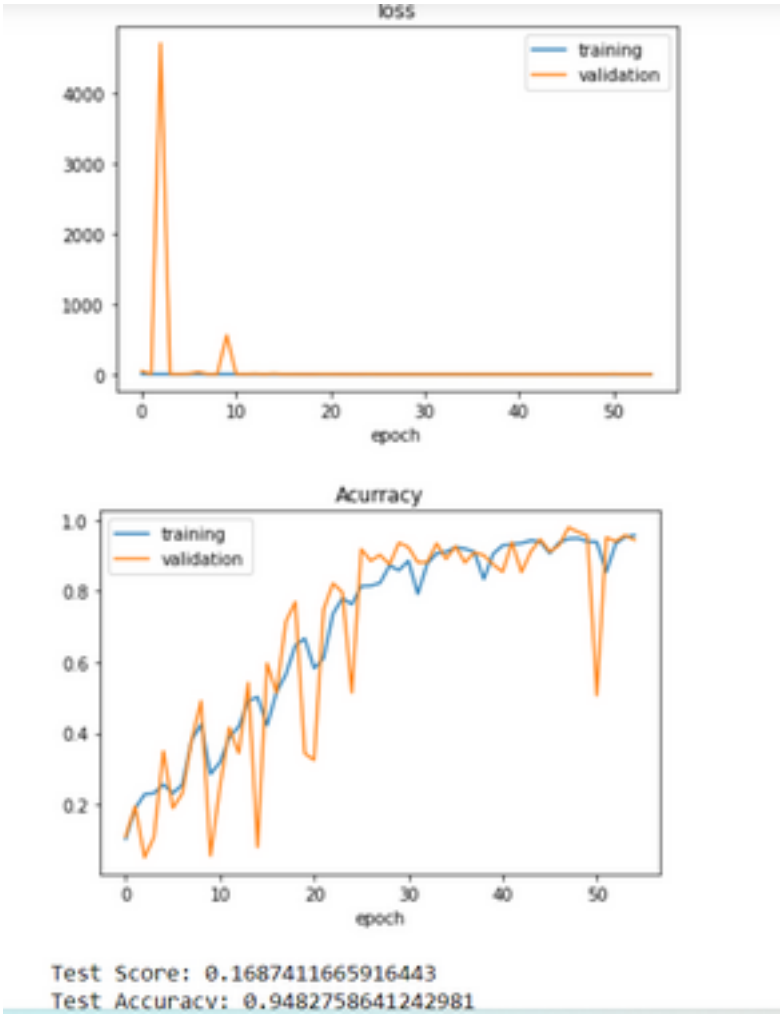
As each image is of different dimensions so up-sampling or down-sampling to 32×32 is done as per the case also data augmentation is done such rotation,

shear, translation, whitening etc. Sample images after pre-processing are shown below.

Figure A.2: Sample Images after pre-processing

A.3 Training

The data set is divided into the training and test sets, and the training set is further divided into the training and validation sets. All splitting of data sets is random and 80% to training and 20% to other part for each split. Finally training set has 22271 images, validation has 5568 images and whereas test has 6960 images.
Model consists of ResNet 50 and it is trained without transfer learning with "Adam" optimizer and "Categorical Cross Entropy" as a loss function and 20 epochs. Also early stopping and drop out were incorporated.
Training loss: 0.1408 Training accuracy: 95.71 percent
Validation loss: 0.1713 Validation accuracy: 87.63 percent
Test score: 0.1687 Test accuracy: 94.82 percent



(a) Accuracy and Loss

Appendix B

Future Work

By combining Res Net 50 with the cutting-edge "Spatial Transformer Network," which renders the model spatially invariant, and cutting-edge data augmentation techniques like "Auto Augment," a significant increase in accuracy can be achieved. The design of the model is shown in figure **??**

Bibliography

[1] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning, Data mining, Inference & Prediction.* Stanford, California: Springer, 2008.

[2] Álvaro Arcos-García, J. A. Álvarez García, and L. M. Soria-Morillo, "Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods," *Neural Networks*, vol. 99, pp. 158–165, 2018.

[3] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," 2016.

[4] E. D. Cubuk, B. Zoph, D. Mané, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning augmentation stratergies from data," 2019.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

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