AI Frameworks for Visual Traffic Surveillance, from Vehicle Classification to Event Detection

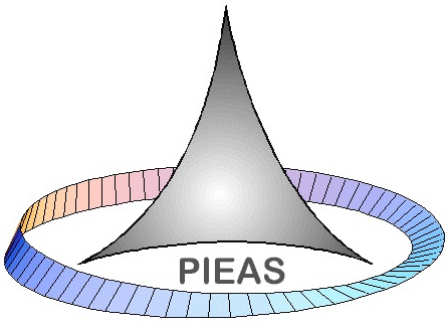
By

Assad Sultan

&

Muneeb ur Rehman

Thesis submitted to the faculty of Engineering in partial fulfillment of requirements for the Degree of BS Electrical Engineering.



Department of Electrical Engineering,

Pakistan Institute of Engineering and Applied Sciences,

Nilore, Islamabad 45650, Pakistan

Januray 3, 2020

This page is left intentionally blank.

Department of Electrical Engineering

Pakistan Institute of Engineering and Applied Sciences (PIEAS)

Nilore, Islamabad 45650, Pakistan

Declaration of Originality

I hereby declare that the work contained in this thesis and the intellectual content of this thesis are the product of my own research. This thesis has not been previously published in any form nor does it contain any verbatim of the published resources which could be treated as infringement of the international copyright law. I also declare that I do understand the terms 'copyright' and 'plagiarism,' and that in case of any copyright violation or plagiarism found in this work, I will be held fully responsible of the consequences of any such violation.

Signature: _____

Name: Assad Sultan

Signature: _____

Name: Muneeb ur Rehman

Date: Januray 3, 2020

Place: PIEAS

Certificate of Approval

This is to certify that the work contained in this thesis entitled

"AI Frameworks for Visual Traffic Surveillance, from Vehicle Classification to Event Detection"

was carried out by

Assad Sultan & Muneeb ur Rehman

under our supervision and that in our opinion, it is fully adequate, in scope and quality, for the degree of BS Electrical Engineering from Pakistan Institute of Engineering and Applied Sciences (PIEAS).

Approved By:

Signature: _____

Supervisor: Dr. Zulfiqar Hassan

Signature: _____

Co-Supervisor: Dr. Sufi Tabassum Gul

Verified By:

Signature: _____

Head, Department of Electrical Engineering

Stamp:

Dedicated to our Parents for their love and affection

Contents

List of Figures

List of Tables

Abstract

The key objective of this project is the detection & classification of vehicles for traffic surveillance purposes. Increasing accidents on roads have made the need of efficient methods to meet the detection & classification purposes significant. Several state of the art alogrithms exist which detect and classify the objects considering some features but to acheive high precision in short time, we need robust algorithms. In the past few years, CNNs have acheived a rapid success in accuracy and speed. In this project, a CNN based detector is used called as YOLO. The algorithm is customized according to the requirements. The still images are used for training made available using web scrapping to make a large dataset to get good results during training. The dataset consists of vehicles of 8 different classes. After the classification mechanism, detection algorithms are used for the detection of objects in a still image and moving frames.

Keywords: Deep Learning, Detection, Vehicles, Convolutional Neural Networks, Deep Neural Networks, YOLO

Chapter 1

Introduction

1.1 Motivation

With the development of science and technology, there may arise crucial problems. Rising traffic in the country gives birth to rising accidents. To avoid/minimize the accidents, proper surveillance is required. Detection and tracking of vehicles have become a key component in traffic surveillance and automatic driving. Traditional algorithms like Gaussian Mixed Model (GMM) has achieved encouraging success in this field [1]. But due to variation in illumination, occlusion, background clutter etc. detection of vehicles is still a challenge. Deep Neural Networks (DNNs) have gained a lot of attention in past few years. Due to the betterment of deep learning, object detection has achieved major advancements in recent years. The work in this project is based on DNNs to detect and classify vehicles from a dataset collected from images and videos.

1.2 Background

Classical methods of vehicle detection were based on some sort of features like symmetry, edges, texture, underbody shadows and corners [2]. These methods were easy to describe and perform in specific kind of applications. These methods fail in many situations like due to less illumination of highway. With the advent of deep learning, a subset of machine learning, the detection and classification mechanism have dramatically improved.

Machine learning and deep learning are sub-branches of computer intelligence which help in the process of vehicle classification to event detection. Using deep learing techniques, we are going to focus on a specific kind of detection (deep learning has grown its branches in almost every kind of image recongnition, text recognition, face detection etc), vehicle detection. Using deep learning based Convolutional Neural Networks (CNNs) which is a wide field for object detection, classification, sementic segmenation etc. These networks take input images, learn features, adjust weights and then detect the targets from test data. In deep learning, algorithms automatically learn features unlike machine learning algorithms, we have to pass features to train the model. Our work is based on Deep Neural Networks (DNNs).

1.3 Goals & Objectives

The goal of this project is to make a deep learning based model (using Python) for vehicle detection and classification. Main objectives are summarized as:

- Collection of datasets (images dataset of car, truck classes etc. and videos).
- Design of Convolutional Neural Network
- Configuration of training options (AlexNet, VGGNet, ResNet etc.)
- Training of detector (RCNN, Fast- RCNN, Faster-RCNN, YOLO etc.)
- Evaluation of the trained detector

1.4 Challenges

Human can understand and interpret images in a second or less. For machines, it is not easy to achieve accurate results unless models are trained properly. Firstly, proper training is a challenging task for a customized data. Detection algorithms are also hard to implement as we must have recommended GPU requirements and best choice of algorithm

for good results. Videos which are moving frames, it is difficult to draw accurate bounding boxes in a video having say 30 frames per second means 30 images are splashed in a second.

1.5 Organization of Thesis

Chapter 2: Chapter 2 is about literature review. It is about the whole reading and understanding of Image classification & detection.

Chapter 3: This chapter is about Convolutional Neural Networks. All CNN architectures with their details are discussed.

Chapter 2

Literature Review

2.1 History of Object Recognition & Detection

The field of computer vision deals with how the computers can be made to understand digital images or videos. The history of Computer Vision can be traced back to 1960s. Before that people had tried for alphabet recognition but those attempts had not led very far. L.G. Roberts, in his Ph.D thesis mentioned about the perception of solid objects. Using the properties of three dimensional transformations and the laws of nature, a procedure was developed which had been able to identify objects as well as determine their orientation and position in space [3].

In 1970s, David Marr wrote a book on visual recognition. In his book he mentioned that in order to take an image and to arrive at a final holistic full 3D representation of the visual world, we must go through several processes. The first process he calls is "Primal sketch" where edges, bars, ends and virtual lines are determined. Next is "two and half-D sketch" leading to "3-D sketch" [4]. Another very important seminal group of work happened in 1970s when people began to ask the question, "how can we move beyond simple block world and start recognizing real objects?" Two group of scientists proposed similar ideas, one is called "generalized structure" and other is called "pictorial structure". The basic idea was that every object is composed of simple geometric primitives [5].

Figure 2.1: SIFT & Object Recognition

In 80s, David Lowe tried to recognize razors by constructing lines and edges and mostly straight lines and their combination. At those times, it was very hard to solve an image recognition problem due to limited resources. From 1999 to 2000, statistical machine learning techniques started to gain momentum. Such techniques are support vector machines, boosting, graphical models etc. In 2006, FujiFilm rolled out first digital camera with face detection feature.

One of the very influential way of thinking in the late 90s till the first 10 years of 21st century was feature based recognition. An important work was done by David Lowe based on a feature called as SIFT feature. The idea was that to match one stop sign shown in Figure 2.1 with another stop sign is very difficult, because there might be all kinds of changes due to camera angles, occlusion, viewpoint, lighting and just the intrinsic variation of the object itself. Despite it is inspired to observe that there are some parts of the object that tend to remain unchanged. So, the task of object recognition began with identifying those features on the object and match these features to a similar object [6].

In 2009, the ILSVRC was rolled out. This data set contains 1.4 million images of different classes. In 2012, the error rate was dropped to almost 16 % [7]. The winning algorithm was based on CNNs. This work was published in 2012 introducing AlexNet,

which brought a tremendous change in the field of image recognition. This thesis is based on image classification and detection using CNNs.

2.2 Basics of an Image

To start with, we must understand all about an image, the information of an image that is important to a machine. A digital image is picture information in digital form. It consists of pixels. The number of pixels in the presentation of a digital image is its spatial resolution, which relates to the image quality. The higher the spatial resolution, the better quality the image has. The resolution can be high, for instance, as high as 1,600 $\times$ 1,200 (1,920,000 pixels = 1.92 megapixels), or as low as 320 $\times$ 200 (64,000 pixels = 64 kilo pixels). In notation, the number to the left of the multiplication symbol represents the width, and that to the right of the symbol represents the height. Image quality also depends on the numbers of bits used in encoding each pixel level [8, 9].

2.2.1 8-Bit Gray Level Images

If a pixel is encoded on a gray scale from 0 to 255, where 0 corresponds to black and 255 corresponds to white. The intermediate numbers represent the shades of gray. For example, for a 640 $\times$ 480 bit image, 307.2 kilobytes are required for storage. Figure 2.2 shows a grayscale image format. The pixel value indicated in the box has an 8-bit value of 25.

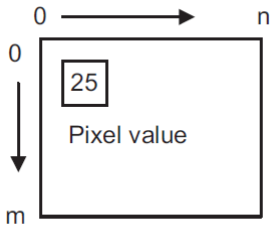


Figure 2.2: Grayscale Image format

2.2.2 2.2 24-bit Color Images

In a 24-bit color representation, each pixel of an image is encoded with Red (R), Green (G) and Blue (B) values. With each component encoded in 8 bits, we have a total of 24 bits for a full color RGB image. Having such an image, we have $2^{24} = 16{:}777216 \times 10^6$ different colors. A 640 × 480 24-bit color image requires 921.6 kilobytes for storage. Figure 2.3 shows the format for the 24-bit color image where the indicated pixel has 8-bit RGB components.

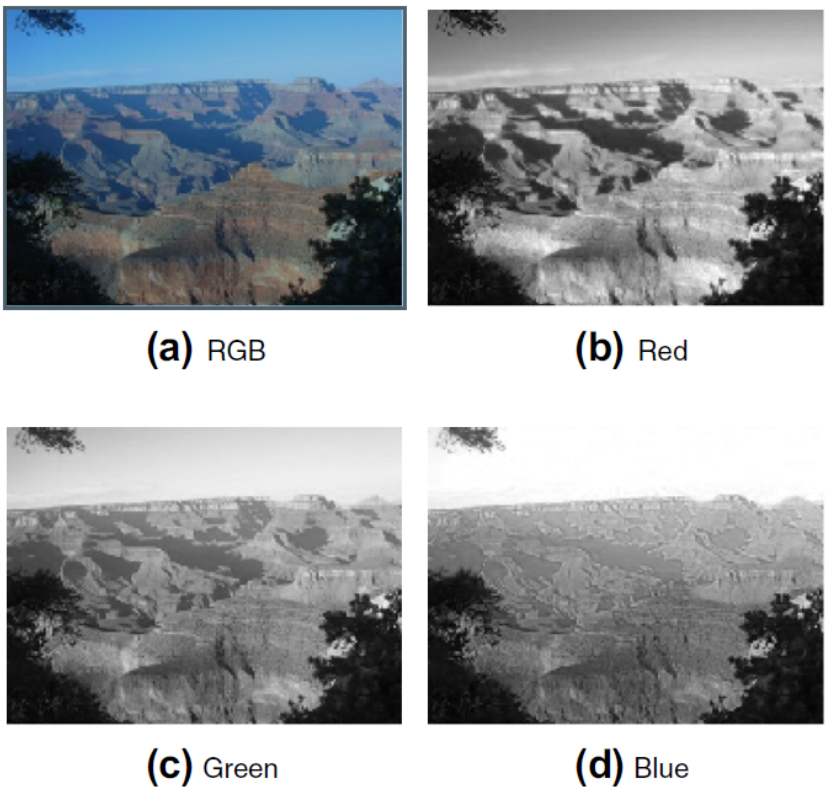


Figure 2.3: The 24-bit color image and its RGB components

2.2.3 8-Bit Color Images

The 8-bit color image is also a popular image format. The pixel values are indexed from 0 to 255. Each index represents an entry of the color map. These images are called color indexed images. As an example, figure 2.4 shows a color indexed image that has a pixel value of 5, which is the index for the entry of the color table. At location 5 in the color table, there are three color components with RGB values of 66, 132 and 134 respectively.

Each color component is encoded in 8 bits. There are only 256 different colors in the image. A 640 × 480 8-bit color image requires 307.2 kilobytes for data storage and 3 × 256 = 768 bytes for color map storage.

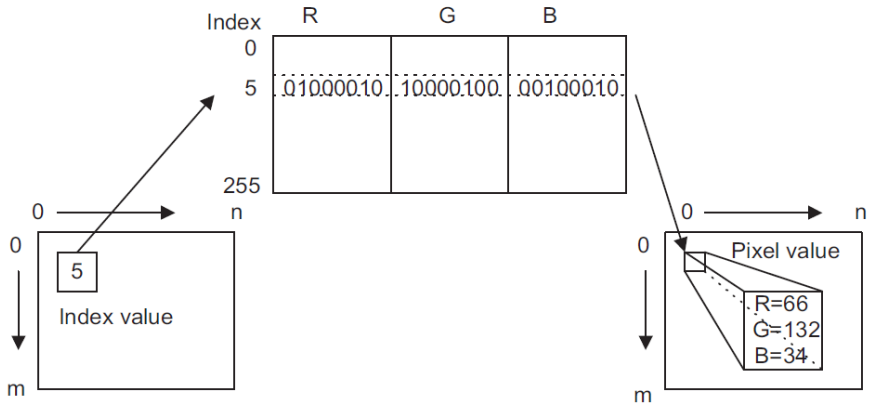


Figure 2.4: The 8-bit color indexed image

2.2.4 Intensity Images

In section 2.1, we mentioned that a grayscale image uses a pixel value in the range 0-255. A 0 value represents black while 255 represents white. In some processing environments, floating point representations are used. The grayscale image has an intensity value normalized from 0 to 1, where 0 is for black and 1 is for white. Figure 2.5 shows the format of the grayscale intensity image, where the indicated pixel shows the intensity value of 0.5988.

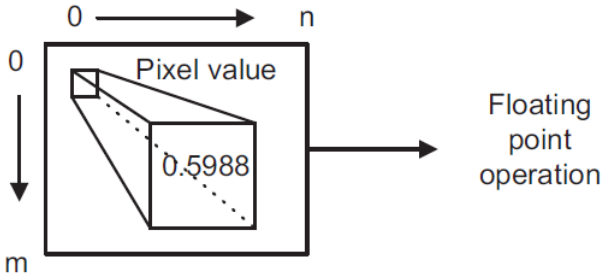


Figure 2.5: The grayscale intensity image format

2.2.5 Red, Green, and Blue Components and Grayscale Conversion

For processing applications, we often need to convert RGB image to grayscale. There are two popular methods, one is average method where a grayscale value is obtained by just averaging the red, green and blue pixel values. The other is luminosity method, where we use approximately 30 % of Red, 59 % of Green and 11% of Blue to form a grayscale image.

$$\text{New grayscale image} = 0.3R + 0.587G + 0.114B \tag{2.1}$$

2.3 Image Classification

Humans can understand and depicts the information contained in an image. On the other hand, machines cannot identify the image and retrieve information from it on its own. To achieve that purpose, we make the machines learn from examples and when it has learned enough, it may extract information from an image from an outside world and recognize it.

There exist many algorithms for image classification. One of them is BoW model, which is commonly used for document classification and natural language processing. In addition to document classification, the BoW model can also be used for image classification. We extract a set of features from an image and count their occurrence. We have different algorithms for feature extraction. For example, in BoW model, SURF algorithm was used [10]. Other interesting image classification methods include texture analysis. However, a gain in performance has been brought by using neural networks. With the introduction of AlexNet architecture in 2012, the error rate was dropped tremendously. Due to robustness of these algorithms, the CNNs are widely used for image classification and recognition nowadays.

The problem of image classification can be stated as: *"Given a set of images labeled with a single category, the machine has to predict a new set of images by assigning it a label and test the accuracy of the predictions"*. For example, in Figure 2.6 an image classification model takes a single image and assigns probabilities to 4 labels, cat, dog,

hat, mug. The computer sees the image as a 3D-array of numbers. The cat example image is 248 pixels wide and 400 pixels in height with three color channels, Red, Blue and Green, therefore, a total of 248 × 400 × 3 = 297,600 numbers. Each number is an integer with values in range 0-255. What our task is to extract out of these thousands of numbers, a single label, such as "cat".

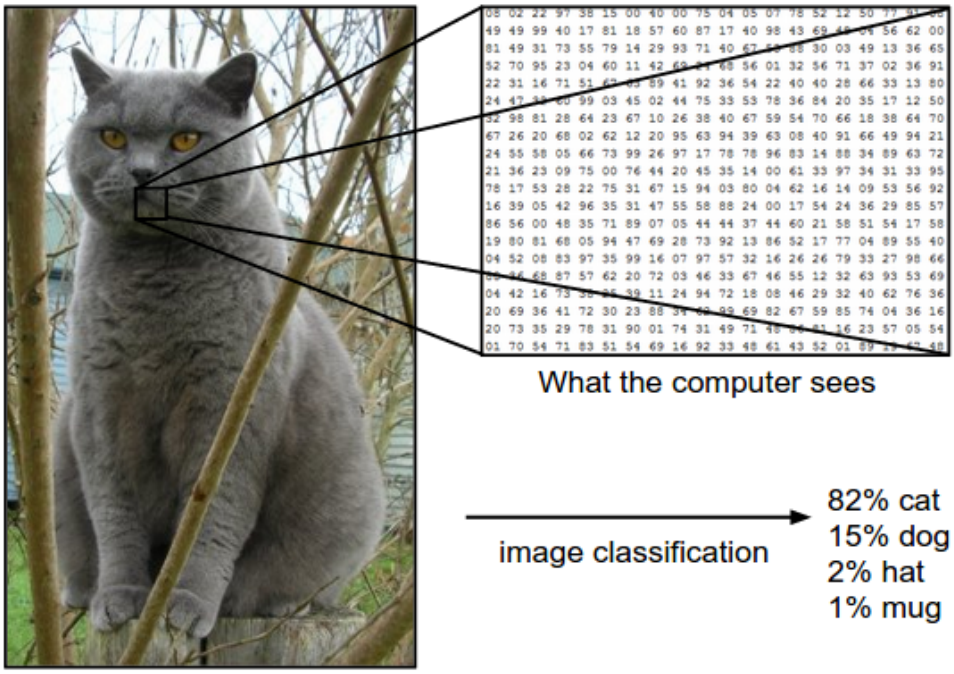


Figure 2.6: A 3D-array of numbers from 0 to 255 of size width × height × 3. 3 represents color channels

There are a variety of challenges associated with this. There may exist viewpoint variation, scale variation, deformation, occlusion (a small portion of object is displaying), illumination conditions, background clutter and intraclass variations. How do we end up writing an algorithm for image classification? Researchers have arrived at the point that in order to solve this problem, use data-driven approach. Instead of writing a code to specify everyone categories of interest look like, we provide computer many examples of images of each class and then develop the algorithm that look at these examples and learn about the visual representation of each class. This approach is known as data-driven approach. So, the image classification pipeline can be summarized as:

- Input of the system is a training dataset composed of N images of labeled with K different classes.
- Using this training set, we train a classifier to learn how do these classes look like.
- After that, test this classifier on a new dataset and then compare the results with the true labels of the images.

2.3.1 Convolutional Neural Networks

CNNs are the most famous network used for classification problems of images. The idea behind CNNs is that a local understanding of an image is good enough. A practical advantage is that if we have fewer parameters, it will reduce the amount of data to train a model and improves the time of learning. A CNN has weights to look at a small patch of image.

A convolution is a weighted sum of the pixel values of an image, as we have a sliding window moving across the whole image. It produces another image of the same size. A CNN typically consists of 3 layers:

1. Convolution Layer
2. Pooling Layer
3. Fully Connected Layer

There are some batch normalization layers in some old CNNs but not used these days. (Details of CNNs are covered in chapter 3).

2.3.1.1 CNN Architectures

As mentioned before, the ImageNet project is a large visual database designed for research purposes. This project runs on an yearly contest named as ILSVRC, where different algorithms compete to classify and detect objects. Here we will talk about the competition top competitors [11].

- LeNet-5 (1998)
- AlexNet (2012)
- ZFNet (2013)
- GoogleNet (2014)
- VGGNet (2014)
- ResNet (2015)

LeNet It is a pioneering 7-layer convolutional network by Yann LeCun et al. in 1998. It classifies digits and was applied by several banks to recognize handwritten numbers on cheques. The processing of higher resolution images we need more convolutional layers, so this technique is constrained by the availability of resources [12].

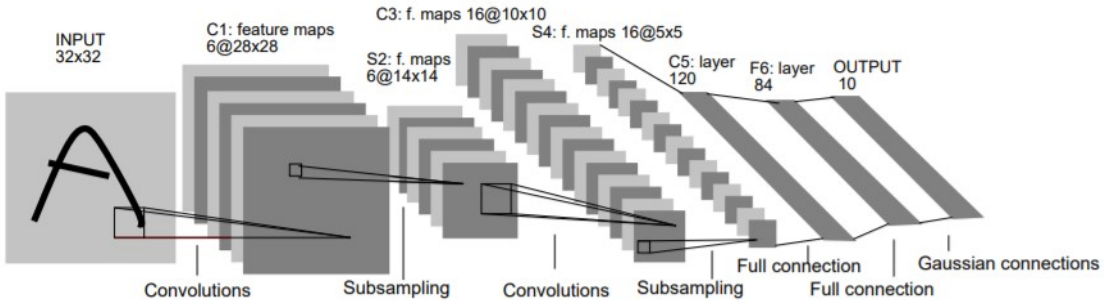


Figure 2.7: Architecture of LeNet-5, original image published in [LeCun et al., 1998]

AlexNet (2012): In 2012, Krizhevsky et al. introduced AlexNet which outperformed all the prior competitors and won the challenge by reducing the error to 15.3 %. This network had a similar architecture like LeNet but was deeper with more filters per layer. It consists of 11 layers between input and output.

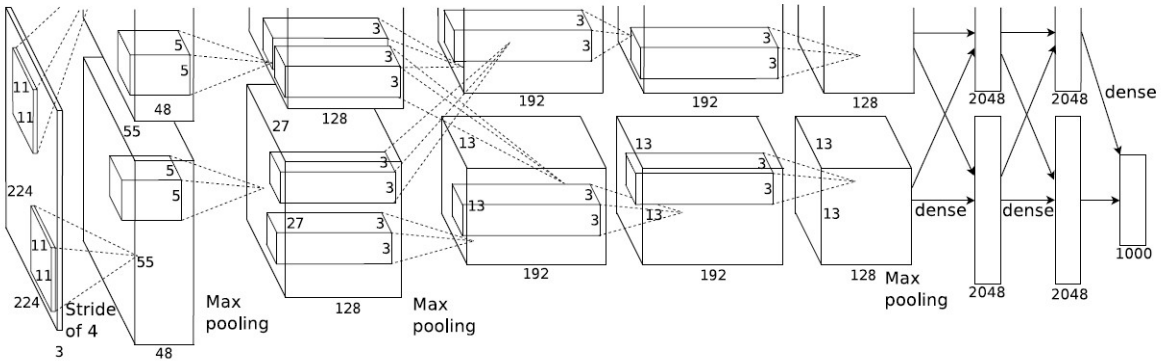


Figure 2.8: An illustration of the architecture of AlexNet CNN

ZFNet (2013): This architecture was the winner of 2013 ILSVRC. It achieved an error rate of 14.8 %. It was achieved by tweaking the hyperparameters of AlexNet while maintaining the same structure and adding deep learning elements [13]. The architecture is shown in 2.9.

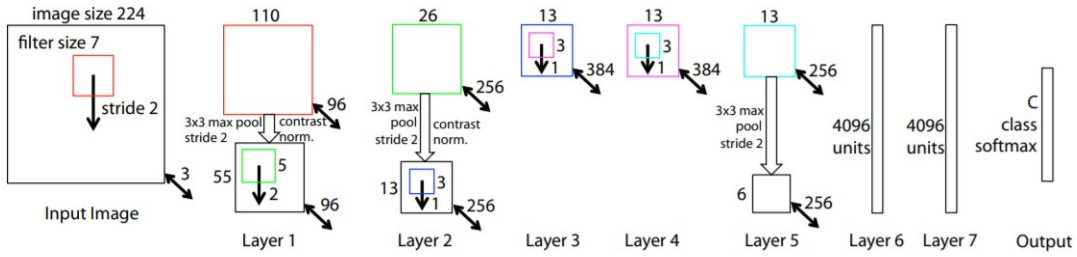


Figure 2.9: Architecture of 8 layer ZFNet Model

GoogleNet (2014): This architecture was the winner of ILSVRC, 2014. It achieved a top-5 error rate of 6.67 %. It was close to human level performance. The network was inspired by LeNet but implemented a new element which is dubbed an inception module. It used batch normalization, image distortions and RMSprop. This architecture consists of 22 layers but a reduced number of parameters from 60 million to 4 million [14].

VGGNet (2014): The VGGNet was developed by Simonyan and Zisserman. It was runner-up in ILSVRC, 2014. It consists of 16 convolutional layers and popular because of

its uniform architecture. A challenging thing in VGGNet is that it consists of 138 million parameters [15].

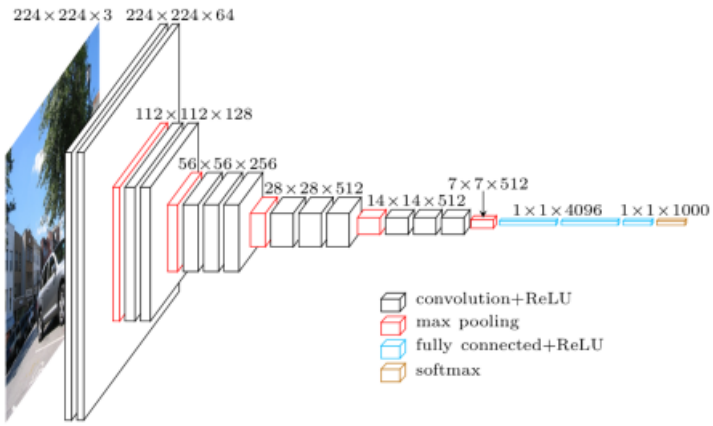


Figure 2.10: An illustration of the architecture of VGGNet CNN

ResNet (2015): In 2015, Residual Neural Network (ResNet) by Kaiming He et al. was introduced. It was a novel architecture with "skip connections" and features heavy batch normalization. These connections are known as gated units. There are 152 layers but still less complex than VGGNet. It achieved a top-5 error rate of 3.57 % which beats human-level performance [16].

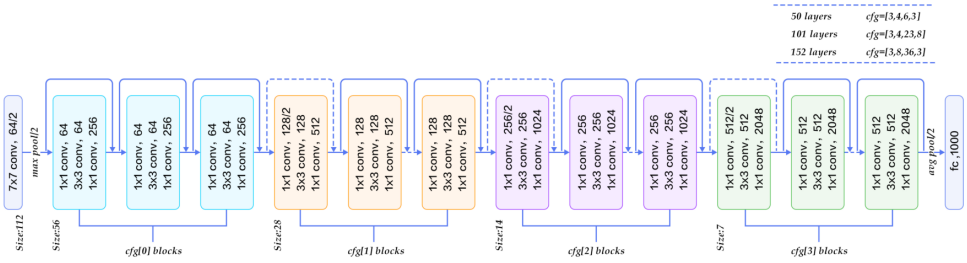


Figure 2.11: ResNet Architecture

AlexNet has parallel two CNN line trained on two GPUs with cross-connections, GoogleNet has inception modules, ResNet has residual connections.

Summary Table

Table 2.1: Comparison of Different CNN Architectures

Year	CNN	Developed by	Top-5 error rate (%)	No. of Parameters
1998	LeNet(8)	Yann LeCun et al.		60 thousand
2012	AlexNet(7)	Alex Krizhevsky et al.	60 million	
2013	ZFNet()	Mathhew, Zeiler & Rob Fergus	9	
2014	GoogleNet(19)	Google	6.67	4 million
2014	VGGNet(16)	Simonyan, Zisserman	7.3	138 million
2015	ResNet(152)	Kaiming He	3.6	

Chapter 3

Convolutional Neural Networks (CNNs)

CNNs are types of artificial neural networks used for object detection and image classification etc. Neural Networks are brain-inspired set of algorithms that are expected to mimic the working of human brain. Billions of neurons in our brain process information through electrical signals. Dendrites receive external information or stimuli in form of electrical impulses which is further processed in the cell body where it integrates all the signals coming in and the output is carried away to the downstream neurons through Axons [17]. The neuron chooses to either reject or accept the signal depending on the signal strength. Likewise, an artificial neural network consists of large number of interconnected nodes known as neurons. The computational node and cell body of neuron works in kind of a similar way where we have large number of signals coming as inputs to neuron. Each signal has a weight W associated with it and the summation of these weighted inputs is passed to the activation function where non-linearity is introduced in the output. Neural networks have the ability to gradually update the weights of the interconnections of the neuron until desirable results are obtained during the training phase of network.

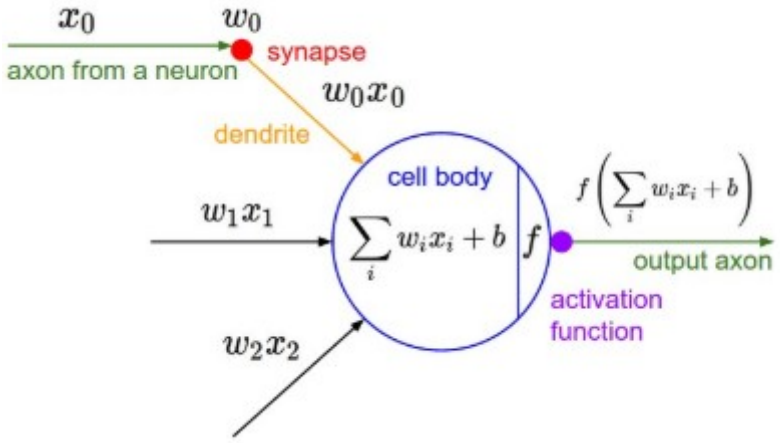


Figure 3.1: Working of a single computational node

These neurons or computational nodes are organized into layer where there are highly interconnected to the neurons of following layer. The input layer of the feed forward neural networks feeds the information to the network and no kind of mathematical computation is performed. The hidden layer also known as distillation layer acts like a cell body and performs computation to extract salient features and patterns from the input data. A simple feed forward neural network with one hidden layer is shown in 3.2 [18]. The output layer simply provides the results for given information.

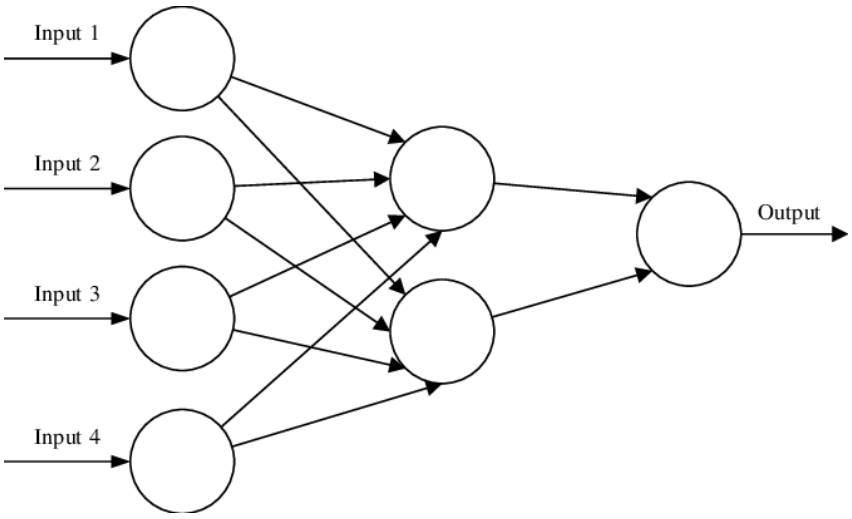


Figure 3.2: Simple Feed Forward Neural Network with one hidden layer

The pixels of the image are arranged in a particular manner i.e. the appearance of the picture will be changed if we change the coordinate of one pixel. However, simple feed

forward neural network loses to learn the inherent spatial relationships within images and fails to extract high level features needed for image classification. CNNs are types of neural networks that learn visual filters to recognize higher level image features and the spatial information is preserved. As a result, you end up learning this hierarchy of filters where filters at the early stage usually represents low level features like edge detection, gradient orientation and learning more and more complex features in the later stages.

3.1 Convolutional Neural Networks Architecture

CNNs consist of three types of layers. These layers include convolutional layers, pooling layers and fully connected layers. When these layers are stacked together, we get a CNN architecture. A general architecture of CNNs with two convolutional layers is shown in 3.3 [18].

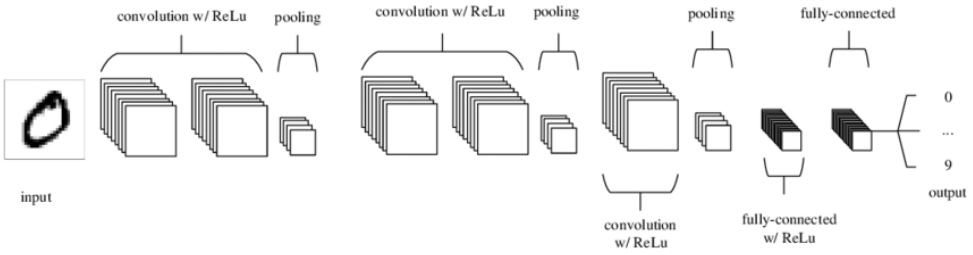


Figure 3.3: Convolutional Neural Network Architecture with two convolutional layers

3.1.1 Convolutional Layer

Convolutional layers are integral part of CNNs and when these layers are stacked together, we start extracting more simple features and then aggregating these into more complex features later on. It consists of the set of learnable filters also known as kernels whose height and width are smaller than the input image and always extend to the full depth of the input volume. We slide the filter over entire image to perform the mathematical operation of convolution i.e. element-wise multiplication of input image with the feature detector. The resultants are summed up and the value is assigned to the top left corner of receptive field as shown in figure 3.4 [19].

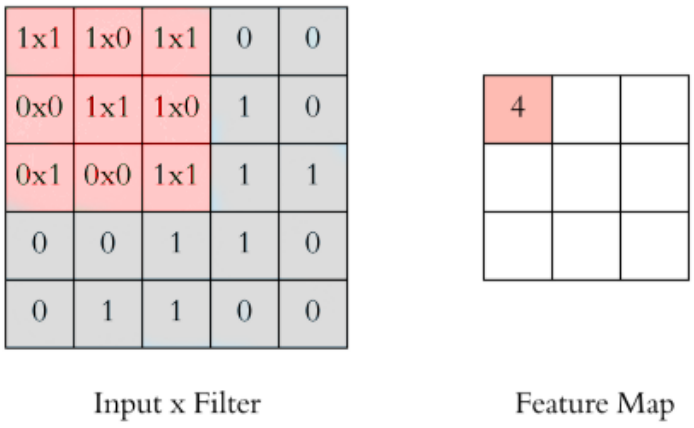


Figure 3.4: 3x3 filter with stride value of 1

The kernel window slides right or downward by a certain value in each step called the stride value and the process is repeated until the entire image is processed. Stride value of 1 moves the filter 1 pixel in each step. This results in a feature map with much smaller size than the input image and keeps on shrinking at every layer which is usually not desirable. Thus, dimensionality of the input image can be preserved by padding zero-pixel values across every side of image.

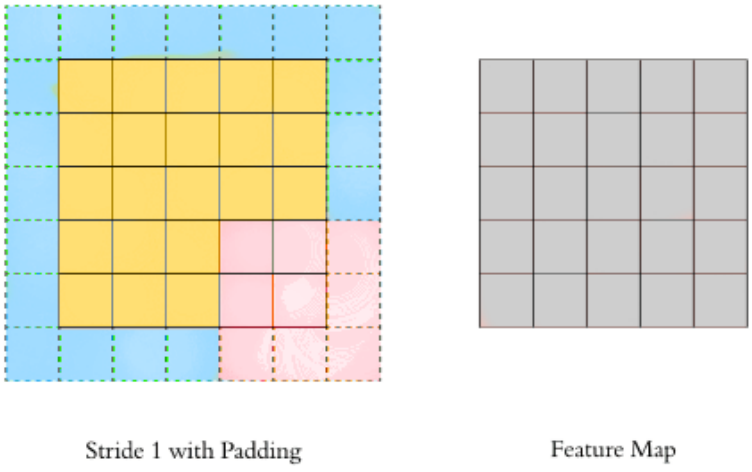


Figure 3.5: Result of 3x3 filter of Stride 1 with padding

In reality, we apply multiple filters to the same input image and feature maps are obtained depending upon the number of filters. In this case the output of convolutional

layer is represented with a 3D matrix with dimensions of height, width and depth, where depth shows the no. of filters applied on the image.

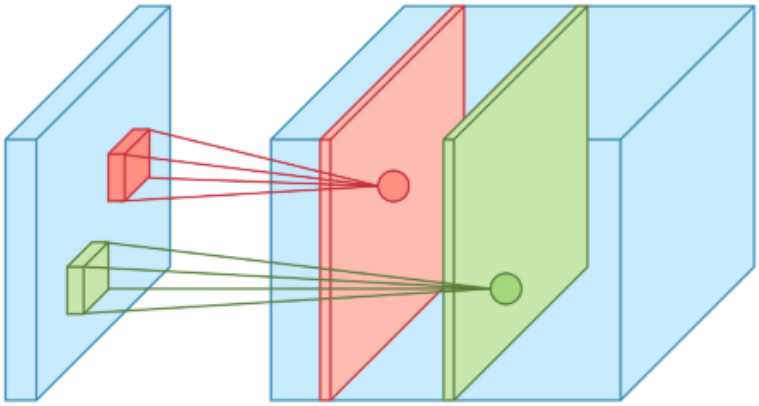


Figure 3.6: Multiple filters applied on one input

3.1.1.1 Activation Functions

It is a mathematical function that is applied to every neuron in the network and decides whether a neuron should be activated or not depending on how much neuron's input is relevant to model's prediction. The main purpose of the activation function is to introduce non-linearity in the output. Without activation function, our network will be reduced to a simple regression model which would be unable to model and learn complex and complicated form of data. Thus, non-linear functions make neural networks capable of performing non-linear transformations to learn complex kinds of data such as images, videos etc. A good activation function should have the following properties:

- It should be zero centered because when back propagating gradient descent, the resulting values of gradient will either be all positive or all negative. Thus, it will affect the gradient based optimization process as we can only follow zig-zag path to reach the optimal point.

- It shouldn't kill the gradient flow i.e. neurons get saturated on the boundaries of activation and local gradient comes out to be zero. As a result, it nullifies the gradient flow during back propagation and dead neurons will stop updating.

- It should be computationally efficient as activation function will be computed across millions of neurons for each input. Moreover, back propagation technique also puts constraint on the choice of activation function i.e. it should be differentiable.

Sigmoid Function: A sigmoid function can be represented by the mathematical function:

$$sig(z) = \frac{1}{1 + e^{-z}} \tag{3.1}$$

This function takes a real value as an input and squeezes it between 0 and -1 i.e. very large values. It is often known as squashing function as the large negative and positive values becomes 0 and 1 respectively [20] . However, the sigmoid function output is not zero-centered, and it also kills the neurons as function gets saturated at higher values. That's why sigmoid function are usually not preferred in neural networks because of these drawbacks.

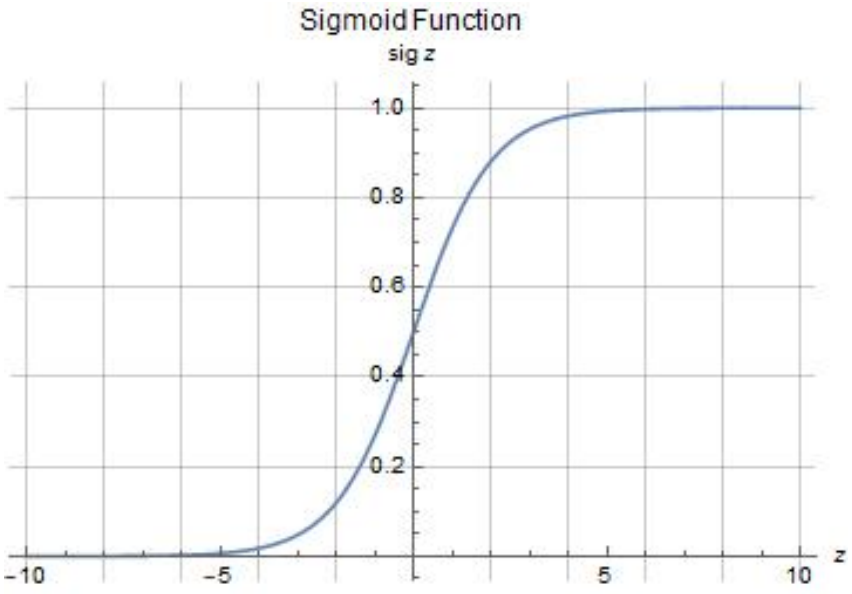


Figure 3.7: Sigmoid Function

Tanh Function: A tanh sigmoid function can be represented by the following mathematical function:

$$tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{3.2}$$

It takes the real input value and squashed it between -1 and 1 i.e. very large negative and positive values are mapped into -1 and 1 respectively [20]. As a result, this non-linear activation function gives zero-mean data, but it still gets saturated at higher values as shown in 3.8. However, tanh activations functions are preferred over sigmoid functions in neural networks.

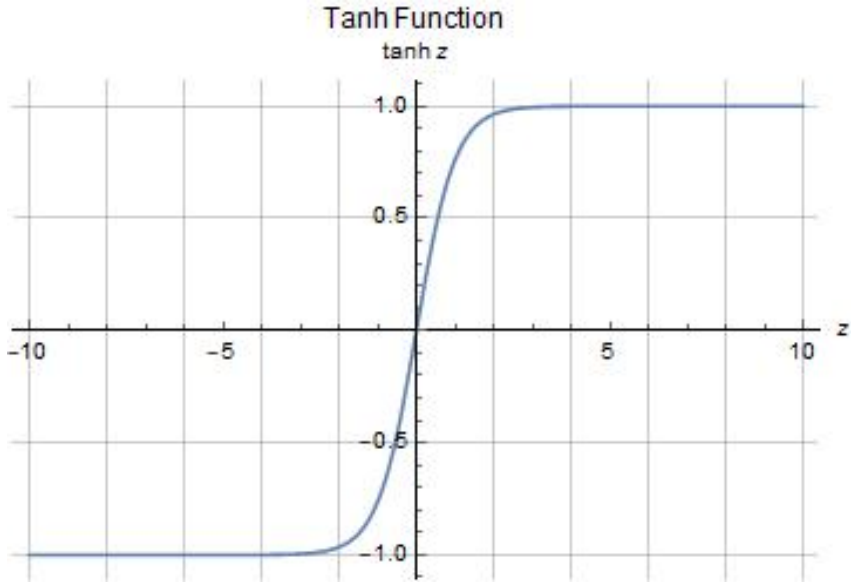
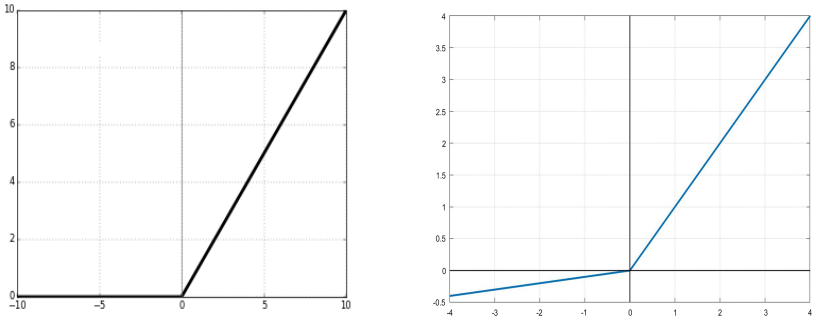


Figure 3.8: Tanh function

ReLU Function: ReLU stands for Rectified Linear unit and is represented by the following equation:

$$ReLU(z) = max(0, z) \tag{3.3}$$

It takes real values as an input and sets all the negative values equal to zero [21]. It is computationally very efficient as no complicated math need to be performed. And it is the most commonly used activation function in neural networks as it increases the convergence rate of gradient descent by six times as compared to sigmoid and tanh activation functions. However, the function doesn't get saturated for larger values of inputs, but the neurons often stuck in a negative side giving zero local gradient and neuron will never get a chance to update in gradient descent learning process. This problem is also known as dying ReLU and it often occurs due to high learning rate or



(a) (b)

Figure 3.9: ReLU and Leaky ReLU Function

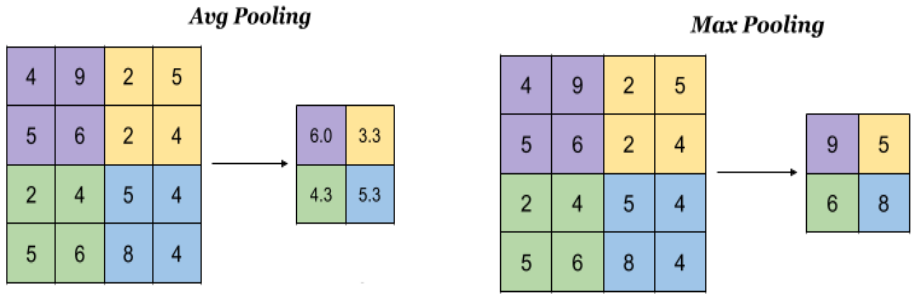
there is a large negative bias. Leaky ReLU can be used to avoid this situation. Leaky ReLU is very similar to original ReLU and the only difference is that now instead of being flat in the negative regime, we are going to give slight negative slope here. This solves a lot of problems we mentioned earlier as it doesn't saturate in the negative space and still very computationally efficient [21].

3.1.2 Pooling Layer

Pooling layer follows the non-linear function introduced in convolutional layer in CNN. It simply down samples the feature map to speed up the computation as well as make some of the features it detects a bit more robust [19]. Rest of the operations will be performed on the summarized features so that the feature map is no more sensitive to the precise location of features in the input. Two most commonly used pooling techniques are described below:

3.1.2.1 Average Pooling

In average pooling, we simply place the filter window and find the average of pixel values covered by the filter on the feature map. We simply keep on sliding the filter and repeat the process until entire input image is processed. The average pooling technique with 2 x 2 filter size and stride value of 2 is illustrated in 3.10a.



(a) (b)

Figure 3.10: Average and Max Pooling

3.1.2.2 Max Pooling

It is a pooling technique in which the most prominent features in the region of feature map are retained. We simply take the maximum value present under filter and process the entire feature map to get a down-sampled output as shown in 3.10b [22].

3.1.3 Flatten Layer

The flatter layer acts as bridge between last convolutional or pooling layer and fully connected layer. The function of the flatten layer is to simply take a two-dimensional feature map and stretch it into a long column vector that acts as an input to the fully connected layer of artificial neural network.

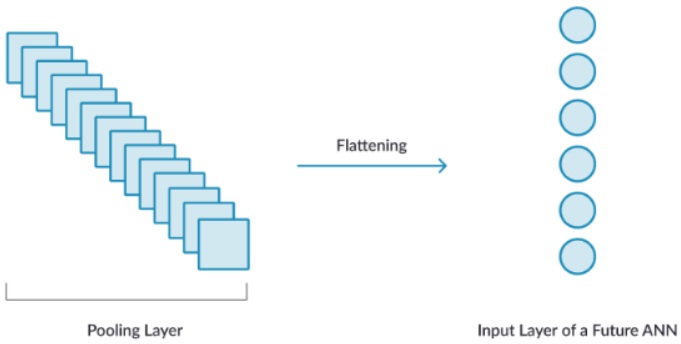


Figure 3.11: Flatten layer

3.1.4 Fully Connected Layer

Fully connected layer is considered as an important part of CNNs and gives the final probabilities for each class. The output feature value vector obtained by the flatting layer is given as input to the fully connected layer. It classifies the input image into various classes using these features. The output is simply a vector of m $\times$ 1 dimension giving the probability of the classification label model is trying to predict. The probabilities of the classes are obtained using Softman activation function which simply squashes the input values between 0 and 1 that sums to one.

Thus we conclude that CNNs are divided into two separate categories i.e. feature learning and classification. In feature learning part, data is passed through convolutional layer, activation functioin and pooling layer over and over again to extract several types of features from the images. The resulting feature matrix is passed to classification layer where it is flattened to a single vector and get probability for each class as an output of fully connected layer.

Chapter 4

Training & Testing of the Classifier

Recall from the last chapter that Convolutional Neural Networks (CNNs) are used in the training of a model for a classifier. Classification is the process of putting a tag on an image from a set of class labels. This chapter is about the training of classifier for a dataset of images of five classes. The step by step process of training a classifier is explained in detail in this chapter. Results obtained from the testing of the classifier by varying certain parameters are recorded and attached in the latter part of the chapter.

4.1 Formation of Dataset

In deep learning techniques for classification of images, we have images examples which are input to to the system for training purposes to train a model and then it is tested for a set of images to get label from the given classes. This set of images is called dataset. The dataset is divided into training and test dataset by a specific ratio.

In this project we have five different classes of vehciles i.e. Bus, Car, Truck, Motorcycle and Hiace. There are a total of 10000 images with 2000 images of each class. These images are downloaded from Google search results and from already available datasets and from websites providing licensed free images like Flickr. APIs are available which can be used along with a simple Python script to download a lot of images. Using matplotlib in

python, we can plot some of the images with the output as shown in listing 1 and plot is attached in figure 4.1.

```
import matplotlib.pyplot as plt
from matplotlib.image import imread
path  = './images/'
for i in range(9):
    plt.subplot(330+1+i)
    filename = path + 'image_'+ str(i) + '.jpg'
    image = imread(filename)
    plt.imshow(image)
    plt.show()
```

Listing 1: Python script to plot some images from each class

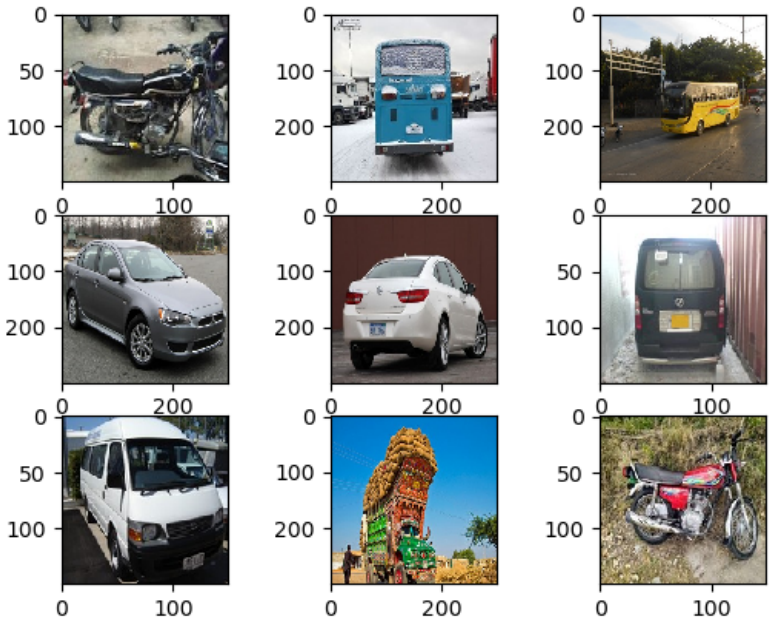


Figure 4.1: Plot of some images from each class

4.1.1 Data Augmentation

Increasing the size of dataset makes the deep learning model to learn and classify images more accurately. This problem of lack of sufficient data can be solved using technique of data augmentation which is used to increase the size of available set of images. Data augmentation is the process of applying techniques like shifting, flipping, rotating and varying the brightness etc on images to introduce variations among them. This technique is used within the code using Keras.

4.2 Setting of Workspace for the Project

There are a lot of images to be processed. To process images, high end GPU is needed. Due to unavailability of an Nvidia GPU on the system, we used Google Colab. It is an online platform having high end GPUs free provided by Google for training up to 12 hours. Enabling a GPU in the notebook settings can accelerate the process of training.

4.3 Process of Training

Keras is one of the leading high level API for neural networks. It contains many modules such as layers, cost functions, activation functions, initialization schemes and regularization schemes. This project uses Keras for training purposes.

4.3.1 Conversion of Images and Labels to Numpy Arrays

After importing all the necessary packages, we are all set to pre process our data. Recall to chapter 2, where we studied the basics of an image. In Keras image has two representations i.e. (width, height, color_channels) and (color_channels, width, height). First approach is called as channels first approach and the latter is called as channels last approach. We will use the channels last approach. First step is to read the images from the folder mounted from Google drive, Labels are assigned to classes from 0 to 4. Keras load_img() loads image from the folder given target size. Target size for this project is chosen as 128 by 128. This means width and height, both are 128. Color channels for RGB images are 3. Images are then converted to numpy arrays using Keras img_to_array() method. Labels are stored as 0, 1, 2, 3, 4 based on the class of image (as data is renamed). A section of code for this implementation is shown in listing 2.

```
folder = '/content/Input_data/'
images = np.zeros((num_of_images,img_rows,img_cols,color_channels))
labels = np.zeros(num_of_images)
i = 0
# enumerate files in the directory
for file in os.listdir(folder):
# determine class
    if file.startswith('bus'):
        output = 0
```

```
    elif file.startswith('car'):
        output = 1
    elif file.startswith('bike'):
        output = 2
    elif file.startswith('hiace'):
        output = 3
    else:
        output = 4
    # load image
    img = load_img(folder + file, target_size=(img_rows,img_cols))
    # convert to numpy array
    img = img_to_array(img)
    # store
    images[i] = img
    labels[i] = output
    i = i+1
```

Listing 2: Conversion of Images to Numpy arrays

The images array has a pattern (num_of_images, rows, cols, color_channels). It means that it has a shape (10000, 128, 128, 3). This is a 4D tensor. There are 10000 images with each image having 128, 128×3 matrices i.e. each 128×3 matrix has 128 rows with each row containing 3 columns with information of 3 colors. The labels array is a 1D tensor with 10000 labels.

4.3.2 Train-Test Division

We have to divide the data into two parts, train and test. The training data is used to for training purposes and when the model is trained then it is used to test the model using the test images and test labels. We used a ratio of 75% and 25% for train and test respectively. The package used for this purpose is the Scikit-learn train_test_split. Listing 3 shows splitting of data into train and test.

```
train_images, test_images, train_labels, test_labels =
train_test_split(images, labels, test_size = 0.25, random_state = 42)
```

Listing 3: Train-test split

There is another technique for splitting of data. We make two directories train and test and randomly divide the images into these directories and then convert them into numpy array. Here we have first converted the images into arrays, split them into train

and test and then we can save them back into directories by converting them into images from array using PIL from_array function.

4.3.3 Mean Correction

After converting to float 32 and normalizing by 255, we have to subtract mean pixel. Listing 4 illustrates the mean correction.

```
train_images = train_images.astype('float32') / 255
test_images = test_images.astype('float32') / 255

if subtract_pixel_mean:
    x_train_mean = np.mean(train_images, axis=0)
    train_images -= x_train_mean
    test_images -= x_train_mean

train_labels = to_categorical(train_labels,num_classes)
test_labels = to_categorical(test_labels,num_classes)
```

Listing 4: Mean correction

The last two lines are categorical labeling for using labels fit for Keras.

4.3.4 Defining the Model

Recall from chapter 3, a CNN model consists of various layers including conv2D, Activation, MaxPoooling2D, Flatten and Dense layer. Listing 5 defines the training Model.

```
#Model
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=train_images.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(64))
```

```
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(num_classes))
model.add(Activation('softmax'))

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

Listing 5: Defining the Model

Last three lines defines the compilation process of the Model. For more than two classes, loss is categorical_crossentropy. For two classes it is binary_crossentropy. There are many optimizers in Keras i.e. SGD, Adam, RMSprop, Adamax etc. Here we have used Adam. Accuracy metrics calculates how often prediction equals labels.

4.3.5 Fitting the Model

Here the process of training starts. We have two functions used in Keras. One is model.fit and other is model.fit_generator. As we discussed in section 4.1.1 about data augmentation. Here we have implemented data augmentation using a an if-else logic. We have also defined a validation set. Validation data tells us about the behavior of the model to an unseen data.

The number of times a model goes through the data is termed as 'epochs'. We have to determine the specific number of epochs for which the training process has to run. If we run the process above that number, the training accuracy will increase. But when we test that model with the test data, the accuracy of the model will be lower than the training accuracy. This process is called '*overfitting*'. We have to avoid the overfitting. We use validation data which shows us the validation accuracy and loss after each epoch. Listing 6 shows the code for training process.

```
if not data_augmentation:
    print('Not using data augmentation.')
    history = model.fit(train_images, train_labels, validation_data =
    (test_images, test_labels), batch_size = batch_size, epochs = n_epochs)
else:
    datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
```

```
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        zca_epsilon=1e-06,  # epsilon for ZCA whitening
        rotation_range=0,  # randomly rotate images in the range (degrees, 0 to 180)
        # randomly shift images horizontally (fraction of total width)
        width_shift_range=0.1,
        # randomly shift images vertically (fraction of total height)
        height_shift_range=0.1,
        shear_range=0.,  # set range for random shear
        zoom_range=0.,  # set range for random zoom
        channel_shift_range=0.,  # set range for random channel shifts
        # set mode for filling points outside the input boundaries
        fill_mode='nearest',
        cval=0.,  # value used for fill_mode = "constant"
        horizontal_flip=True,  # randomly flip images
        vertical_flip=False,  # randomly flip images
        # set rescaling factor (applied before any other transformation)
        rescale=None,
        # set function that will be applied on each input
        preprocessing_function=None,
        # image data format, either "channels_first" or "channels_last"
        data_format=None,
        # fraction of images reserved for validation (strictly between 0 and 1)
        validation_split=0.0)

    datagen.fit(train_images)

    history = model.fit_generator(datagen.flow(train_images, train_labels,
    batch_size = batch_size),validation_data=(test_images, test_labels),
    epochs = n_epochs)
```

Listing 6: Training the Model

4.3.6 Plotting the Accuracy and Loss

A specific model has a specific accuracy upto which it can detect an unknown data. Increasing epochs will not increase the accuracy, it basically overfits the model. For a certain model, we have to find a specific number of epochs. For this we plot the accuracy and loss of training and validation data. Listing 7 shows the script for plotting the accuracy and loss of training and validation data.

```
plt.grid()
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
```

```
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.grid()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

Listing 7: Training, validation accuracy & loss vs. epochs

4.3.7 Saving Model & Weights

After successful training of model, we have to resuse it. We can directly test it for one time when the Colab is not recycled. So the better approach is to save the model and then evalaute it on test data. Model is stored as .h5 file in the root directory. Listing 8 shows the script for saving the weights.

```
# Save model and weights
if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
print('Saved trained model at %s ' % model_path)
```

Listing 8: Saving the Model

4.4 Evaluation of Model

Loading the Model saved in the root directory and test it using evalaute function tells us the accuracy of Model on the test data. Listing 9 shows the script for evalaution.

```
#Load Model
loaded_model = load_model("saved_models/model.h5")
#Evaluate Model
test_loss, test_acc = loaded_model.evaluate(test_images, test_labels)
print('Accuarcy of the model:', test_acc)
```

Listing 9: Evaluating the Model

4.4.1 Two Class Problem & Results

We started with two class problem, Bus & Car and recorded the results from 5 to 30 epochs. The loss was binary_crossentropy in this case. The training accuracy recorded after 20 epochs was 95.08% and training loss was 14.7%. The test set was choosen as validation data and validation accuracy after 5 epochs was 96.10% and validation loss was 10.48%. Here we attach the results only for 5 epochs and 20 epochs in figure 4.2 and 4.3.

```
Epoch 1/5
94/94 [==============================] - 10s 105ms/step - loss: 0.6449 - acc: 0.6364 - val_loss: 0.4691 - val_acc: 0.8290
Epoch 2/5
94/94 [==============================] - 9s 101ms/step - loss: 0.5267 - acc: 0.7553 - val_loss: 0.3685 - val_acc: 0.8500
Epoch 3/5
94/94 [==============================] - 9s 100ms/step - loss: 0.4414 - acc: 0.8023 - val_loss: 0.2771 - val_acc: 0.8880
Epoch 4/5
94/94 [==============================] - 9s 100ms/step - loss: 0.3933 - acc: 0.8338 - val_loss: 0.3090 - val_acc: 0.8590
Epoch 5/5
94/94 [==============================] - 9s 100ms/step - loss: 0.3562 - acc: 0.8488 - val_loss: 0.2304 - val_acc: 0.9090
```

Figure 4.2: Training results for two class problem after 5 epochs

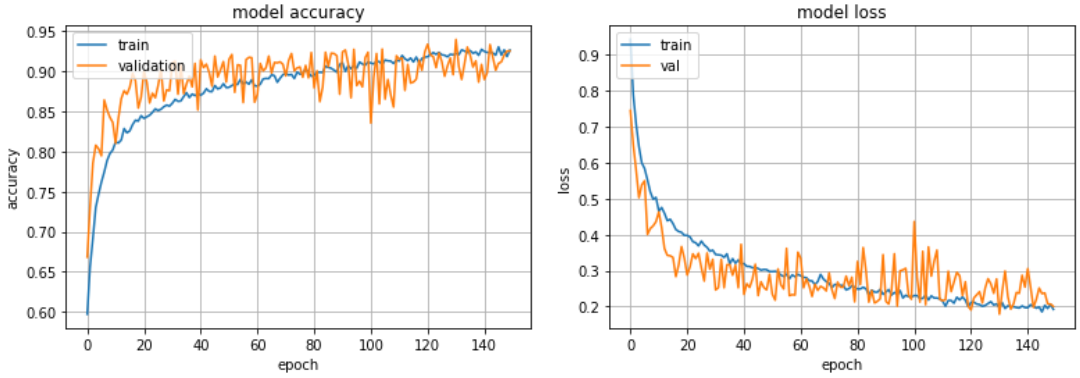
```
Epoch 1/20
94/94 [==============================] - 10s 105ms/step - loss: 0.6554 - acc: 0.6458 - val_loss: 0.5158 - val_acc: 0.755(
Epoch 2/20
94/94 [==============================] - 9s 101ms/step - loss: 0.5238 - acc: 0.7563 - val_loss: 0.4120 - val_acc: 0.8290
Epoch 3/20
94/94 [==============================] - 9s 101ms/step - loss: 0.4327 - acc: 0.8170 - val_loss: 0.2812 - val_acc: 0.8780
Epoch 4/20
94/94 [==============================] - 9s 100ms/step - loss: 0.3903 - acc: 0.8299 - val_loss: 0.2476 - val_acc: 0.9110
Epoch 5/20
94/94 [==============================] - 10s 101ms/step - loss: 0.3464 - acc: 0.8616 - val_loss: 0.2508 - val_acc: 0.906(
Epoch 6/20
94/94 [==============================] - 10s 102ms/step - loss: 0.3114 - acc: 0.8829 - val_loss: 0.2160 - val_acc: 0.921(
Epoch 7/20
94/94 [==============================] - 9s 100ms/step - loss: 0.2733 - acc: 0.8934 - val_loss: 0.1962 - val_acc: 0.9400
Epoch 8/20
94/94 [==============================] - 10s 102ms/step - loss: 0.2582 - acc: 0.8979 - val_loss: 0.1478 - val_acc: 0.953(
Epoch 9/20
94/94 [==============================] - 10s 101ms/step - loss: 0.2330 - acc: 0.9130 - val_loss: 0.1551 - val_acc: 0.943(
Epoch 10/20
94/94 [==============================] - 9s 101ms/step - loss: 0.2086 - acc: 0.9162 - val_loss: 0.1309 - val_acc: 0.9540
Epoch 11/20
94/94 [==============================] - 9s 100ms/step - loss: 0.1799 - acc: 0.9331 - val_loss: 0.1407 - val_acc: 0.9510
Epoch 12/20
94/94 [==============================] - 9s 98ms/step - loss: 0.2114 - acc: 0.9249 - val_loss: 0.1341 - val_acc: 0.9610
Epoch 13/20
94/94 [==============================] - 9s 97ms/step - loss: 0.1806 - acc: 0.9299 - val_loss: 0.1236 - val_acc: 0.9610
Epoch 14/20
94/94 [==============================] - 9s 98ms/step - loss: 0.1783 - acc: 0.9359 - val_loss: 0.1221 - val_acc: 0.9580
Epoch 15/20
94/94 [==============================] - 9s 96ms/step - loss: 0.1722 - acc: 0.9337 - val_loss: 0.1472 - val_acc: 0.9500
Epoch 16/20
94/94 [==============================] - 9s 97ms/step - loss: 0.1659 - acc: 0.9372 - val_loss: 0.2406 - val_acc: 0.9190
Epoch 17/20
94/94 [==============================] - 9s 97ms/step - loss: 0.1694 - acc: 0.9407 - val_loss: 0.1144 - val_acc: 0.9630
Epoch 18/20
94/94 [==============================] - 9s 99ms/step - loss: 0.1596 - acc: 0.9434 - val_loss: 0.1426 - val_acc: 0.9560
Epoch 19/20
94/94 [==============================] - 9s 100ms/step - loss: 0.1499 - acc: 0.9404 - val_loss: 0.1075 - val_acc: 0.9640
Epoch 20/20
94/94 [==============================] - 9s 97ms/step - loss: 0.1479 - acc: 0.9508 - val_loss: 0.1048 - val_acc: 0.9610
```

Figure 4.3: Training results for two class problem after 20 epochs

After 20 epochs for this model, the validation loss increases which indicate that we have to train that model up to 20 epochs.

4.4.2 Five Class Probem & Results

We started with two class problem and extended the model to five classes which is our desired task. The loss function in this case is categorical_crossentropy. The model explained in listing 5 is used for training. We started with 20 epochs, then 50, 80, 100 and 150. The plot of training and validation accuracy and training and validation loss is plotted using Python matplotlib and it is observed that nearly after 80 epochs, the validation loss increases and validation accuracy decreases. The plots are shown in figure 4.4. The



(a) Training & validation accuracy (b) Training & validation loss

Figure 4.4: Model accuracy & model loss

loss was binary_crossentropy in this case. The training accuracy recorded after 80 epochs was 95.08% and training loss was 14.7%. The test set was choosen as validation data and validation accuracy after 5 epochs was 96.10% and validation loss was 10.48%. Here we attach the results only for 20 epochs and 80 epochs in figure 4.5 and 4.6. The results for 80 epochs are attached for only last 10 epochs.

```
Epoch 1/20
235/235 [==============================] - 35s 147ms/step - loss: 0.9324 - accuracy: 0.5967 - val_loss: 0.7398 - val_accuracy: 0.7360
Epoch 2/20
235/235 [==============================] - 28s 121ms/step - loss: 0.7528 - accuracy: 0.6653 - val_loss: 0.5637 - val_accuracy: 0.7740
Epoch 3/20
235/235 [==============================] - 29s 122ms/step - loss: 0.6938 - accuracy: 0.6999 - val_loss: 0.5678 - val_accuracy: 0.7712
Epoch 4/20
235/235 [==============================] - 29s 122ms/step - loss: 0.6440 - accuracy: 0.7251 - val_loss: 0.5629 - val_accuracy: 0.7864
Epoch 5/20
235/235 [==============================] - 29s 122ms/step - loss: 0.6041 - accuracy: 0.7436 - val_loss: 0.4861 - val_accuracy: 0.8328
Epoch 6/20
235/235 [==============================] - 29s 122ms/step - loss: 0.5852 - accuracy: 0.7587 - val_loss: 0.5316 - val_accuracy: 0.7756
Epoch 7/20
235/235 [==============================] - 29s 122ms/step - loss: 0.5691 - accuracy: 0.7679 - val_loss: 0.4899 - val_accuracy: 0.8084
Epoch 8/20
235/235 [==============================] - 29s 122ms/step - loss: 0.5456 - accuracy: 0.7732 - val_loss: 0.3831 - val_accuracy: 0.8624
Epoch 9/20
235/235 [==============================] - 29s 121ms/step - loss: 0.5181 - accuracy: 0.7889 - val_loss: 0.4162 - val_accuracy: 0.8552
Epoch 10/20
235/235 [==============================] - 29s 122ms/step - loss: 0.4982 - accuracy: 0.8040 - val_loss: 0.3896 - val_accuracy: 0.8660
Epoch 11/20
235/235 [==============================] - 29s 123ms/step - loss: 0.4937 - accuracy: 0.8017 - val_loss: 0.3648 - val_accuracy: 0.8604
Epoch 12/20
235/235 [==============================] - 29s 122ms/step - loss: 0.4857 - accuracy: 0.8023 - val_loss: 0.3743 - val_accuracy: 0.8668
Epoch 13/20
235/235 [==============================] - 28s 121ms/step - loss: 0.4550 - accuracy: 0.8164 - val_loss: 0.4719 - val_accuracy: 0.7876
Epoch 14/20
235/235 [==============================] - 28s 121ms/step - loss: 0.4607 - accuracy: 0.8219 - val_loss: 0.3511 - val_accuracy: 0.8712
Epoch 15/20
235/235 [==============================] - 29s 121ms/step - loss: 0.4440 - accuracy: 0.8244 - val_loss: 0.3202 - val_accuracy: 0.8796
Epoch 16/20
235/235 [==============================] - 28s 121ms/step - loss: 0.4318 - accuracy: 0.8247 - val_loss: 0.3210 - val_accuracy: 0.8752
Epoch 17/20
235/235 [==============================] - 28s 121ms/step - loss: 0.4246 - accuracy: 0.8367 - val_loss: 0.3730 - val_accuracy: 0.8464
Epoch 18/20
235/235 [==============================] - 29s 122ms/step - loss: 0.4312 - accuracy: 0.8261 - val_loss: 0.3332 - val_accuracy: 0.8696
Epoch 19/20
235/235 [==============================] - 29s 122ms/step - loss: 0.4113 - accuracy: 0.8385 - val_loss: 0.3146 - val_accuracy: 0.8844
Epoch 20/20
235/235 [==============================] - 29s 122ms/step - loss: 0.4211 - accuracy: 0.8341 - val_loss: 0.3443 - val_accuracy: 0.8620
```

Figure 4.5: Training results for five class problem after 20 epochs

```
Epoch 71/80
235/235 [==============================] - 29s 122ms/step - loss: 0.2596 - accuracy: 0.9012 - val_loss: 0.2835 - val_accuracy: 0.8900
Epoch 72/80
235/235 [==============================] - 28s 121ms/step - loss: 0.2638 - accuracy: 0.8947 - val_loss: 0.1971 - val_accuracy: 0.9300
Epoch 73/80
235/235 [==============================] - 29s 122ms/step - loss: 0.2730 - accuracy: 0.8983 - val_loss: 0.2692 - val_accuracy: 0.8940
Epoch 74/80
235/235 [==============================] - 29s 122ms/step - loss: 0.2502 - accuracy: 0.9029 - val_loss: 0.2214 - val_accuracy: 0.9188
Epoch 75/80
235/235 [==============================] - 29s 121ms/step - loss: 0.2628 - accuracy: 0.8993 - val_loss: 0.2118 - val_accuracy: 0.9232
Epoch 76/80
235/235 [==============================] - 29s 121ms/step - loss: 0.2552 - accuracy: 0.9015 - val_loss: 0.2094 - val_accuracy: 0.9208
Epoch 77/80
235/235 [==============================] - 29s 122ms/step - loss: 0.2534 - accuracy: 0.8991 - val_loss: 0.2551 - val_accuracy: 0.8988
Epoch 78/80
235/235 [==============================] - 28s 120ms/step - loss: 0.2619 - accuracy: 0.8985 - val_loss: 0.3005 - val_accuracy: 0.8792
Epoch 79/80
235/235 [==============================] - 28s 121ms/step - loss: 0.2688 - accuracy: 0.8907 - val_loss: 0.2306 - val_accuracy: 0.9148
Epoch 80/80
235/235 [==============================] - 28s 120ms/step - loss: 0.2544 - accuracy: 0.9003 - val_loss: 0.2386 - val_accuracy: 0.9096
```

Figure 4.6: Training results for five class problem after 80 epochs

4.5 Graphical User Interface

To observe the results in a concrete way, we have to pass test images one by one and obtain the corresponding labels. For this purpose, we made a Graphical User Interface (GUI) using tkinter package of Python. As discussed earlier that we have saved the model

in a folder named as saved_models in the root directory. This model can be downloaded to local disks of the computer (As tkinter uses the GUI of computer, it can not run on Colab). Then from the test directory, all the images one by one are passed and corresponding labels are displayed. An example image is shown in figure **??**.

References

[1] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking, in computer vision and pattern recognition," *IEEE Computer Society Conference on, Fort Collins*, vol. 2, p. 252, 1999.

[2] H. Wang, Y. Yu, Y. Cai, L. Chen, and X. Chen, "A vehicle recognition algorithm based on deep transfer learning with a multiple feature subspace distribution," *Sensors*, vol. 18, p. 4109, 11 2018.

[3] L.G. Roberts, "Machine perception of three-dimensional solids," 1963.

[4] S. E. Palmer, *Vision Science – Photons to Phenomenology.* 1st ed., 1999.

[5] D. G. Lowe, "Three-dimensional object recognition from single two-dimensional images," 1987.

[6] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," 2004.

[7] I. S. Alex Krizhevsky and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," 2012.

[8] L. Tan and J. Jiang, *Image basics Digital signal Processing Fundamentals & Applications.* 2nd ed., 2013.

[9] S. W. Smith, *The Engineers & scientists guide to Digital Signal Processing.* 2nd ed., 1999.

[10] S. W. J. Nursabillilah Mohd AliI *et al.*, "Object classification and recognition using bag-of-words (bow) model," 4 - 6 March 2016.

[11] A. Khan, A. Sohail, *et al.*, "A survey of the recent architectures of deep convolutional neural networks,"

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," 1998.

[13] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," 2013.

[14] C. Szegedy, W. Liu, *et al.*, "Going deeper with convolutions," 2014.

[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scaleimage recognition," 2015.

[16] K. He, X. Zhang, *et al.*, "Deep residual learning for image recognition," 2015.

[17] M. Mishra and M. Srivastava, "A view of artificial neural network," 1-2 Aug.2014.

[18] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *ArXiv e-prints*, 11 2015.

[19] S. Indolia, A. K. Goswami, S. Mishra, and P. Asopa, "Conceptual understanding of convolutional neural network- a deep learning approach," *Procedia Computer Science*, vol. 132, pp. 679 – 688, 2018. International Conference on Computational Intelligence and Data Science.

[20] A. Olgac and B. Karlik, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," *International Journal of Artificial Intelligence And Expert Systems*, vol. 1, pp. 111–122, 02 2011.

[21] J. Gua, Z. Wangb, *et al.*, "Recent advances in convolutional neural networks," vol. 6, 2017.

[22] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," 2010.