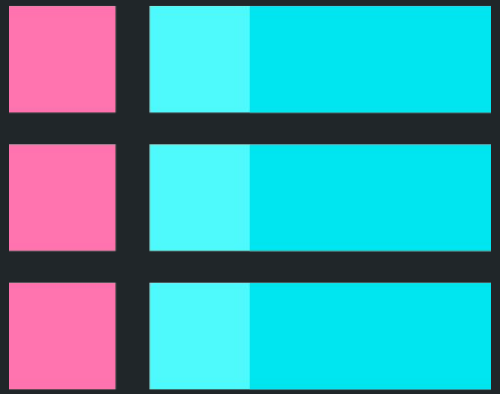
BATCH #5

Q-Sol

As the name suggests: “an Equation Solver”

Table of Contents

1. Meet our Team
2. What's our Project
3. Requirements
4. UML Diagram
5. Block Diagram
6. Front End
7. Interfacing
8. Back End
9. Live Demonstration
10. Conclusion
11. References



Meet the team

Guide

Divya Prasad

Omkar R

Akshay T.A

Karun Krishnan

Adhithya T

What’s the Project

- EQ-sol is a Web Application where the user scans handwritten mathematical equations and the corresponding solutions are displayed.

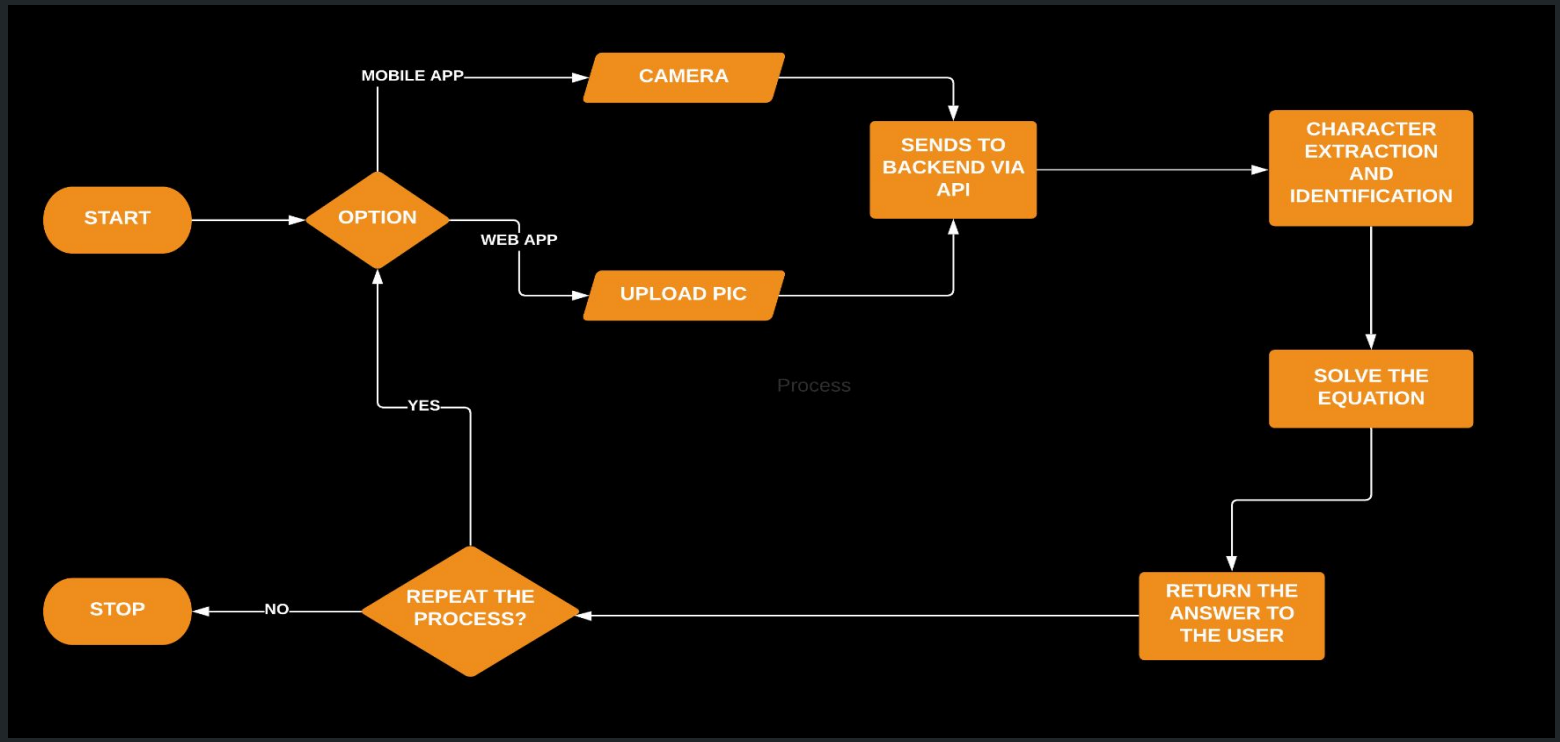
- The features include:
  a. Simple expressions supporting operations:
     - Addition
     - Subtraction
     - Multiplication
  b. Linear equations in the form:
     - ax + b = c
     - Equation on both hand sides

Requirements

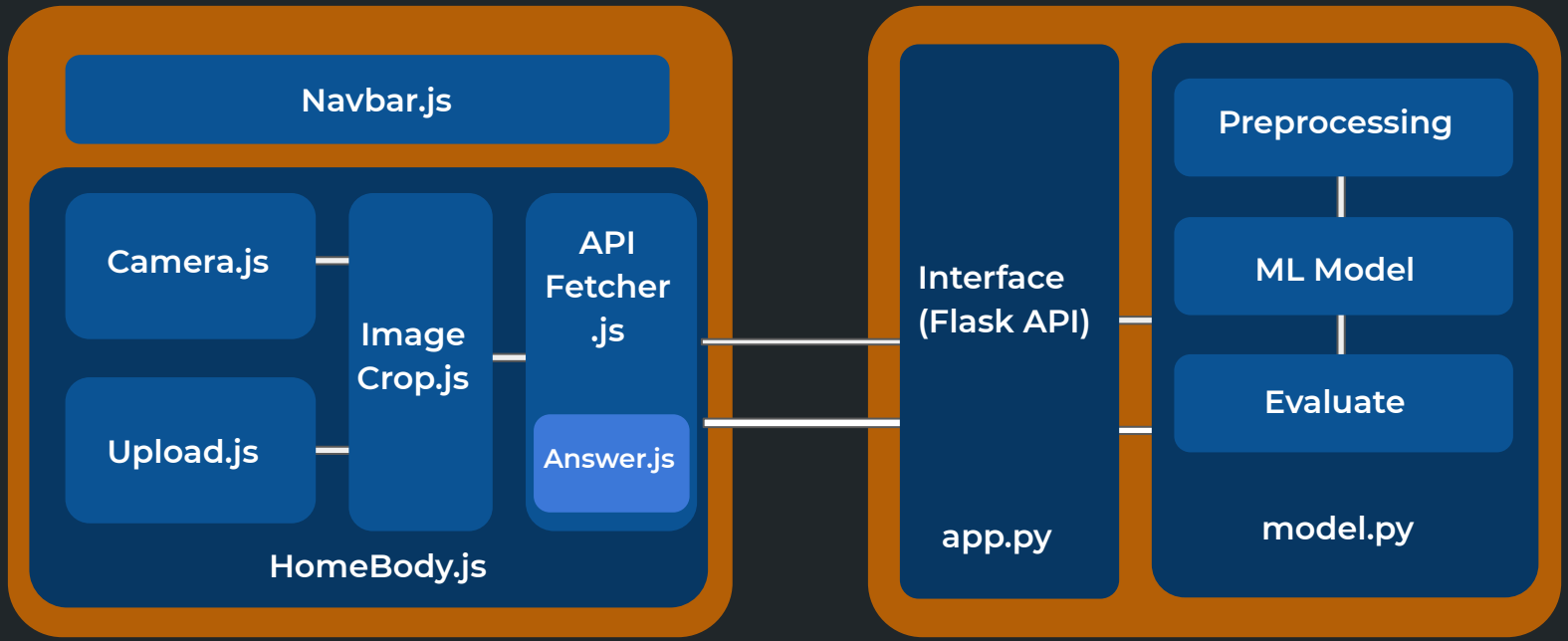
- **Hardware Requirements**
  - **CPU:** Intel i5 8th gen
  - **GPU:** Nvidia GeForce MX 150(2GB Ram)
  - **RAM:** 8GB

- **Software Requirements**
  - **OS:** Windows 10
  - **Language:** JavaScript (Front-End)
    Python 3.7 (Interfacing and Backend)
  - **Libraries:**
    - **Front-End:** react, react-router-dom,reactstrap, react-webcam react-dropzone, react-image-crop
    - **Interface:** flask, flask-cors, flask-restful
    - **Back-End:** opencv-python, keras, sympy

UML Diagram

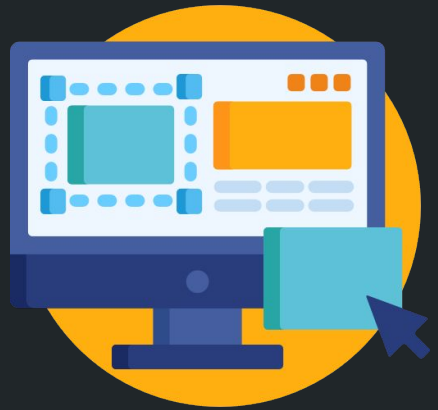

MOBILE APP
CAMERA
SENDS TO BACKEND VIA API
CHARACTER EXTRACTION AND IDENTIFICATION
START
OPTION
WEB APP
UPLOAD PIC
Process
SOLVE THE EQUATION
YES
STOP
NO
REPEAT THE PROCESS?
RETURN THE ANSWER TO THE USER

Block Diagram



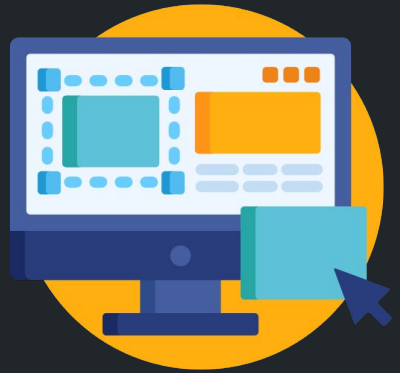
Front End

- The Front-End is developed using React and it’s modules.
- On the home page, there are two options for input
  - Capture using Webcam: go to Camera.js
  - Upload the picture: go to Upload.js
- There is an additional option to crop the image for a cleaner input using react-image-crop



Front End(Contd.)

- Upload.js file uses the ‘react-dropzone’ library to handle the feature of uploading an image via a simple drag and drop interface.
- On selecting the webcam, control goes to Camera.js that uses the ‘react-webcam’ library
- ApiFetcher.js makes API calls and sends the cropped image as a request to the back end. The corresponding response - the solution - is given by Answer.js as a JSON object



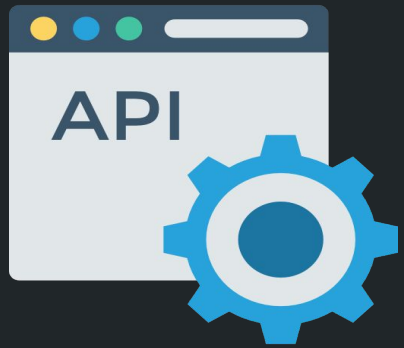
Front End(Home Page)

Interfacing

- The Front-End is connected with the Back-End here. The function fetch('api_url, parameters') to send a POST request to the Back-End using the url and the image as parameters

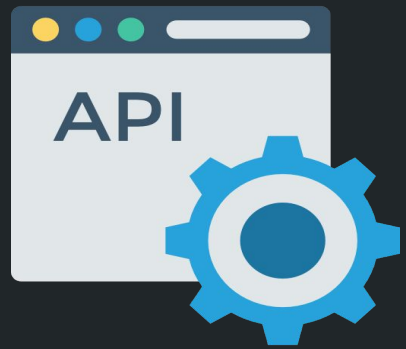
- In the Back-End, Flask API is used along with the 'flask-cors' module, which supports different platforms like React, Angular etc. Arguments are passed using 'flask-restful'.

- The flask API configuration is done in the file app.py such as CORS and routing.



Interfacing(Contd.)

- In app.py, the image received is of type base64 string. This is converted to binary format and is sent as a parameter to model.
- Inside model( ) in model.py, the image is saved in backend server as ‘input.png’. Url of this image is sent as parameter of another function processor(image) which is the main equation solver.
- The return value ie. solutions is passed as return value, of model() and in app.py response of model() as response of the API to Front-End.



Interfacing(Contd.)

ApiFetcher.js

```
fetch('http://127.0.0.1:5000/api',{ //fetch() with backend api as parameter
  method:'POST', //method 'POST'
  headers:{
    'Content-Type': 'application/json'
  },
  body:JSON.stringify({   //body is the object going with api request
    image:EqnImg          //image EqnImg in base64 string attached
  })
})
  .then(Response => Response.json()) //response converted to json
  .then(
    (data) => {
      SetisLoaded(true);
      SetAns(data);       //SetAns assigns content of data to react state(a variable)
    }
  )
```

Interfacing(Contd.)

app.py

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

CORS(app)     #CORS module added to app

image_req_args = reqparse.RequestParser()       #access request parameter from frontend
image_req_args.add_argument("image", type=str)  #add image parameter


@app.route('/api', methods=['POST'])            #set api name and method
def index():

    args = image_req_args.parse_args()
    image = BytesIO(base64.urlsafe_b64decode(args['image'])) #convert base64 image string to binar
    Result  = model(image)              # assign return object from model() to Result

    return Result                       # return Result as response to frontend
```

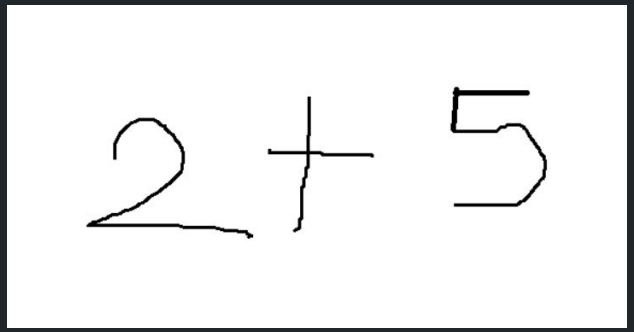
model.py

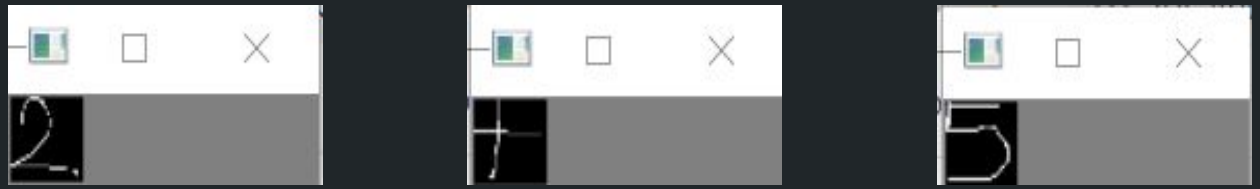
```python
def model(image):


    Image.open(image).save('input.png')    #save the binary image in local directory of server
    Result = processor('input.png')     #call processor(image_url) which is main model and
                                        #solver, return object in Result
    return Result                       # return Result
```

Back End

- The back end is implemented using Python, Keras and OpenCV
- OpenCV performs character segmentation and binarization
- The Keras model is used to predict the segmented characters and returns the result as a string
- The string is solved and is sent back to the frontend for display

Back End - Preprocessing





Back End - ML Model features

- Type -> Sequential Model
- Two Convolution 2D layers -> for feature extraction
- Two Max Pooling Layers -> for dimension reduction
- One drop out layer -> to deal with overfitting

Layers

- Convolutional
  - 30 filters
  - Input shape = (28,28,1)
  - Kernel size (5,5)
  - Activation function
    - Relu

Layers(Continued.)

0	0	0	0	0	0
0	105	102	100	97	96
0	103	99	103	101	102
0	101	98	104	102	100
0	99	101	106	104	99
0	104	104	104	100	98

Image Matrix

Kernel Matrix

0	-1	0
-1	5	-1
0	-1	0

320					

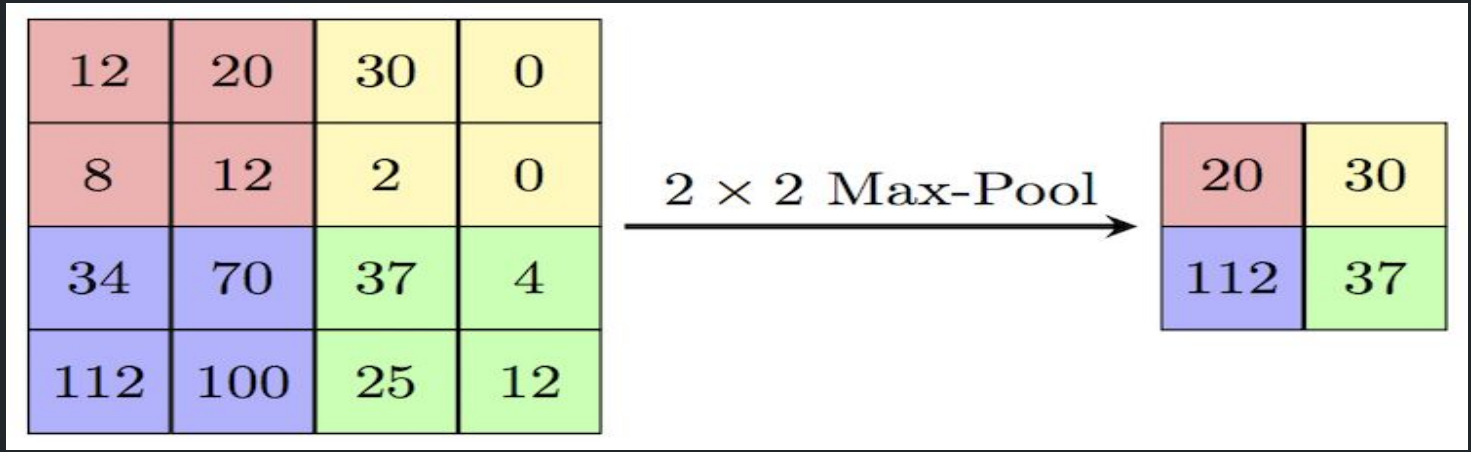
Output Matrix

$$0*0+0*-1+0*0$$
$$+0*-1+105*5+102*-1$$
$$+0*0+103*-1+99*0 = 320$$

Convolution with horizontal and vertical strides = 1

Layers(Continued.)

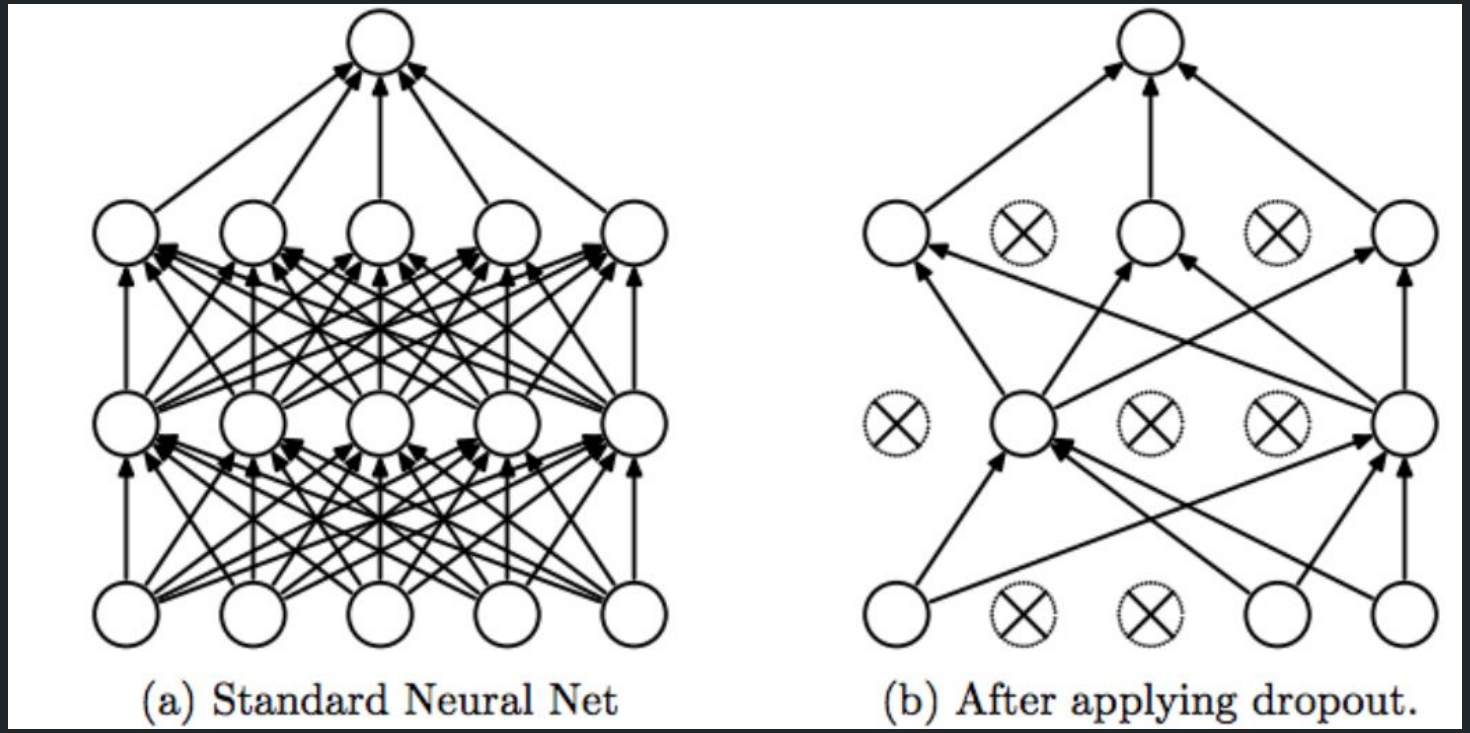
- Pooling
  - Maxpooling layer
    - Pool size (2,2)



Layers(Continued.)

- Convolutional layer
  - 15 filters
  - Kernel (3,3)
  - Activation Function
    - Relu
- Maxpooling layer
  - Pool size (2,2)
- Dropout layer
  - Dropout rate 0.2

Layers(Continued.)



Layers(Continued.)

- Flatten layer
- Fully connected layers
  - Activation Function
    - Relu
    - softmax

Back End - Summary

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, 128)	48128
dense_1 (Dense)	(None, 90)	11610
dense_2 (Dense)	(None, 45)	4095

Total params: 68,678
Trainable params: 68,678
Non-trainable params: 0

Training Accuracy

```
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>
```

Validation accuracy

```
model.evaluate(np.array(m),tac)
```

```
432/432 [==============================] - 3s 6ms/step - loss: 0.0720 - accuracy: 0.9841

[0.0720311626791954, 0.9840545058250427]
```

Solving the equation

- The mathematical expression is returned as string after recognition. This string is solved depending on it's type
- If it is a simple expression, the in-built Python eval() function is used
- If it is a linear equation, the Sympy Python library is used

LIVE DEMO

Conclusion

- Recognizes the handwritten digits and characters.
- Solves the handwritten expressions and linear equations.
- Interface is user-friendly which is implemented using flask.
- Overcame challenges and improved accuracy.

References

- Reference paper -> https://www.researchgate.net/publication/326710549_Recognition_and_Solution_for_Handwritten_Equation_Using_Convolutional_Neural_Network
- Dataset (EMNIST) -> https://www.nist.gov/itl/products-and-services/emnist-dataset
- Other references -> https://aihubprojects.com/handwriting-recognition-using-cnn-ai-projects/ https://www.geeksforgeeks.org/handwritten-equation-solver-in-python/ https://github.com/sabari205/Equation-Solver

THANK YOU