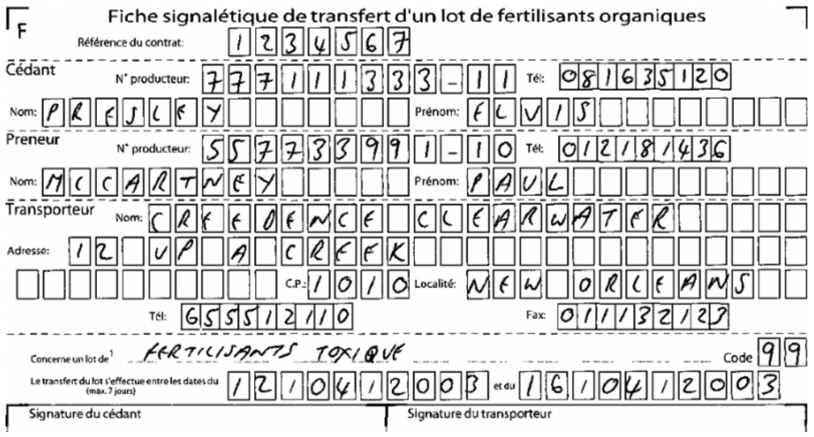
INDIAN INSTITUTE OF TECHNOLOGY (BHU) - VARANASI
EVEN SEMESTER, 2020-2021
EXPLORATORY PROJECT REPORT
(Prince Kumar Gond, Priyank Sisodia, Pratap Singh, Raghav Soni)

SUPERVISOR -
Dr. Amit Kumar Singh
Associate Professor
Department of Electronics Engineering Indian Institute of Technology
aksingh.ece@iitbhu.ac.in

HANDWRITTEN TEXT SEGMENTATION AND RECOGNITION

ABSTRACT
The problem of recognising handwritten characters is an important one to be solved. Recognising handwritten characters can bridge the gap between humans and machines to a large extent. Be it interpreting the numbers written on a cheque, addresses written on a mail or courier, reading number plates of the vehicles for security and surveillance or simply interpreting someone's handwritten notes.
The problem of handwritten character recognition can make a lot of applications very easy for humans and save time to a great extent. Therefore, in our exploratory project we try to build an end to end solution to recognise handwritten characters and interpret them for a computer. This is achieved mainly by using tools like Computer Vision to separate the characters and Deep learning modules to recognise the characters and interpret them for the computer.


F

Fiche signalétique de transfert d'un lot de fertilisants organiques

Référence du contrat: 1234567

Cédant N° producteur: 777111333-11 Tél: 081635120

Nom: PRESLEY Prénom: ELVIS

Preneur N° producteur: 5577339 91-10 Tél: 01/2/8/436

Nom: MCCARTNEY Prénom: PAUL

Transporteur Nom: CREEDENCE CLEARWATER

Adresse: 12 UP A CREEK

C.P.: 1010 Localité: NEW ORLEANS

Tél: 6555121/10 Fax: 011/1/32/127

Concerne un lot de[1] FERTILISANTS TOXIQUE Code 99

Le transfert du lot s'effectue entre les dates du (max. 7 jours) 12/04/2003 et du 16/04/2003

Signature du cédant Signature du transporteur

Fig1. A Sample cheque that would require character recognition

1 BACKGROUND

The problem that has been tackled is not a linear problem belonging to one domain. It requires knowledge of different domains in order to be tackled efficiently. There is knowledge from multiple fields that we used in our project. The major fields that were explored are -

1. PYTHON

   Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

   Python has a multitude of libraries that can be imported and they can make things very easy for the programmer to implement. It has libraries like OpenCV [1] for computer vision, numpy for matrix manipulation and tensorflow/keras [2] for deep learning. Hence, due to its convenience, support and availability of major libraries, Python is our choice as a programming language for this project.

Fig2. Python Symbol

2. <u>COMPUTER VISION</u>

   Computer vision is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do.

   Computer vision is an umbrella term which contains a lot of sub domains in it. In the problem of recognising handwritten characters, we majorly use it for separating characters from each other. This is the first step to recognise handwritten characters, to separate individual characters so that we can recognise each one of them separately.

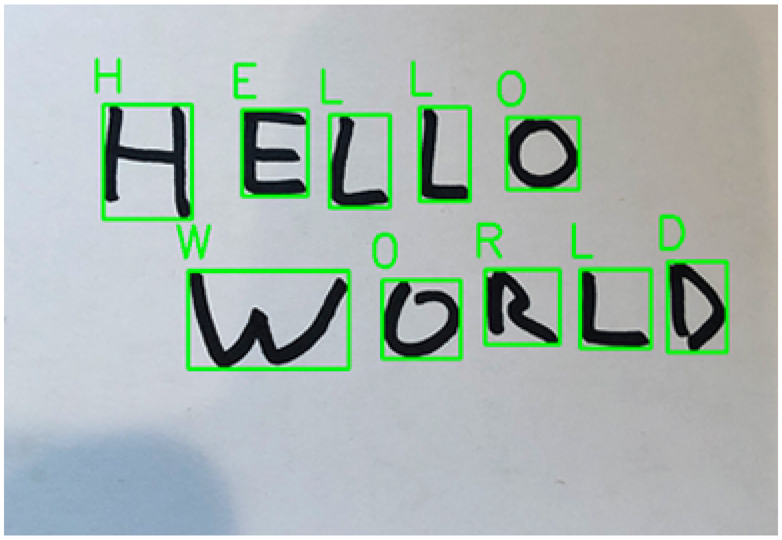


Fig3. Segmenting Individual Characters

3. <u>DEEP LEARNING</u>

   Deep learning (also known as deep structured learning) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Such algorithms are trained on a set of sample data that we already have. Once it is trained, it can operate on unseen data and make predictions about it. We apply deep learning algorithms to identify a single character after separating it from the rest. Deep learning modules try to learn little features of the image they see and then predict the unseen images based on the features that they have learnt. This helps us in recognising letters and alphabets.

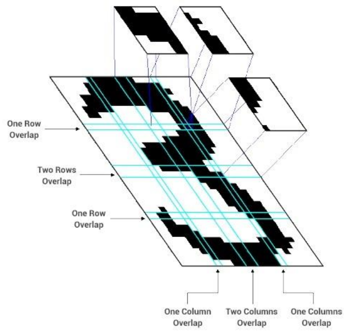


Fig4. An example as how such modules work

2 METHOD

Firstly, the problem is divided into two parts. These two parts are -

1. CHARACTER SEGMENTATION [3]
   The deep learning module that predicts the character from it’s image can only work on a single character. Hence, first we have to separate out different characters from a word. The process of separating individual characters from a word is called character segmentation and it is an important problem in itself. So, first we tackle this problem.

2. CHARACTER RECOGNITION
   Once we have individual characters, we feed them into a deep learning based Convolutional neural network one by one and it returns the predictions. This problem majorly involves building a robust neural network and training it on the sample data so that it can make predictions about the unseen and novel data.

So, we tackle both of these problems and explore these problems one by one in depth.

CHARACTER SEGMENTATION

The problem of character segmentation was an interesting one to explore. We not only tried to extract individual characters from a single image, but we also tried to extract them from a noisy image with a lot of disturbances over the characters and we were successful in doing that. An example of a noisy image that we worked on is given here.

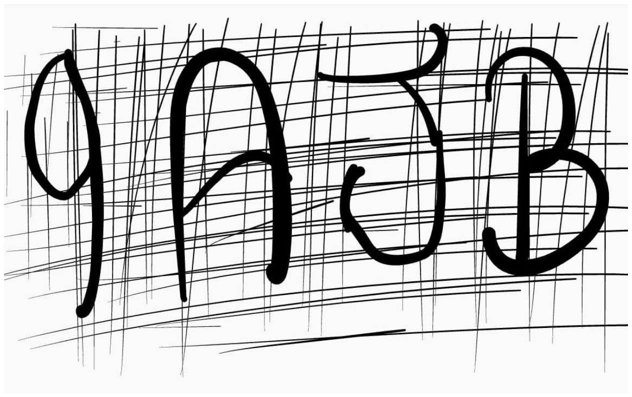


Fig5. Example of noisy image

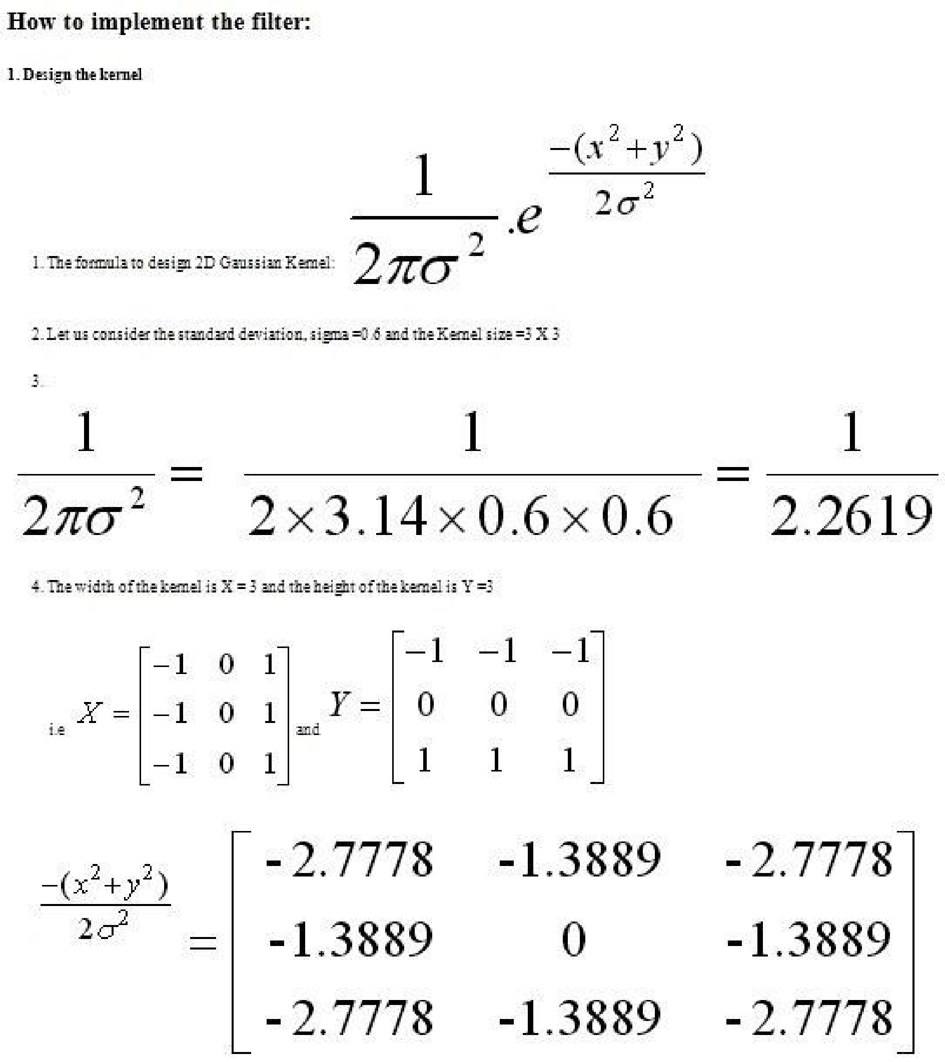
This problem is tackled using computer vision and OpenCV/Numpy libraries in the python programming language. There is a twofold goal that we are trying to pursue here, reducing noise and trying to extract character without letting the noise distort them.

So first we try to reduce the noise from the image. We apply the following methods one by one to the image in order to reduce the noise -

1. Gaussian Blur -

Applying gaussian blur [4] to an image makes it a little smooth. The sharp edges and lines in characters as well as in noise make it difficult for us to segment the characters and to reduce the noise. Hence first we apply the gaussian blur on our image to make it smooth.

We do this with the help of a gaussian filter. The filter is designed in the following way -



How to implement the filter:

1. Design the kernel

1. The formula to design 2D Gaussian Kernel: $\frac{1}{2\pi\sigma^2}.e^{\frac{-(x^2+y^2)}{2\sigma^2}}$

2. Let us consider the standard deviation, sigma =0.6 and the Kernel size =3 X 3

3.

$$\frac{1}{2\pi\sigma^2} = \frac{1}{2\times 3.14\times 0.6\times 0.6} = \frac{1}{2.2619}$$

4. The width of the kernel is X = 3 and the height of the kernel is Y =3

i.e $X = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ and $Y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$

$$\frac{-(x^2+y^2)}{2\sigma^2} = \begin{bmatrix} -2.7778 & -1.3889 & -2.7778 \\ -1.3889 & 0 & -1.3889 \\ -2.7778 & -1.3889 & -2.7778 \end{bmatrix}$$

This filter that is designed is then convoluted over the whole image. The convolution process can be visualized with the help of given image -

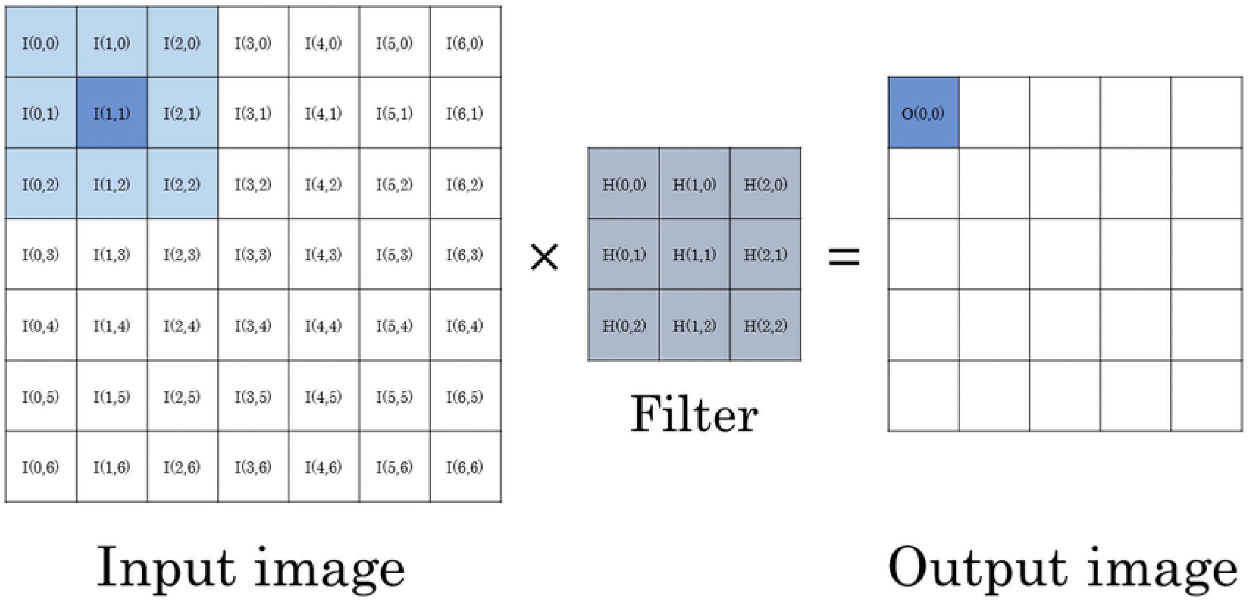


Fig6. Kernel Convolution Operation

Hence, the designed filter is taken over the image again and again, and with each iteration, a new image cell is created.

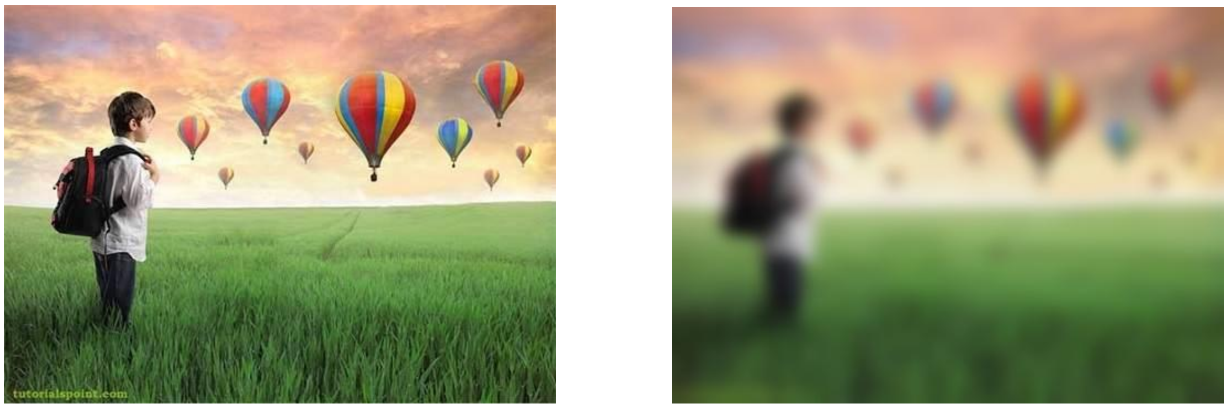
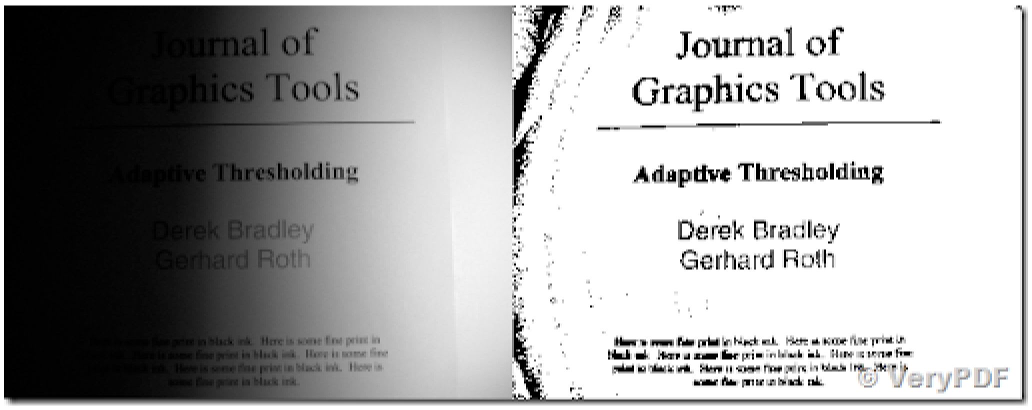


Fig7. Effect of Gaussian Blur on an Image

<u>2. Thresholding -</u>

An image usually consists of three channels that are Red, Green and Blue. However, the color information is useless to us and it just makes the data more complicated. Hence, we convert our image into a Binary image whose pixels can only have two values, either 1 or 0. This compresses the image and makes it easier to work on.

First the image is converted into grayscale, i.e it is left with only one channel with pixel value ranging from 0 to 255 denoting the degree of blackness, then we obtain a binary image from it by converting the values above a certain threshold (say 200) to 1 and rest to zero. Hence, we get a pure black and white image.


Journal of
Graphics Tools

Adaptive Thresholding

Derek Bradley
Gerhard Roth

Here is some fine print in black ink. Here is some fine print in black ink. Here is some fine print in black ink. Here is some fine print in black ink. Here is some fine print in black ink.

Fig8. Effects of Thresholding on an image

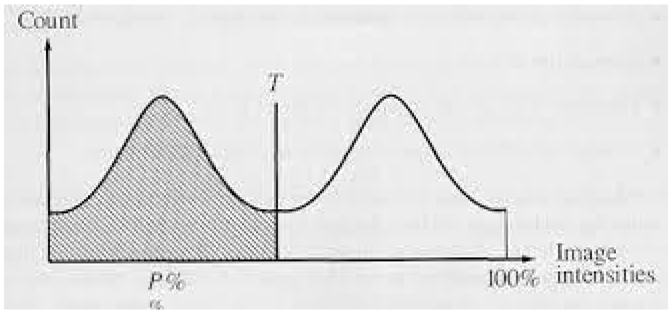


Fig9. Working of thresholding

3. Morphological Operations-

These operations [5] are applied to remove the noise from an image. There are two major morphological operations. These are -

1. Dilation

   The value of the output pixel is the maximum value of all pixels in the neighborhood. In a binary image, a pixel is set to 1 if any of the neighboring pixels have the value 1.

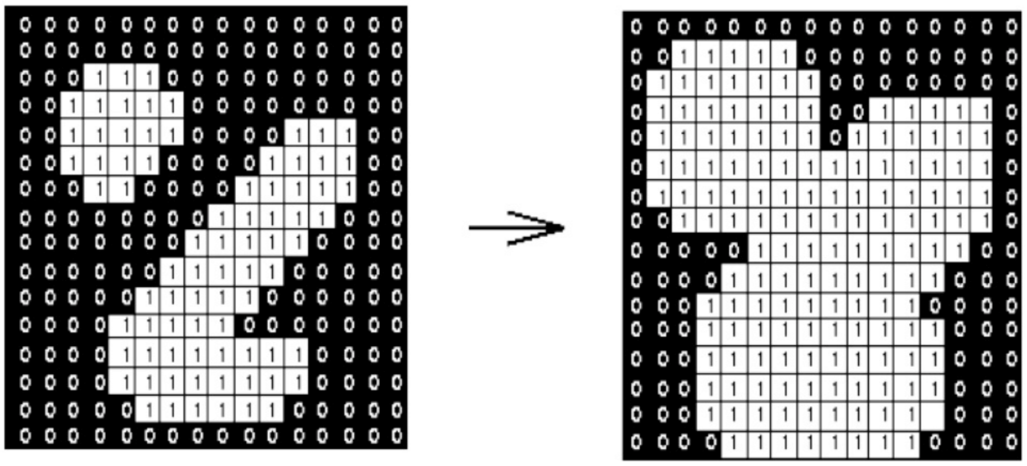
   Morphological dilation makes objects more visible and fills in small holes in objects.

The dilation of A by the structuring element B is defined by

$$A \oplus B = \bigcup_{b \in B} A_b .$$

The dilation is commutative, also given by $A \oplus B = B \oplus A = \bigcup_{a \in A} B_a$

The effect of dilation on an image can be observed as -



Effect of dilation using a 3×3 square structuring element

Fig10. Effect of Dilation

2. Erosion

   The value of the output pixel is the minimum value of all pixels in the neighborhood. In a binary image, a pixel is set to 0 if any of the neighboring pixels have the value 0.

   Morphological erosion removes islands and small objects so that only substantive objects remain.

   The erosion of the binary image A by the structuring element B is defined by

$$A \ominus B = \{z \in E | B_z \subseteq A\},$$

where Bz is the translation of B by the vector z, i.e., $B_z = \{b + z \mid b \in B\}$

The effect of erosion on an image can be observed as-

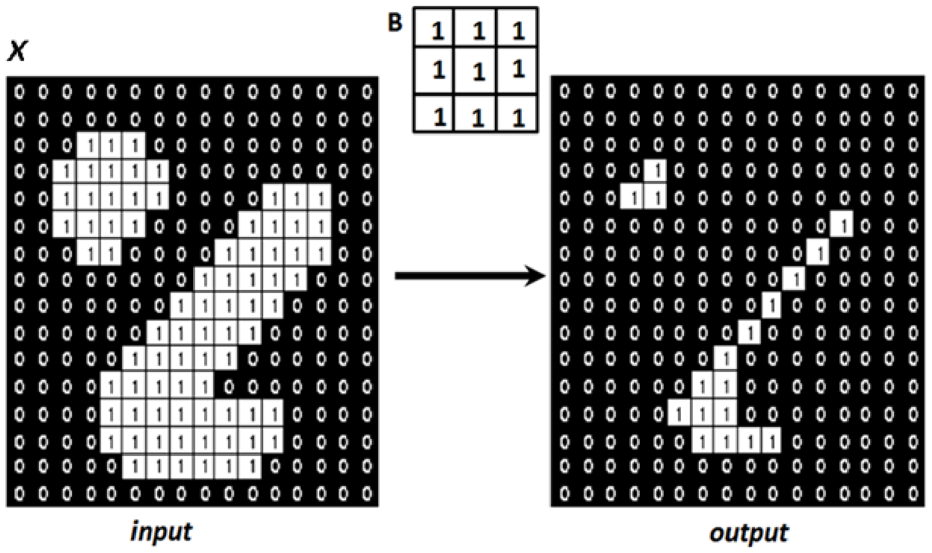


Figure 2. Effect of erosion using a 3X3 square structural element B.

Fig11. Effect of Erosion

<u>4. Median Filter</u>

The median filter is a non-linear digital filtering technique, often used to remove noise from an image or signal. Such noise reduction is a typical pre-processing step to improve the results of later processing (for example, edge detection on an image).
We use this to remove circular noise from the image.

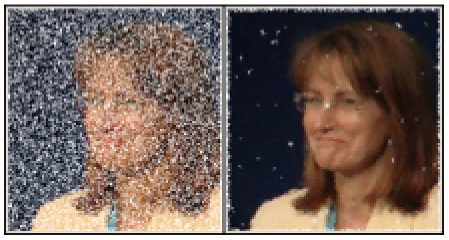


Fig12. Effect of Median Filter

5. Contour Detection

Now using all the previous operations, most of the noise is removed from the image. We apply contour detection on characters to finally segment them and obtain individual characters.

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having the same color or intensity. In our case, the contours are the individual characters. We detect all the contours and operate on contours bigger than a minimum area. We do this to neglect noisy contours that would have been detected in the image.

After contour detection, the image looks something like this and we get our individual characters -

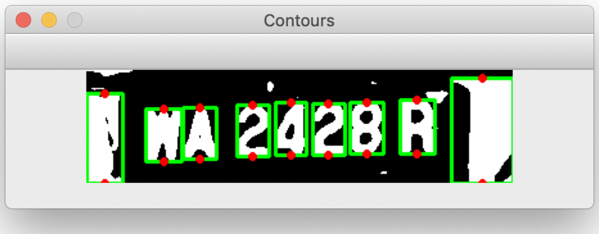


Fig11. Contour Detection on filtered Image

With this we obtain our segmented characters and now we are ready to predict which characters they are using a deep learning model.

CHARACTER RECOGNITION

Now we have obtained individual characters from our character segmentation model. Now the next step is to identify each individual character. We use a Convolutional Neural Network for this.

Convolutional Neural Network

A Convolutional Neural Network (ConvNet/CNN) [6] is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. Here, we employ this neural network to differentiate between and recognise all the alphabets and numbers.

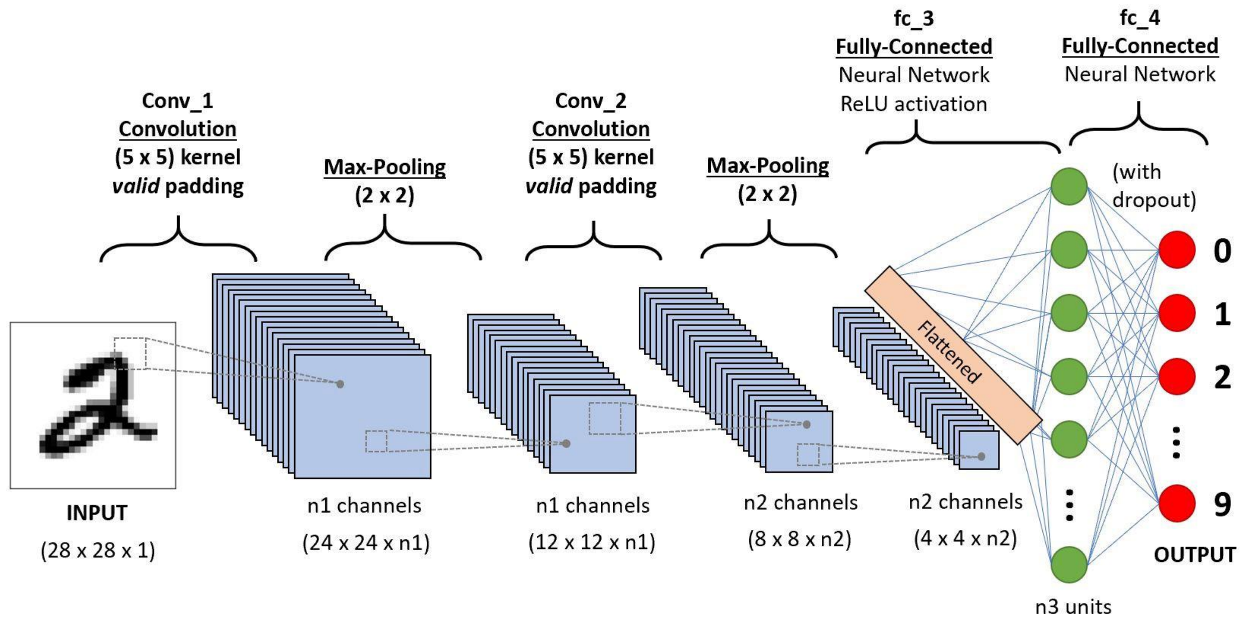


Fig12. Example of a CNN (Convolutional Neural Network)

It has some weights and biases which are adjusted while training to get the right predictions. After training, the weights and biases are fixed and we use these weights and biases to make predictions about the unseen data.
A neural network consists of singular units known as a neuron which are interconnected. The individual neuron and the mathematics behind it’s value can be denoted as -

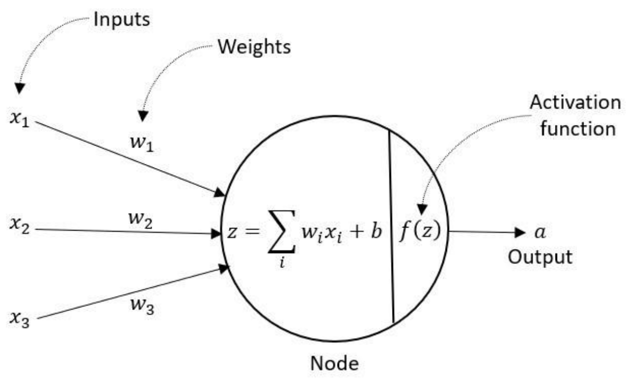


Fig13. A single Neuron

A single neuron works in the following way -

Step 1: For each input, multiply the input value x_i with weights w_i and sum all the multiplied values. Weights — represent the strength of the connection between neurons and decide how much influence the given input will have on the neuron's output. If the weight w_1 has a higher value than the weight w_2, then the input x_1 will have a higher influence on the output than w_2.

$$\sum = (x_1 \times w_1) + (x_2 \times w_2) + \cdots + (x_n \times w_n)$$

The row vectors of the inputs and weights are x = [x_1, x_2, … , x_n] and w =[w_1, w_2, … , w_n] respectively and their dot product is given by

$$x.w = (x_1 \times w_1) + (x_2 \times w_2) + \cdots + (x_n \times w_n)$$

Hence, the summation is equal to the dot product of the vectors x and w

$$\sum = x.w$$

Step 2: Add bias b to the summation of multiplied values and let's call this z. Bias — also known as the offset is necessary in most of the cases, to move the entire activation function to the left or right to generate the required output values.

$$z = x.w + b$$

Step 3: Pass the value of z to a non-linear activation function. Activation functions — are used to introduce non-linearity into the output of the neurons, without which the neural network will just be a linear function. Moreover, they have a significant impact on the learning speed of the neural network. Perceptrons have binary step function as their activation function. However, we shall use sigmoid — also known as logistic function as our activation function.

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

where σ denotes the sigmoid activation function and the output we get after the forward prorogation is known as the predicted value ŷ.

BACKPROPAGATION

The neural network learns through backpropagation and it forms the backbone of the neural network. This backpropagation works as -
The backpropagation carried out in a perceptron is explained in the following two steps.

Step 1: To know an estimation of how far we are from our desired solution a loss function is used. Generally, mean squared error is chosen as the loss function for regression problems and cross entropy for classification problems. Let’s take a regression problem and its loss function be mean squared error, which squares the difference between actual (y_i) and predicted value ($\hat{y}_i$).

$$MSE_i = (y_i - \hat{y}_i)^2$$

Loss function is calculated for the entire training dataset and their average is called the Cost function C.

$$C = MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Step 2: In order to find the best weights and bias for our Perceptron, we need to know how the cost function changes in relation to weights and bias. This is done with the help of the gradients (rate of change) — how one quantity changes in relation to another quantity. In our case, we need to find the gradient of the cost function with respect to the weights and bias.
Let’s calculate the gradient of cost function C with respect to the weight w_i using partial derivation. Since the cost function is not directly related to the weight w_i, let’s use the chain rule.

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial w_i}$$

The gradients are calculated and the error is propagated backwards to the weights and biases and is therefore minimised with learning.

LAYERS

A convolutional Neural network consists of many layers [7]. Some of the important layers that make up the CNN are -

1. Conv2d layer -
   This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs.

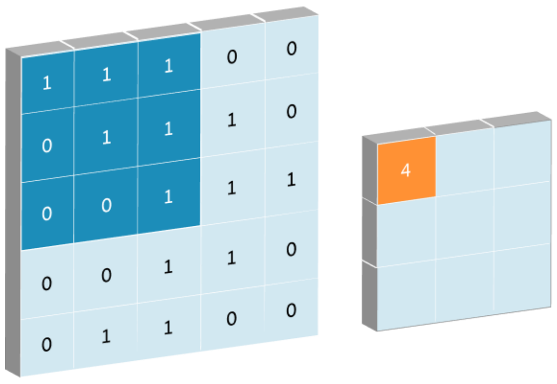


Fig14. A Conv2d layer

2. Batch Normalisation-
   Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.

   Importantly, batch normalization works differently during training and during inference.During training (i.e. when using fit() or when calling the layer/model with the argument training=True), the layer normalizes its output using the mean and standard deviation of the current batch of inputs. That is to say, for each channel being normalized, the layer returns (batch - mean(batch)) / (var(batch) + epsilon) * gamma + beta, where:

   - epsilon is small constant (configurable as part of the constructor arguments)
   - gamma is a learned scaling factor (initialized as 1), which can be disabled by passing scale=False to the constructor.

- beta is a learned offset factor (initialized as 0), which can be disabled by passing center=False to the constructor.

3. Flatten
   This layer flattens the input and makes it linear. It is usually employed at the end to make the convoluted layers linear.

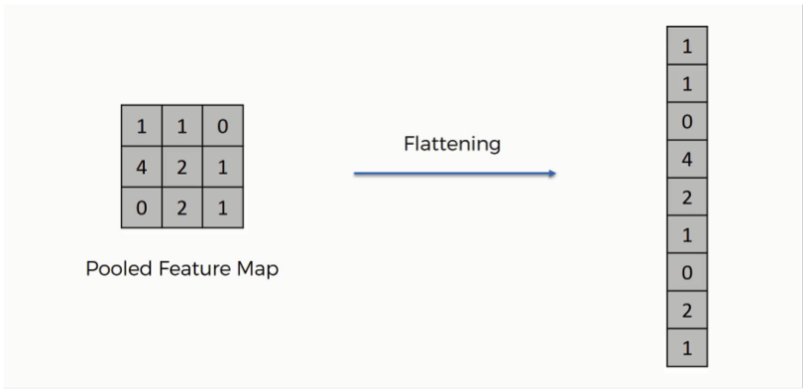
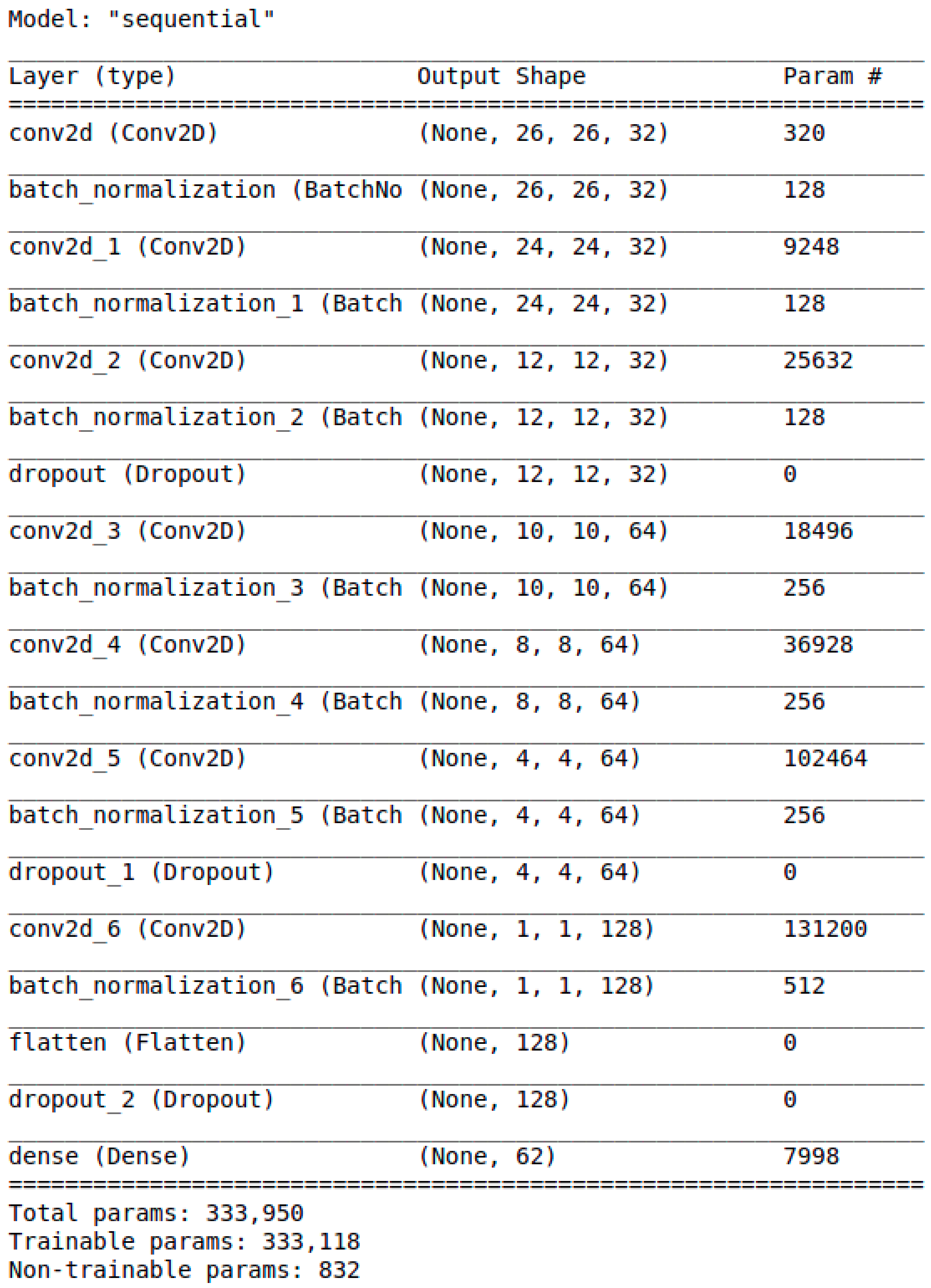


Fig15. Example of Flatten Layer

A convolutional Neural Network is a combination of multiple layers of different kinds. The combination of layers is called the architecture of the neural network. The architecture is decided experimentally and there can be different architecture for different applications.
We read some research papers and did some experiments to decide the architecture best suited for our application and the following neural network architecture was finalised -



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
batch_normalization (BatchNo (None, 26, 26, 32)        128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 32)        9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 24, 24, 32)        128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 32)        25632
_________________________________________________________________
batch_normalization_2 (Batch (None, 12, 12, 32)        128
_________________________________________________________________
dropout (Dropout)            (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 64)        18496
_________________________________________________________________
batch_normalization_3 (Batch (None, 10, 10, 64)        256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 64)          36928
_________________________________________________________________
batch_normalization_4 (Batch (None, 8, 8, 64)          256
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 64)          102464
_________________________________________________________________
batch_normalization_5 (Batch (None, 4, 4, 64)          256
_________________________________________________________________
dropout_1 (Dropout)          (None, 4, 4, 64)          0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 1, 1, 128)         131200
_________________________________________________________________
batch_normalization_6 (Batch (None, 1, 1, 128)         512
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 62)                7998
=================================================================
Total params: 333,950
Trainable params: 333,118
Non-trainable params: 832
```

Fig16. The architecture of neural network

HYPERPARAMETERS

These are the adjustable parameters that are to be decided by the user, these are learning rate, batch size, optimizer, loss etc. These are usually set experimentally and intuitively. Over the training of the network, these hyperparameters need to be tuned for maximum efficiency.

DATASET

We train our CNN on a EMNIST [8] dataset, which consists of more than 8 lakhs samples of small, capital alphabets and digits. We remove the alphabets and digits from the dataset which are similar to make it easier for our network to classify them.

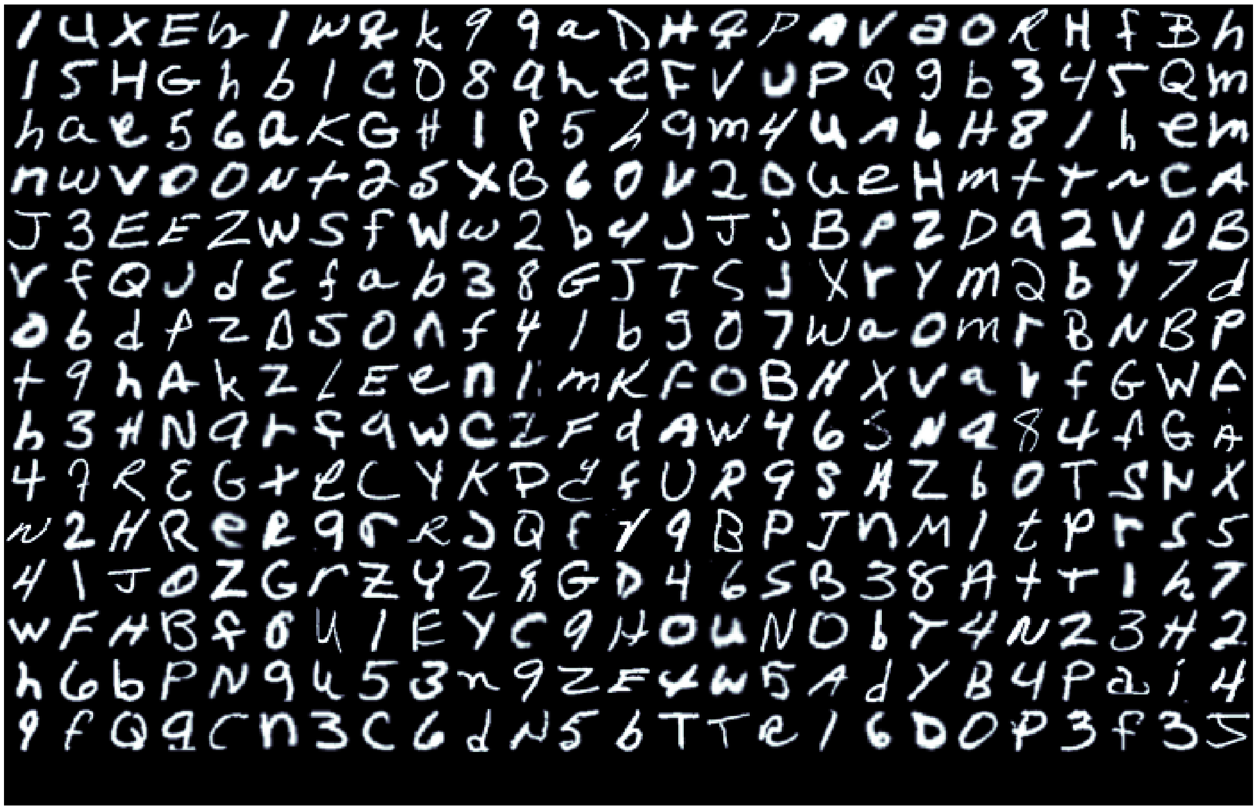


Fig17. EMNIST dataset

The alphabets and numbers that our network was able to classify with over 99% accuracy are -

0 1 2 3 4 5 6 7 8 9

A B C D E F G H J K M N P Q R T U V W X Y

b e h

TRAINING

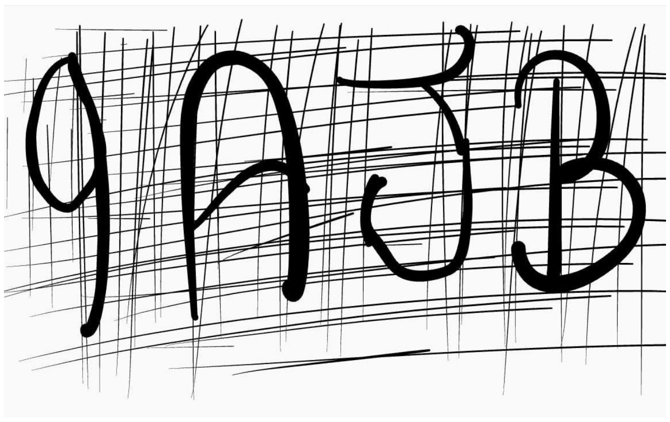
So we train our CNN with the given dataset and tune the hyperparameters to get the maximum efficiency. In the end the weights and biases of the CNN are freezed and we load them to make predictions while implementing the solution to this problem.

3 RESULTS

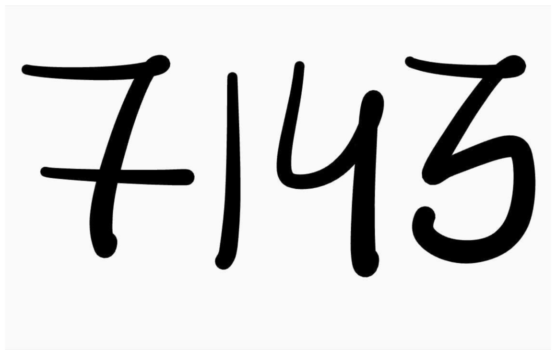
We were able to develop an end to end module which was able to do the following-

1. This model can recognize characters consisting of letters and numbers rotated at a max. angle of 45 deg.
2. It can be also used for characters consisting of lots of noise in the form of lines and dots.
3. Can detect characters having letters of variable thickness and size

Some of the sample results from the model are given here -

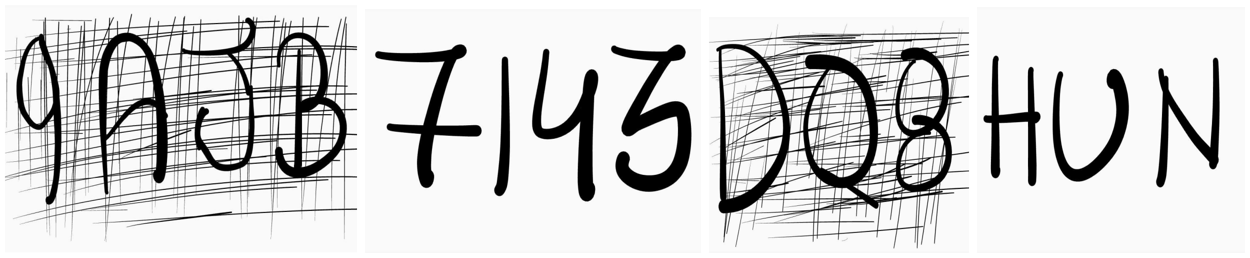


Segmented letter -

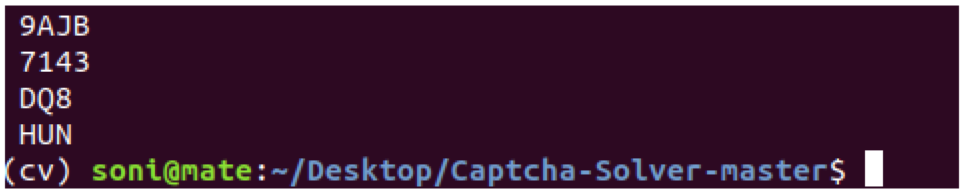


Segmented letters -

We fed these images into the model -



And we got the following predictions from our model -



```
 9AJB
 7143
 DQ8
 HUN
(cv) soni@mate:~/Desktop/Captcha-Solver-master$ 
```

This shows that our model is working robustly both for normal and for noisy handwritten words.

4 APPLICATIONS

This ability to recognise handwritten characters from an image has many applications. It can help in processing the cheques quickly, recognise the vehicle number from the number plate for security and surveillance, processing mails faster by reading the addresses written on them, solving captchas etc.
This module, if made robust enough can help a lot in day to day tasks that we have to carry out.

5 CONCLUSION/FUTURE WORK

It was a profoundly knowledgeable experience to approach and try to solve this problem. In future, we would like to improve the accuracy of our model, make it more robust, include more characters and increase the speed of the module. A lot of work has been done in this field but it is a long way to go before we can have a robust handwriting recognition model that is good enough to be relied on.

The github link to our project and its files-
https://github.com/Raghav-Soni/Handwitten-character-Segmentation-And-Recognition

REFERENCES

1. OpenCV - OpenCV
2. TensorFlow
3. Extraction of Line Word Character Segments Directly from Run Length Compressed Printed Text Documents
4. Gaussian blur.
5. Types of Morphological Operations - MATLAB & Simulink
6. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way
7. Keras layers API
8. The EMNIST Dataset