

COMPUTER VISION BASED ROADWAYS SURVEY SYSTEM USING ARTIFICIAL INTELLIGENCE

A Project Report Phase – II

Submitted in partial fulfillment of the requirements for the award of Bachelor of
Engineering Degree in Electronics and Communication Engineering

by

AVANTHIKA M (39130046)

ARTHIESWARI P (39130038)



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

SCHOOL OF ELECTRICAL AND ELECTRONICS

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited with Grade “A” by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI-600119

APRIL - 2023



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC
Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600 119

www.sathyabama.ac.in

DEPARTMENT OF ELECTRONICS AND COMMUNICAITON ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of AVANTHIKA M (39130046) and ARTHIESWARI P (39130038) who carried out the project entitled "COMPUTER VISION BASED ROADWAYS SURVEY SYSTEM USING ARTIFICIAL INTELLIGENCE" under my supervision from November 2022 to April 2023.


Internal Guide

DR.V.VIJAYAKUMAR, MTech., Ph.D.,

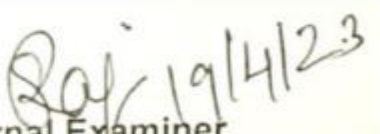
ASSOCIATE PROFESSOR


Head of Department

Dr. T. RAVI M.E., Ph.D.,

Submitted for Viva voce Examination held on _____


Internal Examiner

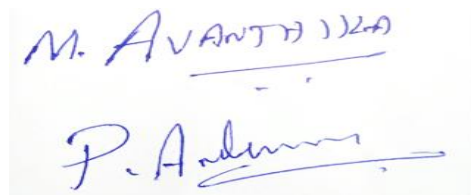

External Examiner

DECLARATION

We, AVANTHIKA M (39130046) and ARTHIESWARI P (39130038) hereby declare that the project report entitled “COMPUTER VISION BASED ROADWAYS SURVEY SYSTEM USING ARTIFICIAL INTELLIGENCE” done by us under the guidance of DR.V.VIJAYAKUMAR ,M.Tech,Ph.D,Associate professor, Department of Electronics and Communication Engineering, SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI, is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Electronics and Communication Engineering.

DATE:

PLACE:

The image shows two handwritten signatures in blue ink. The top signature is 'M. AVANTHIKA' and the bottom signature is 'P. Arthieswari'. Both signatures are underlined.

SIGNATURE OF THE CANDIDATES

ACKNOWLEDGEMENT

We are pleased to acknowledge our sincere thanks to the Board of Management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. We are grateful to them.

We convey our thanks to **Dr. N. M. NANDHITHA, M.E., Ph.D. Dean, School of Electrical and Electronics Engineering** and **Dr. T. RAVI, M.E., Ph.D. Head of the Department, Department of Electronics and Communication Engineering** for providing us necessary support and details at the right time during the progressive reviews.

We would like to express my sincere and deep sense of gratitude to our Project Guide **Dr.V.VIJAYAKUMAR, M.Tech., Ph.D., Associate professor, Dept. of Electronics and Communication Engineering**, for his valuable guidance, suggestions and constant encouragement for the successful completion of our project work.

We wish to express my thanks to all Teaching and Non-teaching staff members of the Department of Electronics and Communication Engineering who were helpful in many ways for the completion of the project.

ABSTRACT

Everyday, an accident occurs on the road due to unanticipated vehicle interference and unpredictable driving by the driver. There is a tonne of study on predicting and detecting auto accidents, but the drivers are never informed beforehand. As a result, a tremendous number of people perish in streetcar accidents every day. Usually, it happens as a result of a driver's negligence or a tardy reaction from the emergency services. For injured persons to be saved, a reliable identification and data correspondence system for traffic accidents is required. It is not conceivable for a framework to provide crisis management organisations with data messages about an accident location in a speedier and more efficient manner. Several scientists have put forth various frameworks for coding accident recognition in exploration writing. Accident identification is made easier with the help of cell phones, GSM, and GPS technologies. Vehicles can be accommodated in specially designed organizations, and mobile applications are also available. Each vehicle must operate a programmed street accident recognition and data communication structure. The employment of sensor technology was extensively covered in research publications, along with attempts to identify mishaps as a result of using AI and computer vision from reconnaissance systems. Vehicles can be accommodated in specially designed organizations, and mobile applications are also available. Each vehicle must operate a programmed street accident recognition and data communication structure. The employment of sensor technology was extensively covered in research publications, along with attempts to identify mishaps as a result of using AI and computer vision from reconnaissance systems.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO.
	ABSTRACT	v
	LIST OF FIGURES	vii
	LIST OF ABBREVIATIONS	viii
	INTRODUCTION	1
	1.1 About Deep Learning	2
	1.2 Convolutional neural networks	5
1	1.3 Reconcurent convolutional neural network	7
	1.4 Fast-Reconcurent convolutional neural network	9
	1.5 Computer vision	11
	1.6 Image Processing	11
2	LITERATURE SURVEY	
	2.1 Literature Survey	13
3	METHODOLOGY	
	3.1 Existing Method	16
	3.2 Proposed Work	16
4	SOFTWARE PACKAGES	
	4.1 Software Packages for Neural Networks	18
5	RESULTS AND DISCUSSION	42
6	CONCLUSION	43
	REFERENCES	45
	APPENDIX	
	A.Source code	57
	B.Output images	53

LIST OF FIGURES

FIGURE NO	TITLE	PAGE NO
1.3	R-CNN MODEL	8
1.4	Fast R-CNN	10
4.1	Tensorflow	18
4.2	Pytorch	23
4.3	Denserly connected network	23
4.4	Capsnet	24
4.5	Fractalnet	25
4.6	Network in network	29
4.9	ZFnet	34
4.10	GoogLeNet	35
4.11	VGGNET	36
4.12	LENET	38

LIST OF ABBREVIATIONS

RCM	-	Road condition monitoring
CNN	-	Convolutional neural network
CAP	-	Capsule neural network
ANN	-	Artificial neural network
SIANN	-	Shift invariant or space invariant artificial neural network
R-CNN	-	Recurrent convolutional neural network
DNN	-	Deep neural network
MFSNN	-	Multi deep neural network fusion siamese neural network
CPU	-	Central processing unit
GPU	-	Graphical processing unit
API	-	Application peripheral interface
CAPSNET	-	Capsule network
NIN	-	Network in Network
RESNET	-	Residual Network
VGGNET	-	Visual geometry group network
IOT	-	Internet of Things
AV	-	Autonomous Vehicle

CHAPTER-1

INTRODUCTION

In order to manage a vast network of transportation infrastructures, Road Condition Monitoring (RCM) has been a challenging and strategic study field. Innovative pavement distress evaluation tools have been developed recently thanks to improvements in computer vision and data mining approaches as well as powerful computing resources for processing the heavy data. The bulk of these systems use measured data to analyse, identify, and pinpoint exact pavement distresses using next-generation distributed sensors and vision-based artificial intelligence (AI) approaches. Next-generation cameras and sensors, which include the image processing, are used in the study and execution of this project. The proposed work is not limited to identify the vandalized road, it further pushes the data i.e., the road on this place is damaged, to the cloud. In the succeeding years, all the cars will be connected, so if there is a problem on an expressway it will send the data to the other cars which will be coming on the same route.

Due to yearly economic damages, India's economic output is reduced by 2.5% as a result of an increase in road deaths. losses. Real street accidents happen once every two seconds in the 16 people often perish on Indian streets, according to the Ministry's Bureau of National Crime Records of Road Transport and Highway, India's Law Commission. India has 238 562 fatalities worldwide in 2013 and the 137 572 traffic deaths were reported. Amount of 16.6 road fatalities per 100,000 inhabitants were recorded. Accidental accidents are the leading cause of death, especially directly affecting consumer expenditure among children.

The delay in disclosing collisions with crisis centre like nearby police stations and hospitals is one of the main factors driving this Fig higher. According to, the 2030 Agenda for Sustainable Development was released in 2015 and aims to reduce the number of deaths and injuries caused by traffic collisions by at least half by the year 2020. Regarding this, the developers provide a review of recent developments that aim to spot accidents naturally and immediately warn crisis populations.

The CV exploration community now places a great deal of emphasis on CV-based scenario interpretation due to the widespread use of observation cameras. Compared to other forms of information, visual data is rich. sources of information including radar signals, GPS, and portable areas forth In light of this, it may play a crucial role. blocking/ predicting accidents, different types of deviations from the collection of quantifiable data on the traffic situation on thestreets.

Several computer vision-based studies focused on information acquisition have been conducted, such extraction scene learning, movement learning, behaviour understanding, and others. These investigations mainly discuss viewpoints like scene examination, video handling techniques, irregularity identification techniques, vehicle discovery and following, multi camera-based techniques and difficulties, action acknowledgment, traffic observing, human conduct investigation, crisis the board, occasion recognition, and so on.

1.1 DEEP LEARNING

Deep learning (also known as deep structured learning) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semisupervised or unsupervised. Deep learning architectures such as deep neural networks, deep belief networks, deep reinforcement learning, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing, machine translation, bioinformatics, drug design, medical image analysis, climate science, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance. The adjective "deep" in deep learning refers to the use of multiple layers in the network. Early work showed that a linear perceptron cannot be a universal classifier, but that a network with a non-polynomial activation function with one hidden layer of unboundedwidth can.

Deep learning is a modern variation which is concerned with an unbounded number of layers of bounded size, which permits practical application and optimized implementation, while retaining theoretical universality under mild conditions. In deep learning the layers are also permitted to be heterogeneous and to deviate widely from biologically informed connectionist models, for the sake of efficiency, trainability and understandability, whence the "structured" part. Most modern deep learning models are based on artificial neural networks, specifically convolutional neural networks (CNN)s, although they can also include propositional formulas or latent variables organized layer-wise in deep generative models such as the nodes in deep belief networks and deep Boltzmann machines.

In deep learning, each level learns to transform its input data into a slightly more abstract and composite representation. In an image recognition application, the raw input may be a matrix of pixels; the first representational layer may abstract the pixels and encode edges; the second layer may compose and encode arrangements of edges; the third layer may encode a nose and eyes; and the fourth layer may recognize that the image contains a face. Importantly, a deep learning process can learn which features to optimally place in which level on its own. This does not eliminate the need for hand-tuning; for example, varying numbers of layers and layer sizes can provide different degrees of abstraction.

The word "deep" in "deep learning" refers to the number of layers through which the data is transformed. More precisely, deep learning systems have a substantial credit assignment path (CAP) depth. The CAP is the chain of transformations from input to output. CAPs describe potentially causal connections between input and output. For a feedforward neural network, the depth of the CAPs is that of the network and is the number of hidden layers plus one (as the output layer is also parameterized). For recurrent neural networks, in which a signal may propagate through a layer more than once, the CAP depth is potentially unlimited. No universally agreed-upon threshold of depth divides shallow learning from deep learning, but most researchers agree that deep learning involves CAP depth higher than 2. CAP of depth 2 has been

shown to be a universal approximate in the sense that it can emulate any function. Beyond that, more layers do not add to the function approximate ability of the network. Deep models ($CAP > 2$) are able to extract better features than shallow models and hence, extra layers help in learning the features effectively. Deep learning architectures can be constructed with a greedy layer-by-layer method. Deep learning helps to disentangle these abstractions and pick out which features improve performance. For supervised learning tasks, deep learning methods eliminate feature engineering, by translating the data into compact intermediate representations akin to principal components, and derive layered structures that remove redundancy in representation.

Deep learning algorithms can be applied to unsupervised learning tasks. This is an important benefit because unlabelled data are more abundant than the labelled data. Examples of deep structures that can be trained in an unsupervised manner are deep belief networks. Deep neural networks are generally interpreted in terms of the universal approximation theorem or probabilistic inference. The classic universal approximation theorem concerns the capacity of feedforward neural networks with a single hidden layer of finite size to approximate continuous functions. In 1989, the first proof was published by George Cybenko for sigmoid activation functions and was generalised to feed-forward multilayer architectures in 1991 by Kurt Hornik. Recent work also showed that universal approximation also holds for non-bounded activation functions such as the rectified linear unit.

The universal approximation theorem for deep neural networks concerns the capacity of networks with bounded width but the depth is allowed to grow. Lu et al. proved that if the width of a deep neural network with ReLU activation is strictly larger than the input dimension, then the network can approximate any Lebesgue integrable function; If the width is smaller or equal to the input dimension, then a deep neural network is not a universal approximator. The probabilistic interpretation derives from the field of machine learning. It features inference as well as the optimization concepts of training and testing, related to fitting and generalization, respectively. More specifically, the probabilistic interpretation considers the activation nonlinearity as a cumulative distribution function. The probabilistic interpretation led to the

introduction of dropout as regularize in neural networks. The probabilistic interpretation was introduced by researchers including Hopfield, Widrow and Narendra and popularized in surveys such as the one by Bishop.

1.2 CONVOLUTIONAL NEURAL NETWORK

In deep learning, a convolutional neural network (CNN) is a class of Artificial Neural Network (ANN), most commonly applied to analyse visual imagery. They are also known as Shift Invariant or Space Invariant Artificial Neural Networks (SIANN), based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation equi-variant responses known as feature maps. Counter-intuitively, most convolutional neural networks are only equi-variant, as opposed to invariant, to translation. CNNs are regularized versions of multilayer perceptrons.

Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "full connectivity" of these networks make them prone to overfitting data. Typical ways of regularization, or preventing overfitting, include: penalizing parameters during training (such as weight decay) or trimming connectivity (skipped connections, dropout, etc.) CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters.

Therefore, on a scale of connectivity and complexity, CNNs are on the lower extreme. Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns to optimize the filters (or kernels) through automated learning, whereas in traditional algorithms these filters are hand-engineered.

After passing through a convolutional layer the image becomes abstracted to a featuremap, also called an activation map, with shape: (number of inputs) x (feature map height) x (feature map width) x (feature map channels). Convolutional layers convolve the input and pass its result to the next layer. This is similar to the response of a neuron in the visual cortex to a specific stimulus. Each convolutional neuron processes data only for its receptive field. Although fully connected feedforward neural networks can be used to learn features and classify data, this architecture is generally impractical for larger inputs such as high resolution images. It would require a very high number of neurons, even in a shallow architecture, due to the large input size of images, where each pixel is a relevant input feature. For instance, a fully connected layer for a (small) image of size 100 x 100 has 10,000 weights for each neuron in the second layer. Instead, convolution reduces the number of free parameters, allowing the network to be deeper.

For example, regardless of image size, using a 5 x 5 tiling region, each with the same shared weights, requires only 25 learnable parameters. Using regularized weights over fewer parameters avoids the vanishing gradients and exploding gradients problems seen during backpropagation in traditional neural networks. Furthermore, convolutional neural networks are ideal for data with a grid-like topology (such as images) as spatial relations between separate features are taken into account during convolution and/or pooling.

Convolutional networks may include local and/or global pooling layers along with traditional convolutional layers. Pooling layers reduce the dimensions of data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, tiling sizes such as 2 x 2 are commonly used. Global pooling acts on all the neurons of the feature map. There are two common types of pooling in popular use: max and average. Max pooling uses the maximum value of each local cluster of neurons in the feature map, while average pooling takes the average value. Fully connected layers connect every neuron in one layer to every neuron in another layer.

It is the same as a traditional multilayer perceptron neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images. In neural networks, each neuron receives input from some number of locations in the previous layer. In a convolutional layer, each neuron receives input from only a restricted area of the previous layer called the neuron's receptive field. Typically, the area is a square (e.g. 5 by 5 neurons). Whereas, in a fully connected layer, the receptive field is the entire previous layer. Thus, in each convolutional layer, each neuron takes input from a larger area in the input than previous layers. This is due to applying the convolution over and over, which takes into account the value of a pixel, as well as its surrounding pixels.

When using dilated layers, the number of pixels in the receptive field remains constant, but the field is more sparsely populated as its dimensions grow when combining the effect of several layers. Each neuron in a neural network computes an output value by applying a specific function to the input values received from the receptive field in the previous layer.

The function that is applied to the input values is determined by a vector of weights and a bias (typically real numbers). Learning consists of iteratively adjusting these biases and weights. The vector of weights and the bias are called filters and represent particular features of the input (e.g., a particular shape).

A distinguishing feature of CNNs is that many neurons can share the same filter. This reduces the memory footprint because a single bias and a single vector of weights are used across all receptive fields that share that filter, as opposed to each receptive field having its own bias and vector weighting.

1.3 RECURRENT BASED CONVOLUTIONAL NEURAL NETWORK(R-CNN)

R-CNN is a Convolutional Neural Network (CNN) and state-of-the-art in terms of image segmentation. This variant of a Deep Neural Network detects objects in an image and generates a high-quality segmentation mask for each instance.

The original goal of R-CNN was to take an input image and produce a set of boundingboxes as output, where each bounding box contains an object and also the category (e.g. car or pedestrian) of the object. More recently, R-CNN has been extended to perform other computer vision tasks.

Recurrent based convolutional neural networks have been used for tracking objects from a drone-mounted camera, locating text in an image, and enabling object detection.

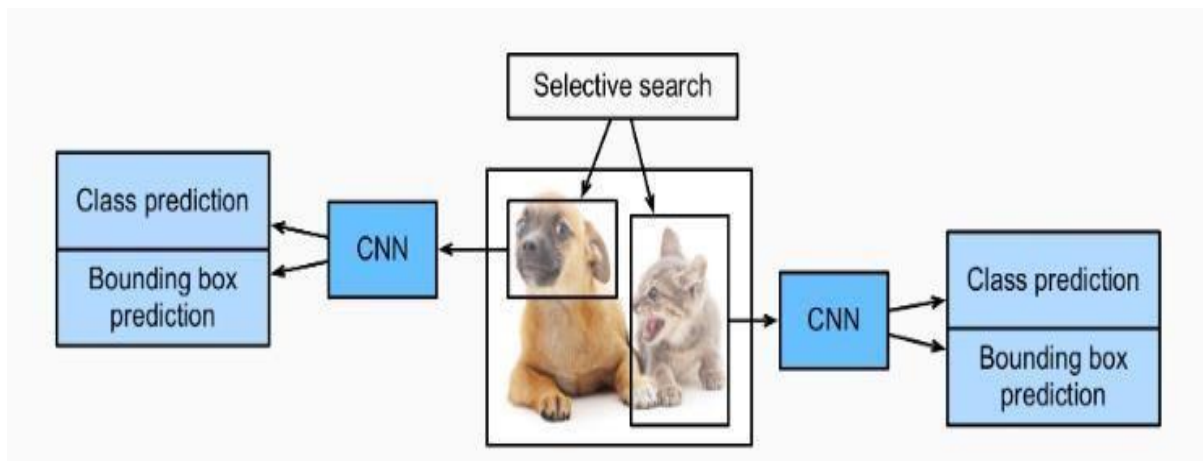


Fig 1.3: R-CNN MODEL

This shows the R-CNN model. More concretely, the R-CNN consists of the following four steps:

1. Perform *selective search* to extract multiple high-quality region proposals on the input image. These proposed regions are usually selected at multiple scales with different shapes and sizes. Each region proposal will be labeled with a class and a ground-truth bounding box.
2. Choose a pretrained CNN and truncate it before the output layer. Resize each region proposal to the input size required by the network.
3. Take the extracted features and labeled class of each region proposal as an example. Train multiple support vector machines to classify objects, where

each support vector machine individually determines whether the example contains a specific class.

4. Take the extracted features and labeled bounding box of each region proposal as an example. Train a linear regression model to predict the ground-truth bounding box.

Although the R-CNN model uses pretrained CNNs to effectively extract image features, it is slow. Imagine that we select thousands of region proposals from a single input image: this requires thousands of CNN forward propagations to perform object detection. This massive computing load makes it infeasible to widely use R-CNNs in real-world applications.

1.4 FAST RECONCURRENT CONVOLUTIONAL NEURAL NETWORK

The main performance bottleneck of an R-CNN lies in the independent CNN forward propagation for each region proposal, without sharing computation. Since these regions usually have overlaps, independent feature extractions lead to much repeated computation.

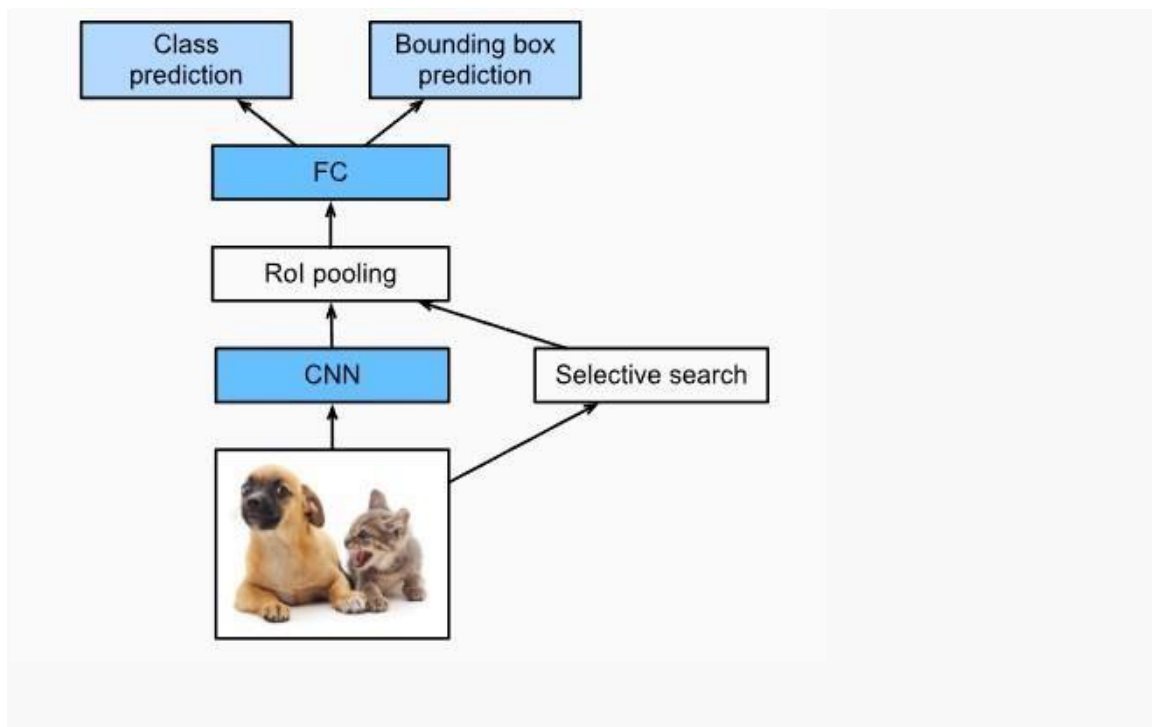


Fig 1.4: Fast R-CNN

This describes the fast R-CNN model. Its major computations are as follows:

1. Compared with the R-CNN, in the fast R-CNN the input of the CNN for feature extraction is the entire image, rather than individual region proposals. Moreover, this CNN is trainable. Given an input image, let the shape of the CNN output be $1 \times c \times h_1 \times w_1$.
2. Suppose that selective search generates n region proposals. These region proposals (of different shapes) mark regions of interest (of different shapes) on the CNN output. Then these regions of interest further extract features of the same shape (say height h_2 and width w_2 are specified) in order to be easily concatenated. To achieve this, the fast R-CNN introduces the *region of interest (RoI) pooling* layer: the CNN output and region proposals are input into this layer, outputting concatenated features of shape $n \times c \times h_2 \times w_2$ that are further extracted for all the region proposals.
3. Using a fully connected layer, transform the concatenated features into an output of shape $n \times d$, where d depends on the model design.
4. Predict the class and bounding box for each of the n region proposals. More concretely, in class and bounding box prediction, transform the fully connected layer output into an output of shape $n \times q$ (q is the number of classes) and a output of shape $n \times 4$, respectively. The class prediction uses softmax regression.

The region of interest pooling layer proposed in the fast R-CNN is different from the pooling layer introduced. In the pooling layer, we indirectly control the output shape by specifying sizes of the pooling window, padding, and stride. In contrast, we can directly specify the output shape in the region of interest pooling layer

1.5 COMPUTER VISION

Computer vision is the field of computer science that focuses on creating digital systems that can process, analyze, and make sense of visual data (images or videos) in the same way that humans do. The concept of computer vision is based on teaching computers to process an image at a pixel level and understand it. Technically, machines attempt to retrieve visual information, handle it, and interpret results through special software algorithms.

Here are a few common tasks that computer vision systems can be used for:

- **Object classification.** The system parses visual content and classifies the object on a photo/video to the defined category. For example, the system can find a dog among all objects in the image.
- **Object identification.** The system parses visual content and identifies a particular object on a photo/video. For example, the system can find a specific dog among the dogs in the image.
- **Object tracking.** The system processes video finds the object (or objects) that match search criteria and track its movement

1.6 IMAGE PROCESSING

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image. Nowadays, image processing is among rapidly growing technologies. It forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps:

- Importing the image via image acquisition tools;
- Analysing and manipulating the image;
- Output in which result can be altered image or report that is based on image analysis.

There are two types of methods used for image processing namely, analogue and digital image processing. Analogue image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques.

Digital image processing techniques help in manipulation of the digital images by using computers. The three general phases that all types of data have to undergo while using digital technique are pre-processing, enhancement, and display, information illustrates the main structural differences between VGG and NiN, and their blocks. Note both the difference in the NiN blocks (the initial convolution is followed by 1×1 convolutions, whereas VGG retains 3×3 convolutions) and in the end where we no longer require a giant fully connected layer.

NiN uses the same initial convolution sizes as AlexNet (it was proposed shortly thereafter). The kernel sizes are 11×11 , 5×5 , and 3×3 , respectively, and the numbers of output channels match those of AlexNet. Each NiN block is followed by a max-pooling layer with a stride of 2 and a window shape of 3×3 . The second significant difference between NiN and both AlexNet and VGG is that NiN avoids fully connected layers altogether. Instead, NiN uses a NiN block with a number of output channels equal to the number of label classes, followed by a *global* average pooling layer, yielding a vector of logits. This design significantly reduces the number of required model parameters, albeit at the expense of a potential increase in training time.

It has dramatically fewer parameters than AlexNet and VGG. This stems primarily from the fact that it needs no giant fully connected layers. Instead, it uses global average pooling to aggregate across all image locations after the last stage of the network body. This obviates the need for expensive (learned) reduction operations and replaces them by a simple average. What was surprising at the time is the fact that this averaging operation did not harm accuracy. Note that averaging across a low-resolution representation (with many channels) also adds to the amount of translation invariance that the network can handle. Choosing fewer convolutions with wide kernels and replacing them by 1×1 convolutions aids the quest for fewer parameters further. It affords for a significant amount of nonlinearity across channels within any given location. Both 1×1 convolutions and global average pooling significantly influenced subsequent CNN designs.

CHAPTER-2

LITERATURE SURVEY

Korada Harish Kumar(2021) presented a work on Computer vision and Machine Learning in Autonomous Vehicle An autonomous vehicle (AV) is an engineering technology that can improve transportation safety, alleviate traffic congestion, and reduce carbon emissions. The automobile industries are trying to eliminate the driving of the vehicle, which will be done by the intelligent system. An autonomous vehicle utilizes a completely automated driving system to permit the vehicle to reply to external conditions that an individual driver would manage. Research on autonomous vehicles is often classified by functionality, for instance, object detection or recognition, navigation, path planning, speed control, lane-keeping, and driver status monitoring. The research topics can also be categorized by the equipment or techniques used, for instance, computer vision, image processing, machine learning, and localization. This paper primarily focuses on computer vision and machine learning algorithms and their uses for autonomous vehicles for object detection and navigation. This paper also focuses on the future direction of machine learning for the navigation system.

Md Khurram monir rabby(2020) presented a work on Smart traffic management system using the Internet of Things (IoT). It works as middleware on the foundation of the IoT and augments the idea of the smart city through the traffic light control, smart parking, smart emergent assistance, anti-theft security system, and others. IoT provides an effective way of interactions among the web devices with the traffic embedded sensors, services, actuators, and other interconnected networks. Hence, the application of IoT in the smart traffic management system is not only limited to the reduction of the traffic congestion, air quality improvement, and traffic flow optimization but also extended to the continuous monitoring and ensuring the security and safety for the elderly people. Acquiring multiple sources of traffic information for data analysis, IoT monitors the traffic flow, controls the traffic operation and stores the correct decision for the future information presentation. Having a combination of advanced machine learning approach and data-driven technique, there are implementation limitations of this technology. However, this survey provides a good insight into the application of IoT in the smart traffic management system based on the existing research perspective.

Zahra Salsabila Hernanda(2022) presented a work at CNN-Based Hyperparameter Optimization Approach for Road Pothole and Crack Detection Systems. Poorly maintained roads contribute to the number of fatal auto accidents that occur each year. The condition of damaged roads in Indonesia reached around 2500 miles or more than 8% of the total national roads. The higher the number of road damages, the probability of a traffic accident due to road damage also rises. In order to avoid this, the roads need to be repaired in a comprehensive and long-term way. However, the way to check for road damage is still based on inefficient methods. In this paper, we propose the optimization of the detection of potholes and cracks using a deep learning convolutional neural network with a pretrained SSD MobileNet V2 model by adjusting the hyperparameter. The optimization was carried out on our previous mobile road inspection system. The effectiveness is confirmed through experiments with the optimal mAP and loss values determined by the model parameter testing process.

D C. Chuang (2019) et al. presented work on "Overtaking Vehicle Detection Based on Deep Learning and Headlight Recognition. When the daytime is full of light, the region-based convolutional neural networks (RCNN) is used to identify and track vehicles. However, due to the lack of light during the nighttime, the headlights are used for vehicle identification and tracking. When the detected position and driving direction of the vehicle satisfy the overtaking condition, the driver can be immediately warned that there will be an overtaking vehicle. In the experiment, the accuracy of the overtaking vehicle detection (precision) can reach more than 90%. Image-based overtaking vehicle detection or blind spot detection systems mainly use image recognition and computer vision related technology.

Traditional methods usually use the shape and texture of vehicles to identify vehicles. In recent years, the method of deep learning can effectively improve the detection accuracy. Currently, there are many methods for deep learning identification such as convolutional neural networks (CNN), deep neural networks (DNN), and region-based convolutional neural networks (RCNN). In terms of neural networks, multiple images of different vehicles with different angles are used as training samples to increase.

Currently, there are many methods for deep learning identification such as convolutional neural networks (CNN), deep neural networks (DNN), and region-based convolutional neural networks (RCNN). In terms of neural networks, multiple images of different vehicles with different angles are used as training samples to increase the recognition rate of RCNN. There is also a refining block used in the fast-RCNN, and the refinement block and the faster-RCNN are combined into a single network to improve the recognition rate. As for vehicle re-identification, a two-branch multi-DNN fusion Siamese neural network (MFSNN) was proposed to fuse the classification outputs of vehicle colors, model and pasted marks []. In the aerial images, fasterRCNN can also be used to detect and identify objects to reduce the difficulty of rescue. H. Leung (2014) et al. In terms of neural networks, multiple images of different vehicles with different angles are used as training samples to increase the recognition rate of RCNN. There is also a refining block used in the fast-RCNN, and the refinement block and the faster-RCNN are combined into a single network to improve the recognition rate. As for vehicle re-identification, a two-branch multi-DNN fusion Siamese neural network (MFSNN) was proposed to fuse the classification outputs of vehicle colors, model and pasted marks []. In the aerial images, fasterRCNN can also be used to detect and identify objects to reduce the difficulty of rescue.

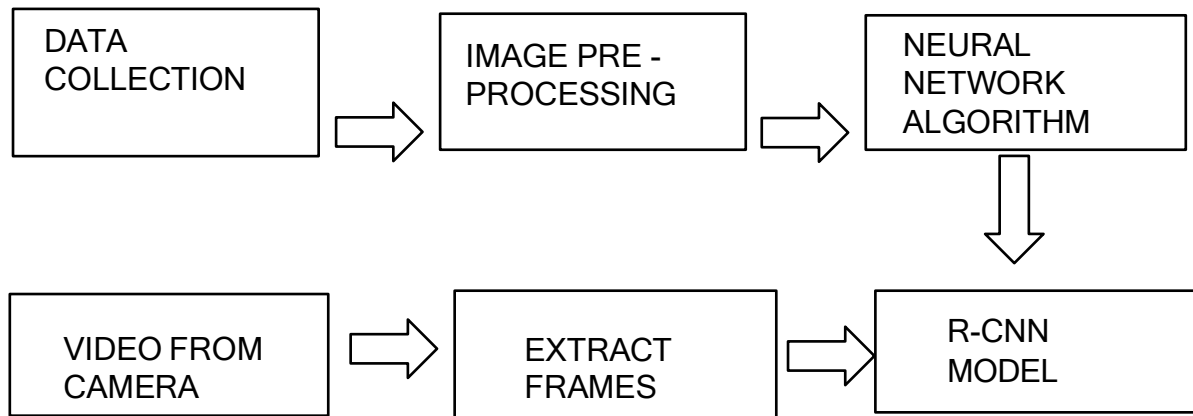
CHAPTER -3

METHODOLOGY

3.1 EXISTING METHOD

In existing system, the system detects the vandalized road condition and it has not been implemented in real time.

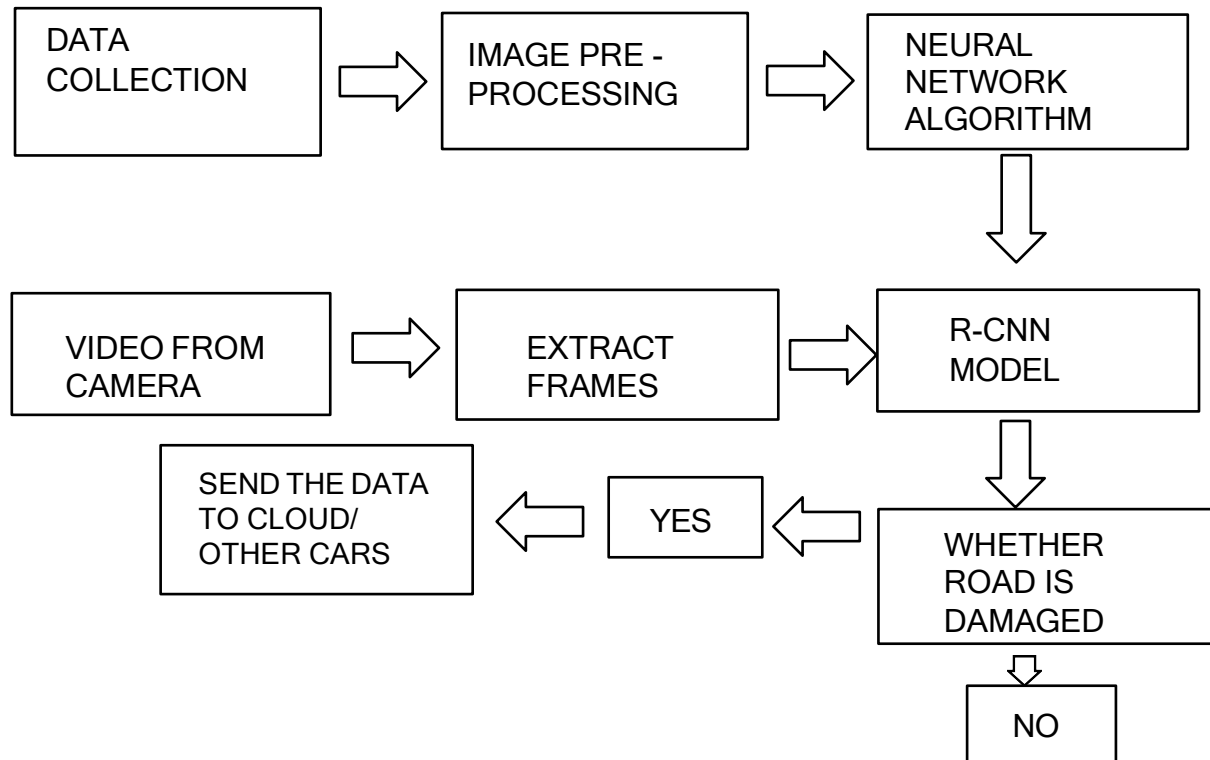
BLOCK DIAGRAM OF EXISTING WORK



3.2 PROPOSED WORK

The proposed work is more accurate than the existing model. It is also not only limited to damage detection; the damaged road data has been sent further to cloud. It helps the fellow drivers who travelling on the way (i.e., pothole) by sending an indication on their cars. This project is mainly focused that in future all cars will be connected to each other, there will be communication among them, either between them or by cloud storages. So that the other cars coming in that way will be alerted to reduce speed, or to change route.

BLOCK DIAGRAM OF PROPOSED WORK



- In this implementation using deep learning approach to image recognition which results in improved accuracy and performance. Specifically Region- based convolution neural network to recognize images.
- Using Computer vision to extract the images and neural network algorithm to compare the original image with collected data's. After identifying the roads condition.itwill push the data to cloud.
- In the upcoming years, all the vehicles will be connected, so if there is a problem on a expressway it will send the data to the other vehicles which will be coming on thesame route.

CHAPTER-4

SOFTWARE PACKAGES

4.1 SOFTWARE PACKAGES FOR NEURAL NETWORKS

4.1.1 TENSORFLOW

TensorFlow is a free and open source platform for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

TensorFlow architecture

TensorFlow architecture works in three significant steps: Data pre-processing - structure the data and brings it under one limiting value. Building the model - build the model for the data. Training and estimating the model - use the data to train the model and test it with unknown data.

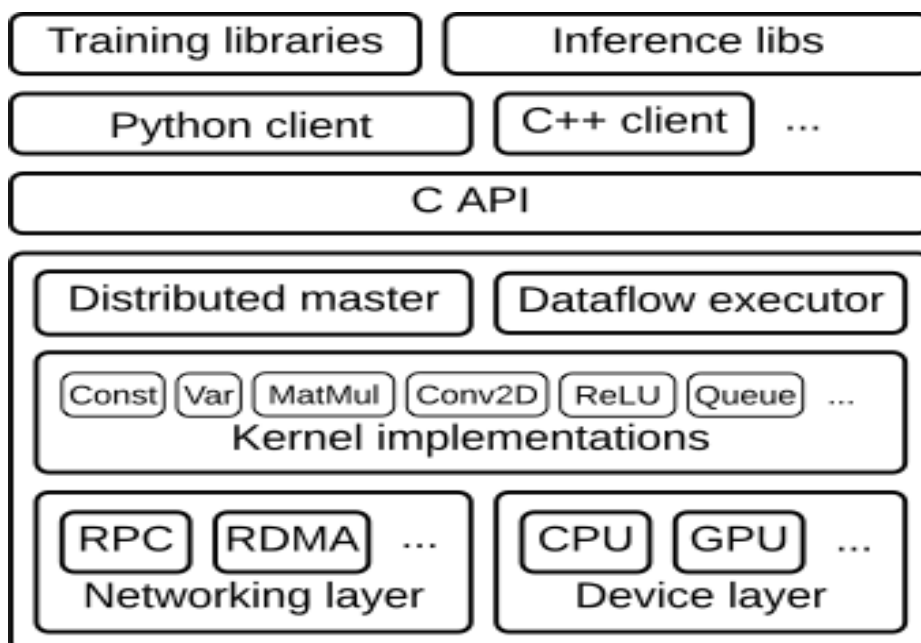


Fig 4.1: Tensorflow

- **Client:**

- Defines the computation as a dataflow graph.
- Initiates graph execution using a session.

- **Distributed Master**

- Prunes a specific subgraph from the graph, as defined by the arguments to `Session.run()`.
- Partitions the subgraph into multiple pieces that run in different processes and devices.
- Distributes the graph pieces to worker services.
- Initiates graph piece execution by worker services.

- **Worker Services (one for each task)**

- Schedule the execution of graph operations using kernel implementations appropriate to the available hardware (CPUs, GPUs, etc).
- Send and receive operation results to and from other worker services.

- **Kernel Implementations**

- Perform the computation for individual graph operations.

Main components of tensorflow:

- Variables: Retain values between sessions, use for weights/bias.
- Nodes: The operations.
- Tensors: Signals that pass from/to nodes.
- Placeholders: Used to send data between your program and the tensorflow graph.
- Session: Place when graph is executed.

Two phase of tensorflow:

The computation graph. TensorFlow programs are usually structured into a construction phase, that assembles a graph, and an execution phase that uses a session to execute ops in the graph.

Types of models in tensorflow:

There are two ways to create a model using the Layers API: A sequential model, and a functional model.

Tensorflow layers

A layer is a callable object that takes as input one or more tensors and that outputs one or more tensors. It involves computation, defined in the `call()` method, and a state (weight variables). Tensorflow supporting different types of Models and different types of Layers. A TensorFlow Model is a Neural Network with one or more Layers.

4.1.2 PYTORCH

PyTorch is an optimized Deep Learning tensor library based on Python and Torch and is mainly used for applications using GPUs and CPUs. PyTorch is favored over other Deep Learning frameworks like TensorFlow and Keras since it uses dynamic computation graphs and is completely Pythonic. It allows scientists, developers, and neural network debuggers to run and test portions of the code in real-time. Thus, users don't have to wait for the entire code to be implemented to check if a part of the code works or not.

The two main features of PyTorch are:

- Tensor Computation (similar to NumPy) with strong GPU (Graphical Processing Unit) acceleration support
 - Automatic Differentiation for creating and training deep neural networks
- Basics of PyTorch

The basic PyTorch operations are pretty similar to Numpy. Let's understand the basics first.

- **Introduction to Tensors**

In machine learning, when we represent data, we need to do that numerically. A tensor is simply a container that can hold data in multiple dimensions. In mathematical terms, however, a tensor is a fundamental unit of data that can be used as the foundation for advanced mathematical operations. It can be a number, vector, matrix, or multi-dimensional array like Numpy arrays. Tensors can also be handled by the CPU or GPU to make operations faster. There are various types of tensors like Float Tensor,

Double Tensor, Half Tensor, Int Tensor, and Long Tensor, but PyTorch uses the 32-bit Float Tensor as the default type.

- **Mathematical Operations**

The codes to perform mathematical operations are the same in PyTorch as in Numpy. Users need to initialize two tensors and then perform operations like addition, subtraction, multiplication, and division on them.

- **Matrix Initialization and Matrix Operations**

To initialize a matrix with random numbers in PyTorch, use the function `randn()` that gives a tensor filled with random numbers from a standard normal distribution. Setting the random seed at the beginning will generate the same numbers every time you run this code. Basic matrix operations and transpose operation in PyTorch are also similar to NumPy.

Common PyTorch Modules

In PyTorch, modules are used to represent neural networks.

- **Autograd**

The autograd module is PyTorch's automatic differentiation engine that helps to compute the gradients in the forward pass in quick time. Autograd generates a directed acyclic graph where the leaves are the input tensors while the roots are the output tensors.

- **Optim**

The Optim module is a package with pre-written algorithms for optimizers that can be used to build neural networks.

- **nn**

The nn module includes various classes that help to build neural network models. All modules in PyTorch subclass the nn module. Computational graphs in PyTorch allow the framework to calculate gradient values for the neural networks built. PyTorch uses dynamic computational graphs. The graph is defined indirectly using operator overloading while the forward computation gets executed.

Dynamic graphs are more flexible than static graphs, wherein users can make interleaved construction and valuation of the graph. These are debug-friendly as it allows line-by-line code execution. Finding problems in code is a lot easier with PyTorch Dynamic graphs – an important feature that makes PyTorch such a preferred choice in the industry.

Computational graphs in PyTorch are rebuilt from scratch at every iteration, allowing the use of random Python control flow statements, which can impact the overall shape and size of the graph every time an iteration occurs. The advantage is – there's no need to encode all possible paths before launching the training. You run what you differentiate.

Working with large datasets requires loading all data into memory in one go. This causes memory outage, and programs run slowly. Besides, it's hard to maintain data samples processing code. PyTorch offers two data primitives - DataLoader and Dataset – for parallelizing data loading with automated batching and better readability and modularity of codes.

Datasets and DataLoader allow users to use their own data as well as pre-loaded datasets. While Dataset houses the samples and the respective labels, DataLoader combines dataset and sampler and implements an iterable around the Dataset so users can easily access samples.

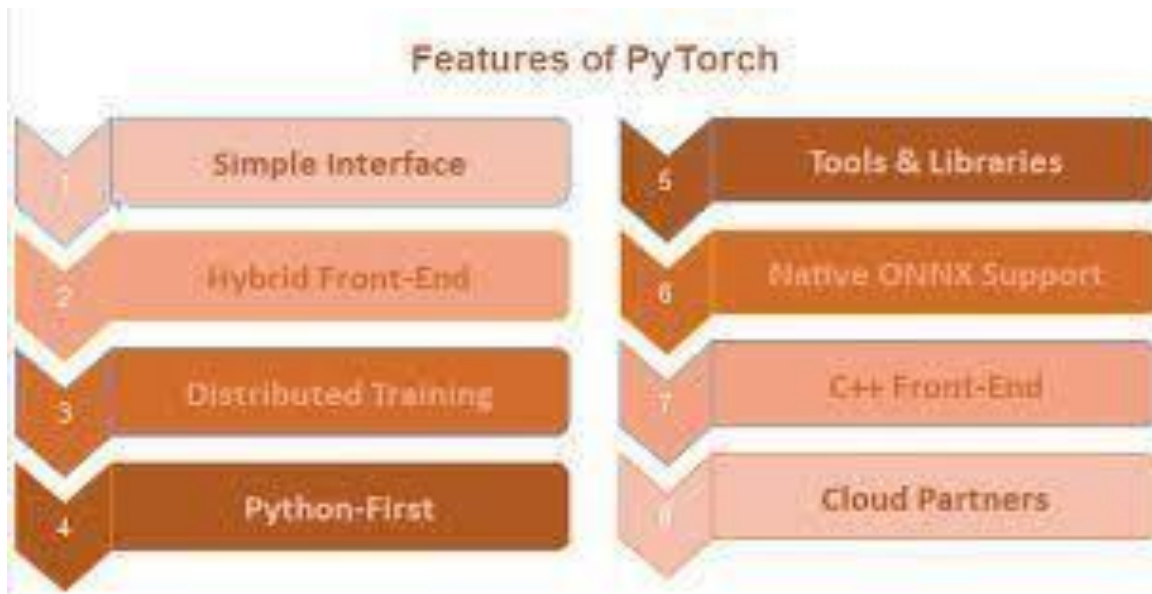


Fig 4.2: Pytorch

4.1.3 DENSELY CONNECTED NETWORK

Densely Connected Convolutional Networks. A DenseNet is a type of convolutional neural network that utilises dense connections between layers, through Dense Blocks, where we connect all layers (with matching feature-map sizes) directly with each other.

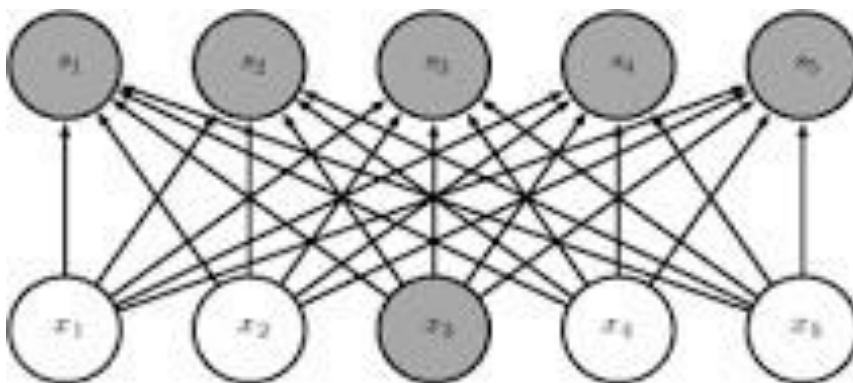


Fig 4.3: Denserly connected network

DenseNet was developed specifically to improve the declined accuracy caused by the vanishing gradient in high-level neural networks. In simpler terms, due to the longer path between the input layer and the output layer, the information vanishes before reaching its destination.

4.1.4 CAPSNET

Formally, a capsule is a set of neurons that collectively produce an activity vector with one element for each neuron to hold that neuron's instantiation value (e.g., hue). Graphics programs use instantiation value to draw an object. Capsnets attempt to derive these from their input.

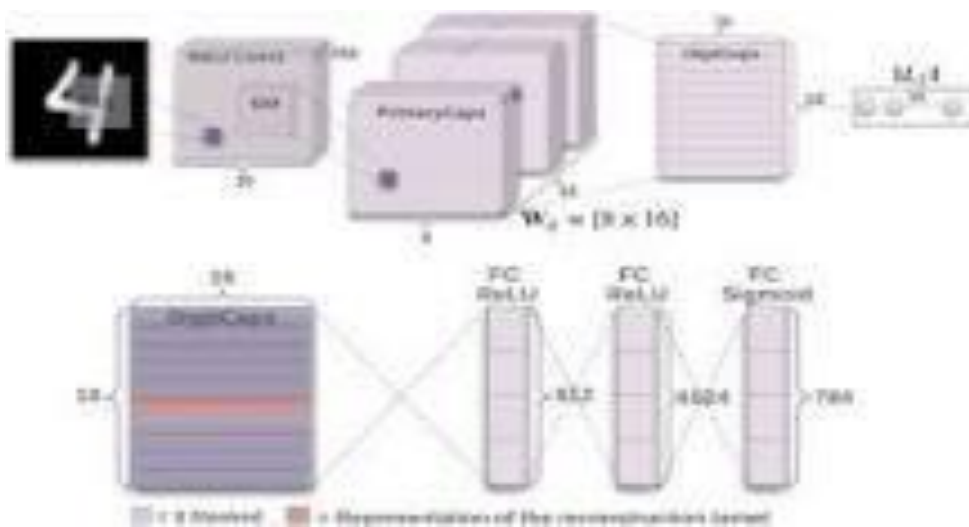


Fig 4.4 Capsnet

In a capsule network, each capsule is made up of a group of neurons with each neuron's output representing a different property of the same feature. This provides the advantage of recognizing the whole entity by first recognizing its parts. The input to a capsule is the output (or features) from a CNN.

Capsule Networks (CapsNet) are the networks that are able to fetch spatial information and more important features so as to overcome the loss of information that is seen in pooling operations. Let us see what is the difference between a capsule and a neuron. Capsule gives us a vector as an output that has a direction.

Advantage:

One of the key benefits of Capsule Network is that they tend to move away from the black-box model of a neural network, and represents more concrete features which can be interpreted to understand what and how is the network learning.

4.1.5 FRACTALNET

FractalNet is a type of convolutional neural network that eschews residual connections in favour of a "fractal" design. They involve repeated application of a simple expansion rule to generate deep networks whose structural layouts are precisely truncated fractals.

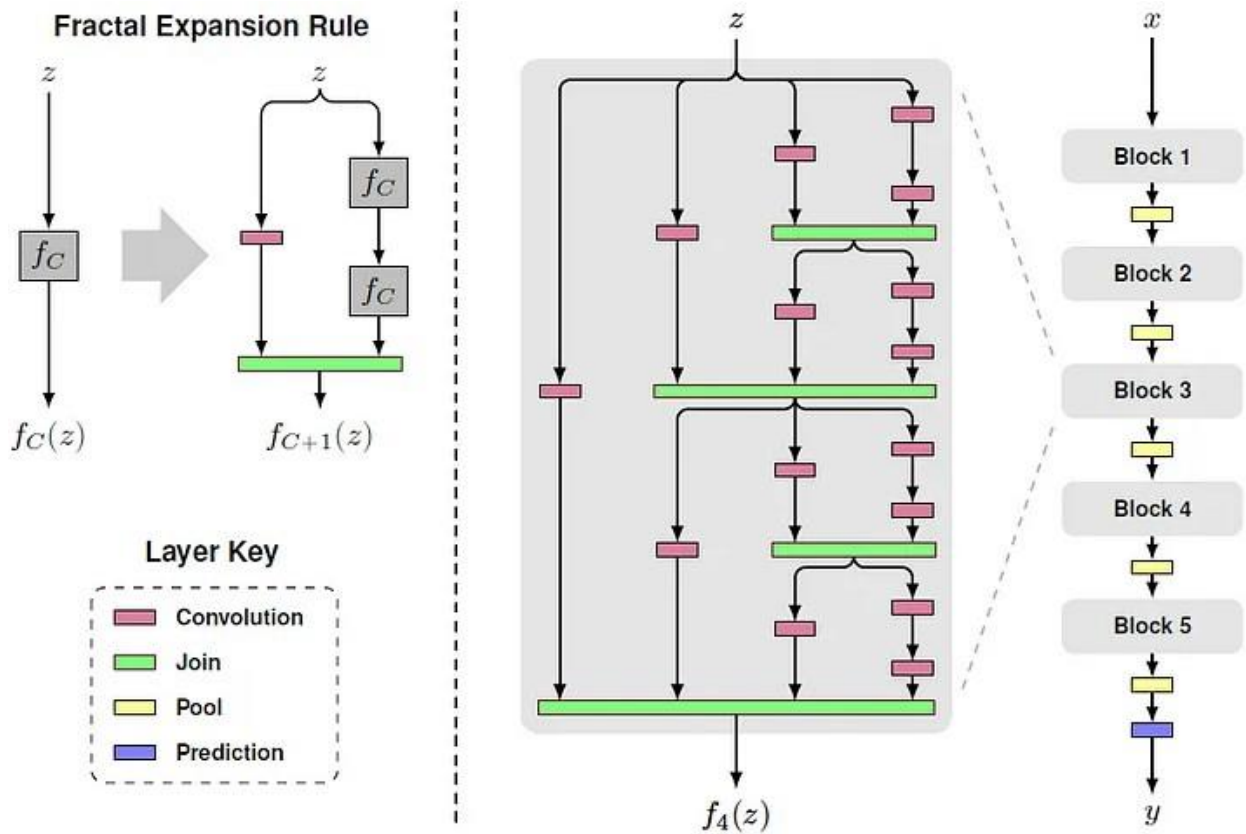


Fig 4.5 Fractalnet

For the base case, $f_1(z)$ is the convolutional layer:

$$f_1(z) = \text{conv}(z)$$

After that, the recursive fractals are:

$$f_{C+1}(z) = [(f_C \circ f_C)(z)] \oplus [\text{conv}(z)]$$

where C is the number of columns as in the middle of the Fig. The number of convolutional layers at the deepest path within a block will have $2^{(C-1)}$. In this case, $C=4$, thereby, the number of convolutional layers is $2^3=8$ layers.

For the join layer (green), element-wise mean is computed. It is not concatenation or addition. With 5 blocks ($B=5$) cascaded as FractalNet at the right of the Fig, then the number of convolutional layers at the deepest path within the whole network is $B \times 2^{(C-1)}$, i.e. $5 \times 2^3 = 40$ layers. In between 2 blocks, 2×2 max pooling is done to reduce the size of feature maps. Batch Norm and ReLU are used after each convolution.

REGULARIZATION VIA DROP-PATH

Dropout (Hinton et al., 2012) and drop-connect (Wan et al., 2013) modify interactions between sequential network layers in order to discourage co-adaptation. Since fractal networks contain additional macro-scale structure, we propose to complement these techniques with an analogous coarse-scale regularization scheme. Fig 2 illustrates drop-path. Just as dropout prevents co-adaptation of activations, drop-path prevents co-adaptation of parallel paths by randomly dropping operands of the join layers. This discourages the network from using one input path as an anchor and another as a corrective term (a configuration that, if not prevented, is prone to overfitting).

We consider two sampling strategies:

- Local: a join drops each input with fixed probability, but we make sure at least one survives.
- Global: a single path is selected for the entire network. We restrict this path to be a single column, thereby promoting individual columns as independently strong predictors. 4 Published as a conference paper at ICLR 2017 Iteration #1 (Local) Iteration #2 (Global) Iteration #3 (Local) Iteration #4 (Global) Fig 2: Drop-path.

A fractal network block functions with some connections between layers disabled, provided some path from input to output is still available. Drop-path guarantees at least one such path, while sampling a subnetwork with many other paths disabled. During training, presenting a different active subnetwork to each mini-batch prevents co-adaptation of parallel paths.

A global sampling strategy returns a single column as a subnetwork. Alternating it with local sampling encourages the development of individual columns as performant stand-alone subnetworks. As with dropout, signals may need appropriate rescaling. With element-wise means, this is trivial; each join computes the mean of only its active inputs. In experiments, we train with dropout and a mixture model of 50% local and 50% global sampling for drop-path. We sample a new subnetwork each mini-batch.

With sufficient memory, we can simultaneously evaluate one local sample and all global samples for each mini-batch by keeping separate networks and tying them together via weight sharing. While fractal connectivity permits the use of paths of any length, global drop-path forces the use of many paths whose lengths differ by orders of magnitude (powers of 2).

The subnetworks sampled by drop-path thus exhibit large structural diversity. This property stands in contrast to stochastic depth regularization of ResNet, which, by virtue of using a fixed drop probability for each layer in a chain, samples subnetworks with a concentrated depth distribution (Huang et al., 2016b). Global drop-path serves not only as a regularizer, but also as a diagnostic tool. Monitoring performance of individual columns provides insight into both the network and training mechanisms, as discusses in more detail.

Individually strong columns of various depths also give users choices in the trade-off between speed (shallow) and accuracy (deep). 3.2 DATA AUGMENTATION Data augmentation can reduce the need for regularization. ResNet demonstrates this, achieving 27.22% error rate on CIFAR-100 with augmentation compared to 44.76% without (Huang et al., 2016b). While augmentation benefits fractal networks, we show that drop-path provides highly effective regularization, allowing them to achieve competitive results even without data augmentation.

IMPLEMENTATION DETAILS

We implement FractalNet using Caffe. Purely for convenience, we flip the order of pool and join layers at the end of a block. We pool individual columns immediately

before the joins spanning all columns, rather than pooling once immediately after them. We train fractal networks using stochastic gradient descent with momentum.

4.1.6 NETWORK IN NETWORK(NIN)

LeNet, AlexNet, and VGG all share a common design pattern: extract features exploiting *spatial* structure via a sequence of convolution and pooling layers and then post-process the representations via fully-connected layers. The improvements upon LeNet by AlexNet and VGG mainly lie in how these later networks widen and deepen these two modules. Alternatively, one could imagine using fully-connected layers earlier in the process. However, a careless use of dense layers might give up the spatial structure of the representation entirely, *network in network (NiN)* blocks offer an alternative. They were proposed based on a very simple insight: to use an MLP on the channels for each pixel separately.

This design poses two major challenges. First, the fully connected layers at the end of the architecture consume tremendous numbers of parameters. For instance, even a simple model such as VGG-11 requires a monstrous 25088×4096 matrix, occupying almost 400MB of RAM in single precision (FP32). This is a significant impediment to computation, in particular on mobile and embedded devices. After all, even high-end mobile phones sport no more than 8GB of RAM. At the time VGG was invented, this was an order of magnitude less (the iPhone 4S had 512MB). As such, it would have been difficult to justify spending the majority of memory on an image classifier.

Second, it is equally impossible to add fully connected layers earlier in the network to increase the degree of nonlinearity: doing so would destroy the spatial structure and require potentially even more memory. The network in network (NiN) blocks offer an alternative, capable of solving both problems in one simple strategy. They were proposed based on a very simple insight: (i) use 1×1 convolutions to add local nonlinearities across the channel activations and (ii) use global average pooling to integrate across all locations in the last representation layer. Note that global average pooling would not be effective, were it not for the added nonlinearities. Let's dive into this in detail.

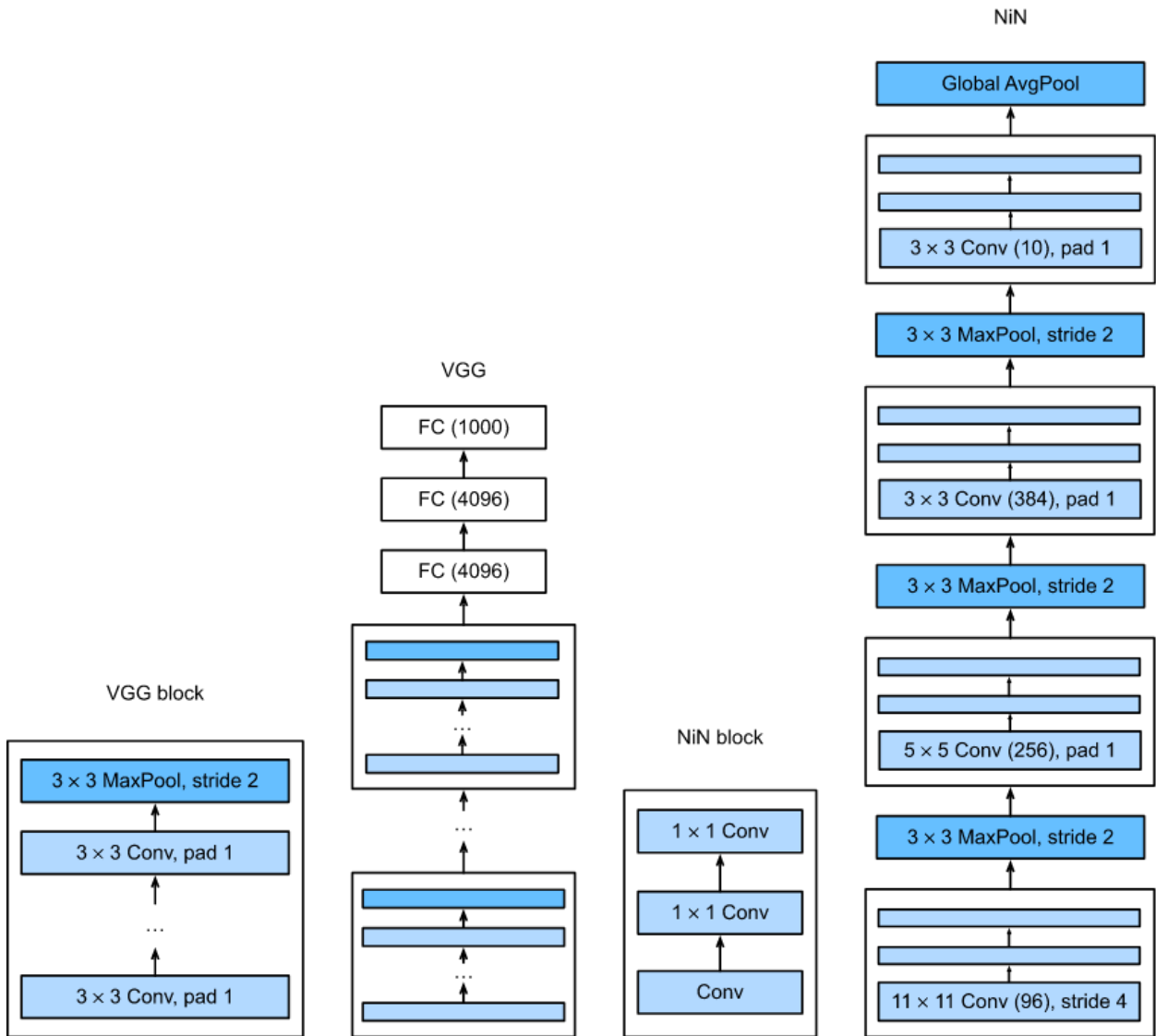


Fig 4.6: Network in Network

In it we discussed that the inputs and outputs of convolutional layers consist of four-dimensional tensors with axes corresponding to the example, channel, height, and width. Also recall that the inputs and outputs of fully connected layers are typically two-dimensional tensors corresponding to the example and feature. The idea behind NiN is to apply a fully connected layer at each pixel location (for each height and width). The resulting 1×1 convolution can be thought as a fully connected layer acting independently on each pixel location.

illustrates the main structural differences between VGG and NiN, and their blocks. Note both the difference in the NiN blocks (the initial convolution is followed by 1×1 convolutions, whereas VGG retains 3×3 convolutions) and in the end where we no longer require a giant fully connected layer. NiN uses the same initial convolution

sizes as AlexNet (it was proposed shortly thereafter). The kernel sizes are 11×11 , 5×5 , and 3×3 , respectively, and the numbers of output channels match those of AlexNet. Each NiN block is followed by a max-pooling layer with a stride of 2 and a window shape of 3×3 .

The second significant difference between NiN and both AlexNet and VGG is that NiN avoids fully connected layers altogether. Instead, NiN uses a NiN block with a number of output channels equal to the number of label classes, followed by a *global* average pooling layer, yielding a vector of logits. This design significantly reduces the number of required model parameters, albeit at the expense of a potential increase in training time.

NiN has dramatically fewer parameters than AlexNet and VGG. This stems primarily from the fact that it needs no giant fully connected layers. Instead, it uses global average pooling to aggregate across all image locations after the last stage of the network body. This obviates the need for expensive (learned) reduction operations and replaces them by a simple average. What was surprising at the time is the fact that this averaging operation did not harm accuracy. Note that averaging across a low-resolution representation (with many channels) also adds to the amount of translation invariance that the network can handle. Choosing fewer convolutions with wide kernels and replacing them by 1×1 convolutions aids the quest for fewer parameters further. It affords for a significant amount of nonlinearity across channels within any given location. Both 1×1 convolutions and global average pooling significantly influenced subsequent CNN designs.

4.1.7 ALEXNET

AlexNet is the name of a convolutional neural network (CNN) architecture, designed by Alex Krizhevsky in collaboration with Ilya Sutskever and Geoffrey Hinton, who was Krizhevsky's Ph.D. advisor. AlexNet competed in the ImageNet Large Scale Visual Recognition Challenge on September 30, 2012. The network achieved a top-5 error of 15.3%, more than 10.8 percentage points lower than that of the runner up. The original paper's primary result was that the depth of the model was essential for its high performance, which was computationally expensive, but made feasible due to the utilization of graphics processing units (GPUs) during training.

AlexNet was not the first fast GPU-implementation of a CNN to win an image recognition contest. A CNN on GPU by K. Chellapilla et al. (2006) was 4 times faster than an equivalent implementation on CPU. A deep CNN of Dan Cireşan et al. (2011) at IDSIA was already 60 times faster and outperformed predecessors in August 2011. Between May 15, 2011 and September 10, 2012, their CNN won no fewer than four image competitions. They also significantly improved on the best performance in the literature for multiple image databases.

According to the AlexNet paper, Cireşan's earlier net is "somewhat similar." Both were originally written with CUDA to run with GPU support. In fact, both are actually just variants of the CNN designs introduced by Yann LeCun et al. (1989) who applied the backpropagation algorithm to a variant of Kunihiro Fukushima's original CNN architecture called "neocognitron." The architecture was later modified by J. Weng's method called max-pooling.

In 2015, AlexNet was outperformed by Microsoft Research Asia's very deep CNN with over 100 layers, which won the ImageNet 2015 contest. AlexNet contained eight layers; the first five were convolutional layers, some of them followed by max-pooling layers, and the last three were fully connected layers. The network, except the last layer, is split into two copies, each run on one GPU. The entire structure can be written as.

where

- CNN = convolutional layer (with ReLU activation)
- RN = local response normalization
- MP = maxpooling
- FC = fully connected layer (with ReLU activation)
- Linear = fully connected layer (without activation)
- DO = dropout

It used the non-saturating ReLU activation function, which showed improved training performance over tanh and sigmoid. AlexNet is considered one of the most influential papers published in computer vision, having spurred many more papers published employing CNNs and GPUs to accelerate deep learning. As of early 2023, the AlexNet paper has been cited over 120,000 times according to Google Scholar.

4.1.8 RESIDUAL NETWORK

Residual Network (ResNet) is a deep learning model used for computer vision applications. It is a Convolutional Neural Network (CNN) architecture designed to support hundreds or thousands of convolutional layers. Previous CNN architectures were not able to scale to a large number of layers, which resulted in limited performance. However, when adding more layers, researchers faced the “vanishing gradient” problem.

Neural networks are trained through a backpropagation process that relies on gradient descent, shifting down the loss function and finding the weights that minimize it. If there are too many layers, repeated multiplications will eventually reduce the gradient until it “disappears”, and performance saturates or deteriorates with each layer added. ResNet provides an innovative solution to the vanishing gradient problem, known as “skip connections”. ResNet stacks multiple identity mappings (convolutional layers that do nothing at first), skips those layers, and reuses the activations of the previous layer. Skipping speeds up initial training by compressing the network into fewer layers.

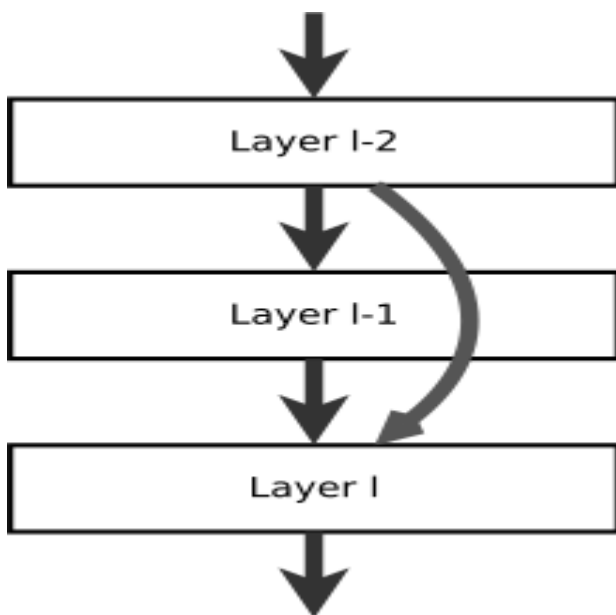


Fig 4.8: Residual Network

A residual neural network (ResNet) is an artificial neural network (ANN). It is a gateless or open-gated variant of the HighwayNet, the first working very deep feedforward neural network with hundreds of layers, much deeper than previous neural networks. *Skip connections* or *shortcuts* are used to jump over some layers (HighwayNets may also learn the skip weights themselves through an additional weight

matrix for their gates). Typical *ResNet* models are implemented with double- or triple-layer skips that contain nonlinearities (ReLU) and batch normalization in between. Models with several parallel skips are referred to as *DenseNets*. In the context of residual neural networks, a non-residual network may be described as a plain network.

Like in the case of Long Short-Term Memory recurrent neural networks there are two main reasons to add skip connections: to avoid the problem of vanishing gradients, thus leading to easier optimization of neural networks, where the gating mechanisms facilitate information flow across many layers ("information highways"), or to mitigate the Degradation (accuracy saturation) problem; where adding more layers to a suitably deep model leads to higher training error. During training, the weights adapt to mute the upstream layer, and amplify the previously-skipped layer. In the simplest case, only the weights for the adjacent layer's connection are adapted, with no explicit weights for the upstream layer. This works best when a single nonlinear layer is stepped over, or when the intermediate layers are all linear. If not, then an explicit weight matrix should be learned for the skipped connection (a HighwayNet should be used).

Skipping effectively simplifies the network, using fewer layers in the initial training stages. This speeds learning by reducing the impact of vanishing gradients, as there are fewer layers to propagate through. The network then gradually restores the skipped layers as it learns the feature space. Towards the end of training, when all layers are expanded, it stays closer to the manifold and thus learns faster. A neural network without residual parts explores more of the feature space. This makes it more vulnerable to perturbations that cause it to leave the manifold, and necessitates extra training data to recover.

4.1.9 ZFNET

ZFNet is a classic convolutional neural network. The design was motivated by visualizing intermediate feature layers and the operation of the classifier. Compared to AlexNet, the filter sizes are reduced and the stride of the convolutions are reduced. Deep learning is a fascinating field of Machine Learning used to solve various complex problems that were previously unthought of. The heart of any Deep learning algorithm is Neural networks. Neural networks are based on the working of the human brain. They contain neurons and nodes that transfer information from one layer to another. The awe-inspiring working of these neural networks was considered to be enclosed in a black box that no one could see through.

There was no clear idea of how they could improve. The only method was using trial and error. But all this was changed when Rob Fergus and Matthew D. Zeiler introduced ZFNet. ZFNet is named after their surname Zeiler and Fergus. ZFNet was a revolutionary way to understand how each layer of a neural network is performing. Let us see how ZFNet uncovers the secrets of neural networks.

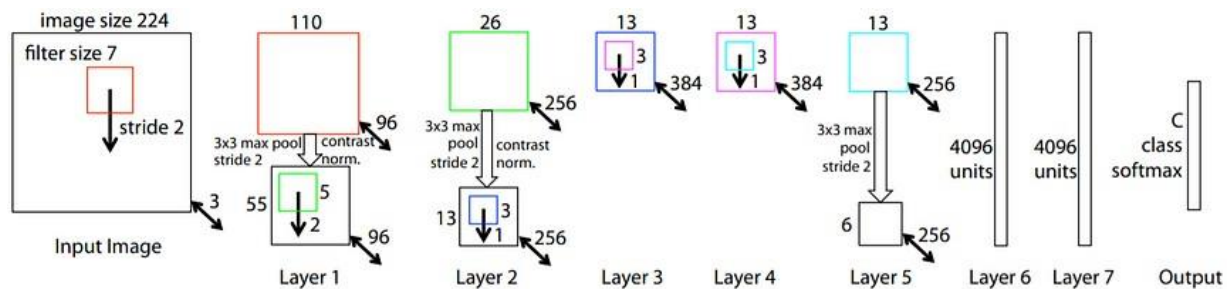


Fig 4.9: ZFnet

- input is 224x224x3 images.
- Next, 96 convolutions of 7x7 with a stride of 2 are performed, followed by ReLU activation, 3x3 max pooling with stride 2 and local contrast normalization.
- Followed by it are 256 filters of 3x3 each which are then again local contrast normalized and pooled.
- The third and fourth layers are identical with 384 kernels of 3x3 each.
- The fifth layer has 256 filters of 3x3, followed by 3x3 max pooling with stride 2 and local contrast normalization.
- The sixth and seventh layers house 4096 dense units each.
- Finally, we feed into a Dense layer of 1000 neurons i.e. the number of classes in ImageNet.

4.1.10 GOOGLE-NET

GoogLeNet is a type of convolutional neural network based on the Inception architecture. It utilises Inception modules, which allow the network to

choose between multiple convolutional filter sizes in each block. An Inception network stacks these modules on top of each other, with occasional max-pooling layers with stride 2 to halve the resolution of the grid.

GoogLeNet is a 22-layer deep convolutional neural network that's a variant of the Inception Network, a Deep Convolutional Neural Network developed by researchers at Google. The GoogLeNet architecture presented in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14) solved computer vision tasks such as image classification and object detection — *find out how well it performed at the conclusion section of this article*. Today GoogLeNet is used for other computer vision tasks such as face detection and recognition, adversarial training etc. The GoogLeNet architecture consists of 22 layers (27 layers including pooling layers), and part of these layers are a total of 9 inception modules.

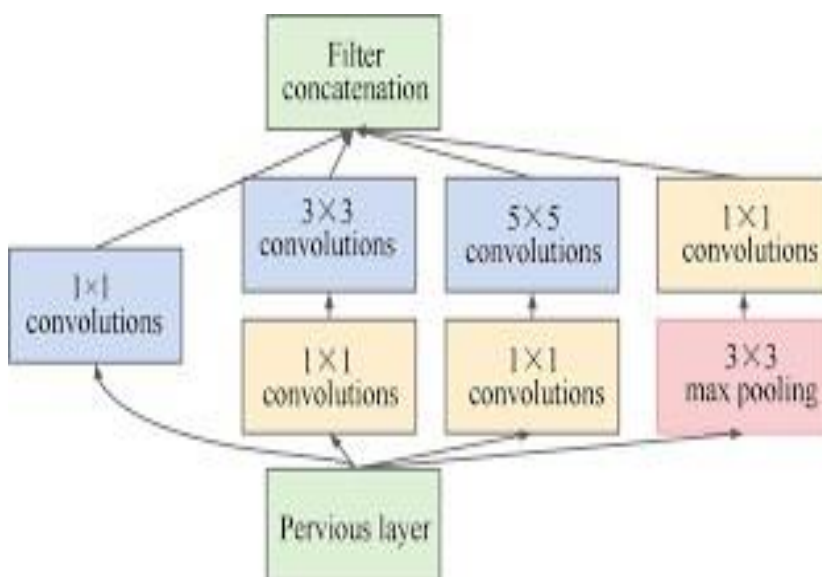


Fig 4.10: GoogLeNet

4.1.11 VGGNET

VGGNet is a Convolutional Neural Network architecture proposed by Karen Simonyan and Andrew Zisserman from the University of Oxford in 2014. This paper mainly focuses on the effect of the convolutional neural network depth on its accuracy. You can find the original paper of VGGNet which is titled Very Deep Convolutional Networks for Large Scale Image Recognition.

Architecture

The input to VGG based convNet is a 224×224 RGB image. Preprocessing layer takes the RGB image with pixel values in the range of 0-255 and subtracts the mean image values which is calculated over the entire ImageNet training set.

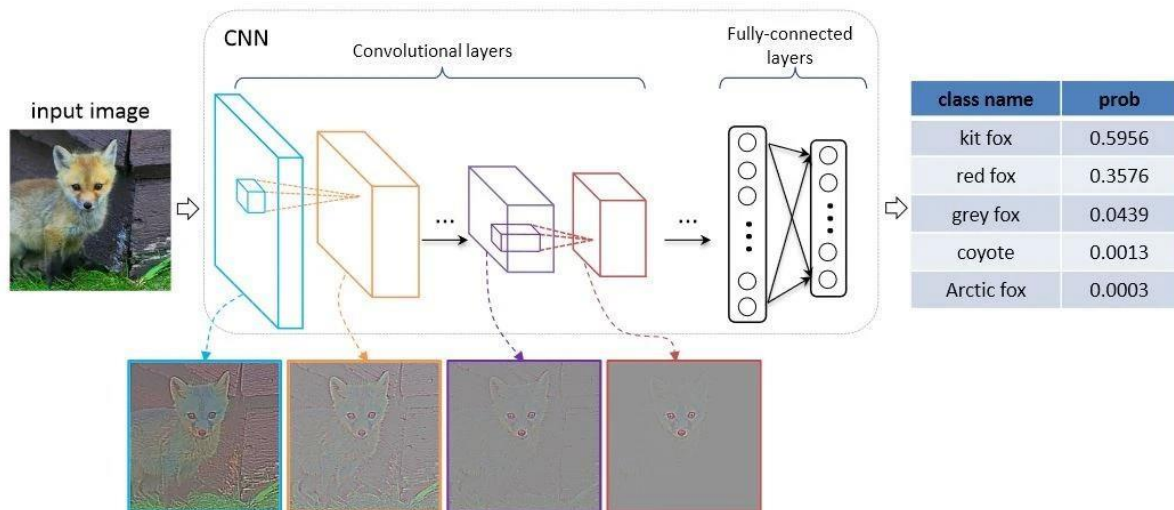


Fig 4.11: VGGNET

The input images after preprocessing are passed through these weight layers. The training images are passed through a stack of convolution layers. There are total of 13 convolutional layers and 3 fully connected layers in VGG16 architecture. VGG has smaller filters (3×3) with more depth instead of having large filters. It has ended up having the same effective receptive field as if you only have one 7×7 convolutional layers.

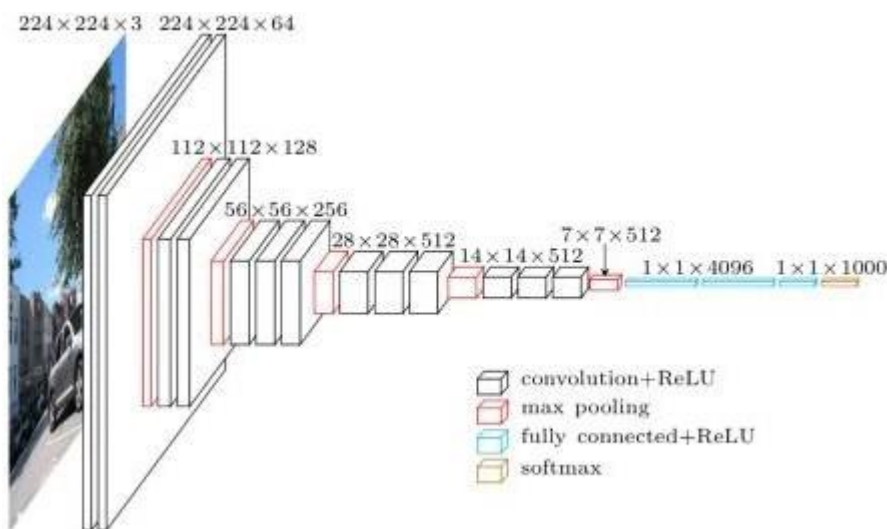


Fig 4.11: FRAMES OF VGGNET

Another variation of VGGNet has 19 weight layers consisting of 16 convolutional layers with 3 fully connected layers and same 5 pooling layers. In both variation of VGGNet there consists of two Fully Connected layers with 4096 channels each which is followed by another fully connected layer with 1000 channels to predict 1000 labels. Last fully connected layer uses softmax layer for classification purpose.

4.1.12 LE-NET

As a representative of the early convolutional neural network, LeNet possesses the basic units of convolutional neural network, such as convolutional layer, pooling layer and full connection layer, laying a foundation for the future development of convolutional neural network. As shown in the Fig (input image data with 32×32 pixels) : LeNet-5 consists of seven layers. In addition to input, every other layer can train parameters.

Layer C1 is a convolution layer with six convolution kernels of 5×5 and the size of feature mapping is 28×28 , which can prevent the information of the input image from falling out of the boundary of convolution kernel. Layer S2 is the subsampling/pooling layer that outputs 6 feature graphs of size 14×14 . Each cell in each feature map is connected to 2×2 neighborhoods in the corresponding feature map in C1. Layer C3 is a convolution layer with 16 5×5 convolution kernels. The input of the first six C3 feature maps is each continuous subset of the three feature maps in S2, the input of the next six feature maps comes from the input of the four continuous subsets, and the input of the next three feature maps comes from the four discontinuous subsets. Finally, the input for the last feature graph comes from all feature graphs of S2. Layer S4 is similar to S2, with size of 2×2 and output of 16 5×5 feature graphs.

Layer C5 is a convolution layer with 120 convolution kernels of size 5×5 . Each cell is connected to the 5×5 neighborhood on all 16 feature graphs of S4. Here, since the feature graph size of S4 is also 5×5 , the output size of C5 is 1×1 . So S4 and C5 are completely connected. C5 is labeled as a convolutional layer instead of a fully connected layer, because if LeNet-5 input becomes larger and its structure remains unchanged, its output size will be greater than 1×1 , i.e. not a fully connected layer.

F6 layer is fully connected to C5, and 84 feature graphs are output.

- Every convolutional layer includes three parts: convolution, pooling, and nonlinear activation functions

- Using convolution to extract spatial features (Convolution was called receptive fields originally)
- Subsampling average pooling layer
- tanh activation function
- Using MLP as the last classifier
- Sparse connection between layers to reduce the complexity of computation
- Recognizing simple digit images is the most classic application of LeNet as it was created because of that.

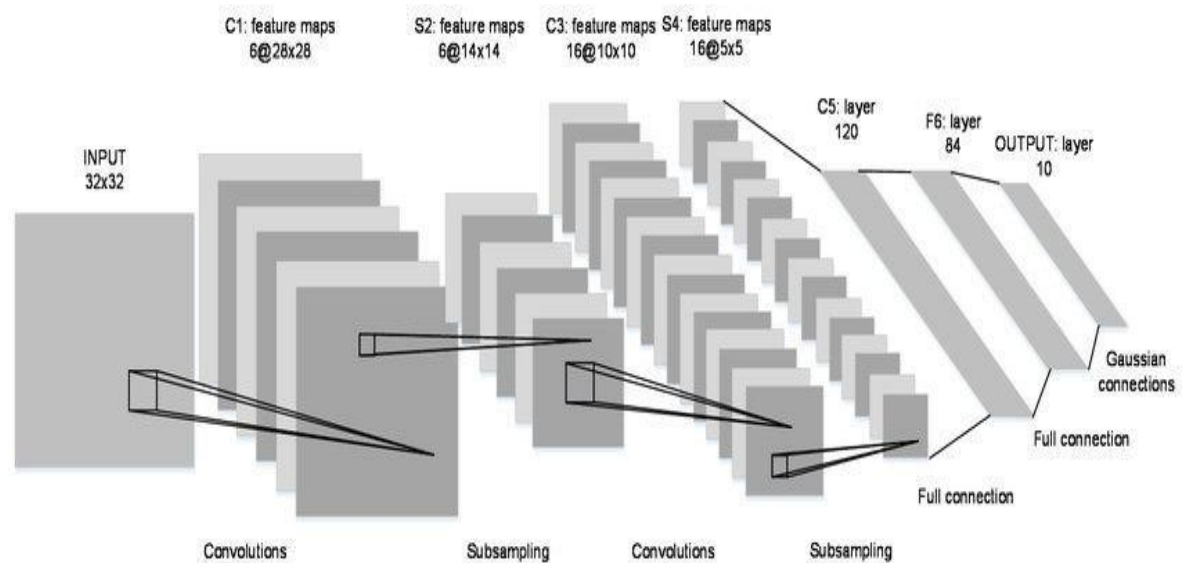


Fig 4.12: LENET

4.1.13 KERAS

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Keras is an open-source high-level Neural Network library, which is written in Python is capable enough to run on Theano, TensorFlow, or CNTK. It was developed by one of the Google engineers, Francois Chollet. It is made user-friendly, extensible, and modular for facilitating faster experimentation with deep neural networks. It not only supports Convolutional Networks and Recurrent Networks individually but also their combination. It cannot handle low-level computations, so it makes use of the Backend library to resolve it. The backend library act as a high-level API wrapper for the low-level API, which lets it run on TensorFlow, CNTK, or Theano.

Advantages of Keras

Keras encompasses the following advantages, which are as follows:

- It is very easy to understand and incorporate the faster deployment of network models.
- It has huge community support in the market as most of the AI companies are keen on using it.
- It supports multi backend, which means you can use any one of them among TensorFlow, CNTK, and Theano with Keras as a backend according to your requirement.
- Since it has an easy deployment, it also holds support for cross-platform. Following are the devices on which Keras can be deployed:
 - iOS with CoreML
 - Android with TensorFlow Android
 - Web browser with .js support
 - Cloud engine
 - Raspberry pi
- It supports Data parallelism, which means Keras can be trained on multiple GPU's at an instance for speeding up the training time and processing a huge amount of data.

Disadvantages of Keras

- The only disadvantage is that Keras has its own pre-conFigd layers, and if you want to create an abstract layer, it won't let you because it cannot handle low- level APIs. It only supports high-level API running on the top of the backend engine (TensorFlow, Theano, and CNTK).

CHAPTER-5

RESULTS AND DISCUSSION

This project is not only limited to damage detection; the damaged road data has been sent further to cloud. It helps the fellow drivers who travelling on the way (i.e., pothole) by sending an indication on their cars. This project is mainly focused that in future all cars will be connected to each other, there will be communication among them, either between them or by cloud storages. So that the other cars coming in that way will be alerted to reduce speed, or to change route.



The condition of the surrounding plays an important factor in traffic congestion. A few studies focused on these factors. Two studies considered social media contribution in input parameter, and five considered weather condition. Events, e.g., national event, school holiday, and popular sports events, play a big role in traffic congestion. For example, Melbourne, Australia, has two public holidays before and during two most popular sports events of the country. The authorities close a few traffic routes to tackle the traffic and the parade, resulting in traffic congestion. Therefore, more focus must be put in including these factors while forecasting.

Dealing with missing data is a challenge in the data processing. Some excluded the respective data altogether, others applied different methods to retrieve the data, and some replaced with other data. Missing data imputation can be a useful research scope in transportation engineering.

CHAPTER-6

CONCLUSION

After studying various methods that have been proposed so far for road detection, most of the region based methods work well for structured road types. Road color properties, lane markings etc are used to differentiate road regions from off road regions we can conclude that mainly there are two type of approaches used so far: region features and texture features. The effectiveness of such algorithms has been improved by various segmentation algorithms and improvements in edge detection algorithms.

But these approaches are not promising same results if roads are unstructured. To overcome these problems texture properties based approaches have been proposed. Use of global information like vanishing point can be very useful to extract the boundaries of the roads even in unstructured environments. The main problem to deal with is computational cost of various orientation filters used to extract the texture properties. Researchers have also highlighted the problem of robust road detection under various environmental and climate changes. Road region recognition is a main feature that is gaining increasing attention from intellectuals because it helps autonomous vehicle to achieve successful navigation without accident. Different techniques based on computer vision have been used by various researchers and outstanding results have been achieved. Despite their success, there is a huge scope of research work which can further enhance the effectiveness of road detection algorithm and make it more general for all the type of roads as well as more robust to withstand various environmental variations and irregularities in road shapes.

Based on the literature reviewed so far, there is a scope of road detection method that will have all the characteristics mentioned below.

- Road detection algorithm should be completely autonomous without any sort of human interface.
- It should work using global features.
- Detection method should be fast enough to meet real time criteria..
- Algorithm should be robust.

Fast vanishing point detection and selection of unique road region features are two major challenges for the further research work in the field of computer vision.

REFERENCES

- [1].EkimYurtsever, Jacob Lambert¹, Alexander Carballo, And Kazuya Takeda, “A Survey of Autonomous Driving: Common Practices and Emerging Technologies”, IEEE-2020.
- [2]. SadanandHowal, Aishwarya Jadhav, ChandrakirtiArthshi, SapanaNalavade, Sonam Shinde, “Object Detection for Autonomous Vehicle Using TensorFlow”, 2019.
- [3]. Mike Daily, Swarup Medasani, Daimler Protics, Trivedi, San Diego, “Self-Driving Cars”,IEEE-2019.
- [4] Yuenan Hou, Zheng Ma, Chunxiao Liu, and Chen Change Loy, “Learning Lightweight Lane Detection CNNs by Self Attention Distillation”, 2019.
- [5]. Geethapriya.S,N.Duraimurugan, S.P. Chokkalingam, “Real-Time Object Detection with YOLO”, IJEAT-2019.
- [6]. Michal Taraba, JurajAdamec, MatusDanko, Peter Drgona, “Utilization of modern sensors in autonomous vehicles”.
- [7]. Izzah Amani Tarmizi, Azrina Abd Aziz, “Vehicle Detection using Convolutional Neural Network for Autonomous Vehicles”, ICIAS-2018.
- [8]. BrilianTafjiraNugraha, Shun-Feng Su, Fahmizal, “Towards Self-driving Car Using Convolutional Neural Network and Road Lane Detector”, ICACOMIT-2017.
- [9]. Aysxegu, Ucxar, Yakup Demir and Cu neyt Gu zelis, “Object recognition and detection with deep learning for autonomous driving applications”.
- [10]. Alex Teichman, Sebastian Thrun, “Tracking-Based Semi-Supervised Learning”.
- [11]. Rajasekhar MV, Anil Kumar Jaswal, “Autonomous Vehicles: The Future Of Automobiles”.

[12]. Seyyed Hamed Naghavi, Cyrus Avaznia, Hamed Talebi, “Integrated Real-Time Object Detection for Self- Driving Vehicles”, 2017.

[13]. Mehdi Masmoudi, Hakim Ghazzai, Mounir Frikha, and Yehia Massoud, “Object Detection Learning Techniques for Autonomous Vehicle Applications”.

[14]. Aditya Singh Rathore, “Lane Detection for Autonomous Vehicles using OpenCV Library”, Jan-2019.

[15]. Zhong-Qiu Zhao, Member, IEEE, Peng Zheng, Shou-tao Xu, and Xindong Wu, Fellow, IEEE, “Object Detection with Deep Learning: A Review”, 2019.

APPENDIX

A. SOURCE CODE

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab

import tensorflow as tf
from tensorflow.contrib.layers import flatten
from keras.layers.pooling import MaxPooling2D
from keras.models import Sequential, Model
from keras.callbacks import EarlyStopping, Callback
from keras.layers import Dense, Dropout, Activation, Flatten, Lambda,
ELU, GlobalAveragePooling2D, regularizers
from keras.layers.convolutional import Convolution2D, Cropping2D, Conv2D
from keras.layers.pooling import MaxPooling2D
from keras.optimizers import adam
from sklearn.utils import shuffle
from keras.utils import np_utils

import time, cv2, glob

global inputShape, size

def kerasModel4():
    model = Sequential()
    model.add(Conv2D(16, (8, 8), strides=(4, 4), padding='valid',
input_shape=(size,size,1)))
    model.add(Activation('relu'))
    model.add(Conv2D(32, (5, 5), padding="same"))
    model.add(Activation('relu'))
    model.add(GlobalAveragePooling2D())
    # model.add(Dropout(.2))
    # model.add(Activation('relu'))
    # model.add(Dense(1024))
    # model.add(Dropout(.5))
    model.add(Dense(512))
    model.add(Dropout(.1))
    model.add(Activation('relu'))
    # model.add(Dense(256))
    # model.add(Dropout(.5))
    # model.add(Activation('relu'))
    model.add(Dense(2))
    model.add(Activation('softmax'))
    return model
```

```
size=100
```

```
## load Training data : pothole
potholeTrainImages = glob.glob("C:/Users/Desktop/pothole-and-plain-rode-images/My
Dataset/train/Pothole/*.jpg")
potholeTrainImages.extend(glob.glob("C:/User/Desktop/pothole-and-plain-rode-
images/My Dataset/train/Pothole/*.jpeg"))
potholeTrainImages.extend(glob.glob("C:/Users/Desktop/pothole-and-plain-rode-
images/My Dataset/train/Pothole/*.png"))
```

```
train1 = [cv2.imread(img,0) for img in potholeTrainImages]
for i in range(0,len(train1)):
    train1[i] = cv2.resize(train1[i],(size,size))
temp1 = np.asarray(train1)
```

```
# ## load Training data : non-pothole
nonPotholeTrainImages = glob.glob("C:/Users/Desktop/pothole-and-plain-rode-
images/My Dataset/train/Plain/*.jpg")
# nonPotholeTrainImages.extend(glob.glob("C:/Users/Desktop/pothole-and-plain-
rode-images/My Dataset/train/Plain/*.jpeg"))
# nonPotholeTrainImages.extend(glob.glob("C:/Users/Desktop/pothole-and-plain-
rode-images/My Dataset/train/Plain/*.png"))
train2 = [cv2.imread(img,0) for img in nonPotholeTrainImages]
# train2[train2 != np.array(None)]
for i in range(0,len(train2)):
    train2[i] = cv2.resize(train2[i],(size,size))
temp2 = np.asarray(train2)
```

```
## load Testing data : non-pothole
nonPotholeTestImages = glob.glob("C:/Users/Desktop/pothole-and-plain-rode-
images/My Dataset/test/Plain/*.jpg")
# nonPotholeTrainImages.extend(glob.glob("C:/Users/Desktop/pothole-and-plain-
rode-images/My Dataset/train/Plain/*.jpeg"))
# nonPotholeTrainImages.extend(glob.glob("C:/Users/Desktop/pothole-and-plain-
rode-images/My Dataset/train/Plain/*.png"))
test2 = [cv2.imread(img,0) for img in nonPotholeTestImages]
# train2[train2 != np.array(None)]
for i in range(0,len(test2)):
    test2[i] = cv2.resize(test2[i],(size,size))
temp4 = np.asarray(test2)
```

```
## load Testing data : potholes
potholeTestImages = glob.glob("C:/Users/Desktop/pothole-and-plain-rode-images/My
Dataset/test/Pothole/*.jpg")
# nonPotholeTrainImages.extend(glob.glob("C:/Users/Desktop/pothole-and-plain-
rode-images/My Dataset/train/Plain/*.jpeg"))
```

```

# nonPotholeTrainImages.extend(glob.glob("C:/Users/Desktop/pothole-and-plain-
rode-images/My Dataset/train/Plain/*.png"))
test1 = [cv2.imread(img,0) for img in potholeTestImages]
# train2[train2 != np.array(None)]
for i in range(0,len(test1)):
    test1[i] = cv2.resize(test1[i],(size,size))
temp3 = np.asarray(test1)

X_train = []
X_train.extend(temp1)
X_train.extend(temp2)
X_train = np.asarray(X_train)

X_test = []
X_test.extend(temp3)
X_test.extend(temp4)
X_test = np.asarray(X_test)

y_train1 = np.ones([temp1.shape[0]],dtype = int)
y_train2 = np.zeros([temp2.shape[0]],dtype = int)
y_test1 = np.ones([temp3.shape[0]],dtype = int)
y_test2 = np.zeros([temp4.shape[0]],dtype = int)

print(y_train1[0])
print(y_train2[0])
print(y_test1[0])
print(y_test2[0])

y_train = []
y_train.extend(y_train1)
y_train.extend(y_train2)
y_train = np.asarray(y_train)

y_test = []
y_test.extend(y_test1)
y_test.extend(y_test2)
y_test = np.asarray(y_test)

X_train,y_train = shuffle(X_train,y_train)
X_test,y_test = shuffle(X_test,y_test)

# X_train.reshape([-1,50,50,1])
# X_test.reshape([-1,50,50,1])/
X_train = X_train.reshape(X_train.shape[0], size, size, 1)
X_test = X_test.reshape(X_test.shape[0], size, size, 1)

y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)

```

```

print("train shape X", X_train.shape)
print("train shape y", y_train.shape)

inputShape = (size, size, 1)
model = kerasModel4()

model.compile('adam', 'categorical_crossentropy', ['accuracy'])
history = model.fit(X_train, y_train, epochs=500, validation_split=0.1)

metrics = model.evaluate(X_test, y_test)
for metric_i in range(len(model.metrics_names)):
    metric_name = model.metrics_names[metric_i]
    metric_value = metrics[metric_i]
    print('{}: {}'.format(metric_name, metric_value))

print("Saving model weights and configuration file")

model.save('sample.h5')

model_json = model.to_json()
with open("truesample.json", "w") as json_file:
    json_file.write(model_json)

model.save_weights("truesample.h5")
print("Saved model to disk")

ric_name, metric_value))import numpy as np
import cv2
import glob
from keras.models import Sequential
from keras.models import load_model
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from keras.utils import np_utils

global size
size = 100
model = Sequential()
model = load_model('C:/Users/Pothole Detection using Machine Learning/Pothole
Detection using Machine Learning/sample.h5')

# X_test = np.load('./models/trainData/128x72x3x10000/X_test.npy')
# y_test = np.load('./models/trainData/128x72x3x10000/y_test.npy')

## load Testing data : non-pothole
nonPotholeTestImages = glob.glob("C:/Users/Desktop/pothole-and-plain-rode-
images/My Dataset/test/Plain/*.jpg")
# nonPotholeTrainImages.extend(glob.glob("C:/Users/Desktop/pothole-and-plain-
rode-images/My Dataset/train/Plain/*.jpeg"))

```

```

# nonPotholeTrainImages.extend(glob.glob("C:/Users/Desktop/pothole-and-plain-
rode-images/My Dataset/train/Plain/*.png"))
test2 = [cv2.imread(img,0) for img in nonPotholeTestImages]
# train2[train2 != np.array(None)]
for i in range(0,len(test2)):
    test2[i] = cv2.resize(test2[i],(size,size))
temp4 = np.asarray(test2)

## load Testing data : potholes
potholeTestImages = glob.glob("C:/Users/Desktop/pothole-and-plain-rode-images/My
Dataset/test/Pothole/*.jpg")
# nonPotholeTrainImages.extend(glob.glob("C:/Users/Desktop/pothole-and-plain-
rode-images/My Dataset/train/Plain/*.jpeg"))
# nonPotholeTrainImages.extend(glob.glob("C:/Users/Desktop/pothole-and-plain-
rode-images/My Dataset/train/Plain/*.png"))
test1 = [cv2.imread(img,0) for img in potholeTestImages]
# train2[train2 != np.array(None)]
for i in range(0,len(test1)):
    test1[i] = cv2.resize(test1[i],(size,size))
temp3 = np.asarray(test1)

X_test = []
X_test.extend(temp3)
X_test.extend(temp4)
X_test = np.asarray(X_test)

X_test = X_test.reshape(X_test.shape[0], size, size, 1)

y_test1 = np.ones([temp3.shape[0]],dtype = int)
y_test2 = np.zeros([temp4.shape[0]],dtype = int)

y_test = []
y_test.extend(y_test1)
y_test.extend(y_test2)
y_test = np.asarray(y_test)

y_test = np_utils.to_categorical(y_test)

tests = model.predict_classes(X_test)
for i in range(len(X_test)):
    print(">>> Predicted=%s" % (tests[i]))

# metrics = model.evaluate(X_test, y_test)
# for metric_i in range(len(model.metrics_names)):
#     metric_name = model.metrics_names[metric_i]
#     metric_value = metrics[metric_i]
#     print('{}: {}'.format(metrics))

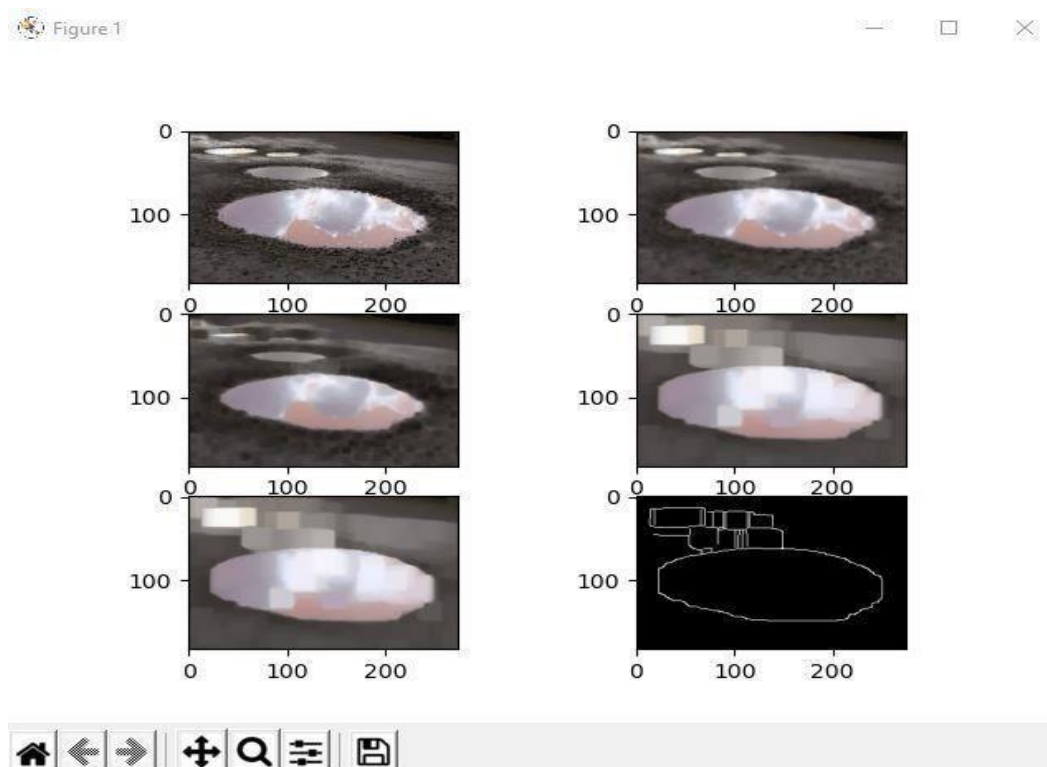
```

B. OUTPUT IMAGES

STEP 1: Inserting the image



STEP 2: Extracting image into frames



STEP 3:

Comparing the captured image and original image, To identify the damaged roads. The accuracy is above 80%.

