FACULTY OF IMAGE PROCESSING

INFORMATION TECHNOLOGY DEPARTMENT

Dr. Mohammed Javed

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY ALLAHABAD

INVESTIGATIVE STUDY ON DETECTION AND RECOGNITION OF NUMBER PLATES:

CNN and TESSERACT

Made by:

DEVESH KATIYAR , INDIAN INSTITUTE OF INFORMATION TECHNOLOGY PUNE

SUMAN KUMARI , INDIAN INSTITUTE OF INFORMATION TECHNOLOGY PUNE

Contents

INVESTIGATIVE STUDY ON DETECTION AND RECOGNITION OF NUMBER PLATES

A PREPRINT

Devesh Katiyar*
Department of CSE
IIIT Pune
Pune(Maharashtra),India 124112
deveshkatiyar9620@gmail.com

Suman Kumari
Department of ECE
IIIT Pune
Pune(Maharashtra),India 124112
sumanku1307@gmail.com

July 5, 2019

INTRODUCTION

This report presents an Investigative study on detection and recognition of number plates using OCR and CNN techniques.License Plate Recognition is a technology that can recognise letters from a digitised license plate image and convert them into ASCII characters to be processed as an editable text.Although there have been many studies on plate detection , character segmentation and character recognition many challenges have still remained.In this paper we will talk about extracting information from a image file(jpeg/png) and create a separate text file consisting of information extracted from image file.Our key idea is to perform detection and segmentation using OPEN CV and recognition using CNN and Tesseract.The presented methods has been evaluated on 100 images(jpeg/png) .The experimental results show that our network achieves an accuracy of 95 percent with CNN model and quite high with the tesseract model.

***K*eywords** Number Plate Recognition · Morphological Operation · Thresholding · Number Plate Detection · Character Segmentation

1 Literature Review

Number Plate Recognition is a form of automatic vehicle identification.A number plate is the unique identification of vehicle.It is an image processing technology used to identify vehicles by their own number plates.Real time number plate recognition plays an important role in maintaining traffic rules.It has wide applications areas such as toll plaza,parking area,highly security areas,border's

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

areas etc.Number plate recognition is designed to detect the number plate and then recognise the vehicle's number plate.The three major parts in this process are vehicle number plate detection and extraction,character segmentation and character recognition.For vehicle number plate detection , extraction and segmentation for both the methods we have used the same algorithm but for character recognition both models are using different algorithms.In the first model we have used CNN model for character recognition in the second model we have used Tesseract for character recognition.

1.1 Learning Paradigm

1.1.1 Supervised Learning

Supervised learning as the name indicates the presence of a supervisor as a teacher. Basically supervised learning is a learning in which we teach or train the machine using data which is well labeled that means some data is already tagged with the correct answer. After that, the machine is provided with a new set of examples(data) so that supervised learning algorithm analyses the training data(set of training examples) and produces a correct outcome from labeled data.

Supervised learning uses a set of example pairs :

$$(x, y), x \in X, y \in Y \tag{1}$$

and the aim is to find a function:

$$f : X \to Y \tag{2}$$

in the allowed class of functions that matches the examples. In other words, we wish to infer the mapping implied by the data; the cost function is related to the mismatch between our mapping and the data and it implicitly contains prior knowledge about the problem domain.

A commonly used cost is the mean-squared error, which tries to minimize the average squared error between the network's output, f(x), and the target value y over all the example pairs. Minimizing this cost using gradient descent for the class of neural networks called multilayer perceptrons (MLP), produces the backpropagation algorithm for training neural networks.

1.1.2 Unsupervised Learning

Unsupervised learning is the training of machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data.Unlike supervised learning, no teacher is provided that means no training will be given to the machine. Therefore machine is restricted to find the hidden structure in unlabeled data by our-self.

In unsupervised learning, some data x is given and the cost function to be minimized, that can be any function of the data x and the network's output, f.The cost function is dependent on the task (the model domain) and any a priori assumptions (the implicit properties of the model, its parameters and the observed variables).

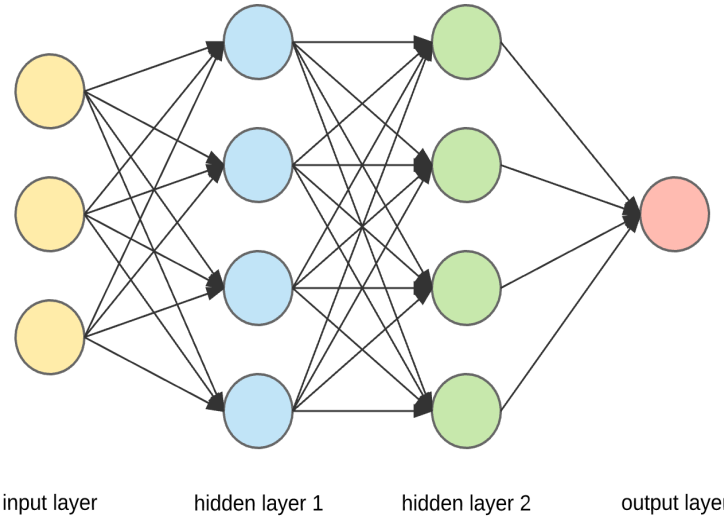
As a trivial example, consider the model f(x)=a where a is a constant and the cost:

$$C = E[(x - f(x))^2] \tag{3}$$

Minimizing this cost produces a value of a that is equal to the mean of the data. The cost function can be much more complicated. Its form depends on the application: for example, in compression it could be related to the mutual information between x and f(x), whereas in statistical modeling, it could be related to the posterior probability of the model given the data (note that in both of those examples those quantities would be maximized rather than minimized).

Tasks that fall within the paradigm of unsupervised learning are in general estimation problems; the applications include clustering, the estimation of statistical distributions, compression and filtering.

1.2 Neural Networks



A neural network is a network or circuit of neurons,or in a modern sense,an artificial neural network,composed of artificial neurons or nodes.Thus a neural network is either a biological neural network ,made up of real biological neurons,or an artificial neural network,for solving artificial intelligence(AI)problems.The connections of the biological neuron are modeled as weights.A positive weight reflects an excitatory connection,while negative values mean inhibitory connections.All inputs are modified by a weight and summed.This activity is referred as a linear combination.Finally,an activation function controls the amplitude of the output.For example,an acceptable range of output is usually between 0 and 1 ,or it could be -1 and 1.

Unlike von Neumann model computations ,artificial neural networks do not separate memory and processing and operate via the flow of signals through the net connections,somewhat akin to biological networks.

These artificial networks may be used for predictive modeling,adaptive control and applications where they can be trained via a dataset.Self-learning resulting from experience can occur within networks,which can derive from a complex and seemingly unrelated set of information.

A neural network (NN), in the case of artificial neurons called artificial neural network (ANN) or simulated neural network (SNN), is an interconnected group of natural or artificial neurons that uses a mathematical or computational model for information processing based on a connectionistic approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network.

In more practical terms neural networks are non-linear statistical data modeling or decision making tools. They can be used to model complex relationships between inputs and outputs or to find patterns in data.

An artificial neural network involves a network of simple processing elements (artificial neurons) which can exhibit complex global behavior, determined by the connections between the processing elements and element parameters. Artificial neurons were first proposed in 1943 by Warren McCulloch, a neurophysiologist, and Walter Pitts, a logician, who first collaborated at the University of Chicago.

One classical type of artificial neural network is the recurrent Hopfield network.

The concept of a neural network appears to have first been proposed by Alan Turing in his 1948 paper Intelligent Machinery in which called them "B-type unorganised machines".

The utility of artificial neural network models lies in the fact that they can be used to infer a function from observations and also to use it. Unsupervised neural networks can also be used to learn representations of the input that capture the salient characteristics of the input distribution, e.g., see the Boltzmann machine (1983), and more recently, deep learning algorithms, which can implicitly learn the distribution function of the observed data. Learning in neural networks is particularly useful in applications where the complexity of the data or task makes the design of such functions by hand impractical.

1.2.1 Artificial Neural Networks

Artificial neural networks (ANN) or connectionist systems are computing systems that are inspired by, but not necessarily identical to, the biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge about cats, for example, that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the learning material that they process.

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it.

In common ANN implementations, the signal at a connection between artificial neurons is a real number, and the output of each artificial neuron is computed by some non-linear function of the sum of its inputs. The connections between artificial neurons are called 'edges'. Artificial neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Artificial neurons may have a threshold such that the signal is only sent if the aggregate signal crosses that threshold. Typically, artificial neurons are aggregated into layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

The original goal of the ANN approach was to solve problems in the same way that a human brain would. However, over time, attention moved to performing specific tasks, leading to deviations from biology. Artificial neural networks have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games and medical diagnosis.

1.2.2 Components of Artificial Neural Network

Neurons

1.A neuron with label j receiving an input

$$p_j(t) \tag{4}$$

from predecessor neurons consists of the following components:

2.An activation

$$a_j(t) \tag{5}$$

the neuron's state, depending on a discrete time parameter

3.Possibly a threshold

$$(\theta)_j \tag{6}$$

which stays fixed unless changed by a learning function,

4.An activation function f that computes the new activation at a given time t+1 from

$$a_j(t), (\theta)_j \tag{7}$$

5.And the net input

$$p_j(t) \tag{8}$$

6.Giving rise to the relation

$$a_j(t+1) = f(a_j(t), p_j(t), (\theta)_j) \tag{9}$$

7.And an output function

$$f_{out} \tag{10}$$

8.Computing the output from the activation

$$o_j(t) = f_{out}(a_j(t)). \tag{11}$$

Often the output function is simply the Identity function.

An input neuron has no predecessor but serves as input interface for the whole network. Similarly an output neuron has no successor and thus serves as output interface of the whole network.

Connections,weights and biases

The network consist of connections,each connection transferring the output of a neuron j.In this sense i is the predecessor of j and j is the successor of i.Each connection is assigned a weight Wij.Sometimes a bias term is added to the total weighted sum of inputs to serve as a threshold to shift the activation function.

Propagation Function

The propagation function computes the input

$$p_j(t) \tag{12}$$

to the neuron j from the outputs

$$o_i(t) \tag{13}$$

of predecessor neurons and typically has the form

$$p_j(t) = \sum_i o_i(t) w_{ij}. \tag{14}$$

When a bias value is added with the function, the above form changes to the following:

$$p_j(t) = \sum_i o_i(t) w_{ij} + w_{0j} \tag{15}$$

where

$$w_{0j} \tag{16}$$

is a bias.

Learning rule

The learning rule is a rule or an algorithm which modifies the parameters of the neural network, in order for a given input to the network to produce a favored output. This learning process typically amounts to modifying the weights and thresholds of the variables within the network.

1.3 Back Propagation

Backpropagation algorithms are a family of methods used to efficiently train artificial neural networks (ANNs) following a gradient descent approach that exploits the chain rule. The main feature of backpropagation is its iterative, recursive and efficient method for calculating the weights updates to improve the network until it is able to perform the task for which it is being trained. It is closely related to the Gauss–Newton algorithm.

Backpropagation requires the derivatives of activation functions to be known at network design time. Automatic differentiation is a technique that can automatically and analytically provide the derivatives to the training algorithm. In the context of learning, backpropagation is commonly used by the gradient descent optimization algorithm to adjust the weight of neurons by calculating the gradient of the loss function; backpropagation computes the gradient(s), whereas (stochastic) gradient descent uses the gradients for training the model (via optimization).

1.3.1 Optimization Problem

To understand the mathematical derivation of the backpropagation algorithm, it helps to first develop some intuition about the relationship between the actual output of a neuron and the correct output for a particular training example. Consider a simple neural network with two input units, one output unit and no hidden units, and in which each neuron uses a linear output (unlike most work on neural networks, in which mapping from inputs to outputs is non-linear)[note 1] that is the weighted sum of its input.

A simple neural network with two input units (each with a single input) and one output unit (with two inputs).Initially, before training, the weights will be set randomly. Then the neuron learns from

training examples, which in this case consist of a set of tuples (x_1, x_2, t) where x_1 , x_2 and are the inputs to the network and t is the correct output (the output the network should produce given those inputs, when it has been trained). The initial network, given x_1 and x_2 , will compute an output y that likely differs from t (given random weights). A loss function $L(t, y)$ is used for measuring the discrepancy between the expected output t and the actual output y. For regression analysis problems the squared error can be used as a loss function, for classification the categorical crossentropy can be used.

As an example consider a regression problem using the square error as a loss:

$$L(t, y) = (t - y)^2 = E, \tag{17}$$

where E is the discrepancy or error.

Consider the network on a single training case: (1, 1, 0), thus the input x_1 and x_2 are 1 and 1 respectively and the correct output, t is 0. Now if the actual output y is plotted on the horizontal axis against the error E on the vertical axis, the result is a parabola. The minimum of the parabola corresponds to the output y which minimizes the error E. For a single training case, the minimum also touches the horizontal axis, which means the error will be zero and the network can produce an output y that exactly matches the expected output t. Therefore, the problem of mapping inputs to outputs can be reduced to an optimization problem of finding a function that will produce the minimal error.

However, the output of a neuron depends on the weighted sum of all its inputs:

$$y = x_1 w_1 + x_2 w_2, \tag{18}$$

where w_1 and w_2 are the weights on the connection from the input units to the output unit. Therefore, the error also depends on the incoming weights to the neuron, which is ultimately what needs to be changed in the network to enable learning. If each weight is plotted on a separate horizontal axis and the error on the vertical axis, the result is a parabolic bowl. For a neuron with k weights, the same plot would require an elliptic paraboloid of k+1 dimensions.

One commonly used algorithm to find the set of weights that minimizes the error is gradient descent. Backpropagation is then used to calculate the steepest descent direction in an efficient way.

1.3.2 Derivation for a single Layered Network

The gradient descent method involves calculating the derivative of the loss function with respect to the weights of the network. This is normally done using backpropagation. Assuming one output neuron,[note 2] the squared error function is

$$E = L(t, y) \tag{19}$$

where

E is the loss for the output y and target value t,

t is the target output for a training sample, and

y is the actual output of the output neuron.

For each neuron j, its output o_j is defined as

$$o_j = \varphi(\text{net}_j) = \varphi\left(\sum_{k=1}^{n} w_{kj} o_k\right). \tag{20}$$

Where the activation function φ is non-linear and differentiable (even if the ReLU is not in one point). A historically used activation function is the logistic function:

$$\varphi(z) = \frac{1}{1 + e^{-z}} \tag{21}$$

which has a convenient derivative of:

$$\frac{d\varphi(z)}{dz} = \varphi(z)(1 - \varphi(z)) \tag{22}$$

The input net_j to a neuron is the weighted sum of outputs o_k of previous neurons. If the neuron is in the first layer after the input layer, the o_k of the input layer are simply the inputs x_k to the network. The number of input units to the neuron is n. The variable w_{kj} denotes the weight between neuron k of the previous layer and neuron j of the current layer.

Finding the Derivative of The Error

Calculating the partial derivative of the error with respect to a weight w_{ij} is done using the chain rule twice:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \text{net}_j}\frac{\partial \text{net}_j}{\partial w_{ij}} \tag{23}$$

In the last factor of the right-hand side of the above, only one term in the sum net_j net_j depends on w_{ij}, so that

$$\frac{\partial \text{net}_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{k=1}^{n} w_{kj} o_k\right) = \frac{\partial}{\partial w_{ij}} w_{ij} o_i = o_i. \tag{24}$$

If the neuron is in the first layer after the input layer, o_i is just x_i.
The derivative of the output of neuron j with respect to its input is simply the partial derivative of the activation function:

$$\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial \varphi(\text{net}_j)}{\partial \text{net}_j} \tag{25}$$

which for the logistic activation function case is:

$$\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial}{\partial \text{net}_j}\varphi(\text{net}_j) = \varphi(\text{net}_j)(1 - \varphi(\text{net}_j)) \tag{26}$$

This is the reason why backpropagation requires the activation function to be differentiable. (Nevertheless, the ReLU activation function, which is non-differentiable at 0, has become quite popular, e.g. in AlexNet)

The first factor is straightforward to evaluate if the neuron is in the output layer, because then $o_j = y$ and

$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y} \tag{27}$$

If the logistic function is used as activation and square error as loss function we can rewrite it as $\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y} = \frac{\partial}{\partial y}\frac{1}{2}(t-y)^2 = y - t$

However, if j is in an arbitrary inner layer of the network, finding the derivative E with respect to o_j is less obvious.

Considering E as a function with the inputs being all neurons $L = \{u, v, \ldots, w\}$receiving input from neuron j,

$$\frac{\partial E(o_j)}{\partial o_j} = \frac{\partial E(\mathrm{net}_u, \mathrm{net}_v, \ldots, \mathrm{net}_w)}{\partial o_j} \tag{28}$$

and taking a total derivative with respect to o_j,a recursive expression for the derivative is obtained:

$$\frac{\partial E}{\partial o_j} = \sum_{\ell \in L}\left(\frac{\partial E}{\partial \mathrm{net}_\ell}\frac{\partial \mathrm{net}_\ell}{\partial o_j}\right) = \sum_{\ell \in L}\left(\frac{\partial E}{\partial o_\ell}\frac{\partial o_\ell}{\partial \mathrm{net}_\ell}\frac{\partial \mathrm{net}_\ell}{\partial o_j}\right) = \sum_{\ell \in L}\left(\frac{\partial E}{\partial o_\ell}\frac{\partial o_\ell}{\partial \mathrm{net}_\ell}w_{j\ell}\right) \tag{29}$$

Therefore, the derivative with respect to o_j can be calculated if all the derivatives with respect to the outputs o_ℓ of the next layer – the ones closer to the output neuron – are known.

Substituting Eq.23, Eq.24 Eq.27 and Eq.29 in Eq.25 we obtain:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}o_i\frac{\partial E}{\partial w_{ij}} = o_i\delta_j\frac{\partial E}{\partial w_{ij}} = o_i\delta_j \tag{30}$$

with

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net}_j} = \begin{cases} \frac{\partial L(o_j,t)}{\partial o_j}\frac{d\varphi(net_j)}{dnet_j} & \text{if } j \text{ is an output neuron,} \\ \left(\sum_{\ell \in L} w_{j\ell}\delta_\ell\right)\frac{d\varphi(net_j)}{dnet_j} & \text{if } j \text{ is an inner neuron.} \end{cases}$$

if φ is the logistic function, and the error is the square error:

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net}_j} = \begin{cases} (o_j - t_j)o_j(1-o_j) & \text{if } j \text{ is an output neuron,} \\ \left(\sum_{\ell \in L} w_{j\ell}\delta_\ell\right)o_j(1-o_j) & \text{if } j \text{ is an inner neuron.} \end{cases}$$

To update the weight w_{ij} using gradient descent, one must choose a learning rate,$\eta > 0$.The change in weight needs to reflect the impact on E of an increase or decrease in w_{ij}. If $\frac{\partial E}{\partial w_{ij}} > 0$,

an increase in w_{ij} increases E; conversely, if $\frac{\partial E}{\partial w_{ij}} < 0$, an increase in w_{ij} decreases E. The new Δw_{ij} is added to the old weight, and the product of the learning rate and the gradient,

multiplied by -1 guarantees that w_{ij} changes in a way that always decreases E. In other words, in the equation immediately below, $-\eta \frac{\partial E}{\partial w_{ij}}$ always changes w_{ij} in such a way that E is decreased:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta o_i \delta_j \tag{31}$$

1.3.3 Loss Functions

Machines learn by means of a loss function. It's a method of evaluating how well specific algorithm models the given data. If predictions deviates too much from actual results, loss function would cough up a very large number. Gradually, with the help of some optimization function, loss function learns to reduce the error in prediction.We will go through several loss functions and their applications in the domain of deep learning.

Loss function helps in optimizing the parameters of the neural networks. Our objective is to minimize the loss for a neural network by optimizing its parameters(weights). The loss is calculated using loss function by matching the target(actual) value and predicted value by a neural network. Then we use the gradient descent method to optimize the weights of the network such that the loss is minimized. This is how we train a neural network.

1.Mean Squared Error

The mean squared error tells you how close a regression line is to a set of points. It does this by taking the distances from the points to the regression line (these distances are the "errors") and squaring them. The squaring is necessary to remove any negative signs. It also gives more weight to larger differences. It's called the mean squared error as you're finding the average of a set of errors. The mean squared error (MSE) or mean squared deviation (MSD) of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors—that is, the average squared difference between the estimated values and what is estimated. MSE is a risk function, corresponding to the expected value of the squared error loss. The fact that MSE is almost always strictly positive (and not zero) is because of randomness or because the estimator does not account for information that could produce a more accurate estimate.

The MSE is a measure of the quality of an estimator—it is always non-negative, and values closer to zero are better.

The MSE is the second moment (about the origin) of the error, and thus incorporates both the variance of the estimator (how widely spread the estimates are from one data sample to another) and its bias (how far off the average estimated value is from the truth). For an unbiased estimator, the MSE is the variance of the estimator. Like the variance, MSE has the same units of measurement as the square of the quantity being estimated. In an analogy to standard deviation, taking the square root of MSE yields the root-mean-square error or root-mean-square deviation (RMSE or RMSD), which has the same units as the quantity being estimated; for an unbiased estimator, the RMSE is the square root of the variance, known as the standard error.

Tasks that fall within the paradigm of supervised learning are pattern recognition (also known as classification) and regression (also known as function approximation). The supervised learning

paradigm is also applicable to sequential data (e.g., for hand writing, speech and gesture recognition). This can be thought of as learning with a "teacher", in the form of a function that provides continuous feedback on the quality of solutions obtained thus far.

The MSE assesses the quality of a predictor (i.e., a function mapping arbitrary inputs to a sample of values of some random variable), or an estimator (i.e., a mathematical function mapping a sample of data to an estimate of a parameter of the population from which the data is sampled). The definition of an MSE differs according to whether one is describing a predictor or an estimator.

Predictor

if a vector of n predictions generated from a sample of n data points on all variables, and Y is the vector of observed values of the variable being predicted, then the within-sample MSE of the predictor is computed as:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2. \tag{32}$$

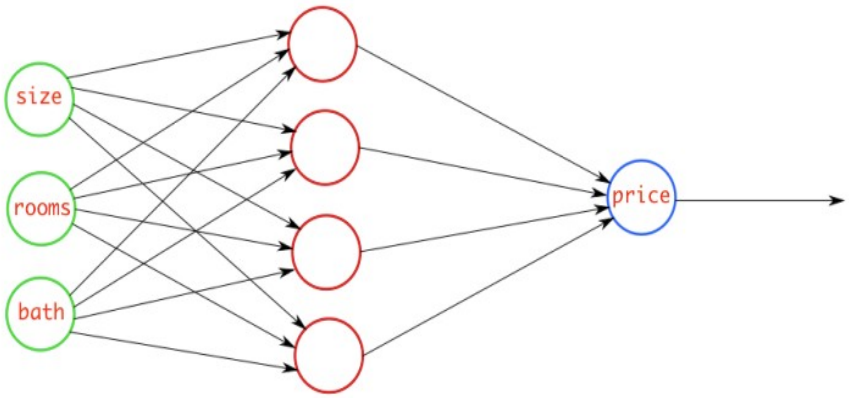
Estimator

The MSE of an estimator theta with respect to an unknown parameter theta is defined as:

$$\mathrm{MSE}(\hat{\theta}) = \mathrm{E}_{\hat{\theta}}\left[(\hat{\theta} - \theta)^2\right]. \tag{33}$$

This definition depends on the unknown parameter, but the MSE is a priori a property of an estimator. The MSE could be a function of unknown parameters, in which case any estimator of the MSE based on estimates of these parameters would be a function of the data and thus a random variable. If the estimator theta is derived from a sample statistic and is used to estimate some population statistic, then the expectation is with respect to the sampling distribution of the sample statistic.

For Example, you have a neural network with which takes some data related to house and predicts it's price. In this case, you can use the MSE loss. Basically, in the case where the output is a real number, you should use this loss function.

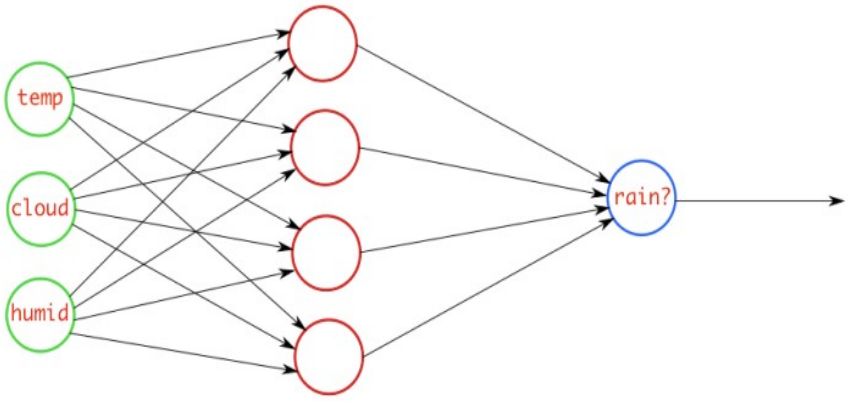


2.Binary Crossentropy

When you have a binary classification task, one of the loss function you can go ahead is this one. If you are using BCE loss function, you just need one output node to classify the data into two classes.

The output value should be passed through a sigmoid activation function so the output is in the range of (0–1).

For example, you have a neural network which takes data related to atmosphere and predicts whether it will rain or not. If the output is greater than 0.5, the network classifies it as rain and if the output is less than 0.5, the network classifies it as not rain. (it could be opposite depending upon how you train the network). More the probability score value, more the chance of raining.

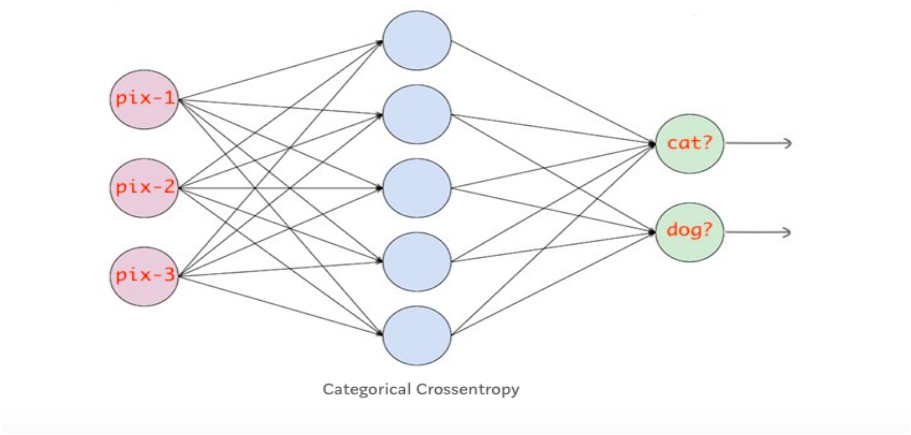


While training the network, the target value fed to the network should be 1 if it is raining otherwise 0. One important thing, if you are using BCE loss function the output of the node should be between (0–1). It means you have to use sigmoid activation function on your final output. Since sigmoid converts any real value in the range between (0–1). If we are not using sigmoid activation on the final layer? Then you can pass an argument called from logits as true to the loss function and it will internally apply the sigmoid to the output value.

3.Categorical Crossentropy

When you have a multi-class classification task, one of the loss function you can go ahead is this one. If you are using CCE loss function, there must be the same number of output nodes as the classes. And the final layer output should be passed through a softmax activation so that each node output a probability value between (0–1).

For example, you have a neural network which takes an image and classifies it into a cat or dog. If cat node has high probability score then the image is classified into cat otherwise dog. Basically, whichever class node has the highest probability score, the image is classified into that class.

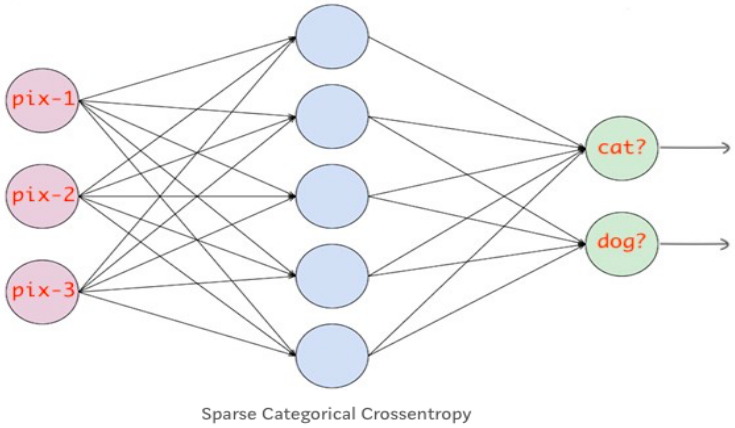


For feeding the target value at the time of training, you have to one hot encode them. If the image is of cat then target vector would be (1, 0) and if the image is of dog, target vector would be (0, 1). Basically, target vector would be of the same size as the number of classes and the index position corresponding to the actual class would be 1 and all other would be zero.

What if you are not using softmax activation on the final layer? Then you can pass an argument called from logits as true to the loss function and it will internally apply the softmax to the output value. Same as in the above case.

4.Sparse Categorical Crossentropy

This loss function is almost similar to CCE except for one change. When you are using SCCE loss function, you do not need to one hot encode the target vector. If the target image is of a cat, you simply pass 0, otherwise 1. Basically, whichever the class is you just pass the index of that class.



1.3.4 Optimizers

The goal of machine learning and deep learning is to reduce the difference between the predicted output and the actual output. This is also called as a Cost function(C) or Loss function. Cost functions are convex functions. As our goal is to minimize the cost function by finding the optimized value for weights. We also need to ensure that the algorithm generalizes well. This will

help make a better prediction for the data that was not seen before. To achieve this we run multiple iterations with different weights. This helps to find the minimum cost. This is Gradient descent.

Gradient Descent

Gradient descent is an iterative machine learning optimization algorithm to reduce the cost function. This will help models to make accurate predictions. Gradient indicates the direction of increase. As we want to find the minimum point in the valley we need to go in the opposite direction of the gradient. We update parameters in the negative gradient direction to minimize the loss.

$$\theta = \theta - \eta \nabla J(\theta; x, y)$$

θ is the weight parameter, η is the learning rate and $\nabla J(\theta; x, y)$ is the gradient of weight parameter θ

Types of Gradient Descent

Different types of Gradient Descent are:

1.Batch Gradient Descent or Vanilla Gradient Descent

2.Stochastic Gradient Descent

3.Mini batch Gradient Descent

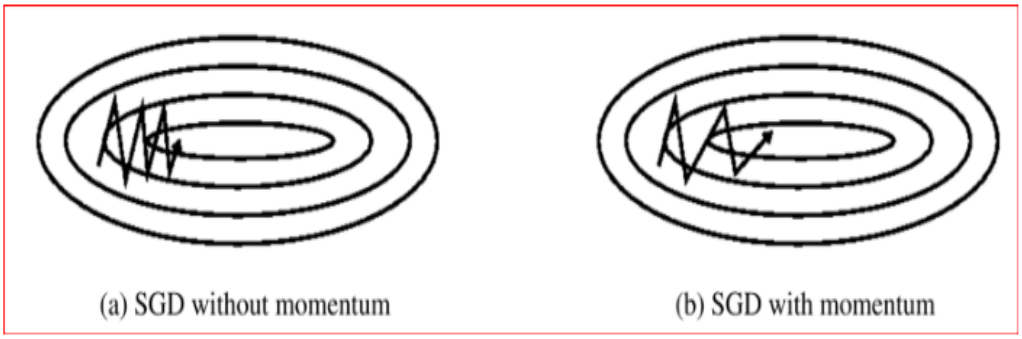
Role of an Optimizer

Optimizers update the weight parameters to minimize the loss function. Loss function acts as guides to the terrain telling optimizer if it is moving in the right direction to reach the bottom of the valley, the global minimum.

Types of Optimizers

1.Momentum

Momentum is like a ball rolling downhill. The ball will gain momentum as it rolls down the hill. Momentum helps accelerate Gradient Descent(GD) when we have surfaces that curve more steeply in one direction than in another direction. It also dampens the oscillation as shown above. For updating the weights it takes the gradient of the current step as well as the gradient of the previous time steps. This helps us move faster towards convergence. Convergence happens faster when we apply momentum optimizer to surfaces with curves.



Momentum helps accelerate Gradient Descent(GD) when we have surfaces that curve more steeply in one direction than in another direction. It also dampens the oscillation as shown above.

For updating the weights it takes the gradient of the current step as well as the gradient of the previous time steps. This helps us move faster towards convergence.

Convergence happens faster when we apply momentum optimizer to surfaces with curves.

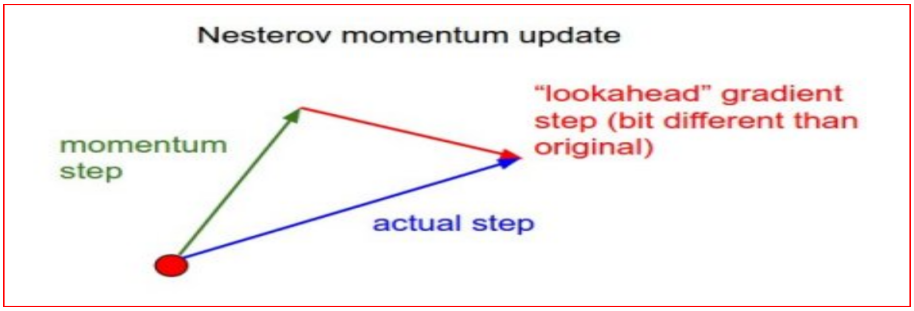
$$v_t = \gamma v_{t-1} + \eta \nabla J(\theta; x, y)$$

$$\theta = \theta - v_t$$

Momentum Gradient descent takes gradient of previous time steps into consideration

2.Nesterov Accelerated Gradient

Nesterov acceleration optimization is like a ball rolling down the hill but knows exactly when to slow down before the gradient of the hill increases again. We calculate the gradient not with respect to the current step but with respect to the future step. We evaluate the gradient of the looked ahead and based on the importance then update the weights.



NAG is like you are going down the hill where we can look ahead in the future. This way we can optimize our descent faster. Works slightly better than standard Momentum.

$$\theta = \theta - v_t$$
$$v_t = \gamma v_{t-1} + \eta \nabla J(\theta - \gamma v_{t-1})$$
$\theta - \gamma v_{t-1}$ is the gradient of looked ahead

3.Adagrad.-.Adaptive Gradient Algorithm

We need to tune the learning rate in Momentum and NAG which is an expensive process.

Adagrad is an adaptive learning rate method. In Adagrad we adopt the learning rate to the parameters. We perform larger updates for infrequent parameters and smaller updates for frequent parameters.

It is well suited when we have sparse data as in large scale neural networks. Glove word embedding uses adagrad where infrequent words required a greater update and frequent words require smaller updates.

For SGD, Momentum, and NAG we update for all parameters θ at once. We also use the same learning rate η. In Adagrad we use different learning rate for every parameter θ for every time step t.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} \cdot g_t$$

G_t is sum of the squares of the past gradients w.r.t. to all parameters θ

Adagrad eliminates the need to manually tune the learning rate.

In the denominator, we accumulate the sum of the square of the past gradients. Each term is a positive term so it keeps on growing to make the learning rate η infinitesimally small to the point that algorithm is no longer able learning. Adadelta, RMSProp, and adam tries to resolve Adagrad's radically diminishing learning rates.

4.Adadelta

1.Adadelta is an extension of Adagrad and it also tries to reduce Adagrad's aggressive, monotonically reducing the learning rate.

2.It does this by restricting the window of the past accumulated gradient to some fixed size of w. Running average at time t then depends on the previous average and the current gradient . 3.In Adadelta we do not need to set the default learning rate as we take the ratio of the running average of the previous time steps to the current gradient.

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

$$\Delta\theta = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g_t]} \cdot g_t$$

5.RMSprop

1.RMSProp is Root Mean Square Propagation. It was devised by Geoffrey Hinton.

2.RMSProp tries to resolve Adagrad's radically diminishing learning rates by using a moving average of the squared gradient. It utilizes the magnitude of the recent gradient descents to normalize the gradient.

3.In RMSProp learning rate gets adjusted automatically and it chooses a different learning rate for each parameter. RMSProp divides the learning rate by the average of the exponential decay of squared gradients.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{(1-\gamma)g^2{}_{t-1} + \gamma g_t + \varepsilon}} \cdot g_t$$

6.Adam.-.Adaptive Moment Estimation

1.Another method that calculates the individual adaptive learning rate for each parameter from estimates of first and second moments of the gradients.

2.It also reduces the radically diminishing learning rates of Adagrad.

3.Adam can be viewed as a combination of Adagrad, which works well on sparse gradients and RMSprop which works well in online and nonstationary settings.

4.Adam implements the exponential moving average of the gradients to scale the learning rate instead of a simple average as in Adagrad. It keeps an exponentially decaying average of past gradients.

5.Adam is computationally efficient and has very little memory requirement.

6.Adam optimizer is one of the most popular gradient descent optimization algorithms.

Adam algorithm first updates the exponential moving averages of the gradient(mt) and the squared gradient(vt) which is the estimates of the first and second moment.

Hyper-parameters $\beta 1$, $\beta 2 \in [0, 1)$ control the exponential decay rates of these moving averages as shown below

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

m_t and v_t are estimates of first and second moment respectively

Moving averages are initialized as 0 leading to moment estimates that are biased around 0 especially during the initial timesteps. This initialization bias can be easily counteracted resulting in bias-corrected estimates.

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{V_t}{1-\beta_2^t}$$

$\hat{m}_t$ and $\hat{v}_t$ are bias corrected estimates of first and second moment respectively

Finally, we update the parameter as shown below

$$\theta_{t+1} = \theta_t - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}}$$

1.3.5 Activation Functions

Activation functions are really important for a Artificial Neural Network to learn and make sense of something really complicated and Non-linear complex functional mappings between the inputs and response variable.They introduce non-linear properties to our Network.Their main purpose is to convert a input signal of a node in a A-NN to an output signal. That output signal now is used as a input in the next layer in the stack.

Specifically in A-NN we do the sum of products of inputs(X) and their corresponding Weights(W) and apply a Activation function f(x) to it to get the output of that layer and feed it as an input to the next layer.

If we do not apply a Activation function then the output signal would simply be a simple linear function. A linear function is just a polynomial of one degree. Now, a linear equation is easy to solve but they are limited in their complexity and have less power to learn complex functional

mappings from data. A Neural Network without Activation function would simply be a Linear regression Model, which has limited power and does not performs good most of the times. We want our Neural Network to not just learn and compute a linear function but something more complicated than that. Also without activation function our Neural network would not be able to learn and model other complicated kinds of data such as images, videos , audio , speech etc. That is why we use Artificial Neural network techniques such as Deep learning to make sense of something complicated ,high dimensional,non-linear -big datasets, where the model has lots and lots of hidden layers in between and has a very complicated architecture which helps us to make sense and extract knowledge form such complicated big datasets.

Non linear functions are those which have degree more than one and they have a curvature when we plot a Non-Linear function. Now we need a Neural Network Model to learn and represent almost anything and any arbitrary complex function which maps inputs to outputs. Neural-Networks are considered Universal Function Approximators. It means that they can compute and learn any function at all. Almost any process we can think of can be represented as a functional computation in Neural Networks.

We need to apply a Activation function f(x) so as to make the network more powerful and add ability to it to learn something complex and complicated form data and represent non-linear complex arbitrary functional mappings between inputs and outputs. Hence using a non linear Activation we are able to generate non-linear mappings from inputs to outputs.

Another important feature of a Activation function is that it should be differentiable. We need it to be this way so as to perform backpropogation optimization strategy while propogating backwards in the network to compute gradients of Error(loss) with respect to Weights and then accordingly optimize weights using Gradient descend or any other Optimization technique to reduce Error.

Some types of Activation Functions:

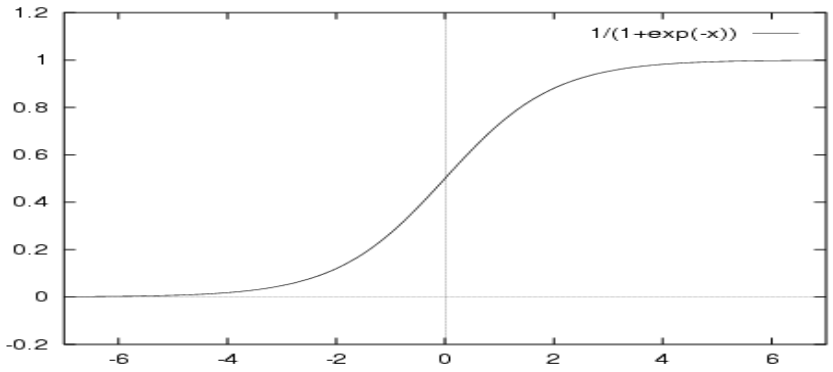
1.Sigmoid or Logistic

2.Tanh.—.Hyperbolic tangent

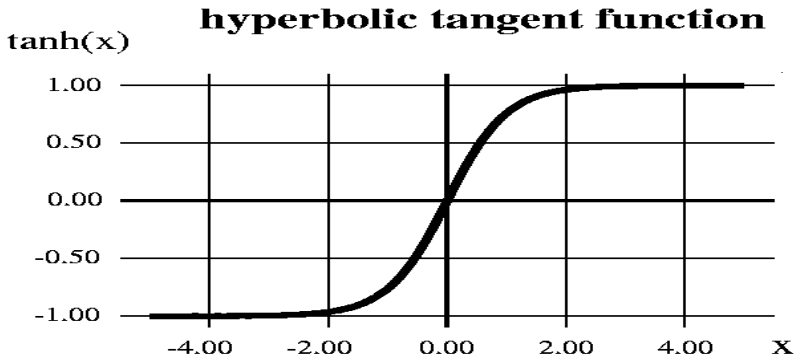
3.ReLu -Rectified linear units

Sigmoid Activation function: It is a activation function of form f(x) = 1 / 1 + exp(-x) . Its Range is between 0 and 1. It is a S—shaped curve. It is easy to understand and apply but it has major reasons which have made it fall out of popularity -

1.Vanishing gradient problem 2.Secondly , its output isn't zero centered. It makes the gradient updates go too far in different directions. 0 < output < 1, and it makes optimization harder. 3.Sigmoids saturate and kill gradients. 4.Sigmoids have slow convergence.



Hyperbolic Tangent function- Tanh : It's mathamatical formula is f(x) = 1-exp(-2x) / 1 + exp(-2x). Now it's output is zero centered because its range in between -1 to 1 i.e -1 < output < 1 . Hence optimization is easier in this method hence in practice it is always preferred over Sigmoid function . But still it suffers from Vanishing gradient problem.

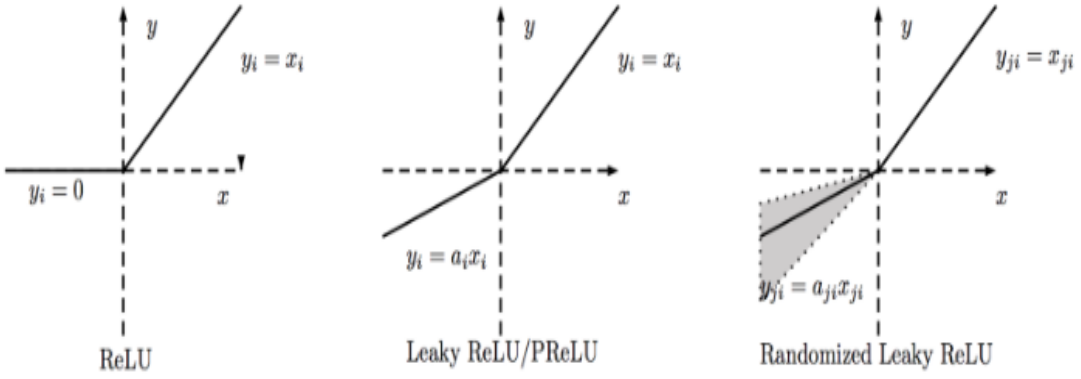


ReLu- Rectified Linear units : It has become very popular in the past couple of years. It was recently proved that it had 6 times improvement in convergence from Tanh function. It's just R(x) = max(0,x) i.e if x < 0 , R(x) = 0 and if x >= 0 , R(x) = x. Hence as seeing the mathamatical form of this function we can see that it is very simple and efficinent . A lot of times in Machine learning and computer science we notice that most simple and consistent techniques and methods are only preferred and are best. Hence it avoids and rectifies vanishing gradient problem . Almost all deep learning Models use ReLu nowadays.

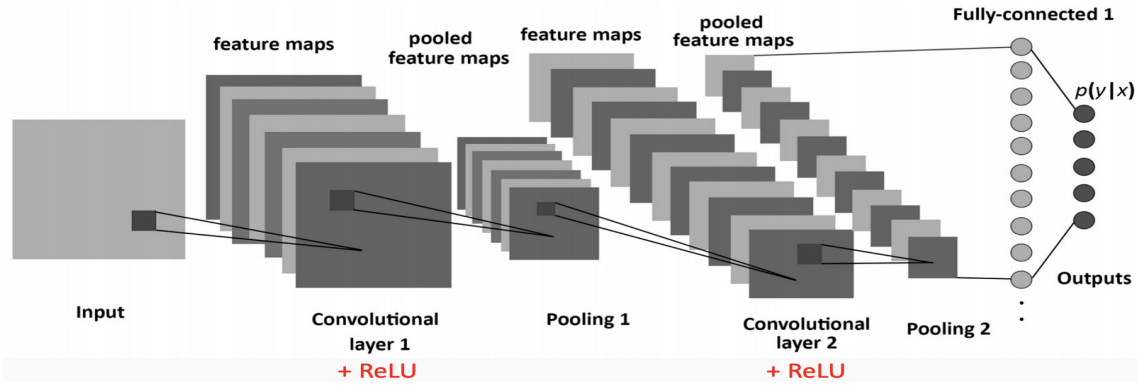
But its limitation is that it should only be used within Hidden layers of a Neural Network Model.

Hence for output layers we should use a **Softmax** function for a Classification problem to compute the probabilites for the classes , and for a regression problem it should simply use a linear function.

To fix this problem another modification was introduced called Leaky ReLu to fix the problem of dying neurons. It introduces a small slope to keep the updates alive.We then have another variant made form both ReLu and **Leaky ReLu** called Maxout function .



1.4 CNN Model



In deep learning a convolutional neural network is a class of deep neural networks , most commonly applied to analyzing visual imagery.CNNs are regularized versions of multilayer perceptrons.Multilayer perceptrons usually refer to fully connected networks ,that is,each neuron in one layer is connected to all neurons in the next layer.The "fully-connectedness" of these networks make them prone to overfitting data.Typical ways of regularizing includes adding some form of magnitude measurement of weights to th loss function.However , CNNs take a different approach towards regularization,they take advantage of the hierarchical pattern in data ad assemble more complex patterns using smaller and simpler patterns.Therefore, on the scale of connectedness and complexity CNNs are on the lower extreme.They are also known as shift invariant or space invariant artificial neural networks,based on their shared-weights architecture and translation invariance characteristics.In physics and mathematics continuos translational symmetry is the invariance(In mathematics,an Invariant is a property ,held by a class of mathematical objects,which remains unchanged when transformations of a certain type are applied to the objects.The particular class of objects and type of transformations are usually indicated by the context in which the term is used.For example,the area of a triangle is an invariant with respect to isometrics of the Euclidean plane.)of a system of equations under any translation.Discrete translational symmetry is invariant under discrete translation.

Convolution Neural Networks (CNNs) are categorized as a special type of feed-forward multi-layer neural network architecture with deep learning capabilities.CNN are primarily inspired by the

outcomes of a series of rigorous biological experiments to demystify the functionality of visual cortex of mammals. The findings of these experiments changed the understanding of vision systems in brain and established the presence of clusters of locally connected neurons (receptive fields) which are responsive only to a particular region of the entire field of vision. Unlike processing on the entire image information obtained through the retinal sensors at single level, visual cortex utilizes a 3-layer hierarchical approach of locally sensitive simple nerve cells and then ascending in the hierarchy more complex and finally the hyper-complex cells each of which works on the activation of the previous layer. Recent Years CNN has achieved Outstanding Performance in the fields of visual recognition, automatic speech recognition and natural language processing. It's the network that has advanced furthest among different types of neural network. At an early stage, because of shortage in data and computing ability, training a high-performance CNN without overfitting is difficult. Along with development of GPU and labeled data, excellent researches on CNN emerge in large numbers.

CNN can derive effective representations of the original image, which makes it identify the visual rules directly from original pixels through few pre-processing procedures. Unlike the input layer of a conventional NN classifier which is fully connected with huge number of weights, CNN employs a locally responsive input layer where neurons work on small regions of the entire input image and produce local responses, which are further processed in successive steps and finally passed through a fully connected layer with much lower dimensions. A CNN usually includes convolutional layers, pooling layers and fully connected layers. And through locally connected neural net and sharing parameters, the CNN can effectively decrease the number of parameters. Unlike other models, convolutional neural networks have the ability of preserving the neighbourhood relations and spatial locality of the input in their hidden high level feature representation.

Each convolutional layer in CNN is often followed by a pooling layer to reduce the dimensionality of responses. These layers are important for many recognition tasks because extracted features after being pooled are invariant to the local transformation of the image. CNN simultaneously learns both feature extraction and classification phases. In particular, convolutional and pooling layers play the role of feature extraction while fully connected layers play the role of classification. A convolutional layer is parametrized by the size of the kernel and the number of feature maps corresponding to kernels. A kernel (or filter) can be considered as a template which is shifted over the image to measure how well it matches each local region of the image. There can be multiple kernels in a convolutional layer to discover different characteristics in the image. There can be multiple kernels in a convolutional layer to discover different characteristics in the image. Inputs from the convolution layer are smoothened to reduce the sensitive nature of the filters towards noise and variations. The process of pooling mainly reduces the size of the image, or the color contrast across red, green, blue (RGB) channels. CNN is compatible to a wide variety of complex activation functions to model signal propagation. One of the common function is the Rectified Linear Unit (ReLU), is favourable for its faster training speed. The last layers in the network are fully connected, where the neurons of preceding layers are connected to every neuron in subsequent layers. Maxout nonlinearity in combination with dropout has achieved significant improvement in computer vision tasks and output of the standard sigmoid and ReLU.

1.5 Tesseract OCR

1.5.1 Introduction

Tesseract package contains an OCR engine-libtesseract and a command line program-tesseract. The lead developer is Ray Smith. Tesseract has unicode (UTF-8) support, and can recognize more than 100 languages "out of the box". It can be trained to recognize other languages. Tesseract supports various output formats: plain-text, hocr(html), pdf.

The Tesseract engine was originally developed as proprietary software at Hewlett Packard labs in Bristol, England and Greeley, Colorado between 1985 and 1994, with some more changes made in 1996 to port to Windows, and some migration from C to C++ in 1998. A lot of the code was written in C, and then some more was written in C++. Since then all the code has been converted to at least compile with a C++ compiler. [14]

Tesseract is available for Linux, Windows and Mac OS X, however, due to limited resources only Windows and Ubuntu are rigorously tested by developers. Tesseract up to and including version 2 could only accept TIFF images of simple one column text as inputs. These early versions did not include layout analysis and so inputting multi-columned text, images, or equations produced a garbled output. Since version 3.00 Tesseract has supported output text formatting, hOCR positional information and page layout analysis. Tesseract can detect whether text is monospaced or proportional. The initial versions of Tesseract could only recognize English language text. V3.04, released in July 2015, added an additional 39 language/script combinations, bringing the total count of support languages to over 100. Tesseract can be trained to work in other languages too. Tesseract is suitable for use as a backend, and can be used for more complicated OCR tasks including layout analysis by using a frontend such as OCRopus.

1.5.2 Architecture

Because of HP's proprietary layout analysis technology, Tesseract did not have it's own dedicated layout analyser. As a result, Tesseract assumes the inputs to be binary image with optional polygonal text regions defined.

Connected Component Analysis is the first step in which the outlines of the components are stored. Outlines are gathered together, purely by nesting, into Blobs.

Blobs are organized into text lines, and the the lines and regions are analyzed for fixed pitch or proportional text. The lines are broken into words differently based on the kind of character spacing. Fixed pitch text is chopped immidiately by character cells. Proportional text is broken into words using definite spaces or fuzzy spaces.

Recognition proceeds as a two-pass process. During the first pass, attempt is made to recognize each word. The words that are satifactorily identified are passed to an adaptive classifier as training data. As a result the adaptive classifier gets a chance at improving results among text lower down on the page. In order to utilize the training of adaptive classifier on the text near the top of the page as second pass is performed, during which words that were not recognized well enough are classified again.

Final Phase resolves the fuzzy spaces, and checks alternative hypotheses for the x-height to locate small-cap text.

1.5.3 Line and Word Finding

1.Line Finding

Algorithm is designed so that skewed page can be recognized without having to deskew, thus preventing any loss of image quality. Blob filtering and line construction are key parts of this process. Under the assumption that most blobs have uniform text size, a simple percentile height filter removes drop-caps and vertically touching characters and median height approximates the text size in the region. Blobs smaller than a certain fraction of the median height are filtered out, being most likely punctuation, diacritical marks and noise. The filtered blobs are more likely to fit a model of non-overlapping, parallel, but sloping lines. Sorting and processing the blobs by x-coordinates makes it possible to assign blobs to a unique text line, while tracking the slope across the page. Once the lines are assigned, a least median of squares fit is used to estimate the baselines, and filtered-out blobs are fitted back into appropriate lines. Final step merges blobs that overlap by at least half horizontally, putting diacritical marks together with the correct base and correctly associating parts of some broken characters.

2.Baseline Fitting

Using the text lines, baselines are fitted precisely using a quadratic spline, which allows Tesseract to handle pages with curved baselines.

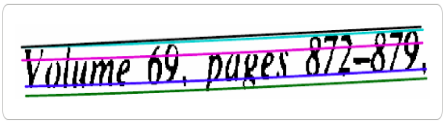


Fig-1: Curved Fitted Baseline

Baseline fitting is done by partitioning the blobs into groups of reasonable continuous displacement for the original straight baseline. A quadratic spline is fitted to the most populous partition by a least square fit.

3. Fixed Pitch Detection and Chopping

Lines are tested to determine whether they are fixed pitch. Where it finds fixed pitch text, Tesseract chops the words into characters using pitch, and disables the chopper and associator on these words for the word recognition step.

Fig-2: Fixed Pitch Chopped Word

4. Proportional Word Finding

Detecting word boundaries in a not-fixed-pitch or proportional text spacing is highly non-trivial task.

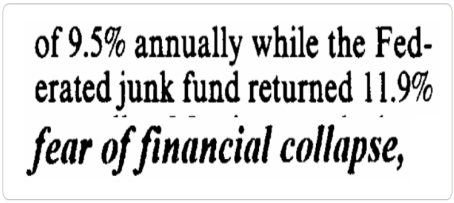


Fig-3: Difficult Word Spacing

For example, the gap between the tens and units of '11.9%' is similar size to general space, but is certainly larger the kerned space between 'erated' and 'junk'. Another case can be noticed that there is no horizontal gap between the bounding box of 'of' and 'financial'. Tesseract solves most of these problems by measuring gaps in a limited vertical range between baseline and mean line. Spaces close to a threshold are made fuzzy, where the decisions are made after word recognition.

1.5.4 Word Recognition

A major part of any word recognition algorithm is to identify how a word should be segmented into characters.

The initial segmented outputs from line finding is classified first. The non-fixed pitch text in the remaining text is classified using other word recognition steps.

1. Chopping Joined Characters

Tesseract attempts to improve the result by chopping the blob with worst confidence from the character classifier.

Chop points are found from concave vertices of a poligonal approximation of the outline, which may have a concave vertex opposite or a line segment. It may take upto 3 pairs of chop points to successfully separate joined characters from ASCII set.

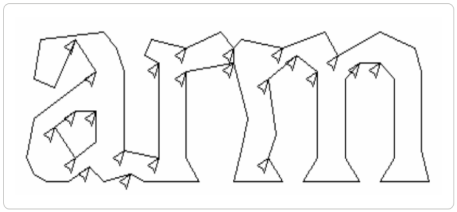


Fig-4: Candidate Chop Points and Chop

Chops are executed in priority order. Any chop that fails to improve the confidence of the result is undone, but not completely discarded so that it can be re-used by the associator if needed.

2. Associating Broken Characters

After the potential chops have been exhausted, if the word is still not good enough, it is given to the associator, which makes a best first search of the segmentation graph of the possible combinations of the maximally chopped blobs into candidate characters.

The search pulls candidate new states from a priority queue and evaluates them by classifying unclassified combinations of fragments.

The chop-then-associate method is inefficient but it gives a benefit of simpler data structures that would be required to maintain the full segmentation graph.

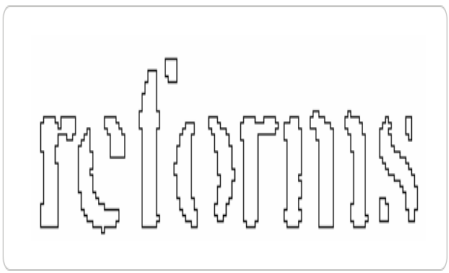


Fig-5: Broken Characters recognized by Tesseract

This ability of Tesseract to successfully classify broken characters gave it an edge over the contemporaries.

1.6 OpenCv

OpenCV (Open Source Computer vision) is free for both academic and commercial use. It is a library of programming functions mainly aimed at real-time computer vision.OpenCV's application has wide areas which includes 2D and 3D feature toolkits, Egomotion estimation, Facial recognition system, Gesture recognition, Motion understanding, Object identification Segmentation and recognition and Motion tracking.OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface.OpenCV contains libraries of pre-defined functions helpful in image processing. Since it is open source, it was chosen as the platform to test the project. Using OpenCV libraries we have implemented image processing mechanisms like RGB to grayscale conversion, erosion, dilation.

2 Problem Statement and Motivation

2.1 Problem Statement

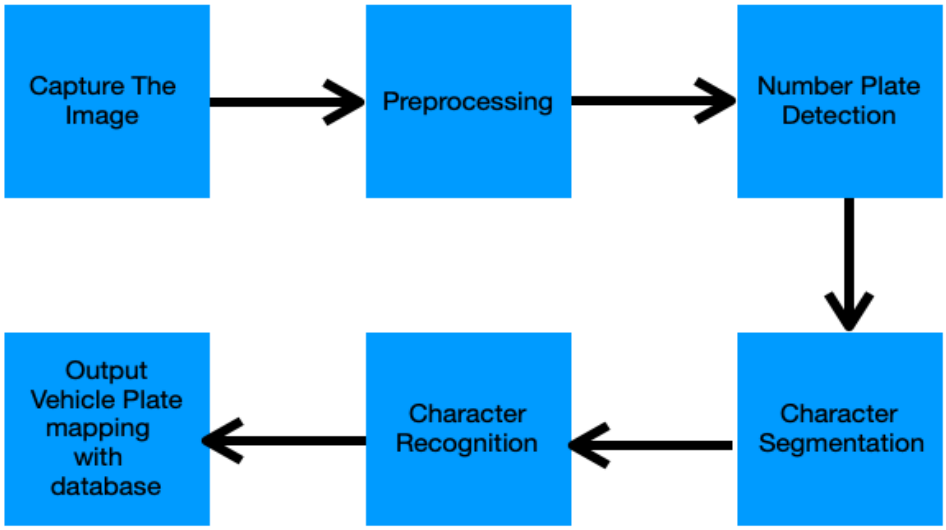
Vehicle license plate detection and recognition is a key technique in most of traffic related applications and is an active research topic in the image processing domain. Different methods, techniques and algorithms have been developed for license plate detection and recognition.Our Aim is to make an algorithm using CNN and Tesseract OCR to implement VLPDR.

2.2 Motivation

In this project we aim to make an application which identifies vehicle by their license plates. It has become a task of prime importance with the increase in number of accidents , traffic rule violations , smuggling of vehicles , use of vehicles in terrorist activities and this has created a need to make an application which will be of certain benefit in resolving this issue.

3 Proposed Methodologies

3.1 CNN model



3.1.1 Image Capturing

The image is captured through a high resolution camera .We save this image in the system which we will use further for pre processing.

Image captured through a 16MP camera

3.1.2 Pre-processing

Pre-processing refers to the transformations applied to our data before feeding it to the algorithm.Pre-processing is the technique in which background illumination conditions and the number plate localization algorithms is used. In this phase mainly focuses on reduce background noise, enhancing of contrast. The system preprocessing uses two processes: Resize – In this section we have to change the size of object according to requirement. Convert Color Space – Images captured by cameras will be either in raw format or encoded into some multimedia standards. These images will be in RGB mode basically i.e. red, green and blue. There should be using OpenCV function in preprocessing phase.

Different Data Processing Techniques Used

In Preprocessing we mainly focused on changing the color space of the image to make it more feasible for detection.

1.RGB to HSV

Unlike RGB and CMYK, which use primary colors, HSV is closer to how humans perceive color. It has three components: hue, saturation, and value. This color space describes colors (hue or tint) in terms of their shade (saturation or amount of gray) and their brightness value. Some color pickers, like the one in Adobe Photoshop, use the acronym HSB, which substitutes the term "brightness" for "value," but HSV and HSB refer to the same color model.

Hue

Hue is the color portion of the model, expressed as a number from 0 to 360 degrees:

Red falls between 0 and 60 degrees.

Yellow falls between 61 and 120 degrees.

Green falls between 121-180 degrees.

Cyan falls between 181-240 degrees.

Blue falls between 241-300 degrees.

Magenta falls between 301-360 degrees.

Saturation

Saturation describes the amount of gray in a particular color, from 0 to 100 percent. Reducing this component toward zero introduces more gray and produces a faded effect. Sometimes, saturation appears as a range from just 0-1, where 0 is gray, and 1 is a primary color.

Value(Brightness)

Value works in conjunction with saturation and describes the brightness or intensity of the color, from 0-100 percent, where 0 is completely black, and 100 is the brightest and reveals the most color.

Changing Colorspace from RGB to HSV

3.1.3 Plate Detection

Rear or front part of the vehicle is captured into an image. The image certainly contains other parts of the vehicle and the environment, which are of no requirement to the system. The area in the image that interests us is the license plate and needs to be detected from the noise.

The Number Plate Detection is the phase in which mainly focuses on ROI(Region of Interest) where we find the contour region.

Morphological Transformations

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called structuring element or kernel which decides the nature of operation.Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing, Gradient etc.

Erosion

The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object (Always try to keep foreground in white) The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).

All the pixels near boundary will be discarded depending upon the size of kernel. So the thickness or size of the foreground object decreases or simply white region decreases in the image. It is useful for removing small white noises (as we have seen in colorspace chapter), detach two connected objects etc.

Dilation

It is just opposite of erosion. Here, a pixel element is '1' if atleast one pixel under the kernel is '1'. So it increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

Opening

Opening is just another name of erosion followed by dilation. It is useful in removing noise.

Closing

Closing is reverse of Opening, Dilation followed by Erosion. It is useful in closing small holes inside the foreground objects, or small black points on the object.

1.Structuring Element

OpenCV has a function cv2.getStructuringElement() in which we pass the shape and size of kernel and get the desired kernel.

Rectangular Kernel cv2.getStructuringElement(cv2.MORPHRECT,(5,5))

array([[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1]], dtype=uint8)

2.Tophat

It is the difference between input image and Opening of the image.

Tophat Image

3.Blackhat

It is the difference between the closing of the input image and input image.

Blackhat Image

4.Image Addition

Addition of two images can be done using OpenCV addition and Numpy addition is a saturated operation,res=img1 + img2.Both images should be of same depth and type, or second image can just be a scalar value.

OpenCv addition is a saturated operation while Numpy addition is a modulo operation.

5.Image Substraction

Substraction of two images by OpenCV function, cv.subtract(). res = img1 - img2. Both images should be of same depth and type.

6.Gaussian Blur

Image blurring is achieved by convolving the image with a low-pass filter kernel. It is useful for removing noise. It actually removes high frequency content (e.g: noise, edges) from the image resulting in edges being blurred when this is filter is applied. (Well, there are blurring techniques which do not blur edges). OpenCV provides mainly four types of blurring techniques.

In this a Gaussian kernel is used. It is done with the function, cv2.GaussianBlur(). We have to specify the width and height of the kernel which should be positive and odd. We also have to specify

the standard deviation in the X and Y directions, sigma X and sigma Y respectively. If only sigma X is specified, sigma Y is taken as equal to sigma X. If both are given as zeros, they are calculated from the kernel size. Gaussian filtering is highly effective in removing Gaussian noise from the image.

7.Adaptive Thresholding

In simple thresholding the threshold value is global,i.e.,it is same for all pixles in the range.Adaptive Thresholding is the method where the threshold value is calculated for smaller regions and therefore there will be different threshold values for different regions.

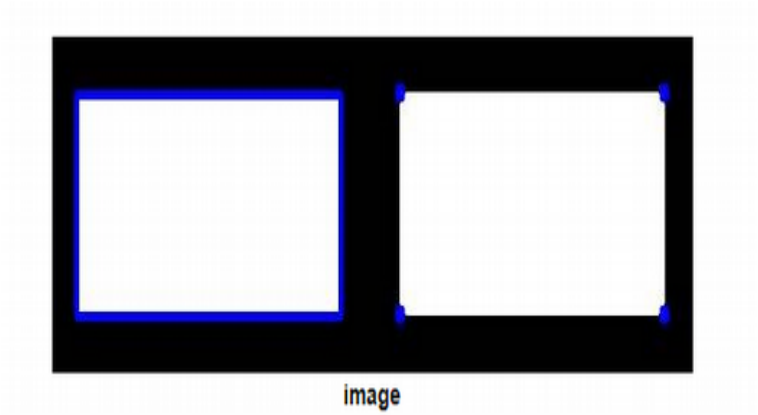
8.Contours

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

8.1.Contour Approximation Method

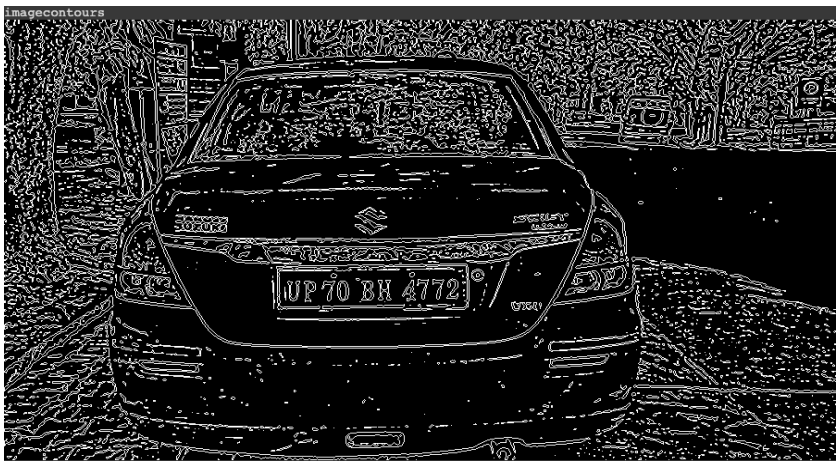
This is the third argument in cv2.findContours function. contours are the boundaries of a shape with same intensity. It stores the (x,y) coordinates of the boundary of a shape.The no. of coordinates being stored by contours is specified by contour approximation method.

If we pass cv2.CHAIN_APPROX_NONE, all the boundary points are stored but cv2.CHAIN_APPROX_SIMPLE removes all redundant points and compresses the contour, thereby saving memory.

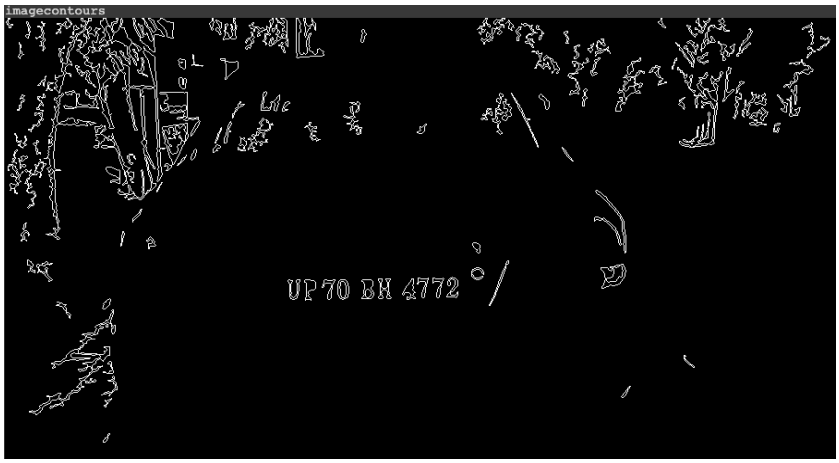


In this step we will check for contours on thresh.

Drawing contours based on actual found contours of thresh image:



After retreiving possible characters ,using those values to draw new contours:

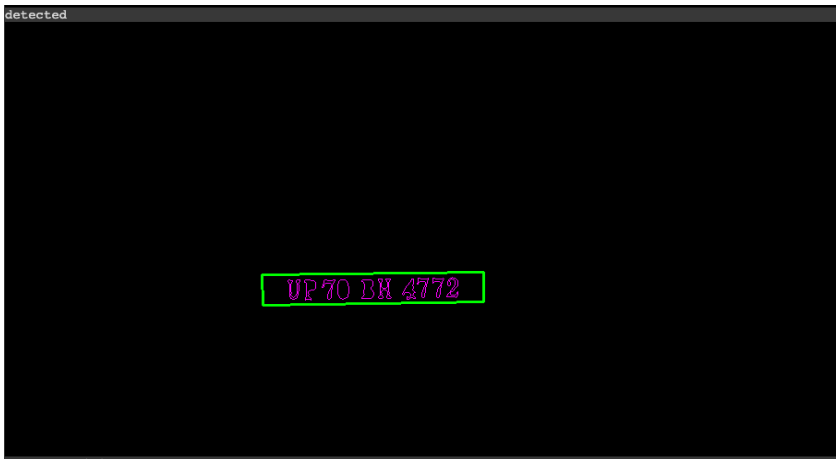


Using a function find all chars in the big list that are a match for the single possible char, and return those matching chars as a list given a possible char and a big list of possible chars.if the char we attempting to find matches for is the exact same char as the char in the big list we are currently checking then we should not include it in the list of matches b/c that would end up double including the current char so do not add to list of matches and jump back to top .

Drawing contours after finding all the characters in the image:

9.Bounding Rectangle

Creating a bounding rectangle around the detected characters after calculating plate width,height and the centre point of the plate.



10.Image Extraction

1.Cropping the image of the detected plate obtained from the contoured image

2.Extracting the image of the plate detected in the original image.

3.1.4 Text Segmentation

1.Read the Image

The Objective of cv2.imread() is to read an image. The image should be in the working directory or a full path of image should be given.

Second argument is a flag which specifies the way image should be read.

..cv2.IMREAD_COLOR : Loads a color image. Any transparency of image will be neglected. It is the default flag.

..cv2.IMREAD_GRAYSCALE : Loads image in grayscale mode

..cv2.IMREAD_UNCHANGED : Loads image as such including alpha channel

2.Thresholding

The function cv2.threshold performs an operation where If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black).First

argument is the source image, which should be a grayscale image. Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value. OpenCV provides different styles of thresholding and it is decided by the fourth parameter of the function. Different types are:

..cv2.THRESH_BINARY

..cv2.THRESH_BINARY_INV

..cv2.THRESH_TRUNC

..cv2.THRESH_TOZERO

..cv2.THRESH_TOZERO_INV

3.Findcontours

Find the contours of binary image with the help of cv2.findcontours() function

3.Contour Perimeter

It is also called arc length. It can be found out using cv2.arcLength() function. Second argument specify whether shape is a closed contour (if passed True), or just a curve.

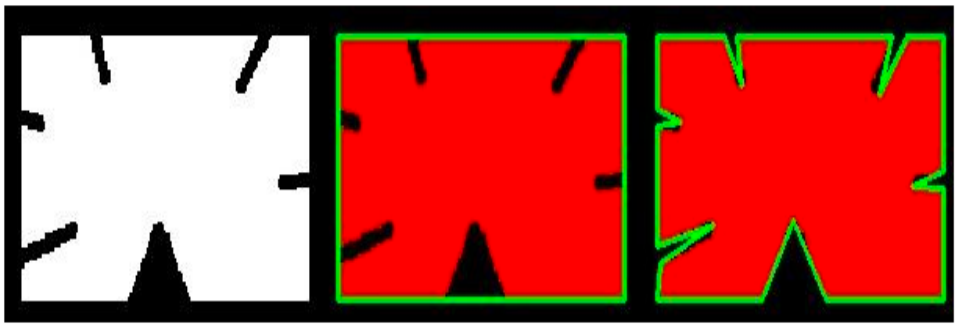
perimeter = cv2.arcLength(cnt,True)

4.Contour Approximation

It approximates a contour shape to another shape with less number of vertices depending upon the precision we specify.second argument is called epsilon, which is maximum distance from contour to approximated contour. It is an accuracy parameter. A wise selection of epsilon is needed to get the correct output.Third argument specifies whether curve is closed or not.

epsilon = 0.1*cv2.arcLength(cnt,True)

approx = cv2.approxPolyDP(cnt,epsilon,True)

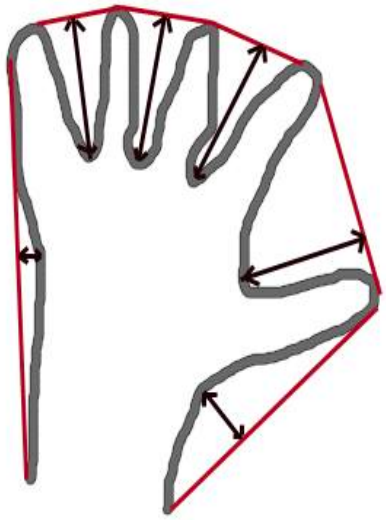


image

5.Convex hull

This functions find the convex hull of a 2D point set using the Sklansky's algorithm that has O(N logN) complexity in the current implementation.

cv2.convexhull() returns convexhull of an image



image

6.Checking Convexity

cv2.isContourConvex() is a function that checks whether a curve is convex or not. It just return whether True or False.

7.Straight Bounding Rectangle

It is a straight rectangle, it doesn't consider the rotation of the object. So area of the bounding rectangle won't be minimum. It is found by the function cv2.boundingRect().It returns top left coordinates of a rectangle and its height and width.cv2.rectangle puts boundary on each digit.

Then using contours hierarchy we will put boundary on each digit.For putting boundary we will use cv2.rectangle() which will use the para metres returned by the function cv2.boundingRect().

8.Crop each image and process

All the images obtained will be cropped,resized and padding will also be applied and the images will be send further for processing.

3.1.5 Text Recognition

Till now we have detected the number plate and also segmented all the characters present on the plate and has refined them for recognition.

For Text Recognition we will use our CNN model which we have trained on Emnist-balanced dataset which is available on kaggle consisting dataset of handwritten characters.For training the model we have test.csv file containing 112800 images and for testing we have test.csv file containing 18800 images.Number of classes is 47, 10 digits, 26 letters, and 11 capital letters that are different looking from their lowercase counterparts.

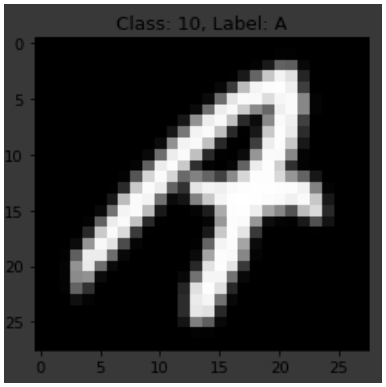
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	...	745
0	45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
1	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
2	43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
3	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
4	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
5	42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
6	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
7	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
8	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0

10 rows x 785 columns

First 10 data in the training set.

The 47 classes are as follows:

class_mapping = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabdefghnqrt'



An example data from the training data set.

Image size is 28*28

The CNN model used is :

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 12, 12, 12)        312
_________________________________________________________________
dropout (Dropout)            (None, 12, 12, 12)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 5, 5, 18)          1962
_________________________________________________________________
dropout_1 (Dropout)          (None, 5, 5, 18)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 24)          1752
_________________________________________________________________
flatten (Flatten)            (None, 384)               0
_________________________________________________________________
dense (Dense)                (None, 150)               57750
_________________________________________________________________
dense_1 (Dense)              (None, 47)                7097
=================================================================
Total params: 68,873
Trainable params: 68,873
Non-trainable params: 0
_________________________________________________________________
```

A single model can be used to simulate having a large number of different network architectures by randomly dropping out nodes during training. This is called dropout and offers a very computationally cheap and remarkably effective regularization method to reduce overfitting and improve generalization error in deep neural networks of all kinds.

The layers of the CNN are as follows:

1.The first layer is the Conv2D layer which contains 12 filters of dimension 5*5 with strides=2

2.The second layer is the Dropout layer with probability .5

3.The third layer is the Conv2D layer with 18 filters of dimension 3*3

4.The fourth layer is the Dropout layer with probability .5

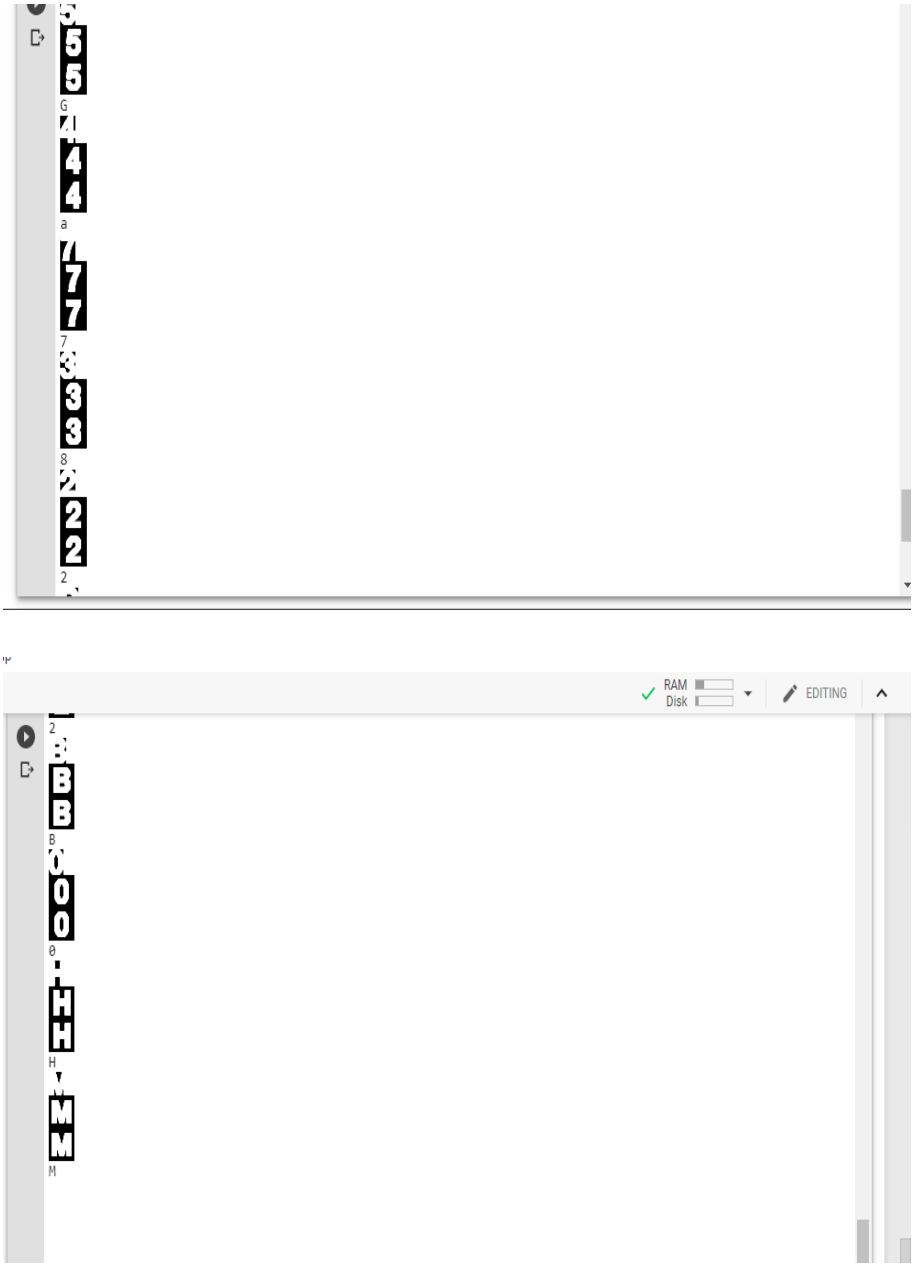
5.The fifth layer is the Conv2D layer which contains 24 filters of dimension 2*2

6.Then we have fully connected network consisting of flatten layer followed by a dense layer consisting of 150 neurons followed by the final output layer consisting of 47 neurons.

The loss function used is categorical_crossentropy and the optimzier used is Adam.

For running this model over the training and testing dataset for 500 epochs we obtained accuracy=84.68 and validation_accuracy=.

Each segmented character is passed in the function predict_digit which evaluates the class with the maximum probability and returns the element at that index in the class_mapping string .



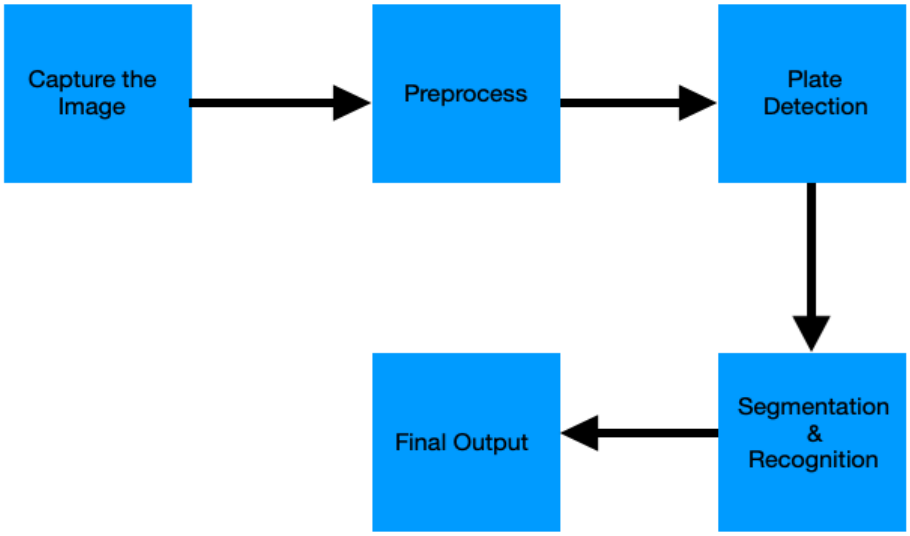
Above are the images of characters followed by their class mapping.In the three consecutive images the first one is the one extracted after applying convex hull and bounding rectangle , the second one is extracted after applying resizing and cropping and the third one is extracted after applying padding.

And the one written in the text format is the predicted character or digit given by the CNN.

3.1.6 Final Output

After successfully completing all the previous operations we now place label on each digit of the image predicted by the CNN.

3.2 Tesseract OCR



3.2.1 Capture The Image

Same as done in CNN model.

3.2.2 Preprocess

Same as done in CNN model.

3.2.3 Plate Detection

Same as done in CNN model.

3.2.4 Segmentation and Recognition

After detection is done we have obtained the number plate from the image and now we can do segmentation and recognition.

Number plate obtained from the original image.

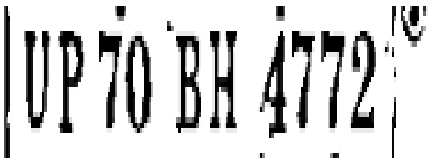
Before Segmentation :

1.We first convert our image from RBG to Grayscale 2.Then we will apply normal thresholding for pixel values greater than 55.

3.Apply adaptive_thresholding using the method cv2.ADAPTIVE_THRESH_MEAN_C

4.Again apply adaptive_thresholding using the method cv2.ADAPTIVE_THRESH_GAUSSIAN_C

6.Now we will resize the image for better segmentation and recognition



7.Add border to increase the white padding

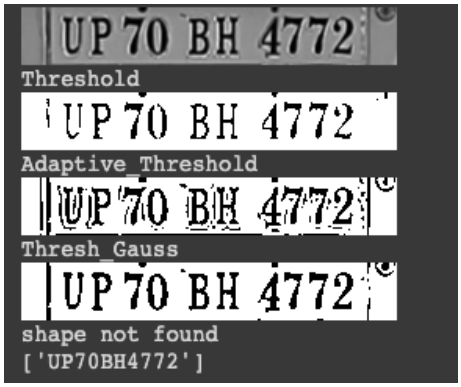
Now we will apply **Canny Edge Detection** and **HoughLinesP** on the bordered image.

After applying Closing Morphological Operation we will get the following result.

Now by using pytesseract.image_to_string() the process of segmentation and recognition will be carried as discussed above.

3.2.5 Final Output

The Final Output obtained is:



4 Implementation

4.1 CNN Model

4.1.1 Capture the Image

capture the image using camera or the code snippets mentioned on google colab for camera capture.

4.1.2 Preprocess

some basic preprocessing is applied as shown below.

```
# hsv transform - value = gray image
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
hue, saturation, value = cv2.split(hsv)
print('hsv_value')
cv2_imshow(value)
```

4.1.3 Plate Detection

The steps are similar to what we discussed in section 2.1.3

1.Applying topHat/blackHat operations

```
# kernel to use for morphological operations
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))

# applying topHat/blackHat operations
topHat = cv2.morphologyEx(value, cv2.MORPH_TOPHAT, kernel)
blackHat = cv2.morphologyEx(value, cv2.MORPH_BLACKHAT, kernel)
print('tophat')
cv2_imshow(topHat)
print('blackhat')
cv2_imshow(blackHat)
# cv2.imwrite(temp_folder + '3 - topHat.png', topHat)
# cv2.imwrite(temp_folder + '4 - blackHat.png', blackHat)
```

Apply these operations on the result obtained after applying the above operation.

2.Add and Subtract between morphological operations

```
add = cv2.add(value, topHat)
subtract = cv2.subtract(add, blackHat)
print('subtract')
cv2_imshow(subtract)
```

This is used to reduce the external noise present in the image.

3.Applying Gaussian blur and Adaptive Threshold

```
# applying gaussian blur on subtract image
blur = cv2.GaussianBlur(subtract, (5, 5), 0)
print('Blur')
cv2_imshow(blur)
# cv2.imwrite(temp_folder + '6 - blur.png', blur)

# thresholding
thresh = cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 19, 9)
print('Thresh')
cv2_imshow(thresh)
```

4.Drawing contours

```
imageContours, contours, hierarchy = cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)

# else, if OpenCV >= 4, try
#contours, hierarchy = cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)

# get height and width
height, width = thresh.shape

# create a numpy array with shape given by threshed image value dimensions
imageContours = np.zeros((height, width, 3), dtype=np.uint8)

# list and counter of possible chars
possibleChars = []
countOfPossibleChars = 0

# loop to check if any (possible) char is found
for i in range(0, len(contours)):

    # draw contours based on actual found contours of thresh image
    cv2.drawContours(imageContours, contours, i, (255, 255, 255))

    # retrieve a possible char by the result ifChar class give us
    possibleChar = ifChar(contours[i])

    # by computing some values (area, width, height, aspect ratio) possibleChars list is being populated
    if checkIfChar(possibleChar) is True:
        countOfPossibleChars = countOfPossibleChars + 1
        possibleChars.append(possibleChar)
```

5.Drawing contours after finding possible chars

```
imageContours = np.zeros((height, width, 3), np.uint8)

ctrs = []

# populating ctrs list with each char of possibleChars
for char in possibleChars:
    ctrs.append(char.contour)

# using values from ctrs to draw new contours
cv2.drawContours(imageContours, ctrs, -1, (255, 255, 255))
print('imagecontours')
cv2_imshow(imageContours)
# cv2.imwrite(temp_folder + '9 - contoursPossibleChars.png', imageContours)
```

6.Final contours

```
imageContours = np.zeros((height, width, 3), np.uint8)

for listOfMatchingChars in listOfListsOfMatchingChars:
    contoursColor = (255, 0, 255)

    contours = []

    for matchingChar in listOfMatchingChars:
        contours.append(matchingChar.contour)

    cv2.drawContours(imageContours, contours, -1, contoursColor)

print("finalContours")
cv2_imshow( imageContours)
# cv2.imwrite(temp_folder + '10 - finalContours.png', imageContours)
```

7.Creating a bounding box

```
for i in range(0, len(plates_list)):
    # finds the four vertices of a rotated rect - it is useful to draw the rectangle.
    p2fRectPoints = cv2.boxPoints(plates_list[i].rrLocationOfPlateInScene)

    # roi rectangle colour
    rectColour = (0, 255, 0)

    cv2.line(imageContours, tuple(p2fRectPoints[0]), tuple(p2fRectPoints[1]), rectColour, 2)
    cv2.line(imageContours, tuple(p2fRectPoints[1]), tuple(p2fRectPoints[2]), rectColour, 2)
    cv2.line(imageContours, tuple(p2fRectPoints[2]), tuple(p2fRectPoints[3]), rectColour, 2)
    cv2.line(imageContours, tuple(p2fRectPoints[3]), tuple(p2fRectPoints[0]), rectColour, 2)

    cv2.line(img, tuple(p2fRectPoints[0]), tuple(p2fRectPoints[1]), rectColour, 2)
    cv2.line(img, tuple(p2fRectPoints[1]), tuple(p2fRectPoints[2]), rectColour, 2)
    cv2.line(img, tuple(p2fRectPoints[2]), tuple(p2fRectPoints[3]), rectColour, 2)
    cv2.line(img, tuple(p2fRectPoints[3]), tuple(p2fRectPoints[0]), rectColour, 2)
    print("detected")
    cv2_imshow(imageContours)
    # cv2.imwrite(temp_folder + '11 - detected.png', imageContours)
    print("detectedOriginal")
    cv2_imshow(img)
    # cv2.imwrite(temp_folder + '12 - detectedOriginal.png', img)
    print('plates')
    cv2_imshow(plates_list[i].Plate)
    print(i)
    cv2.imwrite(temp_folder+'/plates.png' , plates_list[i].Plate)
```

A bounded box will be created around each character.

4.1.4 Segmentation

get_output_image()

These steps explain what we introduced in section 2.1.4

1.Thresholding and Finding Contours

```
ret,thresh = cv2.threshold(img,127,255,0)
im2,contours,hierarchy = cv2.findContours(thresh, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(img, contours, -1, (0,255,0), 3)
cv2_imshow(im2)
cv2_imshow(img)
```

2.Contour Perimeter , Approximation and Convex Hull

To separate out each character

```
for j,cnt in enumerate(contours):
    epsilon = 0.01*cv2.arcLength(cnt,True)
    approx = cv2.approxPolyDP(cnt,epsilon,True)

    hull = cv2.convexHull(cnt)
    k = cv2.isContourConvex(cnt)
    x,y,w,h = cv2.boundingRect(cnt)
```

3.Bounding Rectangle

```
    x,y,w,h = cv2.boundingRect(cnt)


    if(hierarchy[0][j][3]!=-1 and w>10 and h>10):
        #putting boundary on each digit
        contours=cv2.rectangle(img_org,(x,y),(x+w,y+h),(0,255,0),2)
        #cv2.drawContours(img,contours , -1, (0,255,0), 3)
        #cv2_imshow(img_org)
        #cropping each image and process
        roi = img[y:y+h, x:x+w]
        roi = cv2.bitwise_not(roi)
```

4.Refining

image_refiner()

```
def image_refiner(gray):
    org_size = 22
    img_size = 28
    rows,cols = gray.shape

    if rows > cols:
        factor = org_size/rows
        rows = org_size
        cols = int(round(cols*factor))
    else:
        factor = org_size/cols
        cols = org_size
        rows = int(round(rows*factor))
    gray = cv2.resize(gray, (cols, rows))
    cv2_imshow(gray)

    #get padding
    colsPadding = (int(math.ceil((img_size-cols)/2.0)),int(math.floor((img_size-cols)/2.0)))
    rowsPadding = (int(math.ceil((img_size-rows)/2.0)),int(math.floor((img_size-rows)/2.0)))

    #apply apdding
    gray = np.lib.pad(gray,(rowsPadding,colsPadding),'constant')
    cv2_imshow(gray)
    return gray
```

4.1.5 Recognition

For recognition we will use the model which we defined in section 2.1.5 and as we have already trained our model on the provided dataset it will predict character one at a time and will return the class mapping with maximum probability.

predict_digit()

```
# loading pre trained model

def predict_digit(img):
    test_image = img.reshape(-1,28,28,1)

    predicted_indices=np.argmax(model.predict(test_image))
    return class_mapping[predicted_indices]
```

4.1.6 Final Ouput

put_label()

```
#putting label
def put_label(t_img,label,x,y):
    font = cv2.FONT_HERSHEY_SIMPLEX
    l_x = int(x) - 10
    l_y = int(y) + 10
    cv2.rectangle(t_img,(l_x,l_y+5),(l_x+35,l_y-35),(0,255,0),-1)
    cv2.putText(t_img,str(label),(l_x,l_y), font,1.5,(255,0,0),1,cv2.LINE_AA)
    return t_img
```

4.2 Tesseract OCR

4.2.1 Detection

Detection will be same as in CNN model

4.2.2 Recognition

1.create four folder wherever the python file will be: plates, processed, resized, borders. These will contain the images for each step.

```
plates = glob.glob("/content/plates/*.png")
processed = glob.glob("/content/processed/*.png")
resized = glob.glob("/content/resized/*.png")
bordered = glob.glob("/content/bordered/*.png")
```

2. The code is split in four main functions.

The first, adaptiveThreshold(), will take all the plates and, for each of them, will apply two main transormations (apart of gray and thresh):

ADAPTIVE_THRESH_MEAN_C and ADAPTIVE_THRESH_GAUSSIAN_C both under cv2.adaptiveThreshold method.

ADAPTIVE_THRESH_MEAN_C value is the is the mean of neighbourhood area.

ADAPTIVE_THRESH_GAUSSIAN_C value is the weighted sum of neighbourhood values where weights are a gaussian window.

adaptiveThreshold is used mainly because there are different lightning conditions.

```
def adaptiveThreshold(plates):
    for i, plate in enumerate(plates):
        img = cv2.imread(plate)

        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        print('gray')
        cv2_imshow(gray)

        ret, thresh = cv2.threshold(gray, 50, 255, cv2.THRESH_BINARY)
        print('thresh')
        cv2_imshow(thresh)

        threshMean = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 5, 10)
        print('threshmean')
        cv2_imshow(threshMean)

        threshGauss = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 51, 27)
        print('threshgauss')
        cv2_imshow(threshGauss)
        cv2.imwrite("/content/processed/plate{}.png".format(i), threshGauss)

        cv2.waitKey(0)
```

Resize

After thresholding the images,resize them to a fixed size using resize() function. The main features of this function are that it resize the image keeping it's ratio and not too much quality is lost.cv2.INTER_CUBIC interpolation is used because it enlarged most of the images.

```
def resize(processed):
    for i, image in enumerate(processed):
        image = cv2.imread(image)

        ratio = 200.0 / image.shape[1]
        dim = (200, int(image.shape[0] * ratio))

        resizedCubic = cv2.resize(image, dim, interpolation=cv2.INTER_CUBIC)

        cv2.imwrite("/content/resized/plate{}.png".format(i), resizedCubic)
```

AddBorder

the addBorder() is used just after the resize. This will add a 10 pixel border to each image.

```
def addBorder(resized):
    print(resized)
    for i, image in enumerate(resized):
        image = cv2.imread(image)

        bordersize = 10
        border = cv2.copyMakeBorder(image, top=bordersize, bottom=bordersize, left=bordersize, right=bordersize,
                                    borderType=cv2.BORDER_CONSTANT, value=[255, 255, 255])

        cv2.imwrite("/content/bordered/plate{}.png".format(i), border)
```

CleanOCR

cleanOCR() function splitted in two main parts:

the cleaning part: for each image calculate the edges (with Canny function) and after, using HoughLinesP detect the lines. If the line is between a certain range delete it.

```
def cleanOCR(borders):
    detectedOCR = []

    for i, image in enumerate(borders):
        image = cv2.imread(image)

        edges = cv2.Canny(image, 50, 150, apertureSize=3)
        lines = cv2.HoughLinesP(image=edges, rho=1, theta=np.pi / 180, threshold=100, lines=np.array([]),
                                minLineLength=100, maxLineGap=80)
        #print(lines.shape)
        try:
          a, b, c = lines.shape
          for i in range(a):
              x = lines[i][0][0] - lines[i][0][2]
              y = lines[i][0][1] - lines[i][0][3]
              if x != 0:
                  if abs(y / x) < 1:
                      cv2.line(image, (lines[i][0][0], lines[i][0][1]), (lines[i][0][2], lines[i][0][3]), (255, 255, 255),
                               1, cv2.LINE_AA)
        except AttributeError:
          print('shape not found')

        se = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2, 2))
        gray = cv2.morphologyEx(image, cv2.MORPH_CLOSE, se)
```

the OCR part which, using PyTesseract wrapper, will detect the characters in the the image For a cleaner output create a valid chars list which will be used to compare the output chars with the one in this list. If they match, each char will be appended to a list.

```
    # OCR
    config = '-l eng --oem 1 --psm 3'
    text = pytesseract.image_to_string(gray, config=config)
    print(text)

    validChars = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S',
                  'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

    cleanText = []

    for char in text:
        if char in validChars:
            cleanText.append(char)

    plate = ''.join(cleanText)
    # print(plate)

    detectedOCR.append(plate)

    # cv2.imshow('img', gray)
    # cv2.waitKey(0)

return detectedOCR
```

3.The respective four function are being called and platelist is being printed

```
adaptiveThreshold(plates)
resize(processed)


addBorder(resized)

platesList = cleanOCR(bordered)

print(platesList)
```

5 Experimental Results

5.1 Figures

Figure 1: Detection

Figure 2: Recognition

Figure 3: Detection

Figure 4: Recognition

Figure 5: Detection

Figure 6: Recognition

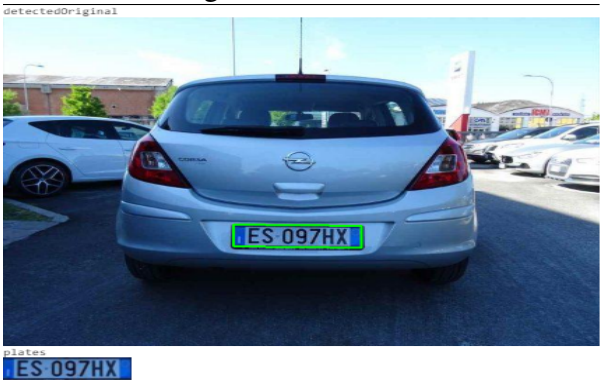


Figure 7: Detection

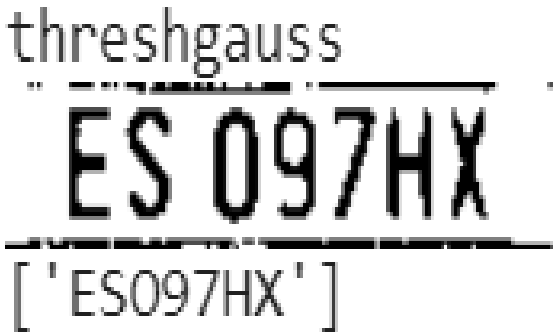


Figure 8: Recognition

Figure 9: Detection

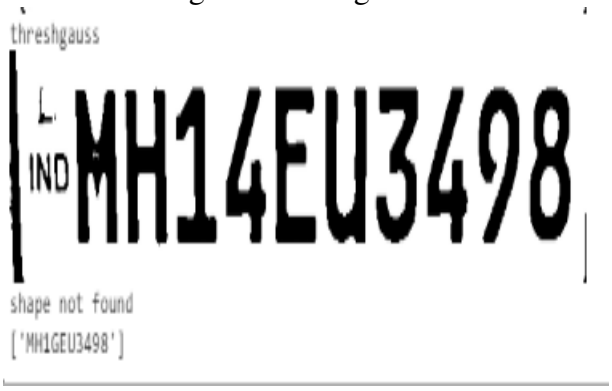


Figure 10: Recognition

5.2 Table

Table 1

Vehicle License Plate Recognition			
CNN Model	Accuracy(%)	Tesseract OCR	Accuracy(%)
CNN Model 1	88	Model 1	<95
CNN Model 2	90		
CNN Model 3	95		

This table shows the Validation Accuracy of different CNN models and Tesseract OCR model.

Table 2

Accuracy Analysis of CNN Model				
Operation	Sample	Success	Fail	Success Ratio(%)
Detection	50	44	8	88
Segmentation	44	42	2	95.45
Recognition	42	30	12	71.4

Table 3

Accuracy Analysis of Tesseract Model				
Operation	Sample	Success	Fail	Success Ratio(%)
Detection	50	44	8	88
Segmentation	44	43	1	97.72
Recognition	43	41	2	95.34

6 References

References

[1] Sweta Kumari, Leeza Gupta, Prena Gupta. Automatic License Plate Recognition Using OpenCV and Neural Network. In *International Journal of Computer Science Trends and Technology (IJCST)*, – Volume 5 Issue 3, May – Jun 2017.

[2] K.M. Sajjad . Automatic License Plate Recognition using Python and OpenCv.

[3] Prof.Amit Kukreja , Swati Bhandari, Sayali Bhatkar, Jyoti Chavda, Smita Lad. Indian Vehicle Number Plate Detection Using Image Processing In *International Research Journal of Engineering and Technology (IRJET)*, Volume: 04 Issue: 04 | Apr -2017 .

[4] Rahul R. Palekar ; Sushant U. Parab ; Dhrumil P. Parikh ; Vijaya N. Kamble. Real time license plate detection using openCV and tesseract. *2017 International Conference on Communication and Signal Processing (ICCSP)*, 6-8 April 2017, IEEE.

[5] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. In *Google Inc., Mountain View, CA*

[6] Thanh-Nga Nguyen, Duc-Dung Nguyen. A New Convolutional Architecture for Vietnamese Car Plate Recognition. In *2018 10THINTERNATIONAL CONFERENCE ON KNOWLEDGE AND SYSTEMS ENGINEERING (KSE)*, pages 312–318. IEEE, 2014.

[7] Nitish Srivastava , Geoffrey Hinton , Alex Krizhevsky , Ilya Sutskever , Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. In *Journal of Machine Learning Research 15 (2014) 1929-1958*, Submitted 11/13; Published 6/14.

[8] LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. In *Neural Computation*, 1(4):541–551, 1989.

[9] P. Sermanet, S. Chintala, and Y. LeCun. Convolutional neural networks applied to house numbers digit classification. In *International Conference on Pattern Recognition (ICPR 2012)*, 2012.