Handwritten Equation Solver using Neural Networks

PROJECT REPORT

submitted by

Adhithya T. (TRV17IT002)

Akshay T. A. (TRV17IT007)

Karun Krishnan (TRV17IT029)

Omkar R. (TRV17IT042)

in partial fulfilment for the award of degree of

Bachelor of Technology

in

Information Technology

of

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

THIRUVANANTHAPURAM

DEPARTMENT OF INFORMATION TECHNOLOGY

GOVERNMENT ENGINEERING COLLEGE BARTON HILL

THIRUVANANTHAPURAM-695 035

JULY 2021

GOVERNMENT ENGINEERING COLLEGE BARTON HILL

THIRUVANANTHAPURAM-695 035

JULY 2021

CERTIFICATE

This is to certify that the project work document entitled "Handwritten Equation Solver using Neural Networks is a bonafide record of the project work done by **Adhithya T. (TRV17IT002), Akshay T. A. (TRV17IT007), Karun Krishnan (TRV17IT029), Omkar R. (TRV17IT042)** in partial fulfilment of the requirements for the award of Degree of **Bachelor of Technology** in **Information Technology** of APJ Abdul Kalam Technological University, Thiruvananthapuram.

Prof. Haripriya A. P.	**Prof. Divya Prasad K. H.**	**Dr. Shamna H. R.**
Project Coordinator,	Project Guide	Head, Project Coordinator
Dept. of IT,	Dept. of IT,	Dept. of IT,
GECBH	GECBH	GECBH

Declaration

We hereby declare that the design of the project titled ***"Handwritten Equation Solver using Neural Networks"*** being submitted in partial fulfilment for the award of B. Tech degree is the original work carried out by us. It has not formed as part of any other thesis submitted for award of any degree or diploma, either in this or any other University.

Signature of the student:

Name : Adhithya T.
(TRV17IT002)

Signature of the student:

Name : Akshay T. A.
(TRV17IT007)

Signature of the student:

Name : Karun Krishnan
(TRV17IT029)

Signature of the student:

Name : Omkar R.
(TRV17IT042)

Acknowledgement

Primarily, we thank God Almighty for all the blessings on the successful completion of this project. A project in a complete form requires a lot of effort and guidance from many people. We would like to express our deep gratitude towards each of them who have been with us from the start to the end of this project.

*We express our sincere gratitude to **Dr. N. Vijayakumar**, Principal, Government Engineering College, Barton Hill for giving us the opportunity to do this project. We owe a great deal of gratitude towards Head of Department of Information Technology, **Dr. Shamna H. R.**, for providing us with the required facilities and her wholehearted support for the project. We express our sincere thanks and a deep sense of gratitude to our project coordinators, **Dr. Shamna H. R.** and **Prof. Haripriya A. P**. for their support and guidance. We also want to especially thank our project guide **Prof. Divya Prasad K. H.** for being kind and patient with us along with her continuous support and encouragement throughout the process.*

At last but not the least, we also thank all our parents, friends and well-wishers who stood beside and encouraged us for the success of the undertaking.

Trivandrum

11-07-2021

Adhithya T.

Akshay T. A.

Karun Krishnan

Omkar R.

Abstract

Robust handwritten character recognition is a tricky job in the area of image processing. Among all the problems, handwritten mathematical expression recognition is one of the complicated issues in the area of computer vision research. In this project a group of handwritten arithmetic equations are recognized and a solution is made for those equations. For classification of specific characters we apply Convolutional Neural Network. Each correct character detection and string operation is used for the solution of the equation. Finally, the experimental results show the great effectiveness of the proposed system.

The main objective of this project is to develop a web app which calculates a Handwritten Equation and solves it by training by handwritten digits and mathematical symbols using Convolutional Neural Network with image processing techniques to achieve an accuracy of 98%.

Table of Contents

Chapter 1

Introduction

Mathematics is broadly used in almost all areas of science, such as physics, engineering, medicine, economics, etc. Digital document analysis and understanding is the major research concern today. Handwritten mathematical expression recognition is still a challenging job to do in the area of computer vision. The primary task for the recognition of mathematical expression is to segment the character and then classify those characters. An operation that seeks to decompose an image of a sequence of characters into sub images of individual symbols is called Character segmentation.

Convolutional neural network (CNN) is one of the most used classification models in the computer vision area. In the last few years, Convolutional Neural Network learning has provided outstanding performance in the field of image classification, machine learning and pattern recognition. Above all existing models, CNN is one of the most popular models and has been providing state-of-the-art recognition accuracy for object recognition, segmentation, human activity analysis, image super resolution, scene understanding, tracking, and image captioning.

For the task of image classification CNN outperforms above all the previous classification methods. CNN extracts features from the image by a series of operations. In this work we mainly focus on the recognition of linear equations and after successful detection of linear equations we apply character string operation for the solution of those equations.

In this project we consider a single equation in the form ax+b=0 with any combination of it's parts. Our job is to recognize the handwritten equation from the image and for each successful detection finding the solution of that equation.

Feature extraction of each individual character is the most difficult part for any handwritten equation due to it's different shape and structure.

To solve this problem we apply a convolutional neural network which doesn't require any predefined feature for classification of specific character. We use the smallest number of hidden layers in order to reduce the training time with tolerable error rate.

This project is aimed at developing software which will be helpful in recognizing handwritten characters for equation solving. It engulfs the concept of neural network. One of the primary means by which computers are endowed with humanlike abilities is through the use of a neural network. Neural networks are particularly useful for solving problems that cannot be expressed as a series of steps, such as recognizing patterns, classifying them into groups, series prediction and data mining. Pattern recognition is perhaps the most common use of neural networks. The neural network is presented with a target vector and also a vector which contains the pattern information, this could be an image or hand written data. The neural network then attempts to determine if the input data matches a pattern that the neural network has memorized. A neural network trained for classification is designed to take input samples and classify them into groups.

Chapter 2

Objective

In the proposed method, we are developing a web app called ‘EQ-SOL’ where the user can give a handwritten mathematical equation as input (image) and get the corresponding answer as output.

EQ-SOL

At first, the noise from the original input image is removed by applying binarization to it. After that we use compact horizontal projection for segmenting each line of the equation from the input image. Then we consider each part of the segmented image as a full image for further process.

For each line of the equation image we then find specific characters in the form of a connected component. Each segmented character is then provided as input to the convolutional neural network model for classification of the character. The resulting character that is the output of CNN is then used for making a character string which is similar to the original equation.

Chapter 3

Requirements

Hardware Requirements

CPU

The CPU (Central Processing Unit) has often been called the brains of the PC. In CPU's the priority is given to the low-latency and it provides more effective results when processing of serial instructions are involved. CPUs are best at handling single, more complex calculations sequentially.

GPU

A GPU (graphics processing unit) is a specialized type of microprocessor. It's optimized to display graphics and do very specific computational tasks. It runs at a lower clock speed than a CPU but has many times the number of processing cores. A GPU can only do a fraction of the many operations a CPU does, but it does so with incredible speed.

Software Requirements

Python 3.7

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code. Python is a programming language that lets you work quickly and integrate systems more efficiently.

ReactJS

React is an open source JavaScript library for building user interfaces or UI components. It is maintained by Facebook and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications. However,

React is only concerned with state management and rendering that state to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

Keras

Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear & actionable error messages. It also has extensive documentation and developer guides.

Chapter 4

Dataset preparation

Preparation of the dataset is the fundamental concern for this work. Characters such as English digit, alphabet and mathematical symbols all can be well defined by their edges.

For this reason we first prepared the dataset as the most precedence given to its edges that enlighten the edges. We prepared some of the dataset by ourselves and also used a modified version of the NIST dataset which is similar to the popular MINIST dataset.

Each element of the dataset is an image of dimension 28x28, which is converted into a grayscale image using OpenCV.

Initially, the dataset consisted of many elements - each of them assigned to a corresponding folder - which was then reduced to the following elements, each having about 4000 data items for training the network:

- Numbers (0-9)
- Mathematical operators (+, -, *)
- The alphabet ‘x’ for solving linear equations

This was done to remove any unwanted detections later on, thereby improving the accuracy of the model.

During training, we gained more than 98.4% as training accuracy.

Pre-Processing

Pre-processing of the input image is the procedure which encompasses changes and modifications to the image to make it fit for recognition. The following techniques may be used for image enhancement.

1. Conversion of RGB to Gray-Scale

First of all this coloured image is transformed into a typical gray-scale image and is represented through a single matrix because the detection of characters on a coloured image is more challenging than on a gray-scale image.

2. Binarization

Binarization is the procedure of choosing a threshold value for adaptation of pixel values into 0's and 1's. In this project for horizontal projection calculation 1's represent the black pixels while 0's characterize the white pixels.

3. Segmentation

Identifying the objects or other significant information in digital images segmentation is mostly used in image processing and computer vision application,which is the process of dividing an image into multiple parts. The segmentation used is character segmentation.

3.1 Character Segmentation

Character segmentation is a procedure that looks to decompose an image of a series of characters into sub images of individual symbols. In our proposed method we use a connected component analysis method for the segmentation of specific characters from the image. Problem arises at the point of extracting the math symbol '=' which is a combination of two connected components. If two consecutive components have the minimum height and are in the same horizontal direction we can consider these two into a single component.

System Design

In this project we use Convolutional Neural Networks (CNNs) for character detection from handwritten images to solve the equation.

Convolutional Neural Networks, a branch of deep learning, has an impressive record for applications in image analysis and interpretation. A convolutional neural network convolves an input image with a defined weight matrix to extract specific image features without losing spatial arrangement information.

The main advantage of CNN compared to its predecessors is that it automatically detects the important features without any human supervision. CNN is also computationally efficient. It uses special convolution and pooling operations and performs parameter sharing. This enables CNN models to run on any device, making them universally attractive.

We evaluate these architectures to determine the best performing CNN for the classification task and aim to achieve good performance levels. We then seek to train multi-class models that enhance sensitivities for the mild or early stage classes, including various methods of data pre-processing and data augmentation to both improve test accuracy as well as increase our effective dataset sample size.

Basic Convolution Neural Network

In the basic convolutional architecture, we deploy two convolutional layers each followed by a max-pooling layer and a dropout layer followed by a flatten layer respectively. It is then followed by fully connected layers.

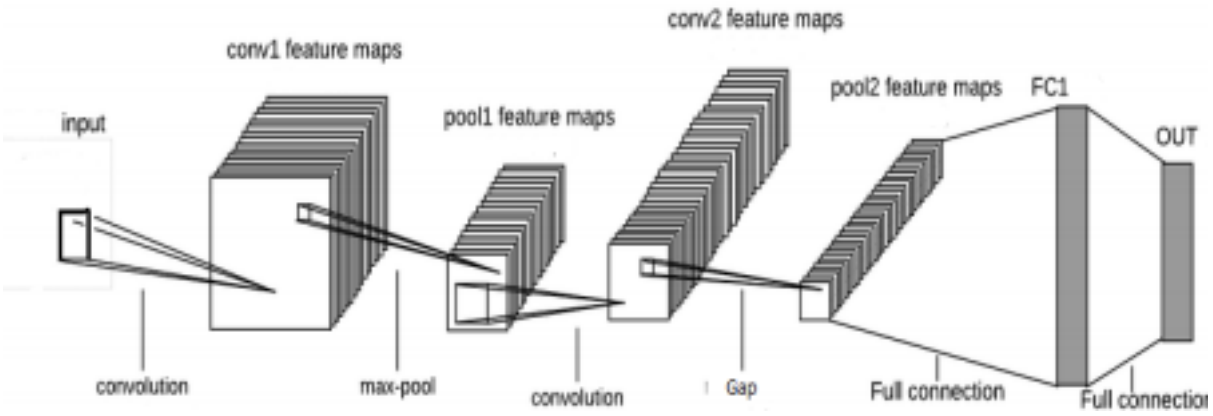


Fig 4.1 Basic CNN Architecture

Convolution

The main building block of CNN is the convolutional layer. Convolution is a mathematical operation to merge two sets of information. In our case the convolution is applied on the input data using a convolution filter to produce a feature map. The convolution layers learn such complex features by building on top of each other.

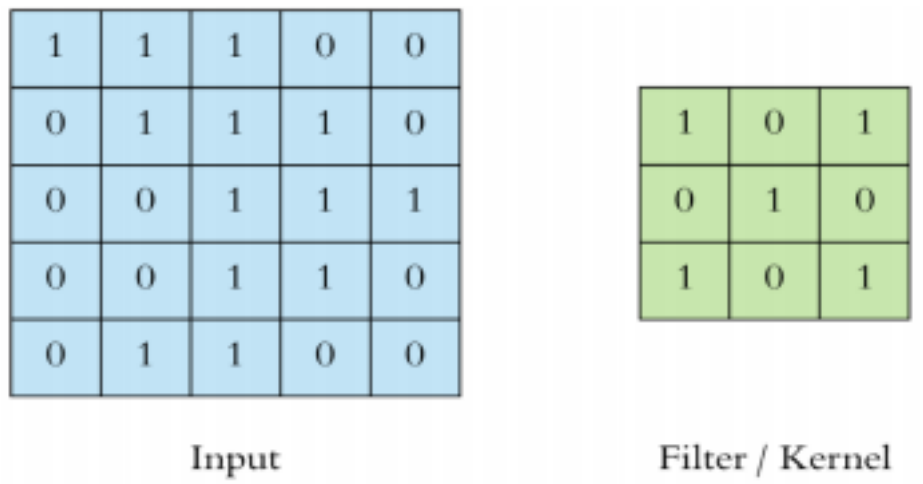


Fig 4.2 Convolution

On the left side is the input to the convolution layer, for example the input image. On the right is the convolution filter, also called the kernel. This is called a 3*3 convolutional filters due to the shape of the filter.

We perform the convolution operation by sliding this filter over the input. At every location, we do element-wise matrix multiplication and sum the result. This sum goes into the feature map. The green area where the convolution operation takes place is called the receptive field. Due to the size of the filter the receptive field is also 3x3.

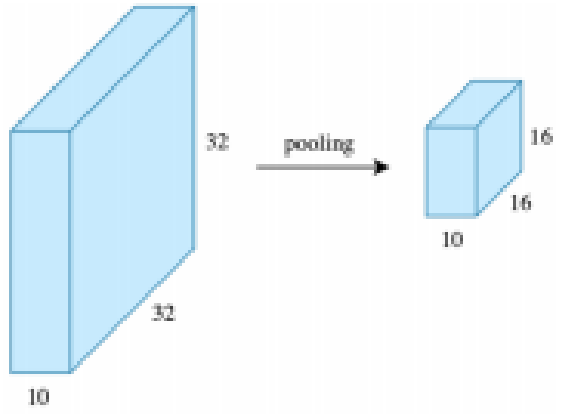
Non-Linearity

The weighted sum of its inputs is passed through an activation function to bring non-linearity in CNN. This result of convolution operation is passed through the RELU activation function. So, the values in the final feature maps are not actually the sums, but the relu function applied to them.

Pooling

Pooling is done after convolution to reduce the dimensionality. This enables us to reduce the number of parameters, which both shortens the training time and combats overfitting. Pooling layers down sample each feature map independently, reducing the height and width, keeping the depth intact. Contrary to the convolution operation, pooling has no parameters. It slides a window over its input, and simply takes the maximum value or average value in the window.

The type of pooling used in this architecture is Max-pooling. Max pooling takes the maximum value in the pooling window. Similar to a convolution, we specify the window size and stride.



Architecture

The network contains nine layers with weights.

The first layer is convolutional layer with 30 filters, kernel size 5x5,with Relu activation function which is followed by max pooling layer with size (2,2). The output of the max pooling layer is given as an input to another convolutional layer of 15 filters, kernel size 3x3 and the activation function Relu, which is followed by another max pooling layer of size (2,2).

A dropout layer and flatten layer is added before connecting to fully connected layers and the remaining three are fully-connected. The output of the last fully-connected layer is fed to a SoftMax which produces a distribution over the class labels. The network maximizes the multinomial logistic regression objective, which is equivalent to maximizing the average across training cases of the log-probability of the correct label under the prediction distribution.

In this research 5x5 filter is used at the convolutional layer after the convolution of the input image with the filter for each input image a 28×28 feature vector is produced. It is concordant to apply a nonlinear layer (or activation layer) instantly after conv layer. The persistence of this layer is to introduce nonlinearity to a system. To perform

down-sampling by separating the input into rectangular pooling regions, and figuring the maximum of each region a max pooling layer is used.

Pooling layers perform down-sampling operations. Calculation of the output size, O of a pooling layer with input size, I Pooling filter size, F Padding, P and stride, S has done by

$$O = (I - F + 2 \times P) / S + 1$$

In our method we use 2×2 pool size and the output of the pooling layer is 12×12. Random selection of input elements to zero with a given probability is done in a dropout layer. It is a simple way to prevent overfitting in the neural network. Overfitting is a stern difficulty in such networks.

Large networks are also sluggish to use, making it more challenging to contract with overfitting by combining the predictions of many different large neural nets at the test time.

The technique for addressing this problem is Dropout. The core idea is to randomly drop units (along with their connections) from the neural network during training. In our proposed method we use the probability 0.3 at the dropout layer during the training. After the convolutional and down-sampling layers one or more fully connected layers is used. The layer in which the neurons associate to all the neurons in the previous layer is fully connected.

Fully connected layers combine all the features learned by the previous layers through the image to recognize the larger patterns. To classify the images the last fully connected layer combines the features. Hence, the Output parameter in the last fully connected layer is equal to the number of classes in the target data.

In our work, the output size is 14, corresponding to the 14 classes. An activation function, Softmax, normalizes the output of the fully connected layer and is also used in the convolutional neural network model.

The softmax layer output comprises positive numbers that sum to one, in the nest that can be used as the classification probabilities at the classification layer. At the final layer, the classification layer uses the probabilities given by the softmax activation function to find the input image classes and find the loss by comparing it with the preassigned ground truth classes.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 24, 24, 30)        780
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 30)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 15)        4065
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 15)          0
_________________________________________________________________
dropout (Dropout)            (None, 5, 5, 15)          0
_________________________________________________________________
flatten (Flatten)            (None, 375)               0
_________________________________________________________________
dense (Dense)                (None, 136)               51136
_________________________________________________________________
dense_1 (Dense)              (None, 49)                6713
_________________________________________________________________
dense_2 (Dense)              (None, 14)                700
=================================================================
Total params: 63,394
Trainable params: 63,394
Non-trainable params: 0
```

Reducing Overfitting

The neural network architecture deployed here has a large number of parameters. Hence, it becomes difficult to learn many parameters without considerable overfitting. To avoid this overfitting, we use:

a) Dropout

Dropout is by far the most popular regularization technique for deep neural networks. It is used to prevent overfitting and the idea is very simple. During training time, at each iteration, a neuron is temporarily “dropped” or disabled with probability p. This means all the inputs and outputs to this neuron will be disabled at the current iteration.

The dropped-out neurons are resampled with probability p at every training step, so a dropped-out neuron at one step can be active at the next one. The hyperparameter p is called the dropout-rate and in this architecture, we have used a dropout-rate of 0.2, corresponding to 20% of the neurons being dropped out.

The dropout prevents the network from being too dependent on a small number of neurons, and forces every neuron to be able to operate independently.

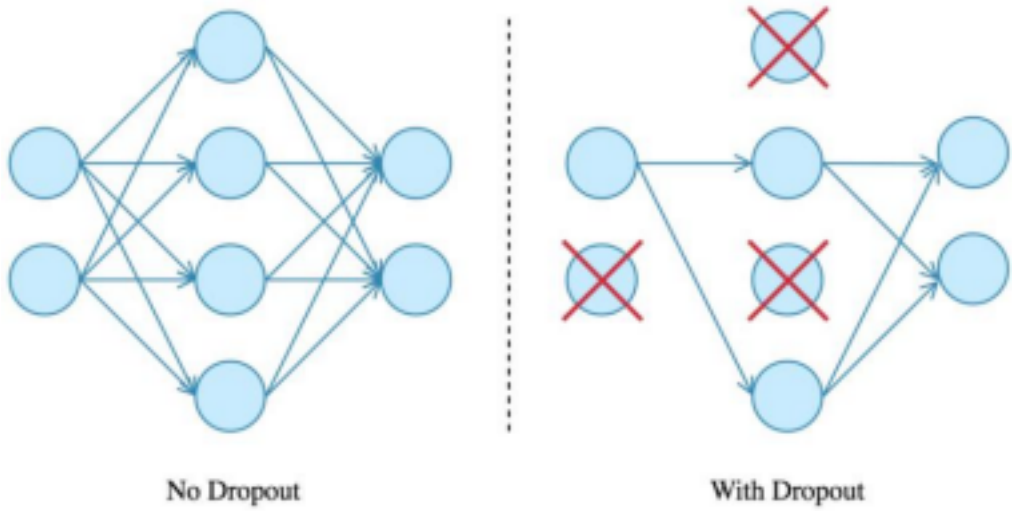


Fig 4.5 Dropout

Dropout can be applied to input or hidden layer nodes but not the output nodes. The edges in and out of the dropped out nodes are disabled.The nodes which were dropped out change at each training step.The dropout is not applied during test time after the network is trained,it is only applied during training.

b) ReLU Non-Linearity

In a neural network, the activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input.

The rectified linear activation function is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. In terms of training time with gradient descent, saturating nonlinearities are much slower than the non-saturating nonlinearity f(x) = max(0, x).

The neurons with such nonlinearity are referred to as Rectified Linear Units (ReLUs). Deep convolutional neural networks with ReLUs train several times faster than their equivalents with tanh units.

However, on this dataset the primary concern is preventing overfitting, so the effect that is observed is different from the accelerated ability to fit the training set reported when using

ReLUs. Faster learning has a great influence on the performance of large models trained on large datasets.

Chapter 5

Training data using CNN

Since convolutional neural networks work on two-dimensional data and our dataset is in the form of 785 by 1. Therefore, we need to reshape it. Firstly, assign the labels column in our dataset to variable y_train. Then drop the labels column from the dataset and then reshape the dataset to 28 by 28. Now, the dataset is prepared and ready.

a. Building Convolutional Neural Network

For building the CNN, import the necessary libraries

```
np.random.seed(1212)
from keras.models import Model
from keras.layers import *
from keras import optimizers
from keras.layers import Input,Dense
from keras.models import Sequential
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
```

b. Creating the model

The model selected is the Keras Sequential model, which is appropriate in applications where there is exactly one input and one output.

```
model=Sequential()
model.add(Conv2D(30,(5,5),input_shape=(28,28,1),activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(15,(3,3),activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(136,activation="relu"))
model.add(Dense(49,activation="relu"))
model.add(Dense(14,activation="softmax"))
```

c. Fitting Model

The training model is compiled using the Adam optimiser and the Categorical Cross Entropy loss function.

```
model.compile(loss="categorical_crossentropy",optimizer="adam",metrics=["accuracy"])
model.fit(np.array(l),cat,shuffle=True,epochs=10)
```

d. Solving the equation

Simple expressions are solved using the eval() function of Python while linear equations are solved using the Sympy Python library.

```
if '*' in s:
    t = s.replace('x','')
    s=t

if "x" in s:
    try:
        equation = s.replace('--','=')
        a = []
        for i in range(0,len(equation)):
            if i==0:
                continue
            if equation[i] == 'x':
                if equation[i-1] == '+' or equation[i-1] == '-' or equation[i-1] == '=':
                    continue
                else:
                    a.append(i)

        counts = 0
        for i in a:
            if counts == 0:
                counts=1
                equation = equation[:i] + '*' + equation[i:]
            else:
                i=i+counts
                equation = equation[:i] + '*' + equation[i:]
                counts+=1

        lhs =  parse_expr(equation.split("=")[0])
        rhs =  parse_expr(equation.split("=")[1])
        solution = solve(lhs-rhs)
        decSol = round(solution[0]*1.0,2)

        print(solution[0]) #fractional solution
        print(decSol) #decimal solution
    except:
        print("invalid equation")
else:
    print(eval(s))
```

Chapter 6

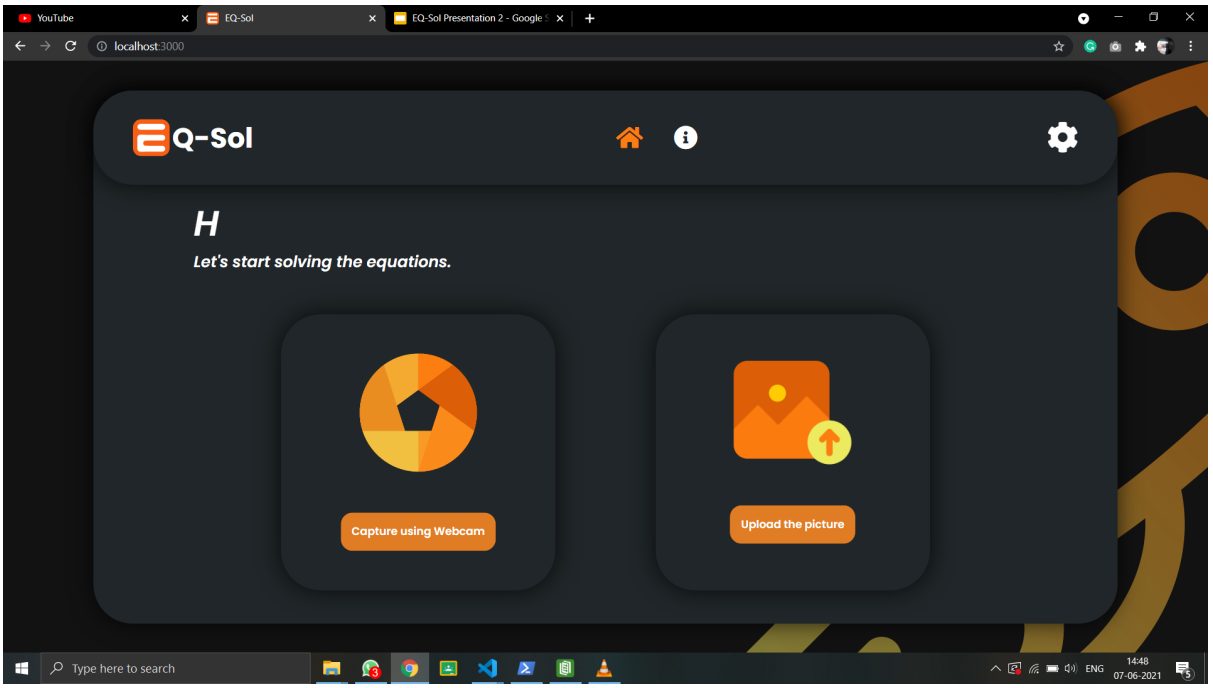
Front-End

On the Front-End side, we used the **ReactJS** framework. Also used here are NPM modules like react-router-dom, react-dropzone, reactstrap, react-image-crop and react-webcam.

The frontend has two features:

1. **Camera**: Here we can click a picture using from the webcam and proceed for upload
2. **Upload a pic**: Here we upload a picture from our computer and proceeds for upload

Also, there is an option to crop the image for selecting out the equation part of the image and proceed with that cropped image. It enhances the User Experience.



Interfacing

Interfacing denotes integrating Frontend with the created **ML Model** in the Backend. We have used **Flask** for implementing it along with modules like flask-cors for making the **RESTful API** that supports any frontend framework. It generates an API which could be used by the frontend for data sending and receiving from the backend using **standard HTTP** protocol.

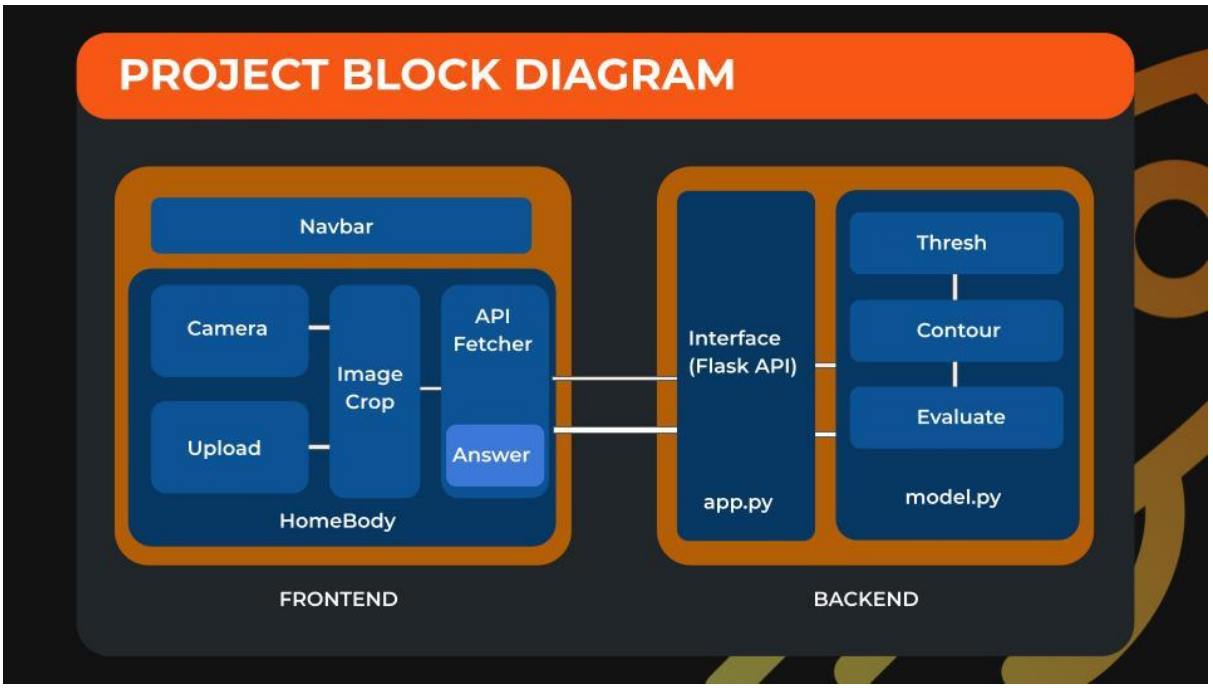
From the frontend part, we use the fetch function to fetch the API by **POST** method with image as a **base64** encoded string to backend. In flask we declare the flask app with method POST. First, access the parameter using request and convert that base64 image string to normal image and save it in localhost directory. Then the url of the created image is passed to the ML model function.

When it reaches the ML model, the backend code works and returns the result as a **JSON** object which contains the detected equation and its corresponding answer. Same result is passed to frontend by returning the result from the ML model function. In frontend the fetch function returns an object from the backend which is converted to normal form and displayed by accessing its attributes.

Chapter 7

Working

The website offers two options to scan the handwritten equation. It can either take a live photo using the webcam or a pic can be uploaded from the computer. It is then sent to the backend via Flask API. The characters/digits are then recognized and extracted. It undergoes several procedures like converting the image into grayscale, contour detection, reshaping etc and is fed to the model.It is then evaluated and sent to the frontend via Flask API where the answer, along with the type of equation, is displayed.



Chapter 8

Model evaluation

Training accuracy

The accuracy of our model based on examples it was constructed on : (using model.fit() function).

```
model.fit(np.array(l),cat,shuffle=True,epochs=10)
```

```
Epoch 1/10
1626/1626 [==============================] - 35s 16ms/step - loss: 1.7619 - accuracy: 0.6431
Epoch 2/10
1626/1626 [==============================] - 28s 17ms/step - loss: 0.2238 - accuracy: 0.9330
Epoch 3/10
1626/1626 [==============================] - 30s 18ms/step - loss: 0.1399 - accuracy: 0.9578
Epoch 4/10
1626/1626 [==============================] - 29s 18ms/step - loss: 0.1091 - accuracy: 0.9671
Epoch 5/10
1626/1626 [==============================] - 29s 18ms/step - loss: 0.0899 - accuracy: 0.9733
Epoch 6/10
1626/1626 [==============================] - 29s 18ms/step - loss: 0.0744 - accuracy: 0.9765
Epoch 7/10
1626/1626 [==============================] - 30s 18ms/step - loss: 0.0690 - accuracy: 0.9793
Epoch 8/10
1626/1626 [==============================] - 29s 18ms/step - loss: 0.0626 - accuracy: 0.9803
Epoch 9/10
1626/1626 [==============================] - 29s 18ms/step - loss: 0.0574 - accuracy: 0.9823
Epoch 10/10
1626/1626 [==============================] - 30s 19ms/step - loss: 0.0503 - accuracy: 0.9846
<tensorflow.python.keras.callbacks.History at 0x216a8787580>
```

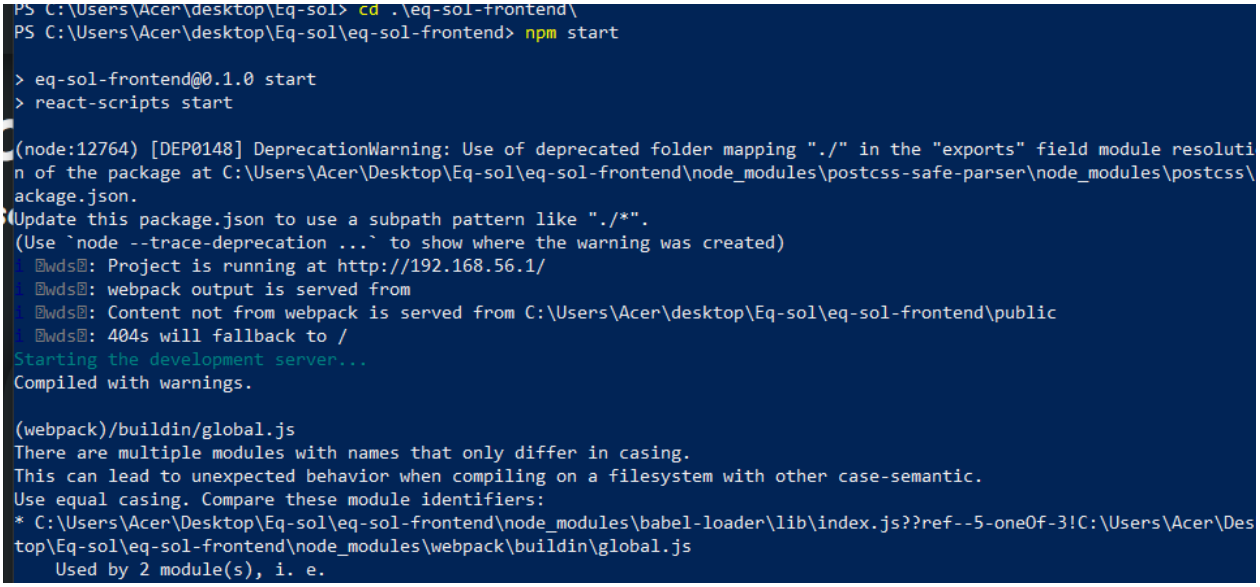
Testing Accuracy

We divided the dataset into two - training and testing dataset in the ratio 7:3. On evaluating, we got the accuracy as 98.4% for the testing dataset.

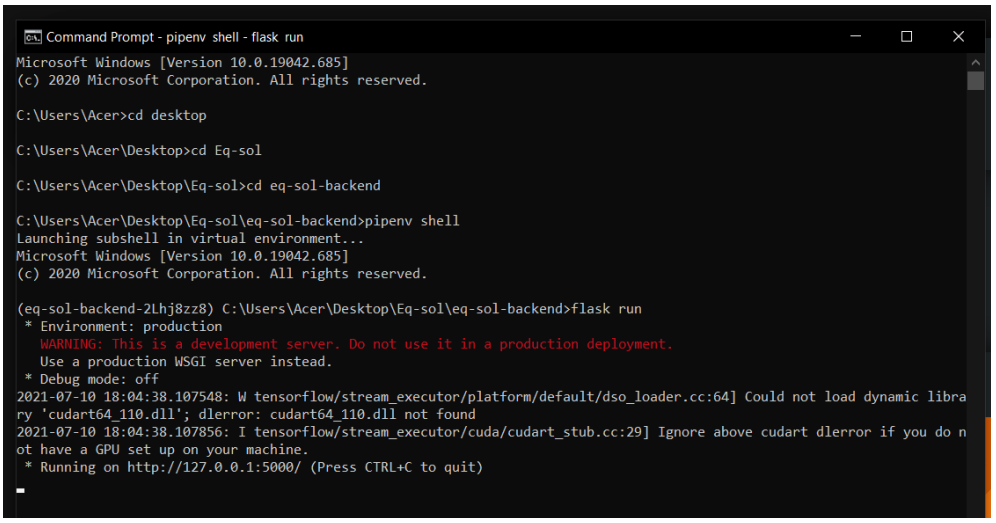
```
In [23]: model.evaluate(np.array(m),tac)
         432/432 [==============================] - 3s 6ms/step - loss: 0.0720 - accuracy: 0.9841
Out[23]: [0.0720311626791954, 0.9840545058250427]
```

Experimental Results

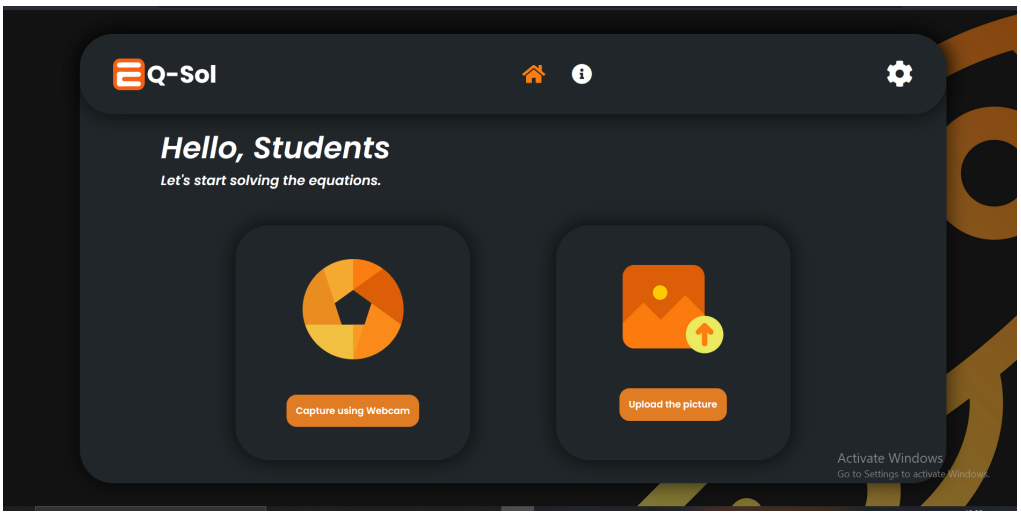
We have added screenshots which show the working of the application. It is able to detect the equation in the picture given both as upload a picture or capture a picture through the webcam and also able to show the solution of it.



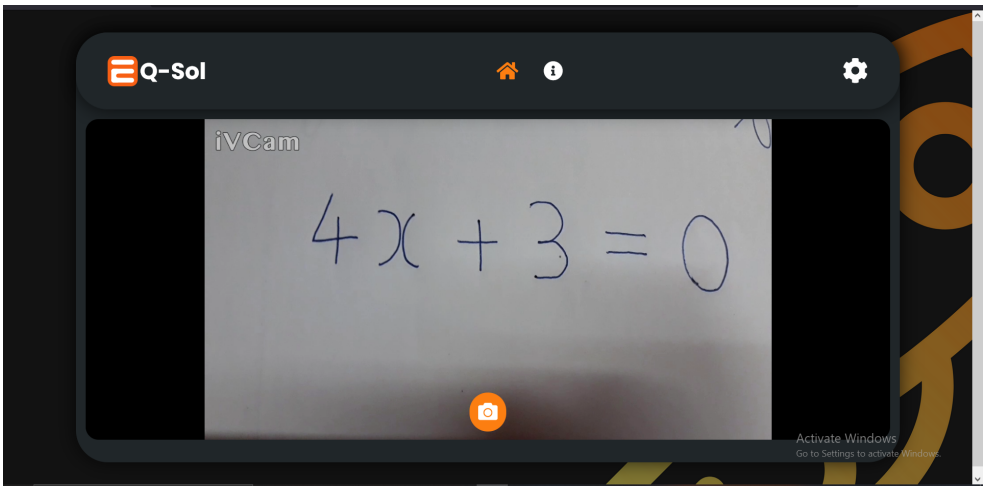
Started the Front-End(React) localhost using command ‘npm start’



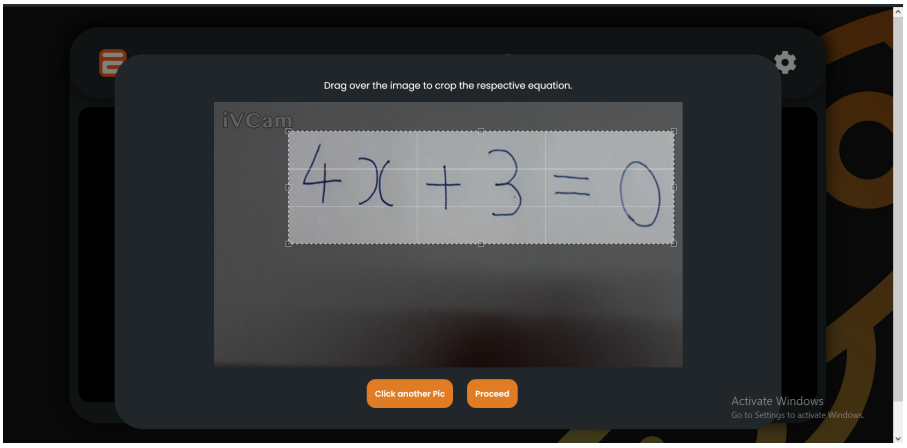
Started the Back-End Server(Flask) using command ‘flask run’



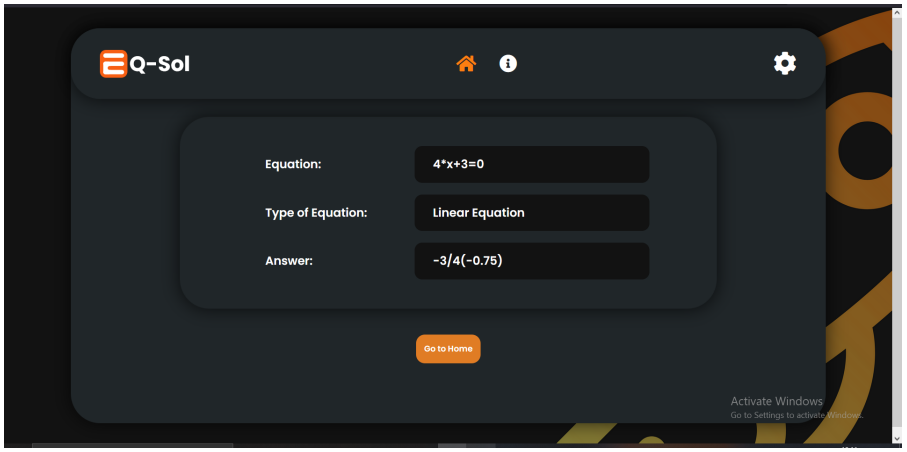
Home page



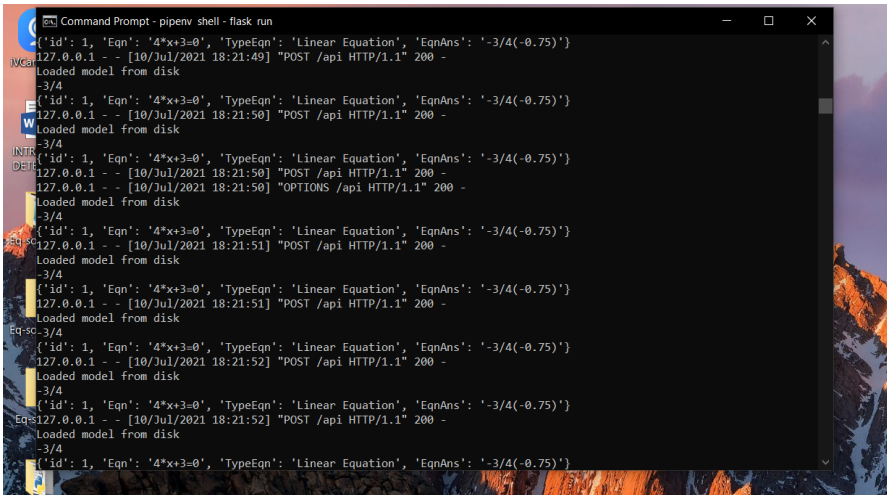
Captured image through camera



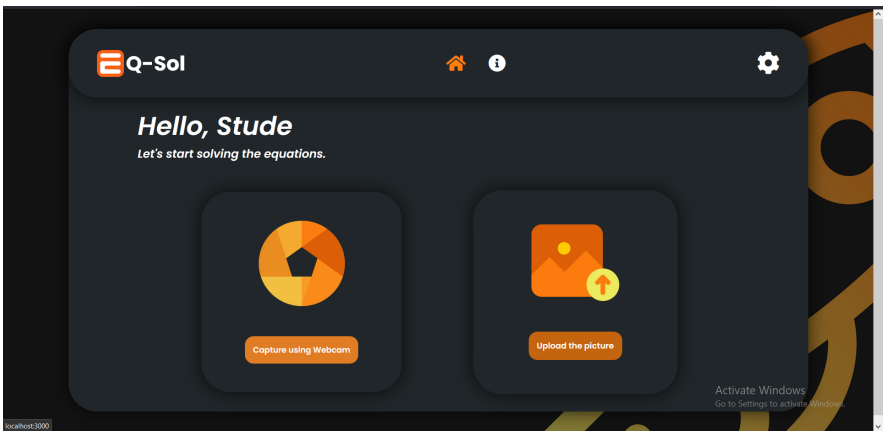
Cropped the image



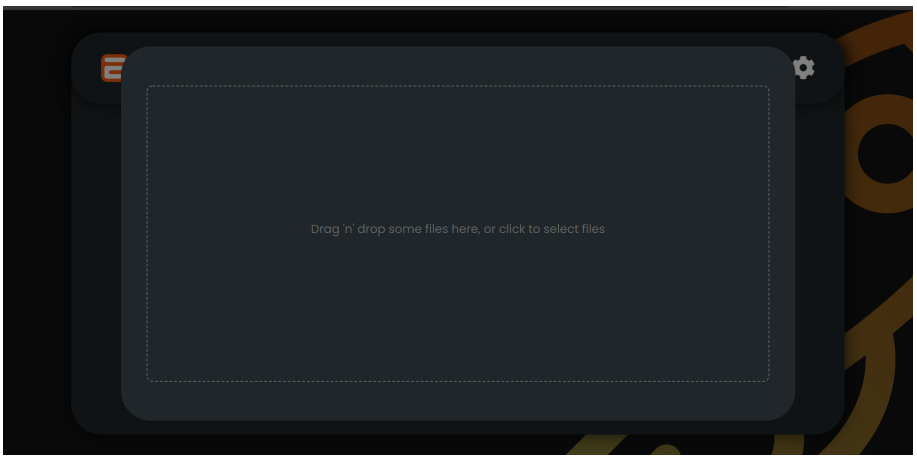
Answer



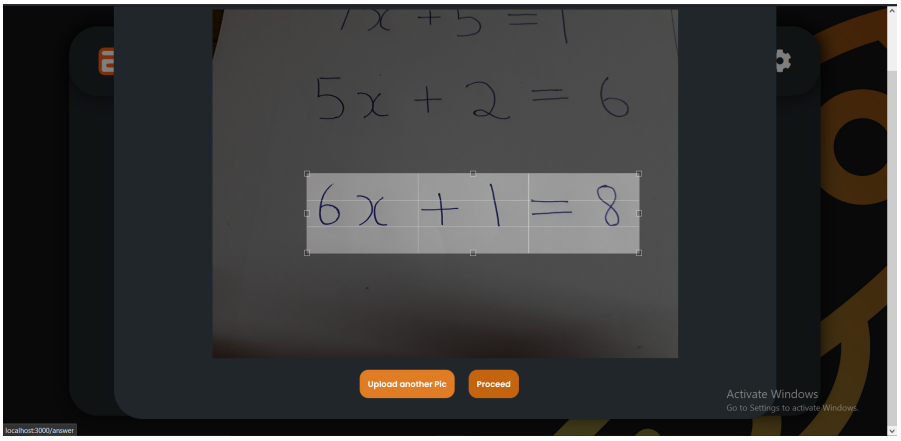
Answer in command prompt



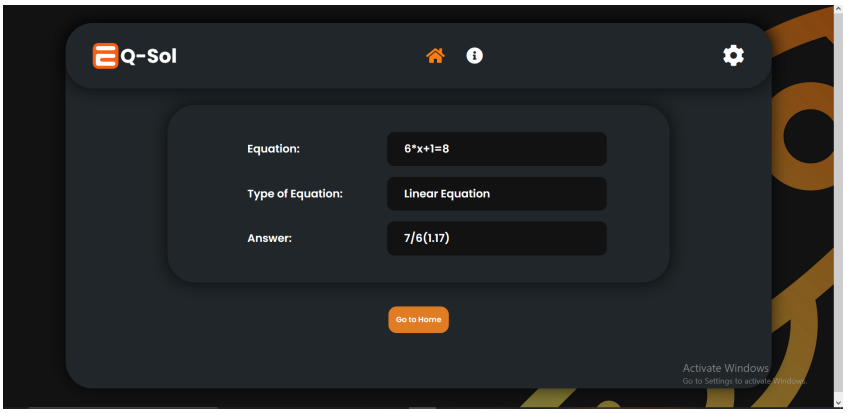
Home Page(Clicked Upload the picture button)



Drag and Drop the image



Cropped the image



Answer

Chapter 9

Conclusion

In this paper we are focusing on recognizing handwritten Mathematical expressions and equations. Feature extraction was the most complicated part of classification. Convolutional Neural Network the most powerful classification model is used in the classification part. Finally we obtained a state of art performance in the detection phases and also in the solution phases. To simplify the math, the main task is the feature extraction from the image and recognition with the help of the CNN model. Any person can easily use this as the process is easy and does not require any advanced knowledge of the technology.

In this paper we mainly focused on recognizing handwritten Mathematical expressions and equations. Projection analysis specially compact horizontal projection is used for the segmentation of each line of quadratics. Connected components which have a very high success rate are used for character segmentation.

Moreover with some predefined features about handwriting it is difficult to recognize handwriting. Convolutional Neural Network the most powerful classification model is used in the classification part.

Appendix

ApiFetcher.js (In Frontend used for fetching Backend API and displaying result)

```
const blobToBase64 = blob => {
  const reader = new FileReader();
  reader.readAsDataURL(blob);
  return new Promise(resolve => {
    reader.onloadend = () => {
      resolve(reader.result);
    };
  });
};
export default function ApiFetcher(props) {
    const [error,Seterror] = useState(0);
    const [isLoaded,SetisLoaded] = useState(false);
    const [test, SetTest] = useState(0);
    const [EqnImg,SetEqnImg] = useState(0);
    const [Ans,SetAns] = useState(0);
    var a = {
      Eqn:"5x + 3 = 23",
      TypeEqn:"Linear Equation",
      EqnAns:"x = 4"
      };
    useEffect(() => {
      let img = (props.location.state.image);
      SetEqnImg(img.split(',')[1])
      // console.log(img)
      fetch('http://127.0.0.1:5000/api',{
        method:'POST',
        headers:{
          'Content-Type': 'application/json'
        },
        body:JSON.stringify({
          image:EqnImg
        })
      })
        .then(Response => Response.json())
        .then(
          (data) => {
            SetisLoaded(true);
```

```
            SetAns(data);

        }
         )
  });
    if (error) {
        return <div className=" container text-center home-para"
style={{marginTop:"50px",fontSize:"40px"}}>Error:
{error.message}</div>;
      } else if (!isLoaded) {
        return(
            <div className="container" >
                <div
class="lds-ring"><div></div><div></div><div></div><div></div></div
></div>
        )
      } else {
        return (
            <div>
                <Answer Ans={Ans}/>
                </div>
        );
      }
}
```

App.py (In backend used to set API link, accept parameters and send image to model.py

```
app = Flask(__name__)
api = Api(app)

CORS(app)

image_req_args = reqparse.RequestParser()
image_req_args.add_argument("image", type=str)


@app.route('/api', methods=['POST'])
def index():
```

```
    args = image_req_args.parse_args()
    image = BytesIO(base64.urlsafe_b64decode(args['image']))
    Result  = model(image)

    return Result
```

Model.py (here the)

```
def processor(url):
    json_file = open('model.json', 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)
    # load weights into new model
    loaded_model.load_weights("model.h5")
    print("Loaded model from disk")

    img = cv2.imread(url,cv2.IMREAD_GRAYSCALE)
    #kernel = np.ones((3,3),np.uint8)
    # cv2.imshow("w",img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    #erosion = cv2.erode(img,kernel,iterations = 3)
    #dilation = cv2.dilate(img,kernel,iterations = 1)
    #img=dilation
    if img is not None:
        #images.append(img)
        img=~img
        ret,thresh=cv2.threshold(img,127,255,cv2.THRESH_BINARY)

ctrs,ret=cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SI
MPLE)
        cnt=sorted(ctrs, key=lambda ctr: cv2.boundingRect(ctr)[0])
        w=int(28)
        h=int(28)
        train_data=[]
        #print(len(cnt))
        rects=[]
        for c in cnt :
            x,y,w,h= cv2.boundingRect(c)
            rect=[x,y,w,h]
```

```
            rects.append(rect)
        #print(rects)
        bool_rect=[]
        for r in rects:
            l=[]
            for rec in rects:
                flag=0
                if rec!=r:
                    if r[0]<(rec[0]+rec[2]+10) and
rec[0]<(r[0]+r[2]+10) and r[1]<(rec[1]+rec[3]+10) and
rec[1]<(r[1]+r[3]+10):
                        flag=1
                    l.append(flag)
                if rec==r:
                    l.append(0)
            bool_rect.append(l)
        #print(bool_rect)
        dump_rect=[]
        for i in range(0,len(cnt)):
            for j in range(0,len(cnt)):
                if bool_rect[i][j]==1:
                    area1=rects[i][2]*rects[i][3]
                    area2=rects[j][2]*rects[j][3]
                    if(area1==min(area1,area2)):
                        dump_rect.append(rects[i])
        #print(len(dump_rect))
        final_rect=[i for i in rects if i not in dump_rect]
        #print(final_rect)
        for r in final_rect:
            x=r[0]
            y=r[1]
            w=r[2]
            h=r[3]
            im_crop =thresh[y:y+h+10,x:x+w+10]


            im_resize = cv2.resize(im_crop,(28,28))
            # cv2.imshow("work",im_resize)
            cv2.waitKey(0)
            cv2.destroyAllWindows()

            im_resize=np.reshape(im_resize,(1,28,28))
```

```
            train_data.append(im_resize)
    s=""
    for i in range(len(train_data)):
        train_data[i]=np.array(train_data[i])
        train_data[i]=train_data[i].reshape(1,28,28,1)
        result=loaded_model.predict_classes(train_data[i])
        if(result[0]==10):
            s=s+"-"
        if(result[0]==11):
            s=s+"+"
        if(result[0]==0):
            s=s+"0"
        if(result[0]==1):
            s=s+"1"
        if(result[0]==2):
            s=s+"2"
        if(result[0]==3):
            s=s+"3"
        if(result[0]==4):
            s=s+"4"
        if(result[0]==5):
            s=s+"5"
        if(result[0]==6):
            s=s+"6"
        if(result[0]==7):
            s=s+"7"
        if(result[0]==8):
            s=s+"8"
        if(result[0]==9):
            s=s+"9"
        if(result[0]==12):
            s=s+"*"
        if(result[0]==13):
            s=s+"x"
    #print(s)
    equation  = s.replace('x','*x')
    if '*' in s:
        t = s.replace('x','')
        s=t
    if "x" in equation:
        try:
            equation = s.replace('--','=')
```

```
            a = []
            for i in range(0,len(equation)):
                if i==0:
                    continue
                if equation[i] == 'x':
                    if equation[i-1] == '+' or equation[i-1] ==
'-' or equation[i-1] == '=':
                        continue
                    else:
                        a.append(i)

            counts = 0
            for i in a:
                if counts == 0:
                    counts=1
                    equation = equation[:i] + '*' + equation[i:]
                else:
                    i=i+counts
                    equation = equation[:i] + '*' + equation[i:]
                    counts+=1

            lhs =  parse_expr(equation.split("=")[0])
            rhs =  parse_expr(equation.split("=")[1])
            solution = solve(lhs-rhs)
            decSol =
str("{:.2f}".format(round(solution[0]*1.0,2)))

            print(solution[0])
            ans = (str(solution[0]) + "(" + decSol + ")")
            eqt = "Linear Equation"
        except:
            print("invalid equation")
            ans = "Invalid equation"
            eqt = "Invalid Equation"
    else:
        print(eval(equation))
        ans = eval(equation)
        eqt = "Simple Expression"

    return { 'id': 1,
        'Eqn':equation,
       'TypeEqn':eqt,
```

```
        'EqnAns':ans
        }



def model(image):

    # Image.open(image).save('input.png')
    # load json and create model
    Image.open(image).save('input.png')
    Result = processor('input.png')
    # img = stringToRGB(image)
    return Result
```

References

- https://www.researchgate.net/publication/326710549_Recognition_and_Solution_for_Handwritten_Equation_Using_Convolutional_Neural_Network
- https://www.nist.gov/itl/products-and-services/emnist-dataset
- https://aihubprojects.com/handwriting-recognition-using-cnn-ai-projects/
- https://www.geeksforgeeks.org/handwritten-equation-solver-in-python/
- Mandal, S Shahnawazuddin, Rohit Sinha, S. R. Mahadeva Prasanna and Suresh Sundaram, "Exploring Sparse Representation for Improved Online Handwriting Recognition", *2018 16th International Conference on Frontiers in Handwriting Recognition*, 2018.
- S.E Benita Galaxy and S. Selvin Ebenzer, "Enhancement of segmentation and zoning to improve the accuracy of handwritten character recognition", *International Conference on Electrical Electronics and Optimization Techniques (ICEEOT)*, 2016.
- R. Sethi and I. Kaushik, "Hand Written Digit Recognition using Machine Learning," *2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT)*, 2020, pp. 49-54, doi: 10.1109/CSNT48778.2020.9115746.
- Catherine Lu and Karanveer Mohan, "Recognition of Online Handwritten Mathematical Expressions Using Convolutional Neural Networks", *cs231n project report stanford*, 2015.
- Martin Riedmiller and Heinrich Braun, "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm", *IEEE international conference on neural networks*, 1993.
- Braun Riedmiller, "A direct adaptive method for faster backpropagation learning: The rprop algorithm", *Neural Networks*, 1993.
- Ahmad-Montaser Awal, Harold Mouchère and Christian Viard-Gaudin, "Towards Handwritten Mathematical Expression Recognition", *2009 10th International Conference on Document Analysis and Recognition*.
- Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan

Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, vol. 15, pp. 1929-1958, 2014.

- S. N. Shuvo, F. Hasan, S. A. Hossain and S. Abujar, "Handwritten Polynomial Equation Recognition and Simplification Using Convolutional Neural Network," *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2020, pp. 1-6, doi: 10.1109/ICCCNT49239.2020.9225587.