A Project report

on

End to End Car License Plate Detection and Recognition with Deep Neural Networks

Submitted in partial fulfillment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY

in

Computer Science & Engineering

by

G.TEJASWINI	**(174G1A0598)**
P.T.USHA	**(174G1A05A3)**
D.SIRI	**(174G1A05B9)**
R.RAGHU NANDINI	**(17F41A0585)**

Under the Guidance of

Dr. B. Hari Chandana, **M.Tech., Ph.D**
Associate Professor

Department of Computer Science & Engineering

SRINIVASA RAMANUJAN INSTITUTE OF TECHNOLOGY : ANANTHAPURAMU
(Affiliated to JNTUA,Approved by AICTE,New Delhi, Accredited by NAAC with 'A' Grade& Accredited by NBA(EEE, ECE &CSE)
Rotarypuram Village, B K Samudram Mandal, Ananthapuramu – 515701

2020-21

SRINIVASA RAMANUJAN INSTITUTE OF TECHNOLOGY : ANANTHAPURAMU

(Affiliated to JNTUA,Approved by AICTE,New Delhi, Accredited by NAAC with 'A' Grade& Accredited by NBA(EEE, ECE &CSE)

Rotarypuram Village, B K Samudram Mandal, Ananthapuramu – 515701

SRIT
Empowering Knowledge

Certificate

This is to certify that the project report entitled End to End Car License Plate Detection and Recognition using Deep Neural Networks is the bonafide work carried out by **G.TEJASWINI** bearing Roll Number 174G1A0598, **P.T.USHA** bearing Roll Number 174G1A05A3, **D.SIRI** bearing Roll Number 174G1A05B9 and **R.RAGHU NANDINI** bearing Roll Number 17F41A0585 in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering** during the academic year 2020-2021.

Guide	**Head of the Department**
Dr.B.Hari Chandana, M.Tech,PhD	Dr. G.K.V. Narasimha Reddy,Ph.D
Associate Professor	Professor & HOD

Date: **EXTERNAL EXAMINER**

Place: Ananthapuramu

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible, whose constant guidance and encouragement crowned our efforts with success. It is a pleasant aspect that we have now the opportunity to express my gratitude for all of them.

It is with immense pleasure that we would like to express our indebted gratitude to our Guide **Dr. B.Hari Chandana, Associate Professor, Computer Science and Engineering Department**, who has guided us a lot and encouraged us in every step of the project work. We thank her for the stimulating guidance, constant encouragement and constructive criticism which have made possible to bring out this project work

.We express our deep felt gratitude to **Mr. P.Chitralingappa, Associate Professor, Mrs. M.Soumya, Assistant Professor,** project coordinator valuable guidance and unstinting encouragement enable us to accomplish our project successfully in time.

.We are very much thankful to **Dr. G.K.V.Narasimha Reddy, Professor & Head of the Department, Computer Science & Engineering,** for his kind support and for providing necessary facilities to carry out the work.

We wish to convey our special thanks to **Dr. G.Balakrishna, Principal of Srinivasa Ramanujan Institute of Technology** for giving the required information in doing our project work. Not to forget, we thank all other faculty and non-teaching staff, and my friends who had directly or indirectly helped and supported us in completing our project in time.

We also express our sincere thanks to the Management for providing excellent facilities**.**

Finally, we wish to convey our gratitude to our family who fostered all the requirements and facilities that we need.

Project Associates

DECLARATION

We, Ms. G.Tejaswini bearing reg no : 174G1A0598, Ms. P.T.Usha bearing reg no : 174G1A05A3, Ms. D.Siri bearing reg no : 174G1A05B9, Ms. R.Raghu Nandini bearing reg no:17F41A0585, students of SRINIVASA RAMANUJAN INSTITUTE OF TECHNOLOGY, Rotarypuram , hereby declare that the dissertation entitled "END TO END CAR LICENSE PLATE DETECTION AND RECOGNITION USING DEEP NEURAL NETWORKS" embodies the report of our project work carried out by us during IV Year Bachelor of Technology under the guidance of Dr.B.Hari Chandana$_{,M.Tech,PhD}$, Department of CSE and this work has been submitted for the partial fulfillment of the requirements for the award of Bachelor of Technology degree.

The results embodied in this project report have not been submitted to any other Universities of Institute for the award of Degree.

G.TEJASWINI	Reg no: 174G1A0598
P.T.USHA	Reg no: 174G1A05A3
D.SIRI	Reg no: 174G1A05B9
R.RAGHU NANDINI	Reg no: 17F41A0585

ABSTRACT

License plate detection and recognition plays a significant role throughout this busy world, owing to the rise in vehicles day by day. Automatic Number Plate Detection and Recognition is a mass surveillance system that captures the image of vehicles and recognizes their license number. In this project, we tackle the problem of car license plate detection and recognition in natural scene images.

Here we propose a unified deep neural network, which can localize license plates and recognize the letters simultaneously in a single forward pass. The whole network can be trained end-to-end. In contrast to existing approaches which take license plate detection and recognition as two separate tasks and settle them step by step, our method jointly solves these two tasks by a single network. It not only avoids intermediate error accumulation but also accelerates the processing speed. For performance evaluation, we took a data set which includes images captured from various scenes under different conditions are tested. Extensive experiments show the effectiveness and the efficiency of our proposed approach

Contents

List of Figures

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Networks
VGG	Visual Geometry Group
RPN	Regional Proposal Networks
ROI	Region Of Interest
FC	Fully Connected
BRNN	Bidirectional Recurrent Neural Network

CHAPTER-1

INTRODUCTION

Day by day automatic car license plate detection and recognition plays an important role in intelligent transportation systems. For intelligent video monitoring and processing, the key content is automated license recognition. A license plate recognition system uses unique identification number which is assigned to each car by administrative authorities. Technically vehicle identification here is an application of image processing technique. Automated image recognition has a wide range of applications such as criminal pursuit, community monitoring, vehicle status checking, automatic toll collection and traffic flow magnitude control.

However, most of the existing algorithms only work well either under controlled conditions or with sophisticated image capture systems. It is still a challenging task to read license plates accurately in an uncontrolled environment. The difficulty lies in the highly complicated backgrounds, like the general text in shop boards, windows, guardrail or bricks,
and random photographing conditions, such as illumination, distortion, occlusion or blurring.

Deep learning has been a huge topic in machine learning research closer to Artificial Intelligence. A few decades ago, computer scientists developed a number of algorithms that trained computers to distinguish multiple instances of the same object. With the growing number of drivers, and an increase of vehicles on the road come problems associated with traffic. Some of these problems, such as accurate bridge and highway tolling, parking lot management, and speed prevention, can now be solved using machine learning. This project will explore the use of deep learning for the purpose of vehicle tracking and license plate recognition.

This project gave us the basic understanding of the modern neural network and how it works with applications in computer vision. By using the building blocks of neural networks, we were able to improve the accuracy of a model with its pre- -trained model using VGG16 network i.e, Convolutional neural network(CNN). We used our understanding of the pre--defined building model of the neutral network to compare the model accuracy using Tensor-flow framework.

What is Machine Learning?

Machine Learning is the field of study that gives computers the capability to learn without being explicitly programmed. ML is one of the most exciting technologies that one would have ever come across. As it is evident from the name, it gives the computer that makes it more similar to humans: *The ability to learn*. Machine learning is actively being used today, perhaps in many more places than one would expect.

Machine learning is about computers being able to perform tasks without being explicitly programmed… but the computers still think and act like machines. Their ability to perform some complex tasks — gathering data from an image or video, for example — still falls far short of what humans are capable of.

History of Machine Learning

Machine Learning is, in part, based on a model of brain cell interaction. The model was created in 1949 by Donald Hebb in a book titled "*The Organization of Behaviour"*. The book presents Hebb's theories on neuron excitement and communication between neurons. Hebb wrote, "When one cell repeatedly assists in firing another, the axon of the first cell develops synaptic knobs (or enlarges them if they already exist) in contact with the soma of the second cell." Translating Hebb's concepts to artificial neural networks and artificial neurons, his model can be described as a way of altering the relationships between artificial neurons (also referred to as nodes) and the changes to individual neurons. The relationship between two neurons/nodes strengthens if the two neurons/nodes are activated at the same time and weakens if they are activated separately. The word "weight" is used to describe these relationships, and nodes/neurons tending to be both positive or both negative are described as having strong positive weights. Those nodes tending to have opposite weights develop strong negative weights (e.g. 1×1=1, -1x-1=1, -1×1=-1).

The Perceptron

In 1957, Frank Rosenblatt – at the Cornell Aeronautical Laboratory – combined Donald Hebb's model of brain cell interaction with Arthur Samuel's Machine Learning efforts and created the perceptron. The perceptron was initially planned as a machine, not a program. The software, originally designed for the IBM 704, was installed in a custom-built machine called the Mark 1 perceptron, which had been constructed for image recognition. This made the software and the algorithms transferable and available for other machines.

Described as the first successful neuro-computer, the Mark I perceptron developed some problems with broken expectations. Although the perceptron seemed promising, it could not recognize many kinds of visual patterns (such as faces), causing frustration and stalling neural network research. It would be several years before the frustrations of investors and funding agencies faded. Neural network/Machine Learning research struggled until a resurgence during the 1990s.

Multilayers Provide the Next Step

In the 1960s, the discovery and use of multi layers opened a new path in neural network research. It was discovered that providing and using two or more layers in the perceptron offered significantly more processing power than a perceptron using one layer. Other versions of neural networks were created after the perceptron opened the door to "layers" in networks, and the variety of neural networks continues to expand. The use of multiple layers led to feed forward neural networks and backpropagation.

Back propagation, developed in the 1970s, allows a network to adjust its hidden layers of neurons/nodes to adapt to new situations. It describes "the backward propagation of errors," with an error being processed at the output and then distributed backward through the network's layers for learning purposes. Back propagation is now being used to train deep neural networks.

An Artificial Neural Network (ANN) has hidden layers which are used to respond to more complicated tasks than the earlier perceptrons could. ANNs are a primary tool used for Machine Learning. Neural networks use input and output layers and, normally, include a hidden layer (or layers) designed to transform input into data that can be used the by output layer. The hidden layers are excellent for finding patterns too complex for a human programmer to detect, meaning a human could not find the pattern and then teach the device to recognize it.

What is deep learning?

Deep learning models introduce an extremely sophisticated approach to machine learning and are set to tackle these challenges because they've been specifically modeled after the human brain. Complex, multi-layered “deep neural networks” are built to allow data to be passed between nodes (like neurons) in highly connected ways. The result is a non-linear transformation of the data that is increasingly abstract. While it takes tremendous volumes of data to ‘feed and build’ such a system, it can begin to generate immediate results, and there is relatively little need for human intervention once the programs are in place.

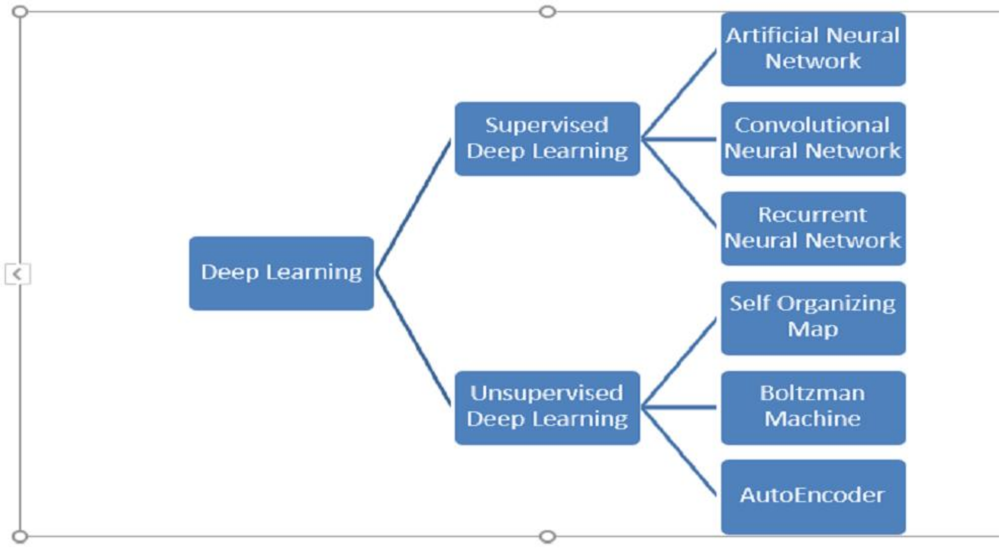


Fig.1.1. Classification of Deep learning

CNNs are a fundamental example of deep learning, where a more sophisticated model pushes the evolution of artificial intelligence by offering systems that simulate different types of biological human brain activity. A convolutional neural network (CNN) is a specific type of artificial neural network that uses perceptrons, a machine learning unit algorithm, for supervised learning, to analyze data. CNNs apply to image processing, natural language processing and other kinds of cognitive tasks. Like other kinds of artificial neural networks, a convolutional neural network has an input layer, an output layer and various hidden layers. Some of these layers are convolutional, using a mathematical model to pass on results to successive layers. This simulates some of the actions in the human visual cortex.

CHAPTER-2

LITERATURE SURVEY

2.1 Introduction

L. Neumann and J. Matas, [1] The proposed a system to localization of number plate mainly for the vehicles in West Bengal (India) and segmented the numbers as to identify each number separately. This paper presents an approach based on simple and efficient morphological operation and edge detection method. He also presents a simple approach to segmented all the letters and numbers used in the number plate. After reducing noise from the input image, we try to enhance the contrast of the binarized image using histogram equalization. We mainly concentrate on two steps:one is to locate the number plate and second is tosegment all the number and letters to identify each number separately.

K. Wang, B. Babenko, and S. Belongie, [3]"End-to-end scene text recognition," Describe a comprehensive survey on existing (Automatic License Plate Recognition) ALPR Techniques by categorizing them according to the features used in each stage. Comparisons of them in the terms of Pros, Cons, Recognition results, & Processing speeds were addressed.

M. Jaderberg, A. Vedaldi, and A. Zisserman, [6]"Deep features for text spotting,". Presented an efficient and robust method of locating license plate is presented. The method makes use of the rich corner information in the plate area and the edge information of license plates. It can deal with more difficult location problems, especially with a license plate existing in a complicated background.

C. N. E. Anagnostopoulos, I. E. Anagnostopoulos, V. Loumos, and E. Kayafas,[11] "A license plate-recognition algorithm for intelligent transportation system applications," they introduced that high level of precision has been required by the number plate recognition when streets are occupied and number of vehicles are passing through. In this paper, by optimizing different parameters, they have accomplished an exactness of 98%

A. Bissacco, M. Cummins, Y. Netzer, and H. Neven[5] developed an ANPR framework focused on the learning capabilities of convolutional neural networks. The self-synthesized function of CNN was used here, as it distinguishes the vehicle states from the number plate. The system was organized in this work in an echelon network of feature detectors that conducted consecutive processing of visual data pertaining to the dominant visual processing experience of the visual cortex, which influenced the computational model of the CNN. The findings of this research were observed with fewer training samples and turned out to be as 90 per cent higher precision rate.

M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman,[7] suggested a device that not only tracks several targets but also obtains high-quality images on plate numbers. A computer with a tuned dual-camera system has been built here by the author; a stationary camera and a pan-tilt-zoom camera are designed to monitor moving conveyance in an open field. The license plate for recognition has been sequentially identified by CNN classifier. As 64 vehicles entered this region illegally, data was composed manually from the science pictures and 59 IDs were accurately detected using this tool.

W. Zhou, H. Li, Y. Lu, and Q. Tian, [10]"Principal visual word discovery for automatic license plate detection," IEEE Trans. The self-synthesized function of CNN was used here, as it distinguishes the vehicle states from the number plate. The system was organized in this work in an echelon network of feature detector. The findings of this research were observed with fewer training samples and turned out to be as 90 per cent higher precision rate.

A. H. Ashtari, M. J. Nordin, and M. Fathy, [14]"An iranian license plate recognition system based on color features," [14] proposed an ANPR method based on the characteristics and variations of the plates therein. The author has proposed in this work an algorithm that is enhanced to perform with Ghanaian license plate for conveyance.

L. Neumann and J. Matas, [2]"Real-time scene text localization and recognition," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Character recognition was rendered with the use of tesseract OCR engine. Feature detection was slightly low but had a good success rate, with an average speed of 0.185s detecting 454 plates with 90.8 per cent accuracy. The optical character recognition provided an average of 0.031s for the procedure and successfully identified approximately 60 per cent of the detected plates.

T. Wang, D. Wu, A. Coates, and A. Y. Ng,[4] "End-to-end text recognition with convolutional neural networks," proposed a three-stage license plate recognition system based on Fast-RCNN which was used for various shooting angles and numerous oblique images.

M. Liao, B. Shi, X. Bai, X. Wang, and W. Liu,[8] "TextBoxes: A fast text detector with a single deep neural network,"suggested a device that not only tracks several targets but also obtains high-quality images on plate numbers. A computer with a tuned dual-camera system has been built here by the author; a stationary camera and a pan-tilt-zoom camera are designed to monitor moving conveyance in an open field. The license plate for recognition has been sequentially identified by CNN classifier. As 64 vehicles entered this region illegally, data was composed manually from the science pictures and 59 IDs were accurately detected using this tool.

2.2 Existing System

In the existing system, it considers plate detection and recognition as two separate tasks, and solves them respectively by different methods. The Existing methods for plate detection are:

- ➢ Edge-based detection
- ➢ Color-based detection
- ➢ Texture-based detection
- ➢ Character-based detection

Edge-based detection:

- o License plates are normally in a rectangular shape with a specific aspect ratio, and they present higher edge density than elsewhere in the image, so edge information is used widely to detect license plates.
- o It selects regions with high edge density and removes spare regions.
- o It is fact in computation but sensitive to the unwanted edges.

Color-based detection:

- Color based approaches are based on the fact that color of the license plate is usually different from that of the car body.
- Color-based methods can be used to detect inclined or deformed license plates.
- However, they are very sensitive to various illumination conditions in natural scene images, and they cannot distinguish other objects in the image with similar color and size as the license plates.

Texture-based detection:

- Texture-based approaches attempted to detect license plates according to the unconventional pixel intensity distribution in plate regions.
- This uses more discriminative characters than edge and color but results in high computational complexity.

Character-based detection:

- In character based method, the characters are extracted from the image and find relationship among those characters.
- This detects license plate in the form of bounding boxes.
- Bounding box is an imaginary rectangle, serves as a point of reference for object detection. Each grid cell predicts a bounding box including(x,y) coordinate(w,h).

Previous work on license plate recognition typically segments characters in the license plate firstly, and then recognizes each segmented character using Optical Character Recognition (OCR) techniques.

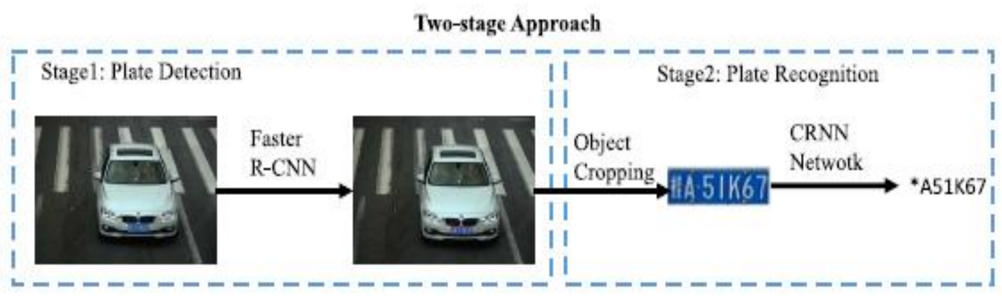


Fig.2.1. Two stage approach

2.3 Proposed System

- Our approach addresses both detection and recognition using a single deep neural network.
- The whole network is trained end-to-end, with both localization loss and recognition loss being jointly optimized, and shows improved performance.
- This can be achieved through a model architecture that consists of layers namely
  - Low-Level Feature Extraction
  - Plate Proposal Generation
  - Proposal Processing and Pooling
  - Plate Detection Network & Plate Recognition Network

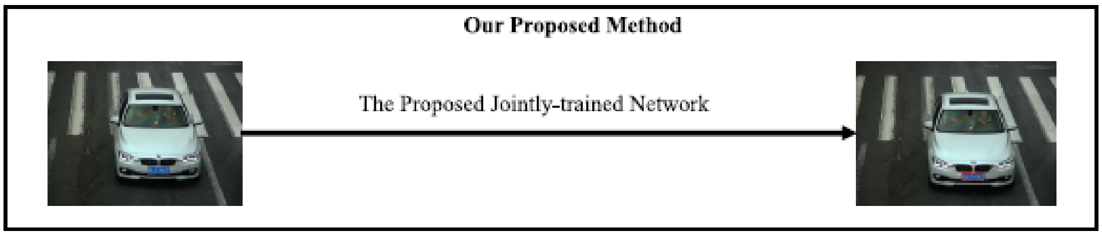


Fig.2.2. The proposed jointly-trained network

2.4 Hardware Requirements

Hardware requirements which we need in this project are:

1. I3 Processor Based Computer or higher
2. Memory: 1GB
3. Hard Drive: 5GB
4. Internet Connection

2.5 Software Requirements

Software requirements which we need in this project are:

1. Operating System	:	Windows 10
2. Tools	:	PyCharm
3. Languages Used	:	Python

CHAPTER-3

CONVOLUTIONAL NEURAL NETWORKS

3.1 Introduction

In the past few decades, Deep Learning has proved to be a very powerful tool because of its ability to handle large amounts of data. The interest to use hidden layers has surpassed traditional techniques, especially in pattern recognition. One of the most popular deep neural networks is Convolutional Neural Networks.

A pre-trained deep CNN model, i.e., VGG-16, is used to extract the deep features of an image. The information about the neural network is strictly concealed by utilizing the lattice-based homomorphic scheme. We implement a real number computation mechanism and a divide-and-conquer CNN evaluation protocol to enable our framework to securely and efficiently evaluate the deep CNN with a large number of inputs.

VGG16 is a convolution neural net (CNN) architecture which was used to win ILSVR(ImageNet) competition in 2014. It is considered to be one of the excellent vision model architecture till date. Most unique thing about VGG16 is that instead of having a large number of hyperparameter they focused on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2.

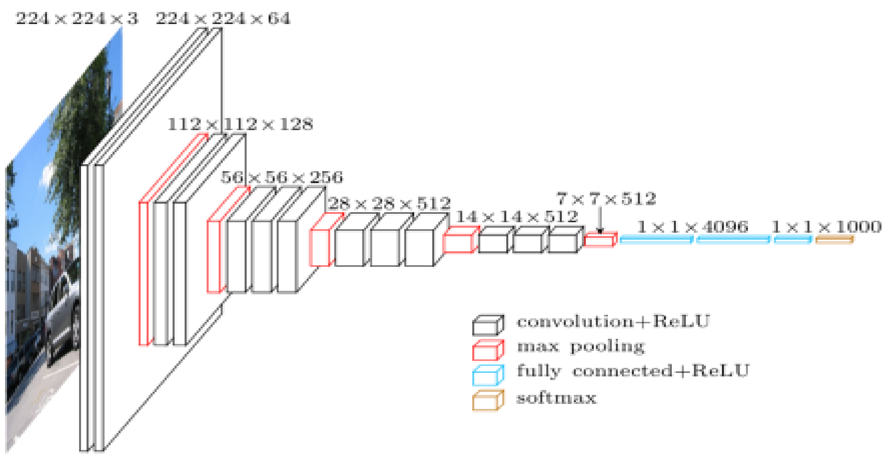


Fig.3.1. VGG16 network architecture

It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 FC(fully connected layers) followed by a softmax for output. The 16 in VGG16 refers to it has 16 layers that have weights. This network is a pretty large network and it has about 138 million (approx) parameters.

CNN's were first developed and used around the 1980s. The most that a CNN could do at that time was recognize handwritten digits.

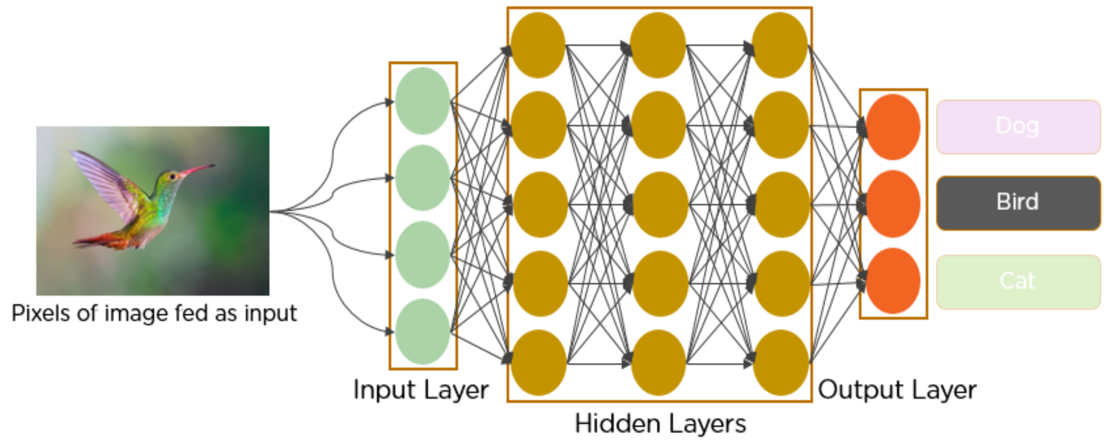


Fig.3.2. Working structure of CNN Architecture

In 2012, computer vision took a quantum leap when a group of researchers from the University of Toronto developed an AI model that surpassed the best image recognition algorithms and that too by a large margin.

Over the years CNNs have become a very important part of many Computer Vision applications. So let's take a look at the workings of CNNs.

3.2 Background of CNNs

CNN's were first developed and used around the 1980s. The most that a CNN could do at that time was recognize handwritten digits. This was a major drawback for CNNs at that period and hence CNNs were only limited to the postal sectors and it failed to enter the world of machine learning.

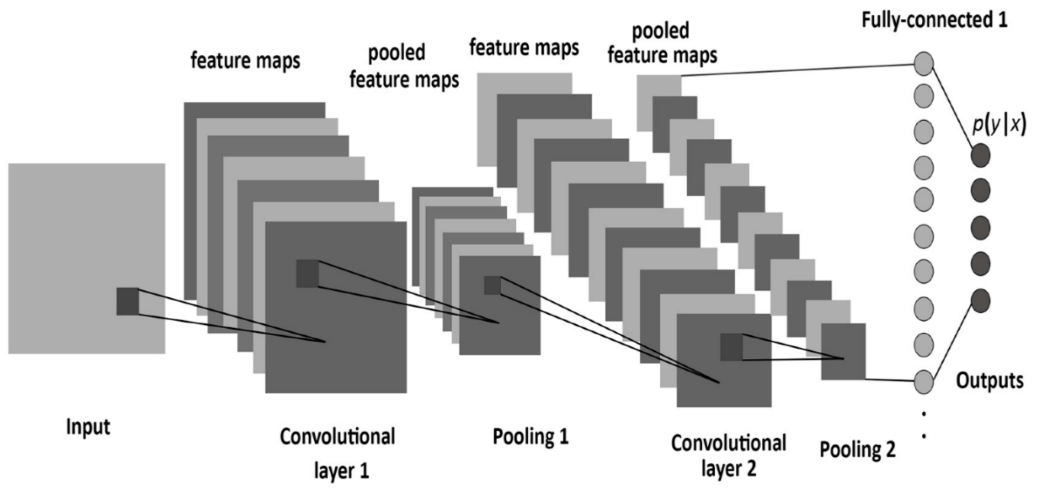


Fig.3.3. Background of CNN

The availability of large sets of data, to be more specific ImageNet datasets with millions of labelled images and an abundance of computing resources enabled researchers to revive CNNs.

What exactly is a CNN?

In deep learning a **convolutional neural network** (**CNN/ConvNet**) is a class deep neural networks, most commonly applied to analyze visual imagery. It uses a special technique called Convolution. Now in mathematics **convolution** is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.

3.3 Working of CNN

CNN is composed of three layers where input image is divided into matrix of pixels.Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. Features of a fully connected layer. Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification**.**The three layers of CNN are:

- Convolutional layer
- Pooling layer
- Fully connected layer

3.3.1 Convolutional Layer

Convolutional layer is the core building block of the CNN. It carries the main portion of the network's computational load. In this layer, the mathematical operation of convolution is performed between input image , the dot product is taken between the filter and parts of the input image with respect to size of filter(MxM).

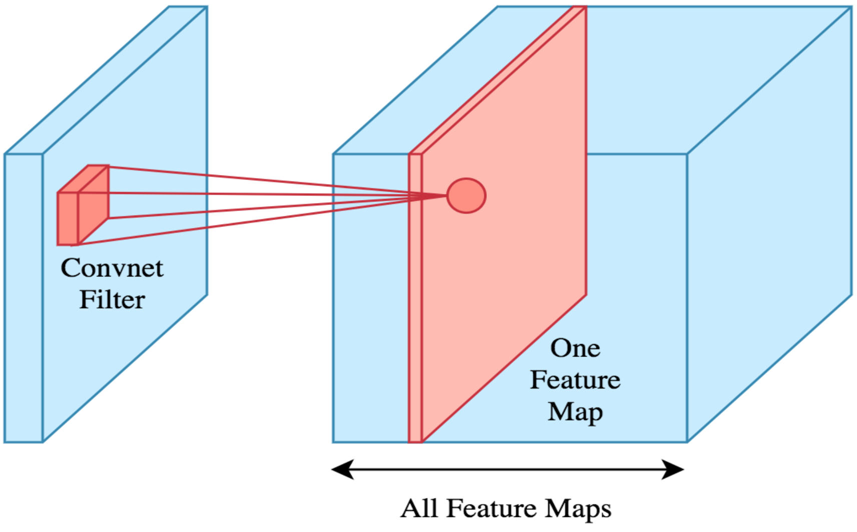


Fig.3.4. Convolutional layer

The output of convolutional layer is the Feature map which gives us information about the image such as corners and images.Later this feature map is fed to other layers to learn several other features of the input image.

3.3.2 Pooling layer

In most cases, a convolutional layer is followed by a Pooling layer. The primary aim of this layer is to reduce the dimensions of the featured maps thereby, reducing the number of parameters to learn and amount of computation to be performed.

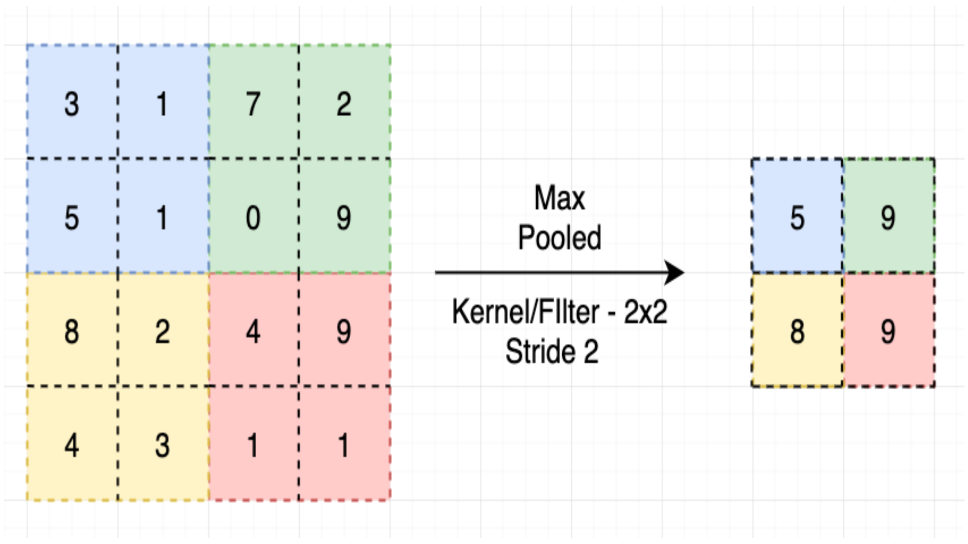


Fig.3.5. Pooling layer with max pooling

This is performed by decreasing the connections between the layers and independently operates on each feature map.

3.3.2.1 Padding

There are two problems arises with convolution:

1. Every time after convolution operation, original image size getting shrinks, as we have seen in above example six by six down to four by four and in image classification task there are multiple convolution layers so after multiple convolution operation, our original image will really get small but we don't want the image to shrink every time.

2. The second issue is that, when kernel moves over original images, it touches the edge of the image less number of times and touches the middle of the image more

   number of timesand it overlaps also in the middle. So, the corner features of any image or on the edges aren't used much in the output.

So, in order to solve these two issues, a new concept is introduced called **padding.** Padding preserves the size of the original image.

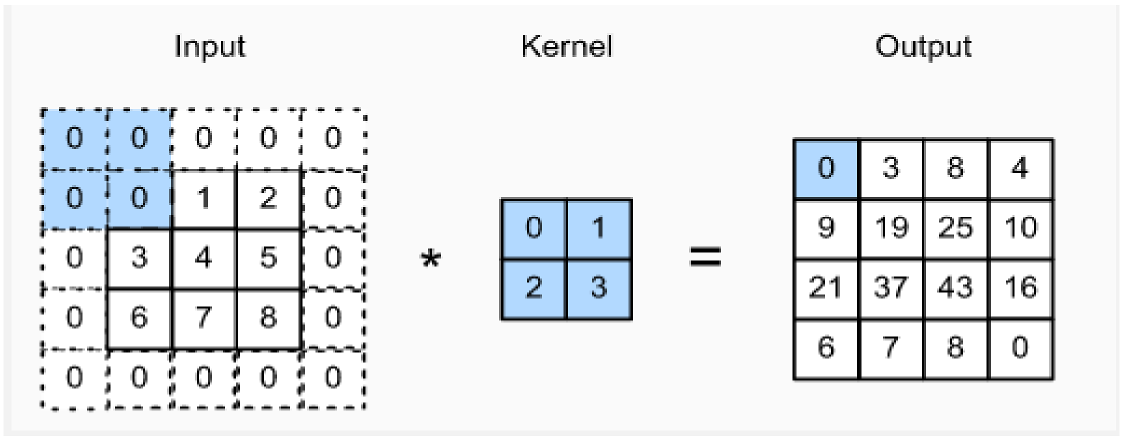
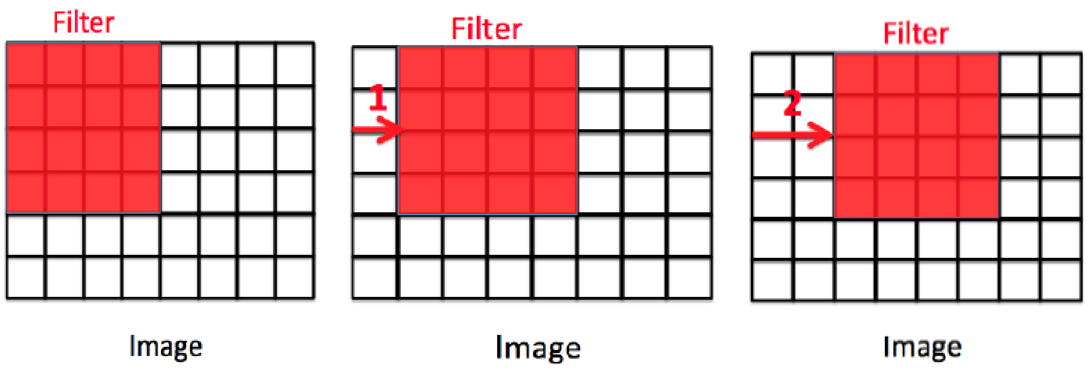


Fig.3.6. Padded image convolved with 2*2 kernel.

So if a $n*n$ matrix convolved with an f*f matrix the with padding p then the size of the output image will be (n + 2p — f + 1) * (n + 2p — f + 1) where p =1 in this case.

3.3.2.2 Striding:



left image: stride =0, middle image: stride = 1, right image: stride =2

Fig.3.7. Striding

Stride is the number of pixels shifts over the input matrix. For padding p, filter size $f*f$ and input image size $n*n$ and stride ‘s’ our output image dimension will be [$\{(n + 2p - f + 1) / s\} + 1] * [\{(n + 2p - f + 1) / s\} + 1]$.Depending upon method used,there are several types pooling operations. However, we are using Max Pooling.In Max-Pooling, the largest element is taken from feature map. When added to a model, max pooling layer reduces the dimensionality of images by reducing the number of pixels in the output from the previous convolutional layer.

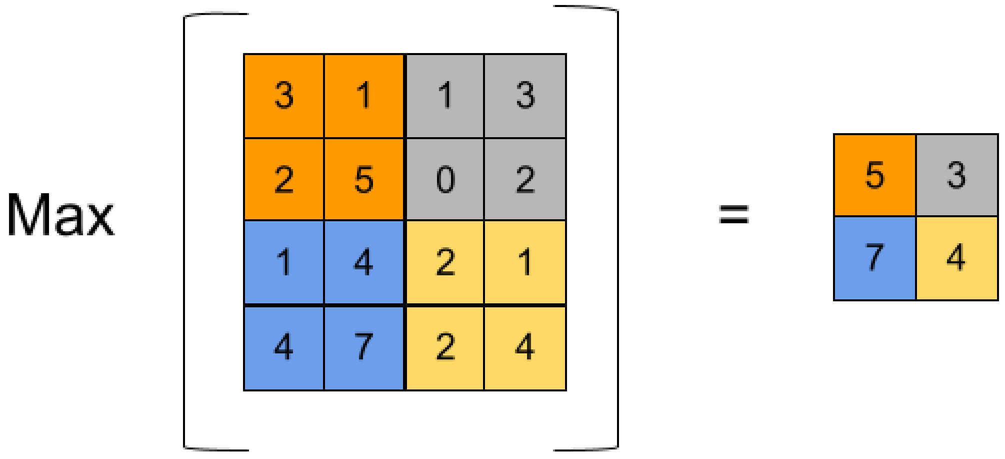


Fig.3.8. Representation of max-pooling layer working

The total sum of elements in a predefined sized image section. The pooling layer usually serves as a bridge between the convolutional layer and the Fully-Connected layer.

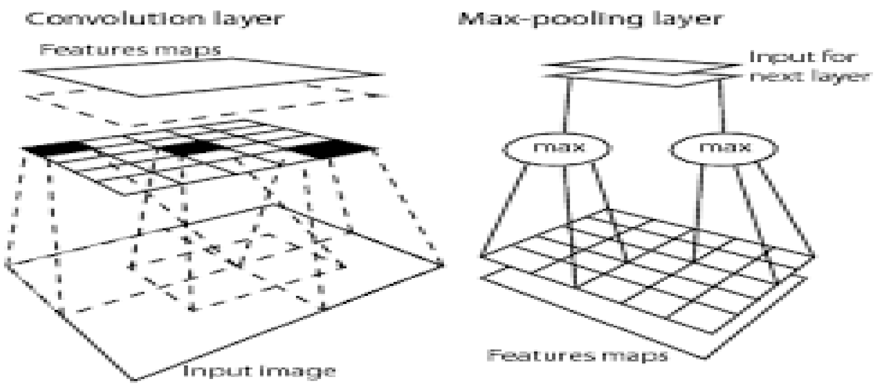


Fig.3.9. Max pooling layer working on feature maps

3.3.3 Fully connected layer

The fully connected layer (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form few layers of a CNN Architecture

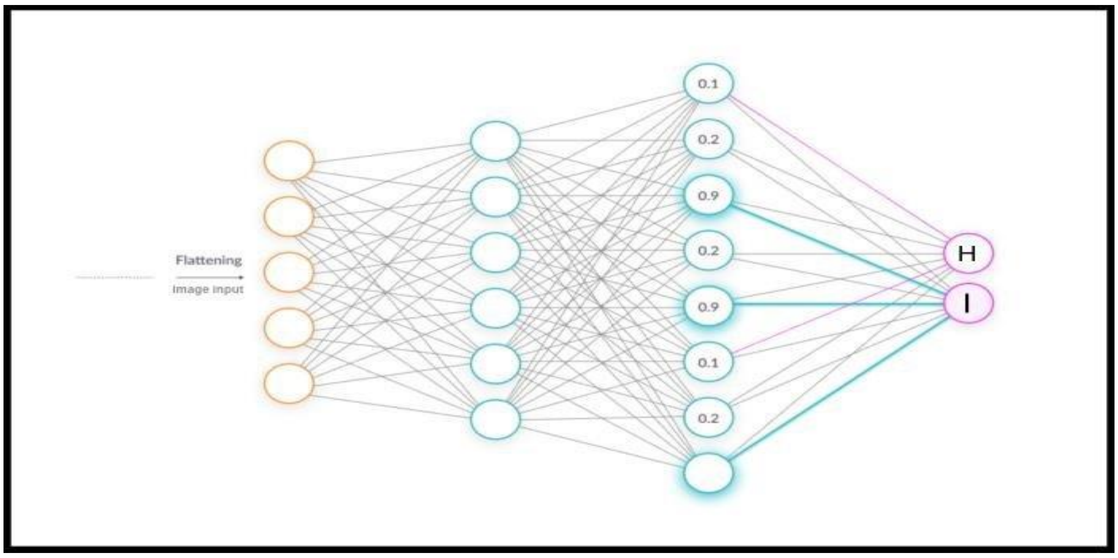


Fig.3.10. Featuring of FC layer

In this, the input image from previous layers are flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, Classification process begins to take place.

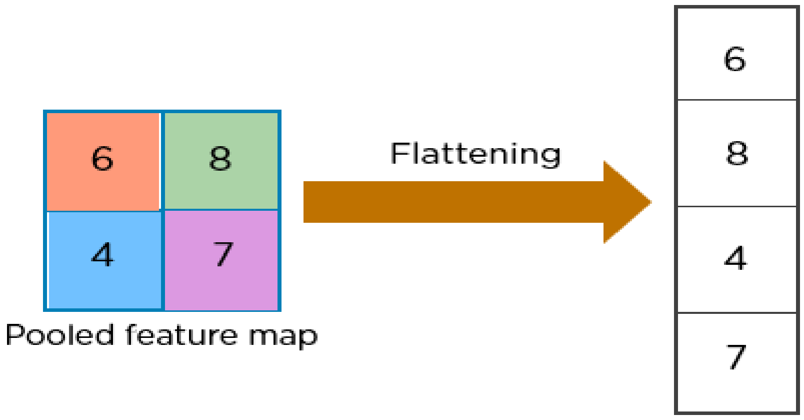


Fig.3.11. Flattening of pooling layer output to vertical vector

Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long featurevector. And it is connected to the final classification model, which is called a fully-connected layer.

3.4 Activation Functions

Finally, one of the most important parameters of the CNN model is the Activation function.

In simple words, it decides which information of the information of the model should fire in the forward direction and which ones should not end of the network.

Here we used ReLU activation function, which maps all negative values to zero and considering only positive values.

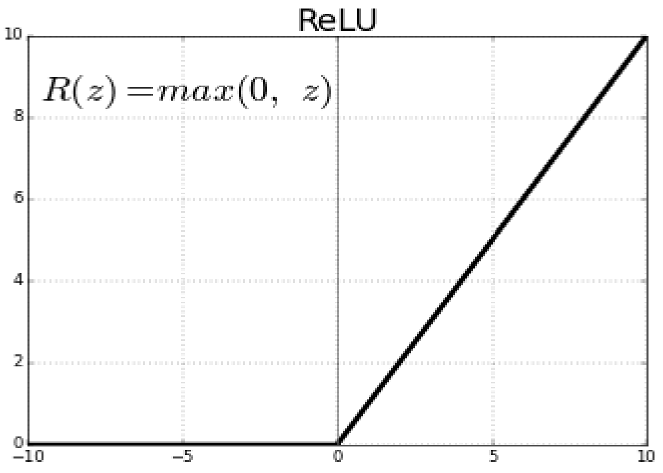


Fig.3.12. Functioning of ReLU Activation Layer

The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time**.**

CHAPTER 4

REGIONAL PROPOSAL NETWORK

4.1 Introduction

Region Proposal Network (RPN). RPN is a fully convolutional network, trained end-to-end, that simultaneously predicts object boundaries and object scores at each detection. With RPN being so important to Faster R-CNN, which continues to be one of the best object detection frameworks available to researchers, the bulk of this piece will focus on the RPN design and the concept of anchor boxes and non-maximum suppression.

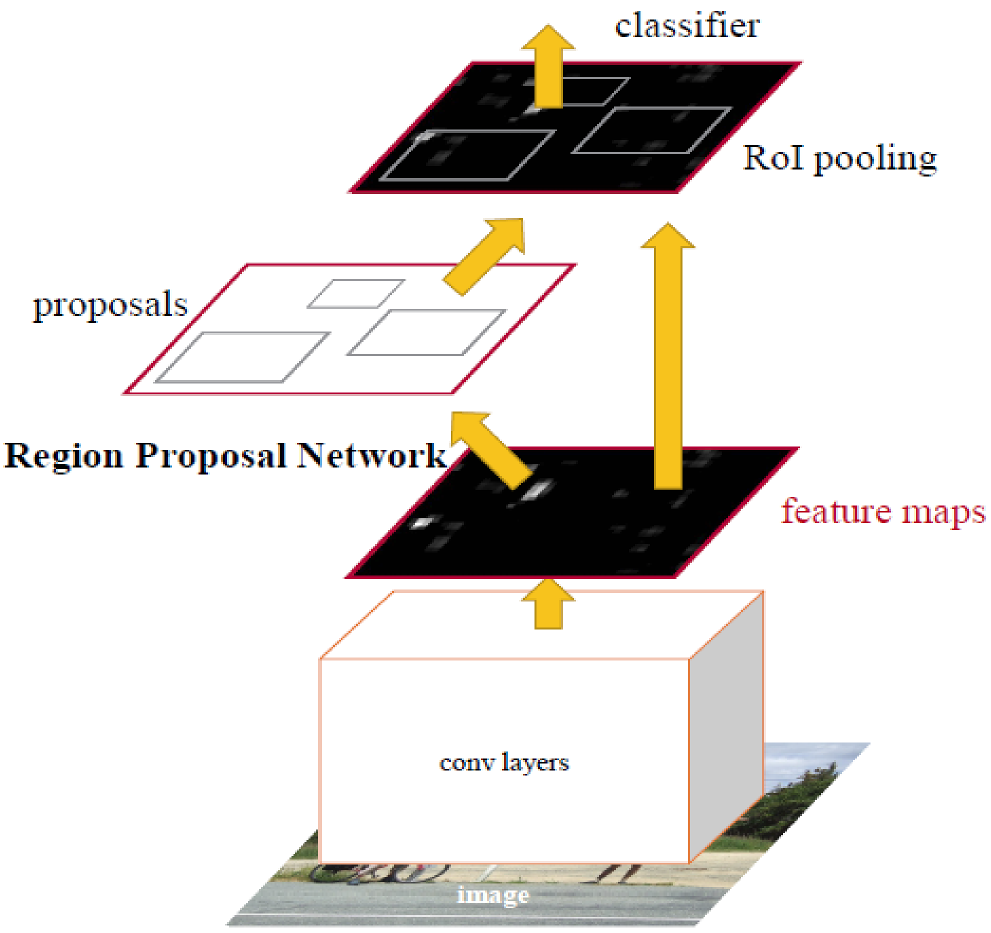


Fig.4.1. Architecture of RPN

The goal of RPN is to output a set of proposals, each of which has a score of its probability of being an object and also the class/label of the object. RPN can take any sized input to achieve this task. These proposals are further refined by feeding to 2 sibling fully connected layers-one for bounding box regression and the other for box classification i.e is the object foreground or background.

The RPN, to generate the proposals, slide a small network over the output of the last layer of feature map. This network uses nxn spatial window as input from the feature map. Each sliding window is mapped to a lower dimensional feature.The position of the sliding window provides localization information with reference to the image while the regression provides finer localization information.

4.2 Anchor point

Every point in the feature map generated by the backbone network is an anchor point. We need to generate anchor boxes for every anchor point. We generate candidate boxes using two parameters — scales and aspect ratios. The boxes need to be at image dimensions, whereas the feature map is reduced depending on the backbone. For example, in the case of vgg16, the image is reduced by 16 times by the end of the backbone. So how do we generate boxes at image dimensions? We use this 16 as the stride in generating anchor boxes at the image level. (Ex: If anchor scales are [8,16,32] and ratios are [0.5,1,2] and stride is 16, then we use the combination of these scales and ratios to generate 9 anchor boxes for each anchor point and then take a stride of 16 over the image to take the next anchor box.)

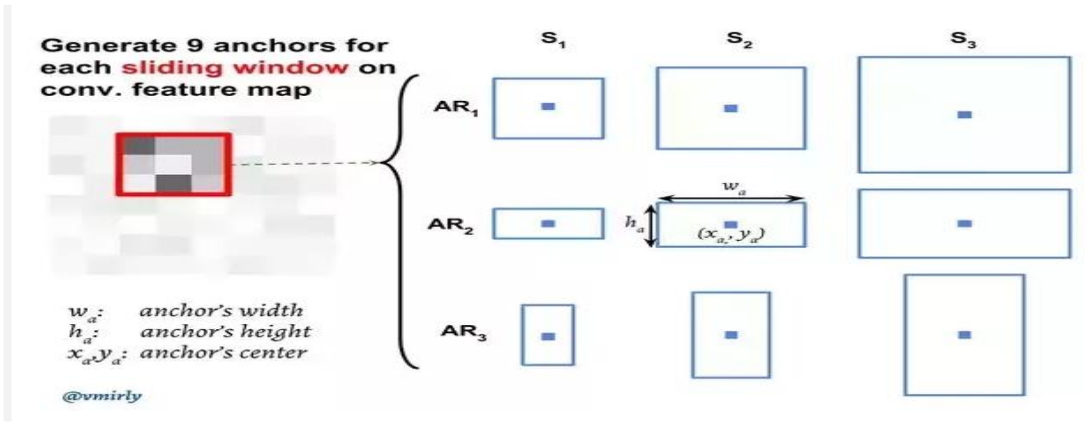


Fig .4.2. Anchor boxes generation

4.3 Anchor Boxes

Anchor box is the one of the most important concept in Faster R-CNN. These are responsible for providing a predefined set of bounding boxes of different sizes and ratios that are going to be used for reference when first predicting object locations for the RPN. These boxes are defined to capture the scale and aspect ratio of specific object classes you want to detect and are typically chosen based on object sizes in the training dataset. Anchor Boxes are typically centered at sliding window.The original implementation uses 3 scales and 3 aspect ratios, which means k=9. If the final feature map from feature extraction layer has width W and height H , then the total number of anchors generated will be W*H*k.

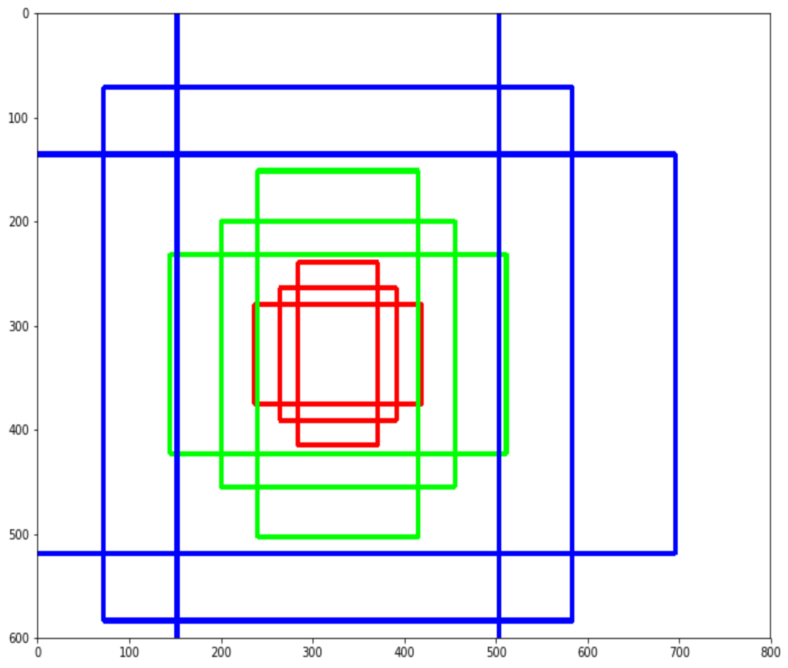


Fig .4.3. Generation of anchor boxes for anchor point

4.4 Need of Anchor Boxes

The main reason to use Anchor Boxes is so that we can evaluate all object predictions at once. They help speed up and improve efficiency for the detection portion of a deep learning neural network framework. Anchor boxes also help to detect multiple objects, objects of different scales, and overlapping objects without the need to scan an image with a sliding window that computes a separate prediction at every potential position as we saw in previous versions of R-CNN. This makes real time object detection possible.

The improvement in speed is made possible because Anchor boxes are translation invariant and so the same ones can be used at every location. As we mentioned before, the information from these anchor boxes is relayed to the regression and classification layer, where regression gives offsets from anchor boxes and classification gives the probability that each regressed anchor shows an object.

4.5 How Anchor Boxes help in identifying an object ?

Although Anchors take the final feature map as input, the final anchors refer to the original image. This is made possible because of the convolution correspondence property of CNN’s, thus, enabling extracted features to be associated back to their location in that image. For a down sampling ratio `d`, the feature map will have dimensions `W/d * H/d`. In other words, in an image, each anchor point, considering we have just one at each spatial location of feature map, will be separated by `d` spatial pixels. A value of 4 or 16 is common for `d`, which also corresponds to the *stride* between tiled anchor boxes. This is a tunable parameter in the configuration of Faster R-CNN. Making it too low or too high can give rise to localization errors. One way to mitigate this localization errors is to learn the offsets applied to each anchor box which is the goal of the regression layer we discussed above.

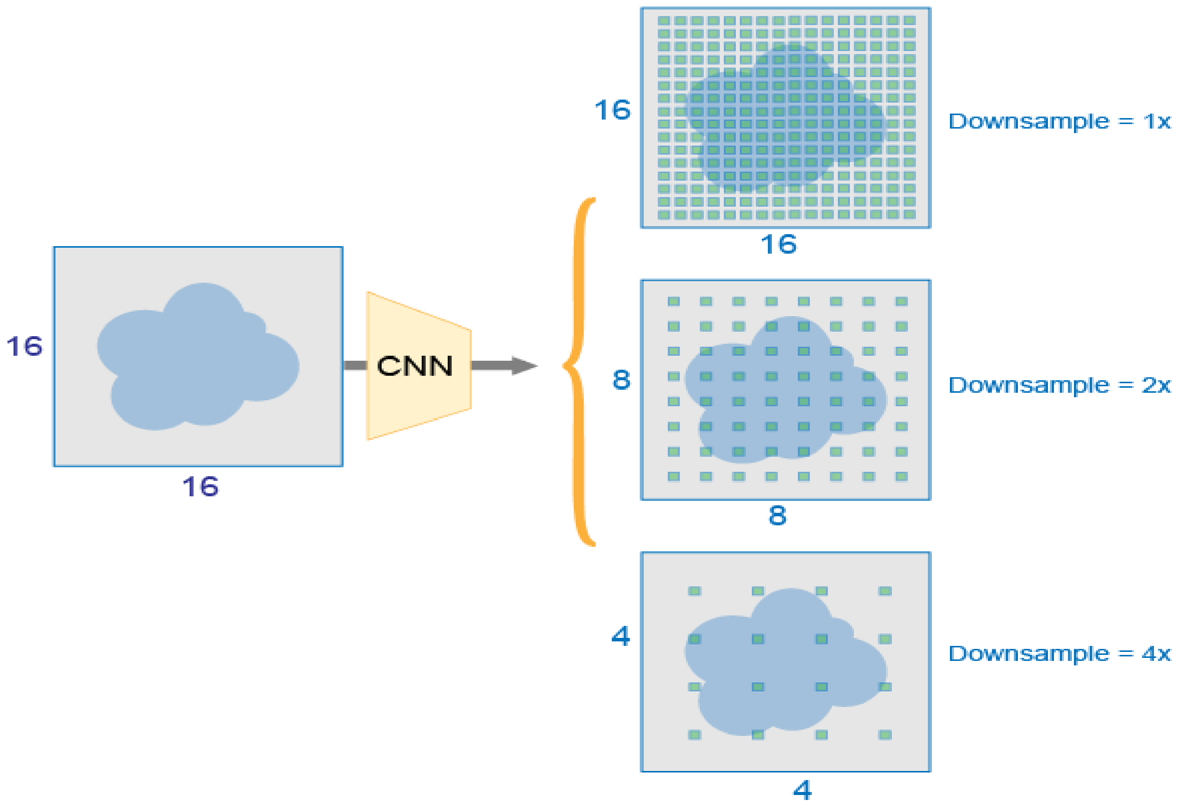


Fig.4.4. Types of down samples

Anchor boxes at each spatial location, mark an object as foreground or background depending on its IOU threshold with the ground truth. All the anchors are places in a mini-batch and trained using softmax cross entropy to learn the classification loss and smooth L1 loss for regression. We use smooth L1 loss as regular L1 loss function is not differentiable at 0.

CHAPTER-5

REGION OF INTEREST POOLING

5.1 Introduction

ROI (Region of Interest) is a proposed region from the original image. We're not going to describe how to extract those regions because there are multiple methods to do only that. The only thing we should know right now is there are multiple regions like that and all of them should be tested at the end.

5.2 How Fast R-CNN works?

5.2.1 Feature extraction

Fast R-CNN is different from the basic **R-CNN** network. It has only one convolutional feature extraction (in our example we're going to use VGG16).

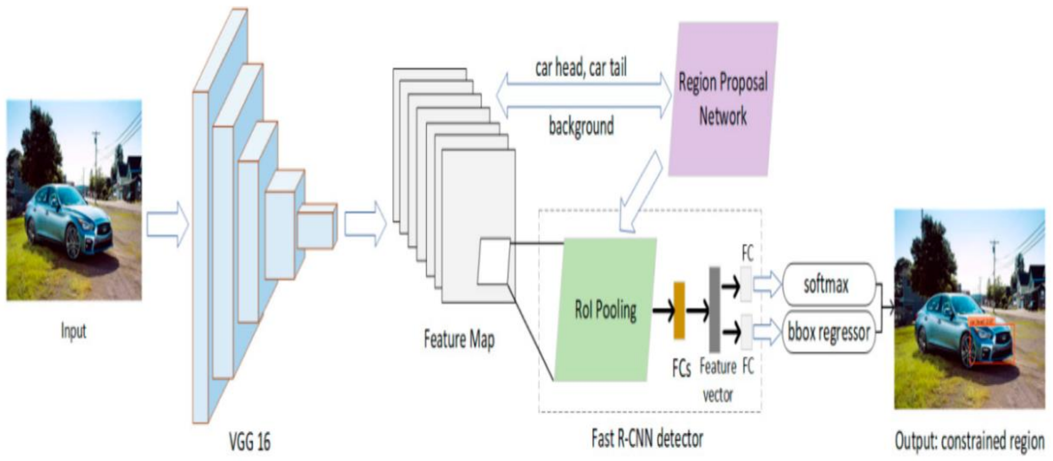


Fig.5.1. VGG 16 feature extraction output size

Our model takes an image input of size **512x512x3** (width x height x RGB) and VGG16 is mapping it into a **16x16x512** feature map. You could use different input sizes (usually it's smaller, default input size for VGG16 in Keras is 224x224).

If you look at the output matrix you should notice that it's **width** and **height** is exactly 32 times smaller than the inputimage (512/32 = 16). That'simportant because all RoIs have to be scaled down by this factor.

5.2.2 Sample ROI

Here we have 4 different RoIs. In the actual Fast R-CNN you might have thousands of them but printing all of them would make image unreadable.

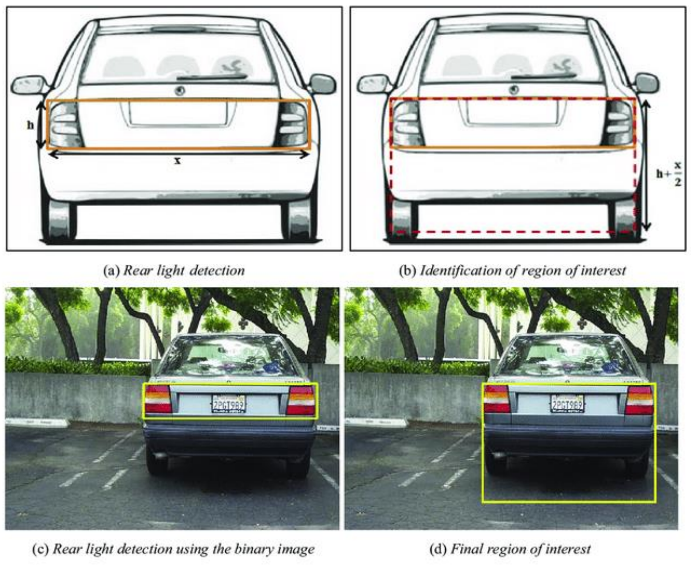

h
x
h+$\frac{x}{2}$
(a) Rear light detection
(b) Identification of region of interest
(c) Rear light detection using the binary image
(d) Final region of interest

Fig.5.2. Region of interest detection in license plate

It's important to remember that **ROI is NOT a bounding box**. It might look like one but it's just a proposal for further processing. Many people are assuming that because most of the papers and blog posts are creating proposals in place of actual objects. It's just more convenient that way, I did it as well on my image. Here is an example of a different proposal area which also is going to be checked by Fast R-CNN (green box).

5.3 How to get ROI's from featured maps?

Now when we know what RoI is we have to be able to map them onto VGG16's output feature map.

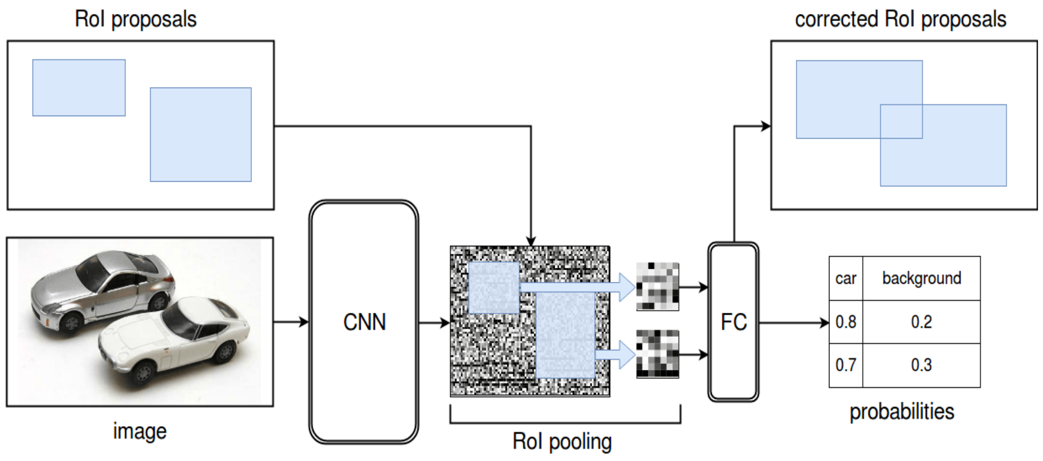


Fig.5.3. Mapping of ROIs onto output of VGG16

Every RoI has it’s original coordinates and size. From now we’re going to focus only on one of them:

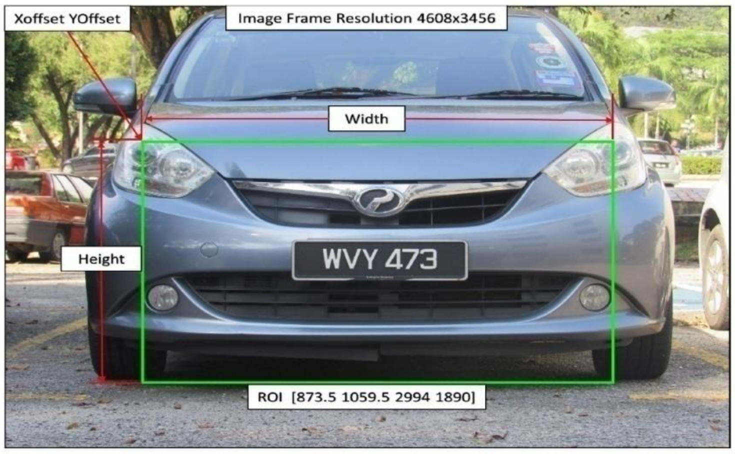

Xoffset YOffset
Image Frame Resolution 4608x3456
Width
Height
WVY 473
ROI [873.5 1059.5 2994 1890]

Fig.5.4. ROI target

Its original size is **4608/3456** and the top left corner is set to be in **(873,1059)**. As you could probably tell, we’re not able to divide most of those numbers by **1024**(scale factor).

- width: 3456/1024 = 3.375
- height: 4608/1024= ~4.5
- x: 1059/64= 16.546
- y: 873/64 = 13.64

Only the last number (Y coordinate of the top left corner) makes sense. That's because we're working on a **32x32** grid right now and only numbers we care about are integers (to be more precise: Natural Numbers).

5.3.1 Quantization of coordinates on the feature map

Quantization is a process of constraining an input from a large set of values (like real numbers) to a discrete set (like integers). If we put our original RoI on feature map it would look like this: We cannot really apply the pooling layer on it because some of the "cells" are divided. What quantization is doing is that every result is rounded down before placing it on the matrix. **9.25** becomes **9**, **4.53** becomes **4**, etc. We don't have to deal with it because it's still going to work but there is a different version of this process called **ROI Align** which fixes that.

RoI Pooling

Now when we have our RoI mapped onto feature map we can apply pooling on it. Once again we're going to choose the size of **RoI Pooling** layer just for our convenience, but remember the size might be different. You might ask "Why do we even apply RoI Pooling?" and that's a good question. If you look at the original design of Fast R-CNN:

After **RoI Pooling Layer** there is a **Fully Connected layer** with a fixed size. Because our RoIs have different sizes we have to pool them into the same size

(**3x3x512** in our example). At this moment our mapped RoI is a size of **4x6x512** and as you can imagine we **cannot divide 4 by 3** : That’s where quantization strikes again.

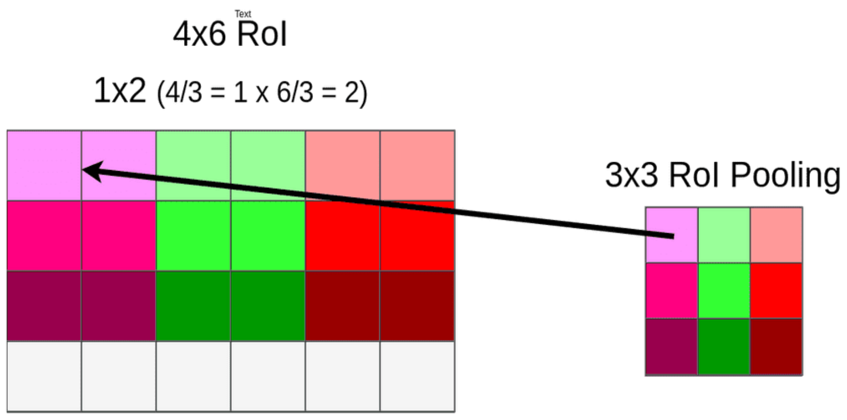


Fig.5.5. Mapped ROI and pooling layer

This time we don’t have to deal with coordinates, only with size. We’re lucky (or just convenient size of pooling layer) that **6** could be divided by **3** and it gives **2**, but when you divide **4** by **3** we’re left with **1.33**. After applying the same method (**round down**) we have a **1x2 vector**.

CHAPTER-6

LICENSE PLATE DETECTION & RECOGNITION

6.1 License plate detection work

Plate detection network aims to judge whether the proposed RoIs are car license plate or not, and refine the coordinates of plate bounding boxes.Two fully connected layers with 2048 neurons and a dropout rate of 0.5 are employed here to extract discriminative features for license plate detection. The features from each RoI are flattened into a vector and passed through the two fully connected layers.

The encoded features are then fed concurrently into two separate linear transformation layers respectively for plate classification and bounding box regression. The plate classification layer has 2 outputs, which indicate the softmax probability of each RoI as plate/non-plate. The plate regression layer produces the bounding box coordinate offsets for each proposal, as used in the region proposal network.

6.2 Fully connected layers

Fully Connected Layer is simply, feed forward neural networks. Fully Connected Layers form the last few layers in the network.

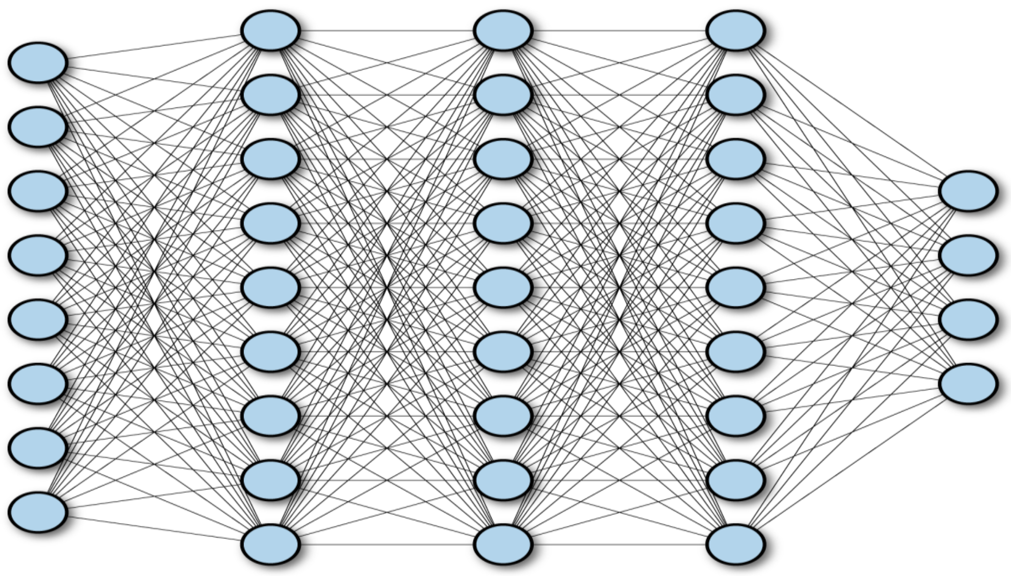


Fig.6.1. Architecture of FC layer

The input to the fully connected layer is the output from the *ROI* Pooling or Convolutional Layer, which is flattened and then fed into the fully connected layer.

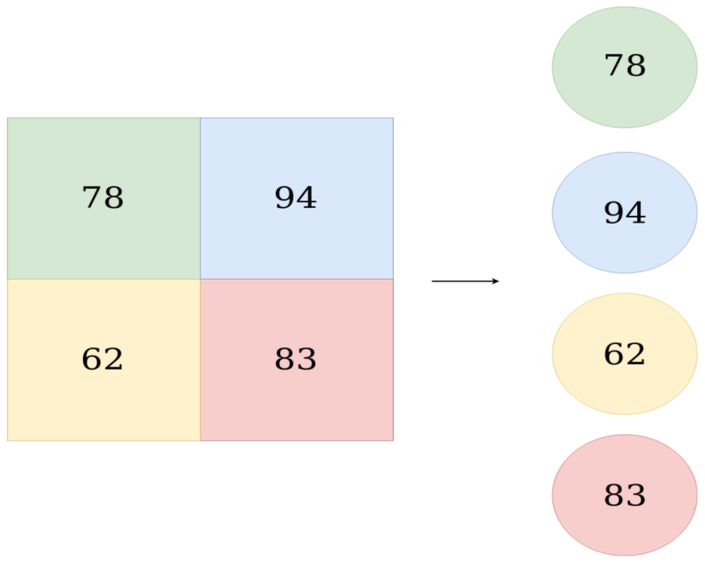


Fig.6.2. Flattening of a matrix

This Flattened vector is then connected to a few fully connected layers which are same as Artificial Neural Networks and perform the same mathematical operations. For each layer of the Artificial Neural Network, the following calculation takes place

ANN Calculation for each layer.

$$g=(Wx+b)$$

where,

x — is the input vector with dimension **[p_l, 1]**

W — Is the weight matrix with dimensions **[p_l, n_l]** where, **p_l** is the number of neurons in the previous layer and **n_l** is the number of neurons in the current layer.

b — Is the bias vector with dimension **[p_l, 1]**

g — Is the activation function, which is usually **ReLU**.

This calculation is repeated for each layer. After passing through the fully connected layers, the final layer uses the **softmax activation function** (instead of ReLU) which is used to get probabilities of the input being a license plate or not. And so finally, we have the probabilities of the object in the image belonging to the different classes.

6.3 Learning Fully Connected Networks with Back propagation

The Back propagation algorithm is a supervised learning method for multilayer feed-forward networks from the field of Artificial Neural Networks. Feed-forward neural networks are inspired by the information processing of one or more neural cells, called a neuron. A neuron accepts input signals via its dendrites, which pass the electrical signal down to the cell body. The axon carries the signal out to synapses, which are the connections of a cell’s axon to other cell’s dendrites.

The principle of the back propagation approach is to model a given function by modifying internal weightings of input signals to produce an expected output signal. The system is trained using a supervised learning method, where the error between the system’s output and a known expected output is presented to the system and used to modify its internal state. Technically, the back propagation algorithm is a method for training the weights in a multilayer feed-forward neural network. As such, it requires a network structure to be defined of one or more layers where one layer is fully connected to the next layer. A standard network structure is one input layer, one hidden layer, and one output layer.

Back propagation can be used for both classification and regression problems, but we will focus on classification in this tutorial. In classification problems, best results are achieved when the network has one neuron in the output layer for each class value. For example, a 2-class or binary classification problem with the class values of A and B. These expected outputs would have to be transformed into binary vectors with one column for each class value. Such as[1, 0] and [0, 1] for A and B respectively. This is called a one hot encoding.

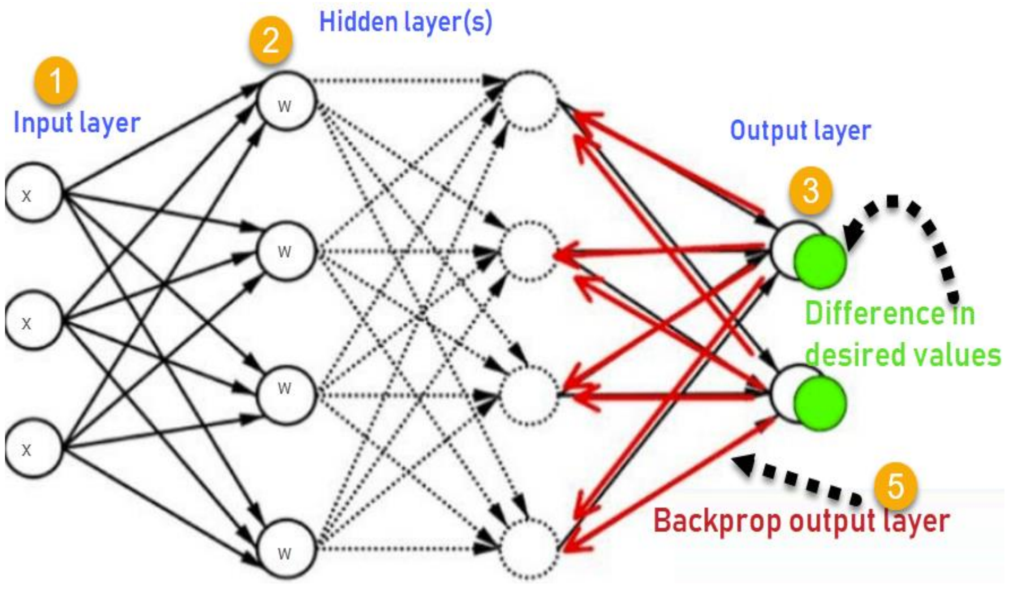


Fig.6.3. Back Propagation functioning

6.4 Fully Connected Networks Memorize

One of the striking aspects about fully connected networks is that they tend to memorize training data entirely given enough time. As a result, training a fully connected network to “convergence” isn’t really a meaningful metric. The network will keep training and learning as long as the user is willing to wait.

For large enough networks, it is quite common for training loss to trend all the way to zero. This empirical observation is one the most practical demonstrations of the universal approximation capabilities of fully connected networks. Note however, that training loss trending to zero does not mean that the network has learned a more powerful model. It’s rather likely that the model has started to memorize peculiarities of the training set that aren’t applicable to any other data points.

It’s worth digging into what we mean by peculiarities here. One of the interesting properties of high-dimensional statistics is that given a large enough dataset, there will be plenty of spurious correlations and patterns available for the picking. In practice, fully connected networks are entirely capable of finding and utilizing these spurious correlations. Controlling networks and preventing them from misbehaving in this fashion is critical for modelling success.

6.4.1 Regularization

Regularization is the general statistical term for a mathematical operation that limits memorization while promoting generalise learning. There are many different types of regularization available, which we will cover in the next few sections.

6.4.2 Dropout

Dropout is a form of regularization that randomly drops some proportion of the nodes that feed into a fully connected layer. Here, dropping a node means that its contribution to the corresponding activation function is set to 0. Since there is no activation contribution, the gradients for dropped nodes drop to zero as well.

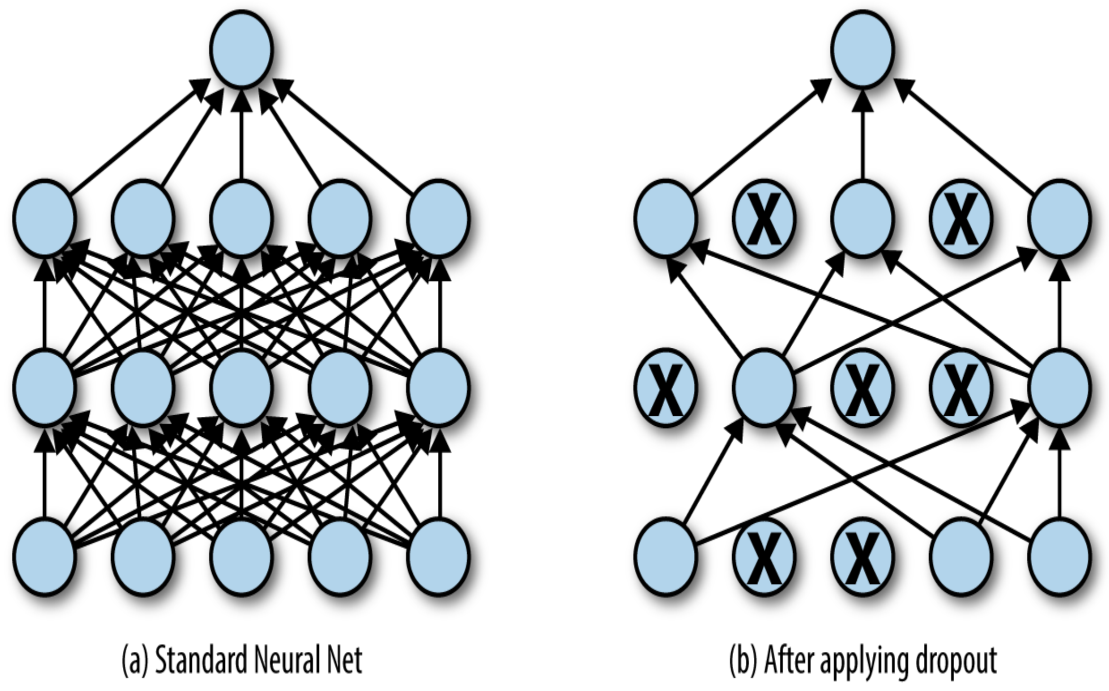


Fig.6.4. Output of FC layer

The nodes to be dropped are chosen at random during each step of gradient descent. The underlying design principle is that the network will be forced to avoid "co-adaptation." Briefly, we will explain what co-adaptation is and how it arises in non-regularized deep architectures. Suppose that one neuron in a deep network has learned a useful representation. Then other neurons deeper in the network will rapidly learn to depend on that particular neuron for information. This process will render the network brittle since the network will depend excessively on the features learned by that neuron, which might represent a quirk of the dataset, instead of learning a general rule.

Dropout prevents this type of co-adaptation because it will no longer be possible to depend on the presence of single powerful neurons (since that neuron might drop randomly during training). As a result, other neurons will be forced to “pick up the slack” and learn useful representations as well. The theoretical argument follows that this process should result in stronger learned models.

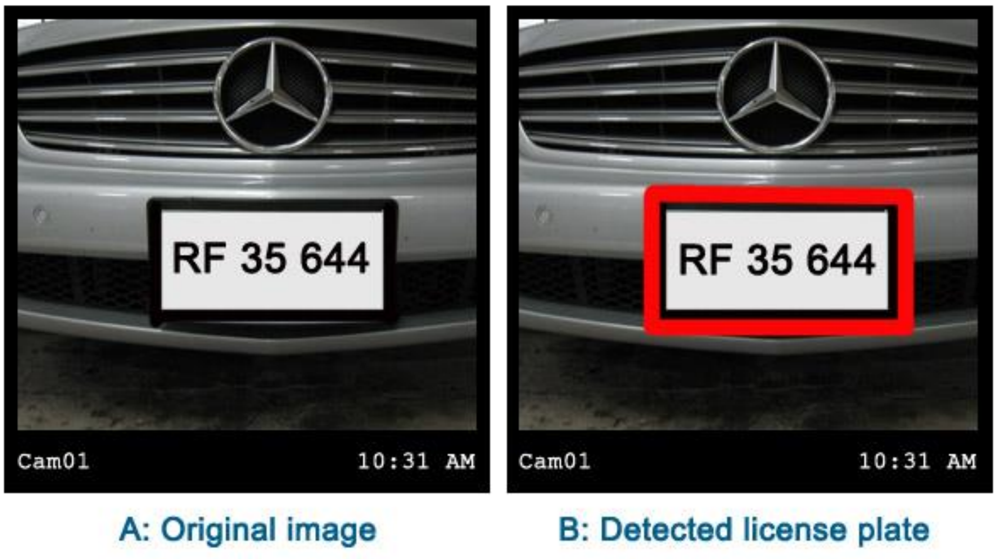


Fig.6.5. License Plate detection

6.5 Bidirectional recurrent neural networks

Bidirectional recurrent neural networks (**BRNN**) connect two hidden layers of opposite directions to the same output. With this form of generative deep learning, the output layer can get information from past (backwards) and future (forward) states simultaneously.

Standard recurrent neural network(RNNs) also have restrictions as the future input information cannot be reached from the current state. On the contrary, BRNNs do not require their input data to be fixed. Moreover, their future input information is reachable from the current state.

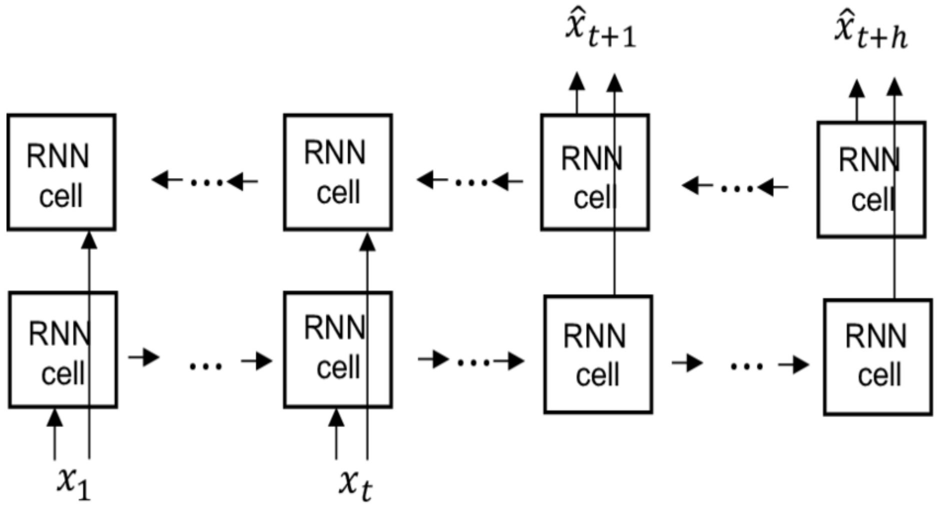


Fig.6.6. Structure of BRNN

The principle of BRNN is to split the neurons of a regular RNN into two directions, one for positive time direction (forward states), and another for negative time direction (backward states). Those two states' output are not connected to inputs of the opposite direction states. The general structure of RNN and BRNN can be depicted in the right diagram. By using two time directions, input information from the past and future of the current time frame can be used unlike standard RNN which requires the delays for including future information.

6.5.1 Training

BRNNs can be trained using similar algorithms to RNNs, because the two directional neurons do not have any interactions. However, when back-propagation through time is applied, additional processes are needed because updating input and output layers cannot be done at once. General procedures for training are as follows: For forward pass, forward states and backward states are passed first, then output neurons are passed. For backward pass, output neurons are passed first, then forward states and backward states are passed next. After forward and backward passes are done weights are updated.

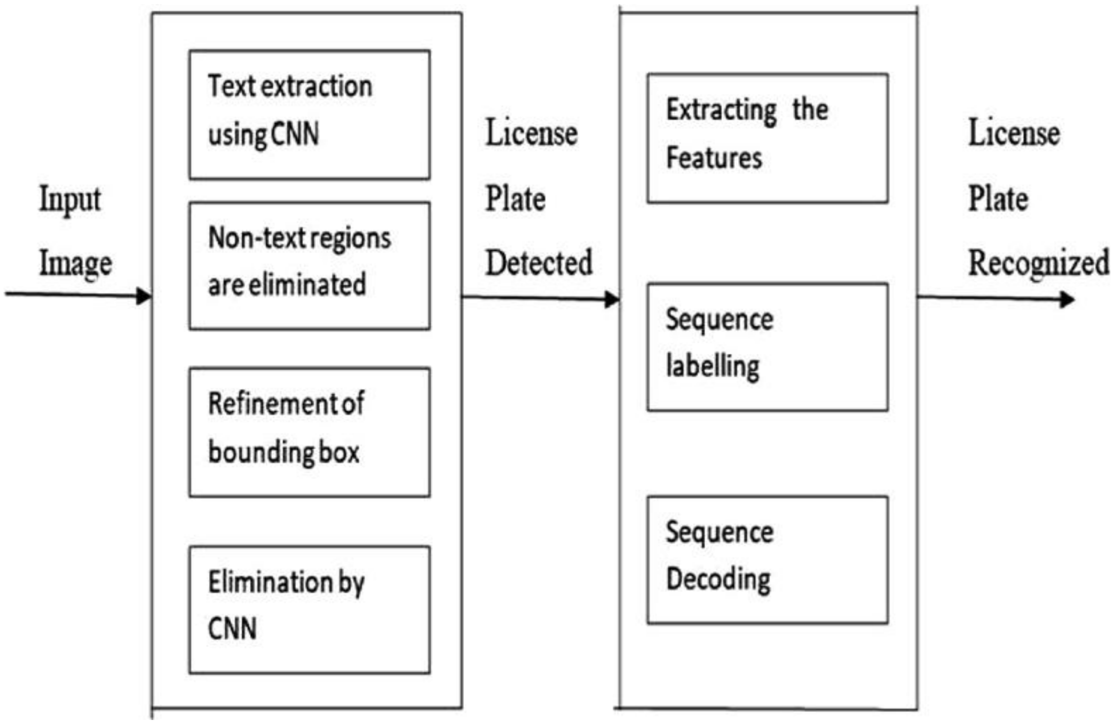


Fig.6.7. Training of datasets using BRNN

6.6 Plate Recognition Work

Plate recognition network aims to recognize each character in RoIs based on the extracted region features. To avoid the challenging task of character segmentation, we regard the plate recognition as a sequence labelling problem. Bidirectional RNNs (BRNNs) with CTC loss are employed to label the sequential features. Q ∈ RC×X×Y, where C is the channel size. First of all, we add two additional convolutional layers with ReLUs. Both of them use 512 filters. The kernel sizes are 3 and 2 respectively, with a padding of 1 used in the first convolutional layer. A rectangular pooling window with kW = 1 and kH = 2 is adopted between them, which would be beneficial for recognizing characters with narrow shapes, such as '1' and 'I', referring to [27]. These operations will reform the region features Q to a sequence with the size as D×L, where D = 512 and L = 19. We denote the resulting features as V = (v1, v2,..., vL), where vi ∈ RD.

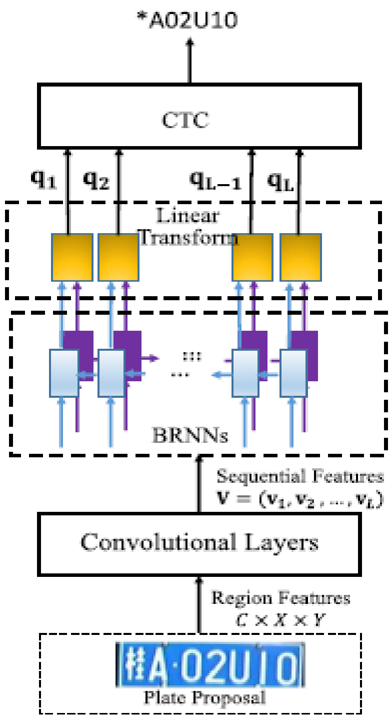


Fig.6.8. Plate Recognition network

Then BRNNs are applied on top of the sequential features. As presented in Figure 4, two separated RNN layers with 512 units are used. One processes the feature sequence forward, with the hidden state updated.

The other one processes it backward, with the hidden state updated via h(b) t = g(vt, h(b) t+1). The two hidden states are concatenated together and fed to a linear transformation with 37 outputs. Softmax layer is followed to transform the 37 outputs into probabilities, which correspond to the distributions over 26 capital letters, 10 digits, and a special non character class. We record the probabilities at each ti.

Hence, after BRNNs encoding, the feature sequence V is transformed into a sequence of probability estimation q = (q1, q2,..., qL) with the same length as V. BRNNs capture abundant contextual information from both directions, which will make the character recognition more accurate. To overcome the shortcoming of gradient vanishing or exploding during traditional RNN training, Long-Short Term Memory (LSTM) is employed here. It defines a new cell structure called memory cell, and three multiplicative gates (i.e., input gate, forget gate and output gate), which can selectively store information for a long time.

Then CTC layer is adopted here for sequence decoding, which is to find an approximately optimal path π∗ with maximum probability through the BRNNs' output sequence q, i.e., π∗≈ B arg max π P(π|q) . (2) Here a path π is a label sequence based on the output activation of BRNNs, and P(π|q) = L t=1 P(πt|q). The operator B is defined as the operation of removing the repeated labels and the non-character label from the path. For example, B(a − a − b−) = B(−aa − −a − bb) = (aab). Details of CTC can refer to [25]. The optimal label sequence π∗ is exactly the recognized plate label.

There is a need to develop Automatic Number Plate Recognition (ANPR) system as a one of the solutions to this problem. There are numerous ANPR systems available today. These systems are based on different methodologies but still it is really challenging task as some of the factors like high speed of vehicle, non-uniform vehicle number plate, language of vehicle number and different lighting conditions can affect a lot in the overall recognition rate.

So, we developed an jointly distributed network using deep convolutional network where increasing efficiency and reducing the parameters as segmentation is completely ignored due to increasing parameters. Our system produces the results from highly resolution images.

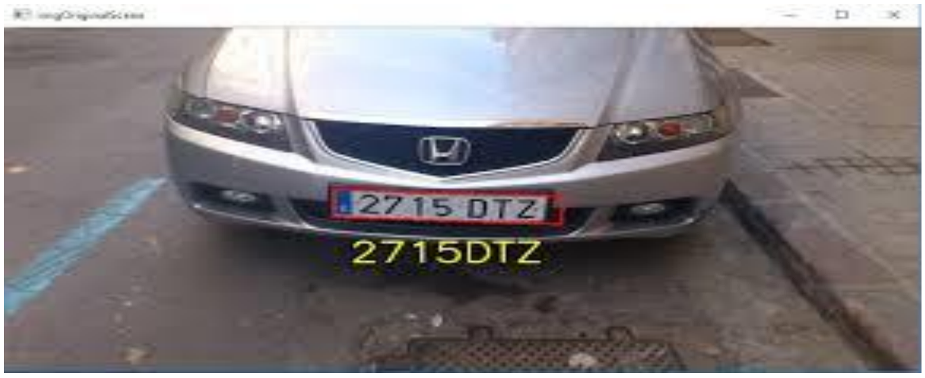


Fig.6.9. Recognition of characters on license plate

CHAPTER 7

IMPLEMENTATION

This is the block diagram of the process that is taking place in the project. At first an image is considered in the form of multiple frames, then segmentation of frames is done by image pre processing. License plate segmentation is done in two stages for characters and numbers, having different classifiers for each which are trained by the samples and getting recognition results is done.

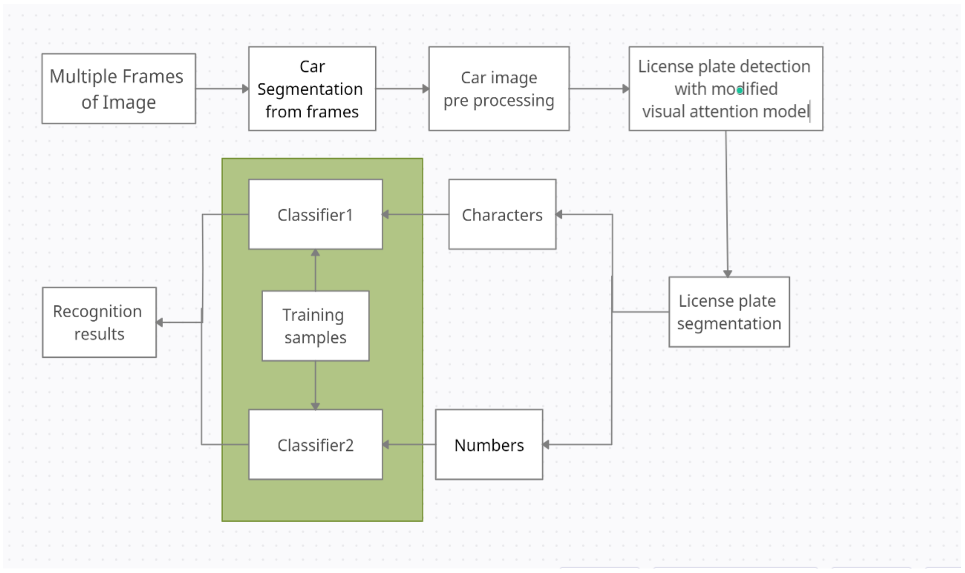
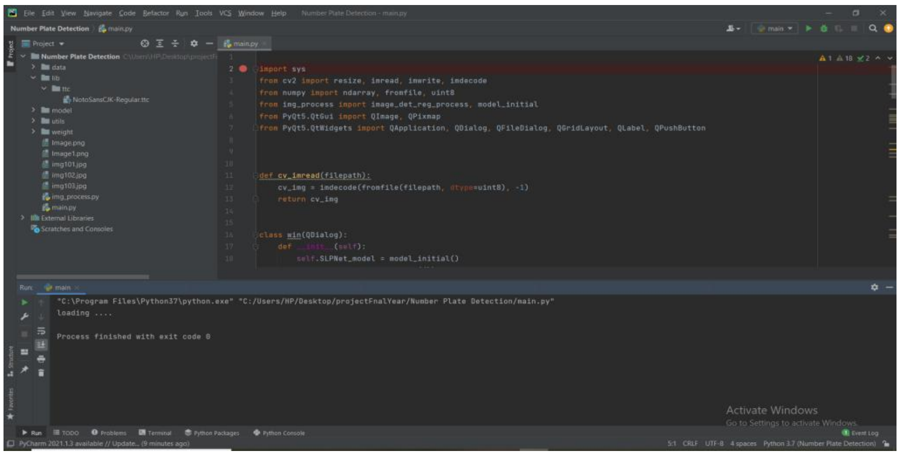


Fig.7.1. Block Diagram of the process

We implement the code in Python Programming Language. We have done our project executionin PyCharm IDE which provides wide range of essential tools for python developers

Here, this is the environment that is set up and when imported all the files, packages , the PyCharm looks exactly like this which means we are good to go for the execution.



```python
import sys
from cv2 import resize, imread, imwrite, imdecode
from numpy import ndarray, fromfile, uint8
from img_process import image_det_reg_process, model_initial
from PyQt5.QtGui import QImage, QPixmap
from PyQt5.QtWidgets import QApplication, QDialog, QFileDialog, QGridLayout, QLabel, QPushButton



def cv_imread(filepath):
    cv_img = imdecode(fromfile(filepath, dtype=uint8), -1)
    return cv_img


class win(QDialog):
    def __init__(self):
        self.SLPNet_model = model_initial()
```

```
"C:\Program Files\Python37\python.exe" "C:/Users/HP/Desktop/projectFnalYear/Number Plate Detection/main.py"
loading ....

Process finished with exit code 0
```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
Number Plate Detection - main.py
Number Plate Detection
main.py
Project
Number Plate Detection
data
lib
ttc
NotoSansCJK-Regular.ttc
model
utils
weight
Image.png
Image1.png
img101.jpg
img102.jpg
img103.jpg
img_process.py
main.py
External Libraries
Scratches and Consoles
Run: main
Activate Windows
Go to Settings to activate Windows.
Run TODO Problems Terminal Python Packages Python Console Event Log
PyCharm 2021.1.3 available // Update... (9 minutes ago)
5:1 CRLF UTF-8 4 spaces Python 3.7 (Number Plate Detection)

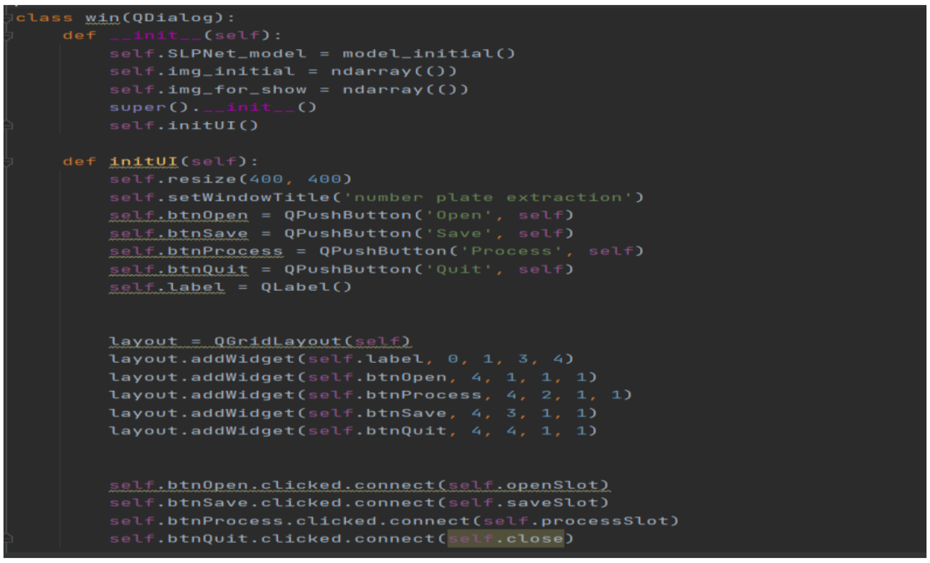
Fig.7.2. Environment setup

Then we run our code by clicking on the green arrow button at the right top of the screen and that gives our output in the run time environment at the bottom. After completion of code compiling and executing we could able to see a dialogue box.

Here we represent the block of code for the process or action that is taking place at the time of execution and its respective output simulataneously.

CODE:

Here is the code for the dialogue box that appears just after the successful running of the code.



```python
class win(QDialog):
    def __init__(self):
        self.SLPNet_model = model_initial()
        self.img_initial = ndarray(())
        self.img_for_show = ndarray(())
        super().__init__()
        self.initUI()

    def initUI(self):
        self.resize(400, 400)
        self.setWindowTitle('number plate extraction')
        self.btnOpen = QPushButton('Open', self)
        self.btnSave = QPushButton('Save', self)
        self.btnProcess = QPushButton('Process', self)
        self.btnQuit = QPushButton('Quit', self)
        self.label = QLabel()


        layout = QGridLayout(self)
        layout.addWidget(self.label, 0, 1, 3, 4)
        layout.addWidget(self.btnOpen, 4, 1, 1, 1)
        layout.addWidget(self.btnProcess, 4, 2, 1, 1)
        layout.addWidget(self.btnSave, 4, 3, 1, 1)
        layout.addWidget(self.btnQuit, 4, 4, 1, 1)


        self.btnOpen.clicked.connect(self.openSlot)
        self.btnSave.clicked.connect(self.saveSlot)
        self.btnProcess.clicked.connect(self.processSlot)
        self.btnQuit.clicked.connect(self.close)
```

OUTPUT:

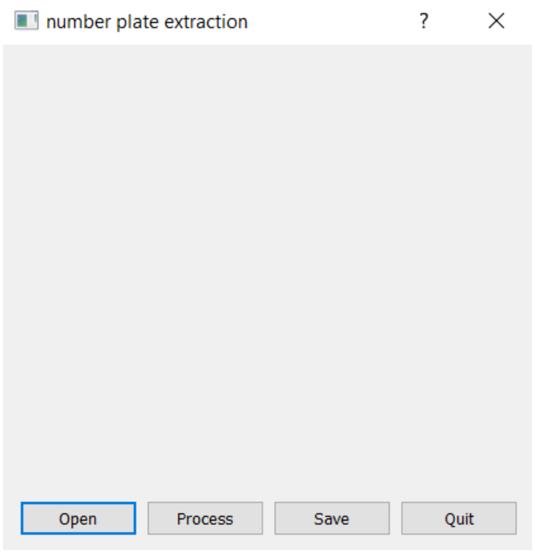
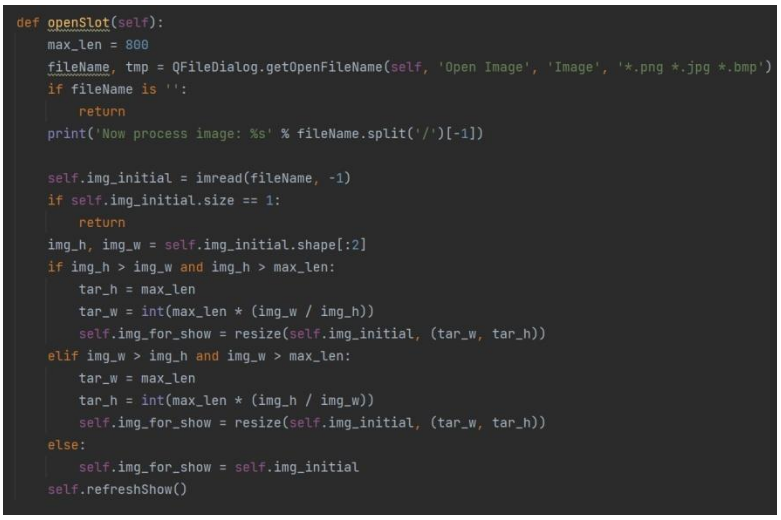


Fig.7.3. Dialogue Box

This is the dialogue box that appears immediately after executing the code. It contains open, process, save and quit buttons. Each has separate function which involves different blocks of code , that are as follows.

CODE:

This is the code for the action that is to be done when open button is clicked.



```
def openSlot(self):
    max_len = 800
    fileName, tmp = QFileDialog.getOpenFileName(self, 'Open Image', 'Image', '*.png *.jpg *.bmp')
    if fileName is '':
        return
    print('Now process image: %s' % fileName.split('/')[-1])

    self.img_initial = imread(fileName, -1)
    if self.img_initial.size == 1:
        return
    img_h, img_w = self.img_initial.shape[:2]
    if img_h > img_w and img_h > max_len:
        tar_h = max_len
        tar_w = int(max_len * (img_w / img_h))
        self.img_for_show = resize(self.img_initial, (tar_w, tar_h))
    elif img_w > img_h and img_w > max_len:
        tar_w = max_len
        tar_h = int(max_len * (img_h / img_w))
        self.img_for_show = resize(self.img_initial, (tar_w, tar_h))
    else:
        self.img_for_show = self.img_initial
    self.refreshShow()
```

OUTPUT:

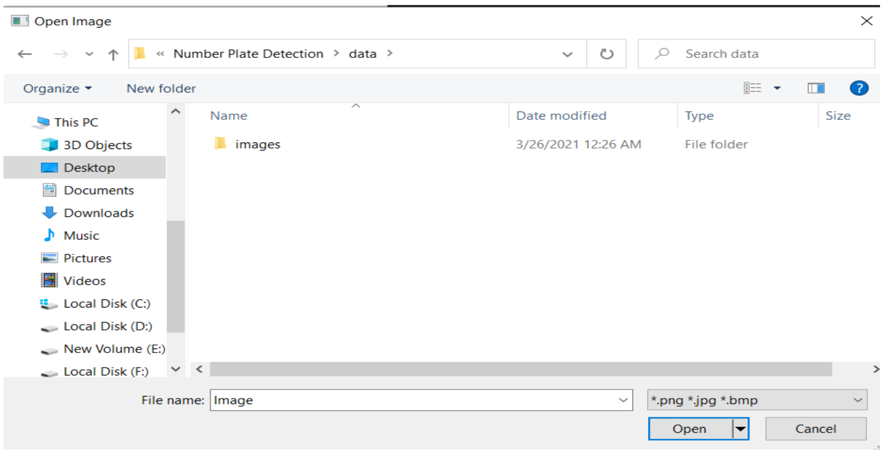


Fig.7.4. Action on clicking open button

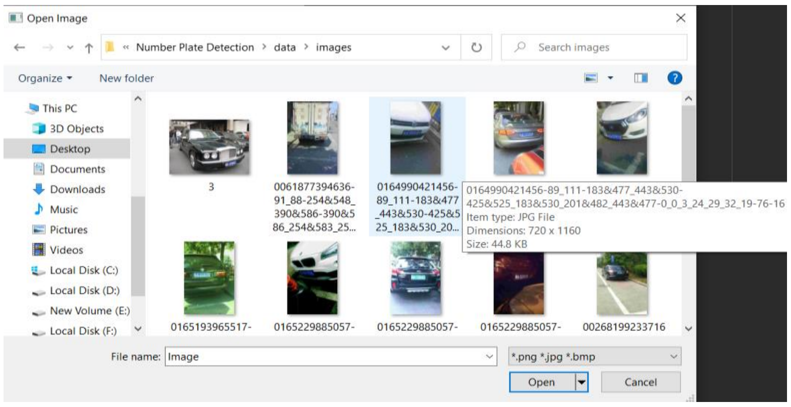
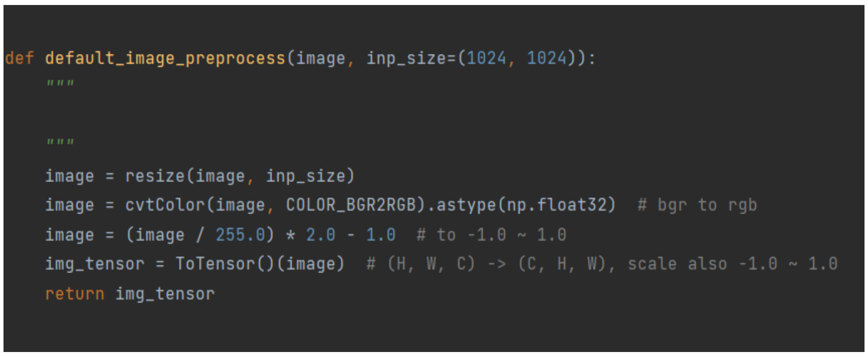


Fig.7.5. Testing dataset

On clicking "open" button , a tab containing folders will be enabled to select the required file. Here we select one such image of the car license plate from the images folder, by double clicking on it.

CODE:



```
def default_image_preprocess(image, inp_size=(1024, 1024)):
    """

    """
    image = resize(image, inp_size)
    image = cvtColor(image, COLOR_BGR2RGB).astype(np.float32)  # bgr to rgb
    image = (image / 255.0) * 2.0 - 1.0  # to -1.0 ~ 1.0
    img_tensor = ToTensor()(image)  # (H, W, C) -> (C, H, W), scale also -1.0 ~ 1.0
    return img_tensor
```

This is the code for preprocessing the image so that it get fit into the dialogue box without actually missing out the important parts of the image.

OUTPUT:

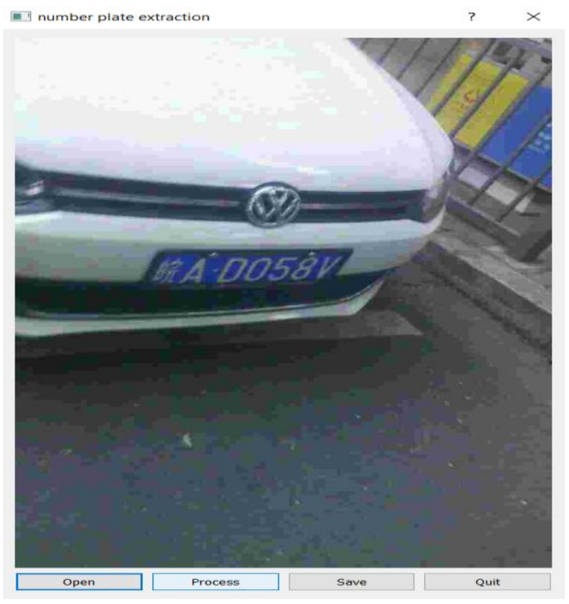
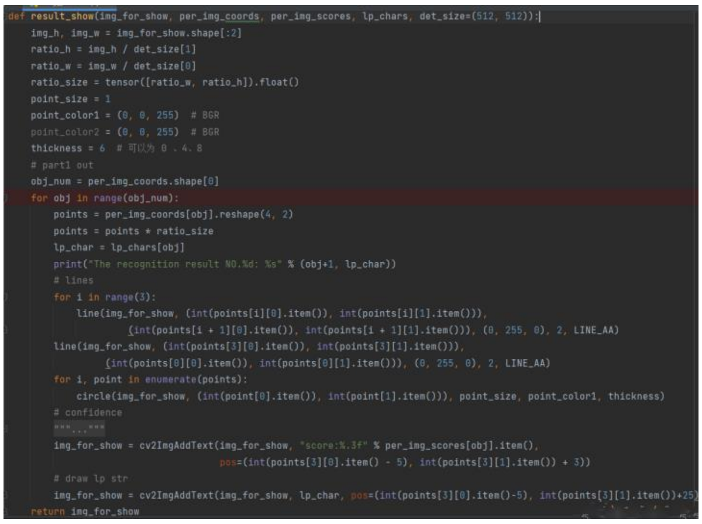


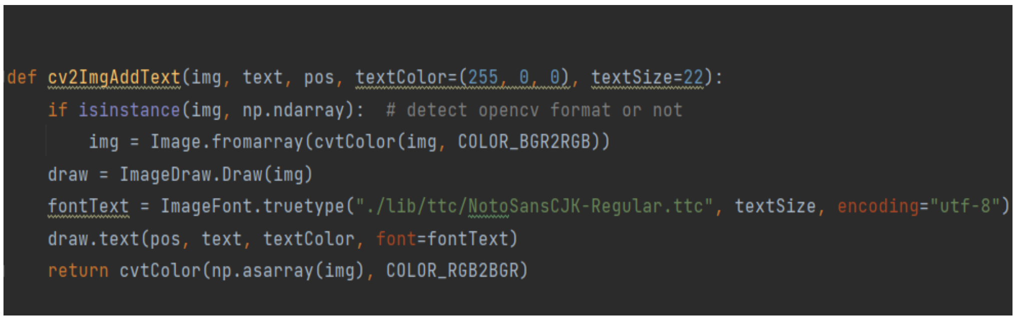
Fig.7.6. Preprocessed car image

CODE:



```python
def result_show(img_for_show, per_img_coords, per_img_scores, lp_chars, det_size=(512, 512)):
    img_h, img_w = img_for_show.shape[:2]
    ratio_h = img_h / det_size[1]
    ratio_w = img_w / det_size[0]
    ratio_size = tensor([ratio_w, ratio_h]).float()
    point_size = 1
    point_color1 = (0, 0, 255)  # BGR
    point_color2 = (0, 0, 255)  # BGR
    thickness = 6  # 可以为 0 . 4. 8
    # part1 out
    obj_num = per_img_coords.shape[0]
    for obj in range(obj_num):
        points = per_img_coords[obj].reshape(4, 2)
        points = points * ratio_size
        lp_char = lp_chars[obj]
        print("The recognition result NO.%d: %s" % (obj+1, lp_char))
        # lines
        for i in range(3):
            line(img_for_show, (int(points[i][0].item()), int(points[i][1].item())),
                     (int(points[i + 1][0].item()), int(points[i + 1][1].item())), (0, 255, 0), 2, LINE_AA)
        line(img_for_show, (int(points[3][0].item()), int(points[3][1].item())),
                 (int(points[0][0].item()), int(points[0][1].item())), (0, 255, 0), 2, LINE_AA)
        for i, point in enumerate(points):
            circle(img_for_show, (int(point[0].item()), int(point[1].item())), point_size, point_color1, thickness)
        # confidence
        """..."""
        img_for_show = cv2ImgAddText(img_for_show, "score:%.3f" % per_img_scores[obj].item(),
                                     pos=(int(points[3][0].item() - 5), int(points[3][1].item()) + 3))
        # draw lp str
        img_for_show = cv2ImgAddText(img_for_show, lp_char, pos=(int(points[3][0].item()-5), int(points[3][1].item())+25
    return img_for_show
```

We use this code for showing the result of the process which is displaying the recognized characters on the number plate including the bounding box.This happens after clicking “process” button. Also for adding the text on the image we use following code.



```
def cv2ImgAddText(img, text, pos, textColor=(255, 0, 0), textSize=22):
    if isinstance(img, np.ndarray):  # detect opencv format or not
        img = Image.fromarray(cvtColor(img, COLOR_BGR2RGB))
    draw = ImageDraw.Draw(img)
    fontText = ImageFont.truetype("./lib/ttc/NotoSansCJK-Regular.ttc", textSize, encoding="utf-8")
    draw.text(pos, text, textColor, font=fontText)
    return cvtColor(np.asarray(img), COLOR_RGB2BGR)
```

OUTPUT:

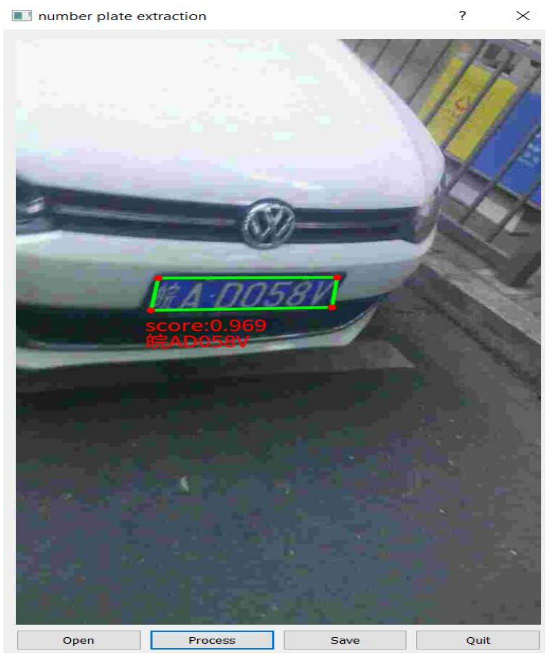
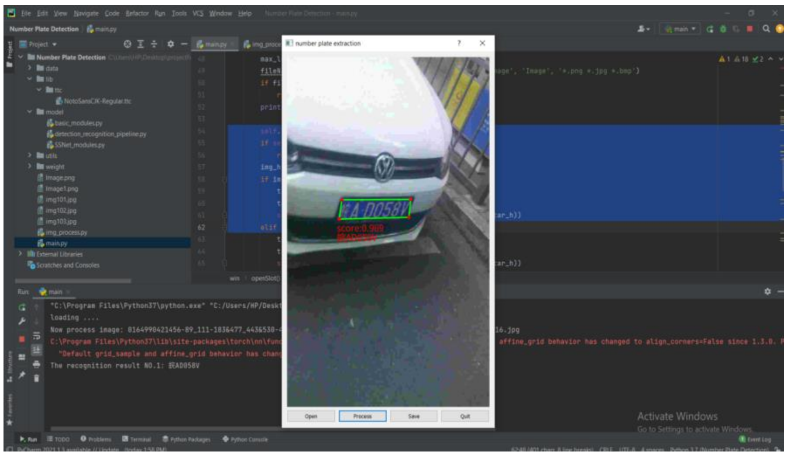


Fig.7.7. Displaying the result on clicking process button



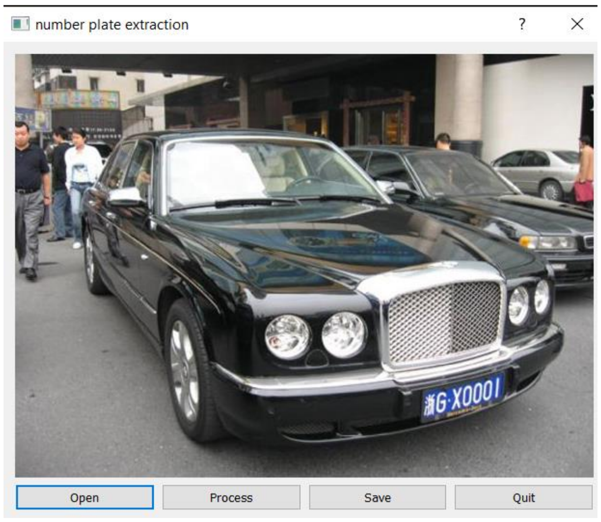
```
"C:\Program Files\Python37\python.exe" "C:/Users/HP/Deskt
loading ....
Now process image: 0164990421456-89_111-1836477_4436530-                16.jpg
C:\Program Files\Python37\lib\site-packages\torch\nn\func
  "Default grid_sample and affine_grid behavior has chang   affine_grid behavior has changed to align_corners=False since 1.3.0.
The recognition result NO.1: 皖AD058V
```

Fig.7.8. Recognition result

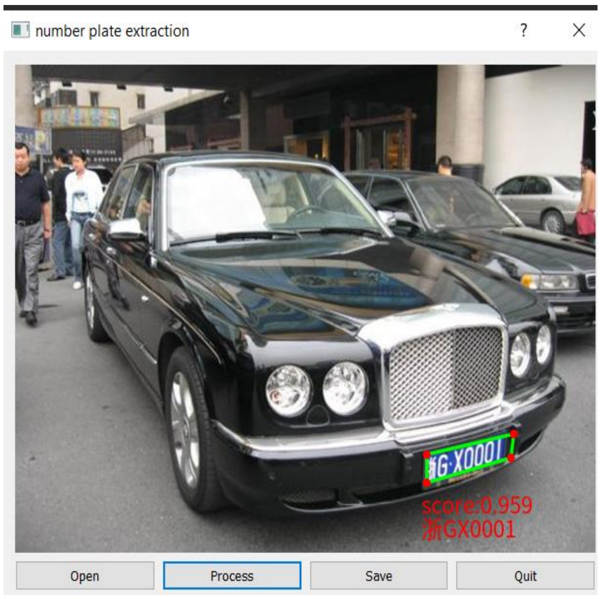
Also we can see the recognized characters of the number plate in the run time environment as well. On clicking the save button , we can save the picture in the required folder.

Here are few more images that are tested and are given successful results. These images are taken under different conditions like light and dark, different ang;les like top, low, dutch etc., and also multiple cars as well.

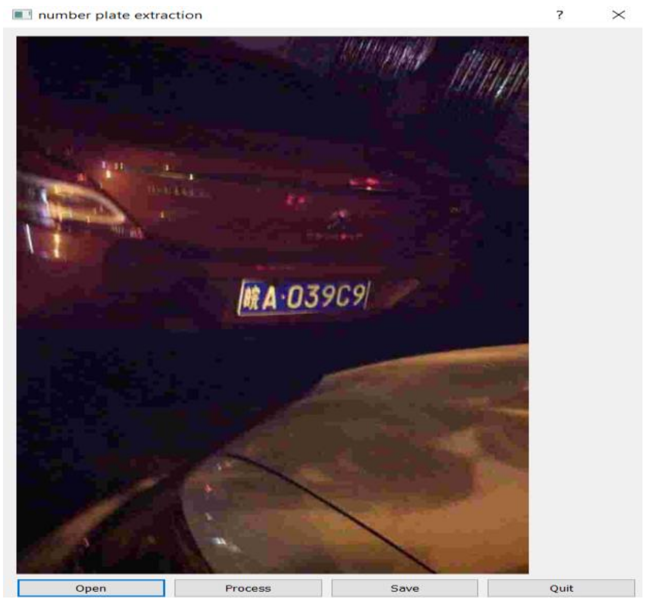
Test-1:



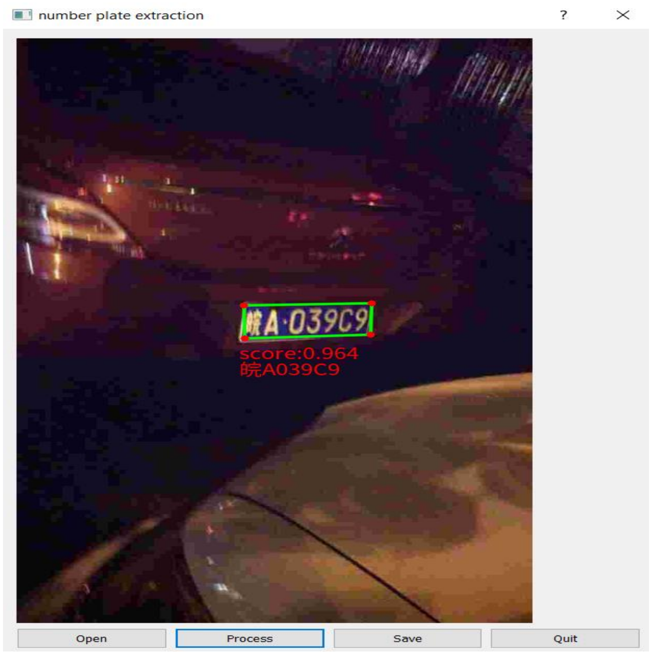
Result-1:



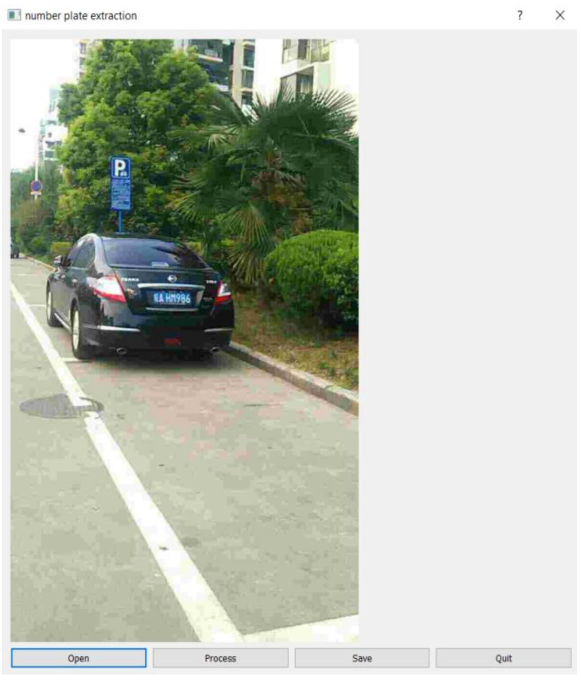
Test-2:



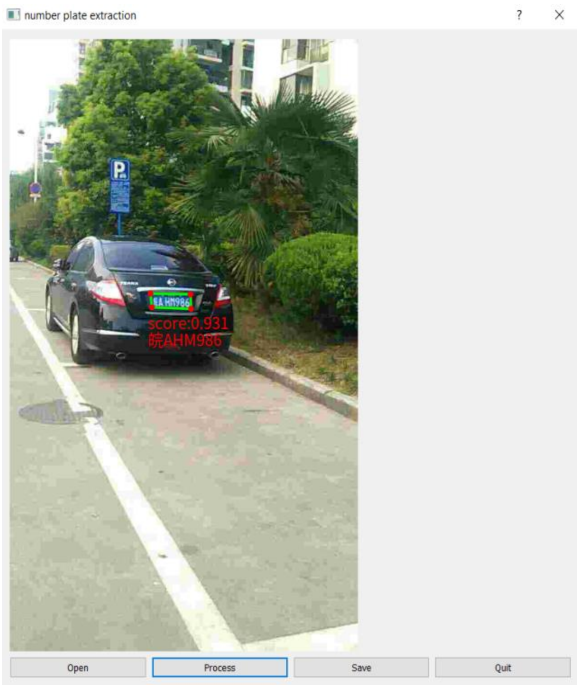
Result-2:



Test-3:

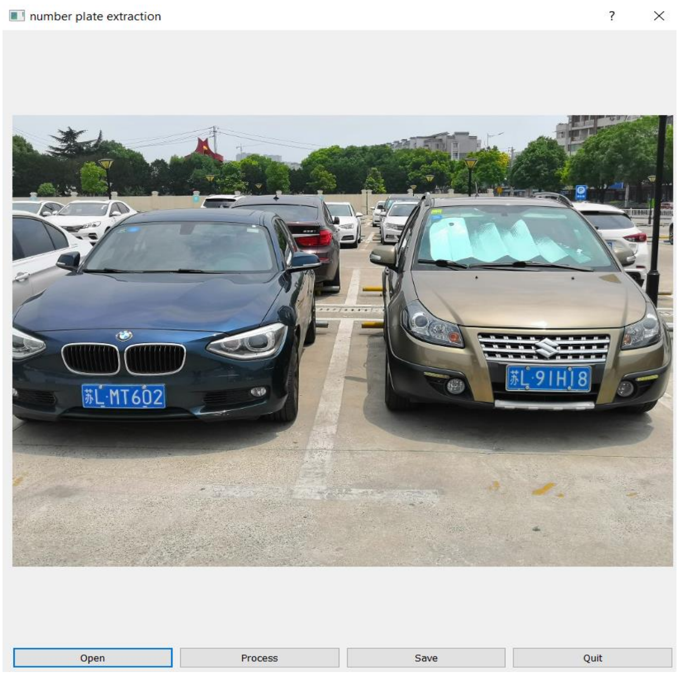


Result-3:

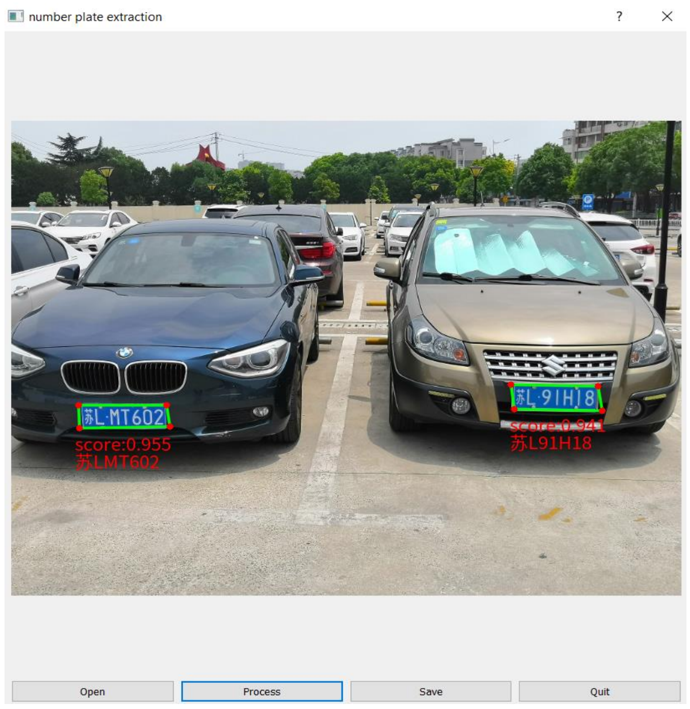


score:0.931
皖AHM986

Test-4:



Result-4:



CONCLUSION

A lot of research has been performed on detection and recognition of license plate. Different researchers provided different methods and techniques for this process. However, every technique has its own advantages and disadvantages. Furthermore each country has its own license plate numbering system, colors, language of characters, style (font) and sizes. Even within the same country the license plate differs from state to state and in terms of types of License plate.

In our proposed work, we have presented a jointly trained networkfor simultaneous car license plate detection and recognition. With this network, car license plates can be detected andrecognized all at once in a single forward pass, with bothhigh accuracy and efficiency. By sharing convolutional featureswith both detection and recognition network, the modelsize decreases largely. The whole network can be trainedapproximately end-to-end, without intermediate processinglike image cropping or character separation. Comprehensiveevaluation and comparison on three datasets with differentapproaches validate the advantage of our method. In the future,we will extend our network to multi-oriented car license plates.

Hence we can conclude that ANPR solutions tend to gain a huge popularity today. There is a immediate need of such kind of ANPR in India as there are problems of traffic, stealing cars. Government should take some interest to develop this system as this system is very economical and eco friendly. Starting from enforcement to providing smoother day-to-day facilities, it has served all. Future work need to be done to detect and recognize the license plates from video clips in real time.

REFERENCES

[1] L. Neumann and J. Matas, “A method for text localization and recognition in real world images,” in Proc. Asian Conf. Comput. Vis., 2011, pp. 770–783.

[2] L. Neumann and J. Matas, “Real-time scene text localization and recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2012, pp. 3538–3545.

[3] K. Wang, B. Babenko, and S. Belongie, “End-to-end scene text recognition,” in Proc. IEEE Int. Conf. Comput. Vis., Nov. 2011, pp. 1.

[4] T. Wang, D. Wu, A. Coates, and A. Y. Ng, “End-to-end text recognition with convolutional neural networks,” in Proc. IEEE Int. Conf. Pattern Recognit., Nov. 2012, pp. 3304–3308.

[5] A. Bissacco, M. Cummins, Y. Netzer, and H. Neven, “PhotoOCR: Reading text in uncontrolled conditions,” in Proc. IEEE Int. Conf. Comput. Vis., Dec. 2013, pp. 785–792.

[6] M. Jaderberg, A. Vedaldi, and A. Zisserman, “Deep features for text spotting,” in Proc. Eur. Conf. Comput. Vis., 2014, pp. 512–528.

[7] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, “Reading text in the wild with convolutional neural networks,” Int. J. Comput. Vis., vol. 116, no. 1, pp. 1–20, 2016.

[8] M. Liao, B. Shi, X. Bai, X. Wang, and W. Liu, “TextBoxes: A fast text detector with a single deep neural network,” in Proc. Nat. Conf. Artif. Intell., 2017, pp. 4161–4167.

[9] S. Du, M. Ibrahim, M. Shehata, and W. Badawy, “Automatic license plate recognition (ALPR): A state-of-the-art review,” IEEE Trans. Circuits Syst. Video Technol., vol. 23, no. 2, pp. 311–325, Feb. 2013.

[10] W. Zhou, H. Li, Y. Lu, and Q. Tian, “Principal visual word discovery for automatic license plate detection,” IEEE Trans. Image Process., vol. 21, no. 9,

pp. 4269–4279, Sep. 2012.

[11] C. N. E. Anagnostopoulos, I. E. Anagnostopoulos, V. Loumos, and E. Kayafas, “A license plate-recognition algorithm for intelligent transportation system applications,” IEEE Trans. Intell. Transp. Syst., vol. 7, no. 3, pp. 377–392, Sep. 2006.

[12] G.-S. Hsu, J.-C. Chen, and Y.-Z. Chung, “Application-oriented license plate recognition,” IEEE Trans. Veh. Technol., vol. 62, no. 2, pp. 552–561, Feb. 2013.

[13] Y. Yuan, W. Zou, Y. Zhao, X. Wang, X. Hu, and N. Komodakis, “A robust and efficient approach to license plate detection,” IEEE Trans. Image Process., vol. 26, no. 3, pp. 1102–1114, Mar. 20

[14] A. H. Ashtari, M. J. Nordin, and M. Fathy, “An iranian license plate recognition system based on color features,” IEEE Trans. Intell. Transp. Syst., vol. 15, no. 4, pp. 1690–1705, Aug. 2014.

[15] S.-L. Chang, L.-S. Chen, Y.-C. Chung, and S.-W. Chen, “Automatic license plate recognition,” IEEE Trans. Intell. Transp. Syst., vol. 5, no. 1, pp. 42–53, Mar. 2004.