Mathematical Equation Detection and Evaluation

Abhishek Kumar (20200544)

The project report is submitted to University College Dublin
in part fulfilment of the requirements for the degree of
MSc Data and Computational Science

School of Mathematics and Statistics
University College Dublin

Supervisor: Dr.Ackay Sarp

August 15,2021

Abstract

Image classification has its roots in many fields from manufacturing, security sites, medical domain, computer vision applications like facial, text recognition etc, and these techniques as part of deep learning and computer vision are efficiently improving the accuracy in computer vision domain especially in the process of handwritten text detection's. Consumers and academics alike will benefit from the ability to identify handwritten sentiments. A student, for example, can use this system to verify the results of a paper-based solution. While a lot of work has gone into character identification, there is still a lot more to be done in the field of handwritten texts recognition. A system that identifies and solves mathematical equations would be extremely useful to the user, but developing such a system would be difficult due to the many types of mathematical symbols that are employed, as well as ambiguities in symbol position and arrangement in handwritten expressions. Artificial intelligence algorithms appear to be capable of teaching such a system to understand a wide range of handwritten mathematical operators/expressions given enough data. What happens in text detection? How image classification is helpful over here? So, the answer is, it is a process in which images are labelled in several labelled or pre-defined classes. There are many techniques that have been used for image classification such as YOLO, SVM, Boltzmann, and Random Forest etc. Many studies have proposed a model in which the Deep Neural Network technique by converting the image into grey scale and is used with the grey scaled segmentation technique or simply converting the image into grey scale using OpenCV and its morphological operations. The combination of these two techniques is yielding better results in minimal computational time.

Acknowledgments

I would like to express my gratitude to my supervisor, professor Dr. Akcay Sarp, for guiding and supporting me throughout this research. He helped me to overcome the obstacles that were faced during this work. I am thankful to him for always inspiring me and steering me in the right direction.

Contents

List of Figures

Chapter 1

General Introduction

1.1 Introduction

1.1.1 Neural Networks, Deep Columnar Convolutional Neural Network(DCCNN) and Computer Vision:

Recent research suggests that deep neural networks may perform well on a variety of difficult tasks, including handwritten number identification using the MNIST dataset and generic picture classification using the CIFAR 10 dataset. The learning period for deep neural networks has been lowered from weeks or months to a few hours or days because to advances in graphic processing units.

As, like MCDNN, the preprocessed input is split to connect divergent DCNNs on the same input or perform separate preprocessing steps, the architecture has been dubbed Deep Columnar Convolutional Neural Network. The distinction is that after each level, the forks are merged using one of two merge procedures, which has a major influence on the network's learning capabilities. Glorot uniform initialization is used to randomly initialize all of the weights at first. After that, each layer is trained with a collection of data known as the training set and verified with an unknown set of data known as the test set. To train the network, a few key approaches have been implemented.

What is computer vision? Computer vision is an interdisciplinary study that studies how computers may be programmed to comprehend digital pictures or movies at a high level. The goal is to automate operations that can be performed by human visual systems. As a result, a computer should be able to detect handwritten mathematical equations on paper.

How Does A Computer Read An Image? Any image is read by the computer as a range of numbers between 0 and 255. There are three major channels in any color image: red, green, and blue. It operates in a very straightforward manner. For each primary color, a matrix is created, and these matrices are then combined to provide a Pixel value for the individual R, G, and B colors. Each of the matrices' elements contains

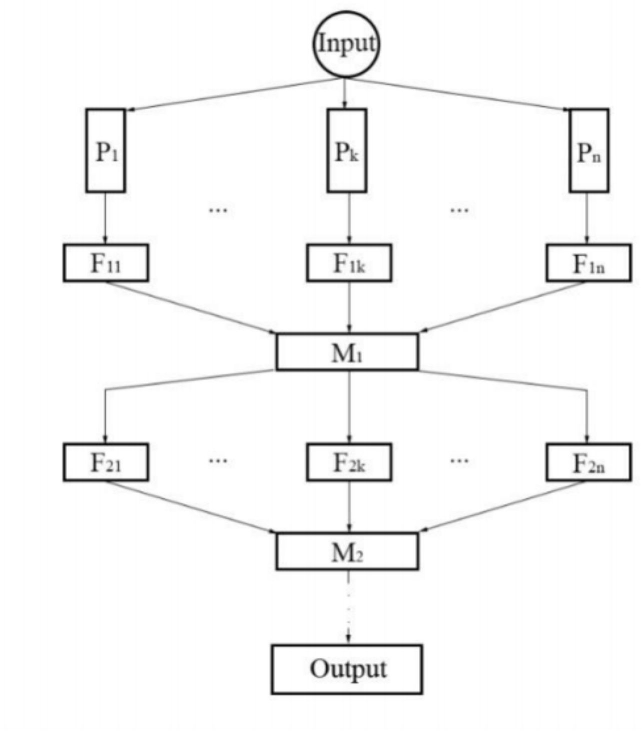


Figure 1.1: **DCCNN Achitecture**:P1 through Pn are the several preprocessing stages that can be done before the first fork layer. At the ith level of the design, Fij represents the jth Fork layer. The Merge layers are represented by M1 through Mn.[1]

information about the pixel's brightness intensity.

1.2 Problem Statement

Consumers and academics alike will benefit from the ability to identify handwritten sentiments. A student, for example, can use this system to verify the results of a paper-based solution. While a lot of work has gone into character identification, there is still a lot more to be done in the field of handwritten texts recognition. A system that identifies and solves mathematical equations would be extremely useful to the user, but developing such a system would be difficult due to the many types of mathematical symbols that are employed, as well as ambiguities in symbol position and arrangement in handwritten expressions. Artificial intelligence algorithms appear to be capable of teaching such a system to understand a wide range of handwritten mathematical operators/expressions given enough data. So, **The project is aimed at building an AI system based on Deep learning (DCCNN) to identify handwritten mathematical equations from a sheet of paper, digitizes the steps of solving them and evaluate them to provide feedback.**

1.3 Project Outline:

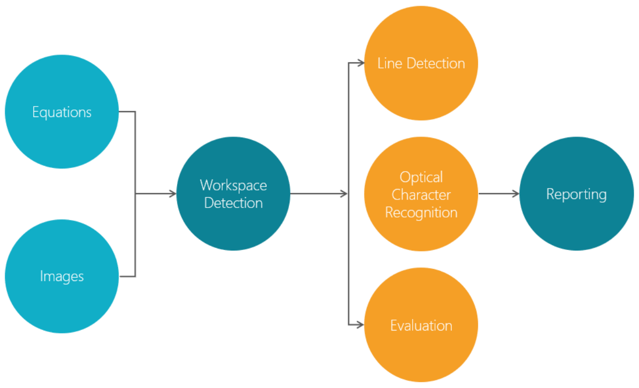


Figure 1.2: Flow of the project

1.4 Functional Blocks:

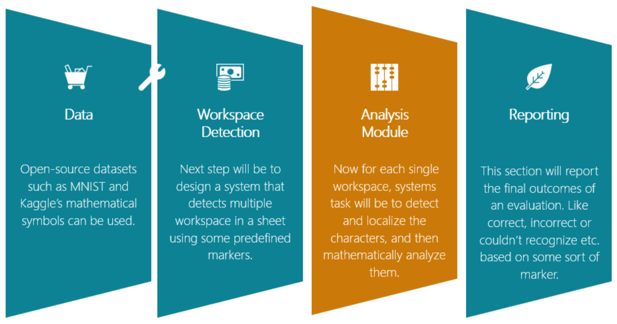

Data
Open-source datasets such as MNIST and Kaggle's mathematical symbols can be used.
Workspace Detection
Next step will be to design a system that detects multiple workspace in a sheet using some predefined markers.
Analysis Module
Now for each single workspace, systems task will be to detect and localize the characters, and then mathematically analyze them.
Reporting
This section will report the final outcomes of an evaluation. Like correct, incorrect or couldn't recognize etc. based on some sort of marker.

Figure 1.3: Building Blocks of the computer vision algorithm

In Chapter 2 we will discuss about how the data was prepossessed for Optical image recognition, work-space detection, analysis and reporting modules in detail

Chapter 2

Building Blocks of the Algorithm

2.1 Datasets Used for the model training:

Open source datasets such as MNIST and Kaggle's mathematical symbols are used for training the model. Around 125k images from each of these 15 character classes was used for model training and testing.

1. Digits [0,1,2,3,4,5,6,7,8,9] : Open source MNIST digits dataset of size (28*28)

2. Symbols [',' , '-', '+', '*']: Kaggle's Handwritten Mathematical Symbols Dataset of size (45*45), considered only a subset of all the symbols available to restrict the complexity of the algorithm.

2.1.1 MNIST Data:

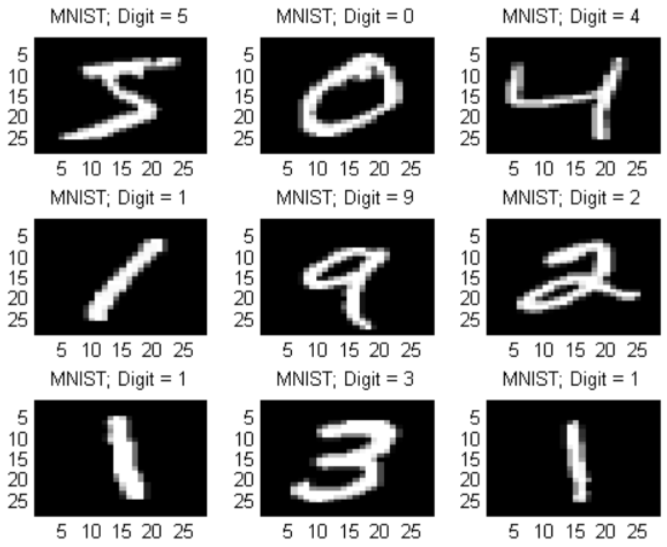


Figure 2.1: MNIST digits data

A common dataset used in computer vision and deep learning is the MNIST handwritten digit classification problems. The Modified National Institute of Standards and Technology dataset is an abbreviation for Modified National Institute of Standards and Technology dataset. It's a collection of 60,000 tiny square grayscale pictures of handwritten single numerals ranging from 0 to 9. 6000 samples are in training set while 10000 in testing set. Pre-processing stages involved in the MNIST dataset are as follows.

For further references see MNIST Data Source

1. 500 different writers provided 128 * 128 pixel handwritten numbers.

2. A Gaussian filter is used to the image to soften the edges.

3. The digit is then placed and centered within a square image while maintaining the aspect ratio.

4. The image is then down-scaled to 28 x 28 pixels using bi-cubic interpolation.

Symbols data from kaggle's data set are pre-processed in the same way as MNIST digits are before training to match its image specifications. If the data sets used have different dimensions, thickness, and line width, pre-processing becomes a must,Since it will be harder for the deep learning model to identify patterns in the data. The use of pre-processing will help to reduce digit and symbol variances.

2.1.2 Kaggle's Symbols Data:

This datasets include 82 symbols but only a few symbols such as "+", "-", "*", "(", ")" are selected to limit the complexity of the project. Each symbol contains at most around 15000 examples approximately. For further references see Kaggle's Data Source

Images has to be processed in the same way as MNIST before training. Steps involved in preprocessing are:

- Making a binary image with a black backdrop and white symbols
- Dilating by a 3 * 3 kernel
- Padding the image to 20 * 20 while keeping the aspect ratio
- The picture is padded to a size of 28 * 28 by the center of mass.

Gaussian blurring : The goal of blurring is to perform noise reduction. Gaussian blurring is nothing but using the kernel whose values have a Gaussian distribution. The values are generated by a Gaussian function so it requires a sigma value for its parameter.

Thresholding: transforms images into binary images. We need to set the threshold value and max values and then we convert the pixel values accordingly.

Erosion: is the technique for shrinking figures and it's usually processed in a grayscale.

Dilation: is the opposite to erosion. It is making objects expand and the operation will be also opposite to that of erosion.

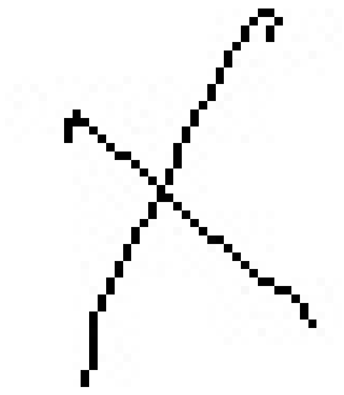


Figure 2.2: Kaggle's symbol image without pre-processing

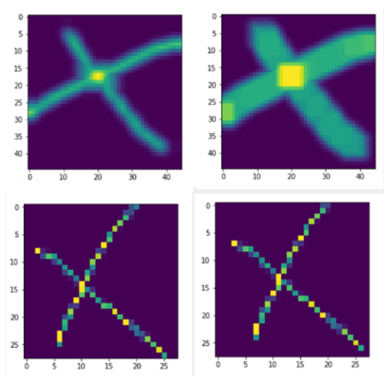


Figure 2.3: Gaussian blur, dilation, erosion,shifting and cropping steps outputs

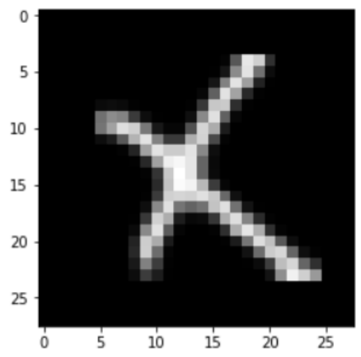


Figure 2.4: Pre-processed symbol image to match MNIST specs

2.2 Optical Character Recognition

2.2.1 DCCNN:

On several image classification problems, such as the MNIST, CIFAR-10, and CIFAR-100 datasets, the suggested model is a single deep and broad neural network architecture that achieves near state-of-the-art performance similar to ensemble models. Using tensor-flow's image data generator, the dataset is divided into 8:2 training and validation subsets for model training and validation.

Wide Architecture: While other models only have a few maps per layer (own a shallow architecture), the suggested model has a high number of maps per layer, layered in both horizontal (fork) and vertical (merge) levels. information loss is reduced due to vertical layers because of which model gets two input versions at the same time. Fork layers are often made up of many maps of the same or various sizes.[1]. Key Features of the model are:

Deep Architecture: The suggested model employs a multi-layered architecture that is further compounded by a high number of forked levels that are combined into a single big layer. The combined layer creates an extremely broad layer after merging. The deeper merge layers might include millions of parameters, which improves model performance.

Pooling via Convolution Subsampling: In certain situations, the max pooling layer is replaced with a convolution layer, and the output of the preceding layer is subsampled to retain information even during pooling operations. This is known as convolutional pooling, and it necessitates that the map size be the same as the previous layer's map size. As can be seen in, it improves convergence speed while also expanding the network's size.

Varible Kernel Size: A 5x5 kernel is present in all forked convolutional layers that take a preprocessed image. This increases the network's convergence speed and accuracy. All intermediate tier fork layers use a 4x4 or 3x3 kernel while increasing the amount of maps at the same time. To increase efficiency, the last tier fork and merge layer uses a 3x3 kernel and frequently has the highest map size.

Fork Layers: The fork layer, also known as a DCNN column, takes a single input and can have several convolutional layers layered in order. At any given level, there may be many fork levels, and the first fork layer may take the same input or information that has been preprocessed using different ways. The latter is favored since it gives each branch a unique perspective on the same facts. If the outputs are concatenated at the merge layer, fork layers may have various kernel sizes and map sizes at the same level.

Merge Layers: The merge layer takes k fork layers from the previous level and merges them together using merge operation: concatenate. This procedures combines the forked layers into a single layer, increasing the network's breadth.

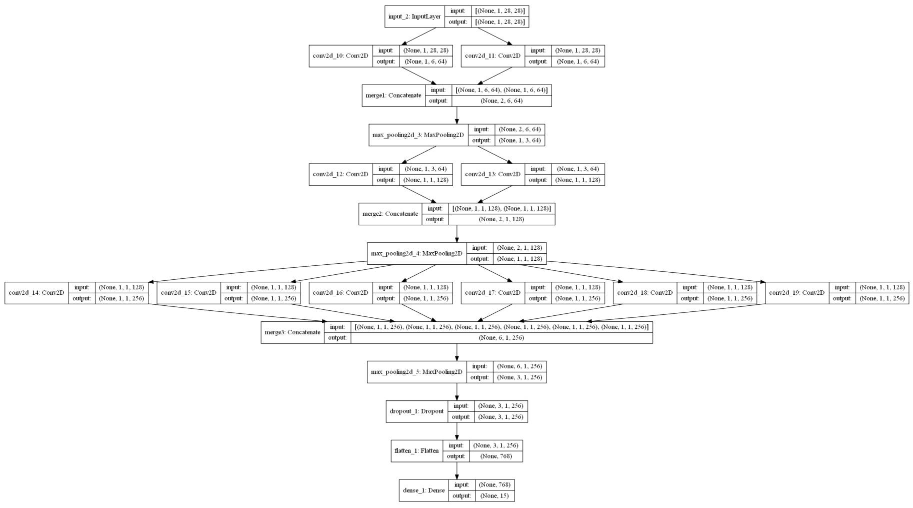
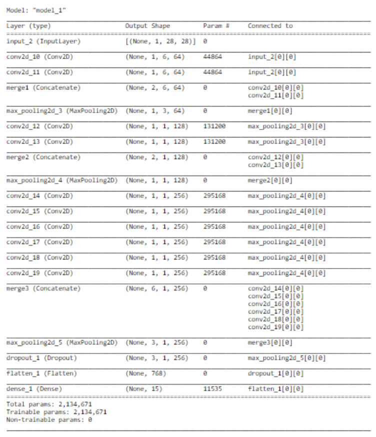


Figure 2.5: DCCNN model used for character recognition model



```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            [(None, 1, 28, 28)]  0
__________________________________________________________________________________________________
conv2d_10 (Conv2D)              (None, 1, 6, 64)     44864       input_2[0][0]
__________________________________________________________________________________________________
conv2d_11 (Conv2D)              (None, 1, 6, 64)     44864       input_2[0][0]
__________________________________________________________________________________________________
merge1 (Concatenate)            (None, 2, 6, 64)     0           conv2d_10[0][0]
                                                                 conv2d_11[0][0]
__________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 1, 3, 64)     0           merge1[0][0]
__________________________________________________________________________________________________
conv2d_12 (Conv2D)              (None, 1, 1, 128)    131200      max_pooling2d_3[0][0]
__________________________________________________________________________________________________
conv2d_13 (Conv2D)              (None, 1, 1, 128)    131200      max_pooling2d_3[0][0]
__________________________________________________________________________________________________
merge2 (Concatenate)            (None, 2, 1, 128)    0           conv2d_12[0][0]
                                                                 conv2d_13[0][0]
__________________________________________________________________________________________________
max_pooling2d_4 (MaxPooling2D)  (None, 1, 1, 128)    0           merge2[0][0]
__________________________________________________________________________________________________
conv2d_14 (Conv2D)              (None, 1, 1, 256)    295168      max_pooling2d_4[0][0]
__________________________________________________________________________________________________
conv2d_15 (Conv2D)              (None, 1, 1, 256)    295168      max_pooling2d_4[0][0]
__________________________________________________________________________________________________
conv2d_16 (Conv2D)              (None, 1, 1, 256)    295168      max_pooling2d_4[0][0]
__________________________________________________________________________________________________
conv2d_17 (Conv2D)              (None, 1, 1, 256)    295168      max_pooling2d_4[0][0]
__________________________________________________________________________________________________
conv2d_18 (Conv2D)              (None, 1, 1, 256)    295168      max_pooling2d_4[0][0]
__________________________________________________________________________________________________
conv2d_19 (Conv2D)              (None, 1, 1, 256)    295168      max_pooling2d_4[0][0]
__________________________________________________________________________________________________
merge3 (Concatenate)            (None, 6, 1, 256)    0           conv2d_14[0][0]
                                                                 conv2d_15[0][0]
                                                                 conv2d_16[0][0]
                                                                 conv2d_17[0][0]
                                                                 conv2d_18[0][0]
                                                                 conv2d_19[0][0]
__________________________________________________________________________________________________
max_pooling2d_5 (MaxPooling2D)  (None, 3, 1, 256)    0           merge3[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 3, 1, 256)    0           max_pooling2d_5[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 768)          0           dropout_1[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 15)           11535       flatten_1[0][0]
==================================================================================================
Total params: 2,134,671
Trainable params: 2,134,671
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 2.6: DCCNN model structure and number of parameters

Data Pre-processing for model training:

Images are first converted into a binary image, then noises are removed from them, then with a use of 3*3 kernel output images are dilated and eroded. Gaussian blur is applied with sigma value of 1. Then all the rows and columns where all the pixels are black are removed, after this by keeping the aspect ratio same images are padded to 20*20 image, then finally the images are padded to 28*28 with center of mass to get the final pre-processed image which matches the specifications of the training dataset.

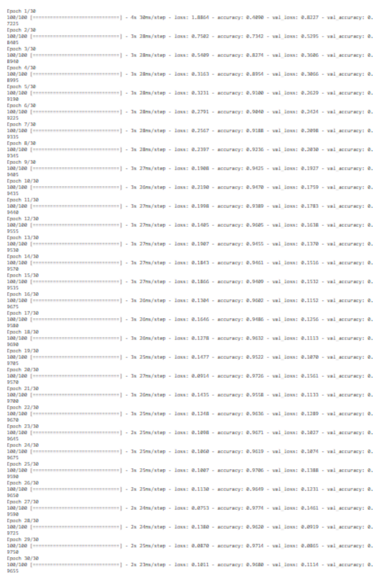
Key parameter definition in the models:

Activation Function: Softmax is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector. softmax is used as the activation function for multi-class classification problems where class membership is required on more than two class labels, 15 in our case. Softmax is utilized in the last layer of the neural network model to predict the multinomial probability distribution. While ReLu activation is used in the other levels.

Optimizers: When compared to other optimizers, the AdaDelta optimizer is an adaptive learning rate approach that does not require human tweaking of a learning rate. Instead of inefficiently storing w previous squared gradients, the sum of gradients is recursively defined as a decaying average of all past squared gradients. The parameters that were utilized were **learning rate = 1** and **rho = 0.95(decay factor)**

Dropout: Deep learning neural networks are prone to over-fitting a training data-set with a small number of instances. Dropout reduces the capacity of the model during training time by by randomly dropping out nodes. A crucial point is that weights of the different sampled networks are shared. Therefore, dropout combines node sampling with weight sharing, reducing the computational burden. Dropout is efficient because typically each sub-networks is trained with a small set of randomly sampled training data points.So before the last dense layer, a 50% **dropout** is applied. Regularization via dropout can be easily implemented in Keras by adding additional layers in the model configuration. This is different from the penalty-based regularization because dropout acts directly on the layout of the network, and not on the parameters, like weight decay.

Validation Accuracy: The model is trained for only 30 epochs and has a **96.55** percent accuracy.



```
Epoch 1/30
100/100 [==============================] - 4s 30ms/step - loss: 1.8864 - accuracy: 0.4090 - val_loss: 0.8227 - val_accuracy: 0.
7225
Epoch 2/30
100/100 [==============================] - 3s 28ms/step - loss: 0.7502 - accuracy: 0.7342 - val_loss: 0.5295 - val_accuracy: 0.
8405
Epoch 3/30
100/100 [==============================] - 3s 28ms/step - loss: 0.5409 - accuracy: 0.8274 - val_loss: 0.3606 - val_accuracy: 0.
8940
Epoch 4/30
100/100 [==============================] - 3s 28ms/step - loss: 0.3163 - accuracy: 0.8954 - val_loss: 0.3066 - val_accuracy: 0.
8995
Epoch 5/30
100/100 [==============================] - 3s 28ms/step - loss: 0.3231 - accuracy: 0.9100 - val_loss: 0.2629 - val_accuracy: 0.
9190
Epoch 6/30
100/100 [==============================] - 3s 28ms/step - loss: 0.2791 - accuracy: 0.9040 - val_loss: 0.2424 - val_accuracy: 0.
9225
Epoch 7/30
100/100 [==============================] - 3s 28ms/step - loss: 0.2567 - accuracy: 0.9188 - val_loss: 0.2098 - val_accuracy: 0.
9335
Epoch 8/30
100/100 [==============================] - 3s 28ms/step - loss: 0.2397 - accuracy: 0.9236 - val_loss: 0.2030 - val_accuracy: 0.
9345
Epoch 9/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1908 - accuracy: 0.9425 - val_loss: 0.1927 - val_accuracy: 0.
9405
Epoch 10/30
100/100 [==============================] - 3s 26ms/step - loss: 0.2190 - accuracy: 0.9470 - val_loss: 0.1759 - val_accuracy: 0.
9435
Epoch 11/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1998 - accuracy: 0.9389 - val_loss: 0.1783 - val_accuracy: 0.
9440
Epoch 12/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1405 - accuracy: 0.9605 - val_loss: 0.1638 - val_accuracy: 0.
9555
Epoch 13/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1907 - accuracy: 0.9455 - val_loss: 0.1370 - val_accuracy: 0.
9530
Epoch 14/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1843 - accuracy: 0.9461 - val_loss: 0.1516 - val_accuracy: 0.
9570
Epoch 15/30
100/100 [==============================] - 3s 27ms/step - loss: 0.1866 - accuracy: 0.9409 - val_loss: 0.1532 - val_accuracy: 0.
9535
Epoch 16/30
100/100 [==============================] - 3s 26ms/step - loss: 0.1304 - accuracy: 0.9602 - val_loss: 0.1152 - val_accuracy: 0.
9675
Epoch 17/30
100/100 [==============================] - 3s 26ms/step - loss: 0.1646 - accuracy: 0.9486 - val_loss: 0.1256 - val_accuracy: 0.
9580
Epoch 18/30
100/100 [==============================] - 3s 26ms/step - loss: 0.1278 - accuracy: 0.9632 - val_loss: 0.1113 - val_accuracy: 0.
9690
Epoch 19/30
100/100 [==============================] - 3s 25ms/step - loss: 0.1477 - accuracy: 0.9522 - val_loss: 0.1070 - val_accuracy: 0.
9705
Epoch 20/30
100/100 [==============================] - 3s 27ms/step - loss: 0.0914 - accuracy: 0.9726 - val_loss: 0.1561 - val_accuracy: 0.
9570
Epoch 21/30
100/100 [==============================] - 3s 26ms/step - loss: 0.1435 - accuracy: 0.9558 - val_loss: 0.1133 - val_accuracy: 0.
9700
Epoch 22/30
100/100 [==============================] - 3s 25ms/step - loss: 0.1248 - accuracy: 0.9636 - val_loss: 0.1289 - val_accuracy: 0.
9670
Epoch 23/30
100/100 [==============================] - 2s 25ms/step - loss: 0.1098 - accuracy: 0.9671 - val_loss: 0.1027 - val_accuracy: 0.
9645
Epoch 24/30
100/100 [==============================] - 3s 25ms/step - loss: 0.1060 - accuracy: 0.9619 - val_loss: 0.1074 - val_accuracy: 0.
9675
Epoch 25/30
100/100 [==============================] - 3s 25ms/step - loss: 0.1007 - accuracy: 0.9706 - val_loss: 0.1388 - val_accuracy: 0.
9590
Epoch 26/30
100/100 [==============================] - 2s 25ms/step - loss: 0.1130 - accuracy: 0.9649 - val_loss: 0.1231 - val_accuracy: 0.
9650
Epoch 27/30
100/100 [==============================] - 2s 24ms/step - loss: 0.0753 - accuracy: 0.9774 - val_loss: 0.1461 - val_accuracy: 0.
9590
Epoch 28/30
100/100 [==============================] - 2s 24ms/step - loss: 0.1380 - accuracy: 0.9620 - val_loss: 0.0919 - val_accuracy: 0.
9725
Epoch 29/30
100/100 [==============================] - 2s 25ms/step - loss: 0.0870 - accuracy: 0.9714 - val_loss: 0.0865 - val_accuracy: 0.
9750
Epoch 30/30
100/100 [==============================] - 2s 23ms/step - loss: 0.1011 - accuracy: 0.9680 - val_loss: 0.1114 - val_accuracy: 0.
9655
```

Figure 2.7: Model accuracy after 30 epochs

Augmentation: The idea behind data augmentation is the following. Given a theoretically infinite amount of data, the model would be exposed to every possible aspect of the data generating process, hence it would never over-fit and will always be able to generalize well. Data augmentation generates additional training data from the available training samples, by augmenting the samples using a number of random transformations that provide realistic-looking images. The aim is that at training time, the model will never encounter the exact same image twice. This helps expose the model to more aspects of the data generating process and generalize better, thus introducing regularization.

Data augmentation was used and the accuracy is marginally enhanced (random rotations, width shift, and height shift). **Rotation degree = 20, width shift = 20% of image width, Height shift = 20% of image height**

2.3 Workspace Detection

OpenCV is a large open-source library for computer vision, machine learning, and image processing, and it currently plays an important part in real-time applications, which are critical in today's systems. It can be used to recognize items, people, and even human handwriting in images and videos. Python can process the OpenCV array structure for analysis when it is combined with different modules such as NumPy. We can employ vector space and conduct mathematical operations on these characteristics to identify picture patterns and their different features.

OpenCV is used for the workspace detection module. Here the task is to identify rectangular boxes, then sort them according to where they are on the worksheet. Because there are so many rectangles in the worksheet,We need to pick the ones that are valid work-spaces among them. Let's have a look at how each stage is completed in the process of workspace detection.

Steps of work-space extraction:

The layers closest to the input are utilized to extract spatial characteristics in a convolution neural network. This behavior is modeled after what happens in the human visual system when we are asked to recognize a specific object. The shape, color, presence of textures, light orientation, and edges are the initial pieces of information that our brain decodes. We extract general information from the environment that, as we process it, allows us to extract more and more abstract information in order to recognize the object. Say I[x,y] is an image in two direction x(height) and y(width), then an edge can be defined as the region in i[x,y] where there is change in colour intensity.

Rectangular boxes Identification:A rectangle is obtained with Two horizontal and vertical lines. The first step is to locate all horizontal and vertical lines on the spreadsheet, ignoring any numerals, symbols, or other writing. This extract_box() will first generate a binary image named "vertical_line" that includes all of the worksheet's vertical lines, and then another binary image called "horizontal_lines" that contains all of the worksheet's horizontal lines and then they are added to get the final image. The findContour() function in OpenCV aids in the extraction of contours from an image. The cv2.RETR

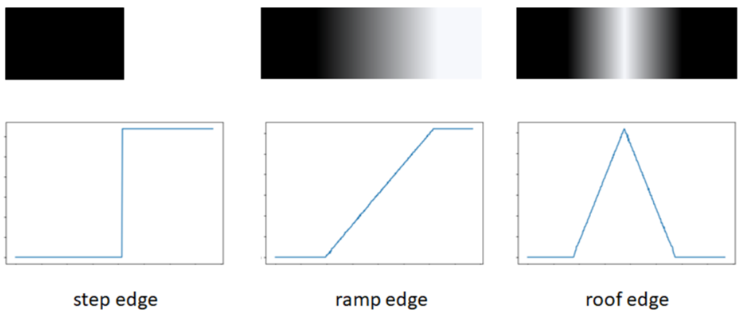


Figure 2.8: Edges in image : https://towardsdatascience.com/image-derivative-8a07a4118550

TREE mode searches for all plausible contour lines and reconstructs a complete hierarchy of nested contours. Only the endpoints required for drawing the contour line are returned by the function cv2.CHAIN APPROX SIMPLE. The contour lines are returned as a list of points in the returned contour. Each contour is a Numpy array of (x,y) coordinates of the object's border points. This may be used to locate all of the items in the final image (Only objects in the final image are the rectangles). The coordinates of the rectangles in the final image are equal to the coordinates of the rectangles in the original image since the final image is just a binary replica of the original image. Now that we have the coordinates, To draw the outer line of the figure, we'll sort the contours by their area. With the selected contour line, cv2.drawContours().

Contours Sorting: Now that we've located all of the rectangles, we can sort them by coordinates from top to bottom. The sort_contours() function returns contours and bounding boxes (top-left and bottom-right coordinates) that have been sorted according to the way we want it to be specified. The approach used in our case is top to bottom.

Area Based Selection: To get the area of the contours, we can implement the function cv2.contourArea(). Now we need to select the three biggest rectangles out of many, it is done by calculating the area of each rectangle and then selecting the top three rectangles with largest area. Then these selected work-spaces are sent to analysis module for line detection from each work-space.

2.4 Analysis module

As previously stated, the analysis module will first identify lines, then predict characters in each line, and lastly build an equation using the anticipated characters, which will be evaluated by marking boxes.

Detection of Line:

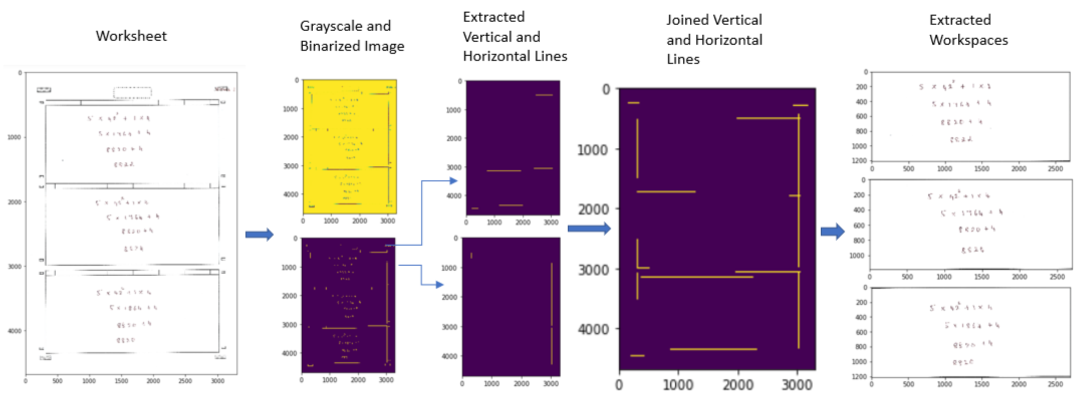
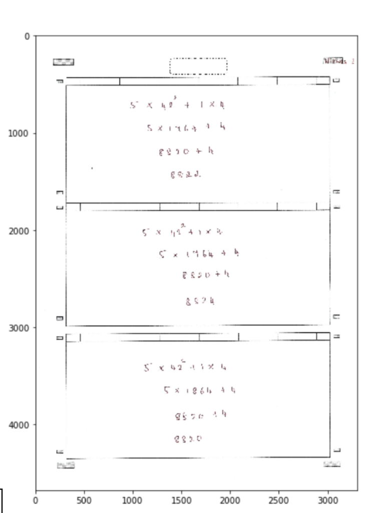

Worksheet
Grayscale and Binarized Image
Extracted Vertical and Horizontal Lines
Joined Vertical and Horizontal Lines
Extracted Workspaces

Figure 2.9: All steps involved during Extraction of Workspaces from a worksheet


[worksheet]

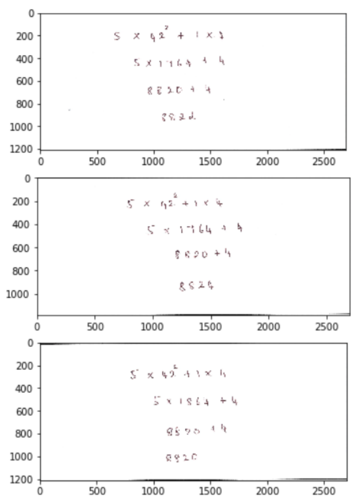

[Workspaces]

Figure 2.10: work-spaces identified after application of contour detection, selection based on area on original worksheet

It becomes difficult to detect a line for a computer program from a sheet of paper; due to the fact that humans can follow different approaches in solving a similar equation, someone may finish it in one line by skipping many intermediate steps, while some may write all the steps involved; the way we write equations is also different from person to person, for eg. orientations of writing texts on paper can differ from person to person, like some may write exponents away from the equation, which will let the module to interpret those exponents as a distinct line which is not ideal as exponents are part of the equation and considering them as separate line will totally change the equation and the calculations related to that mathematical equations would change.

The line detection program assumes that there is a sufficient space between lines and that exponential characters and lines meet at some point. To take the forward derivative, the identified work-spaces are first transformed to binary pictures, then compressed in a single array. This method of line detection is known as ”Line detection using forward derivative”, There will be a modification in the derivative whenever there is a line.

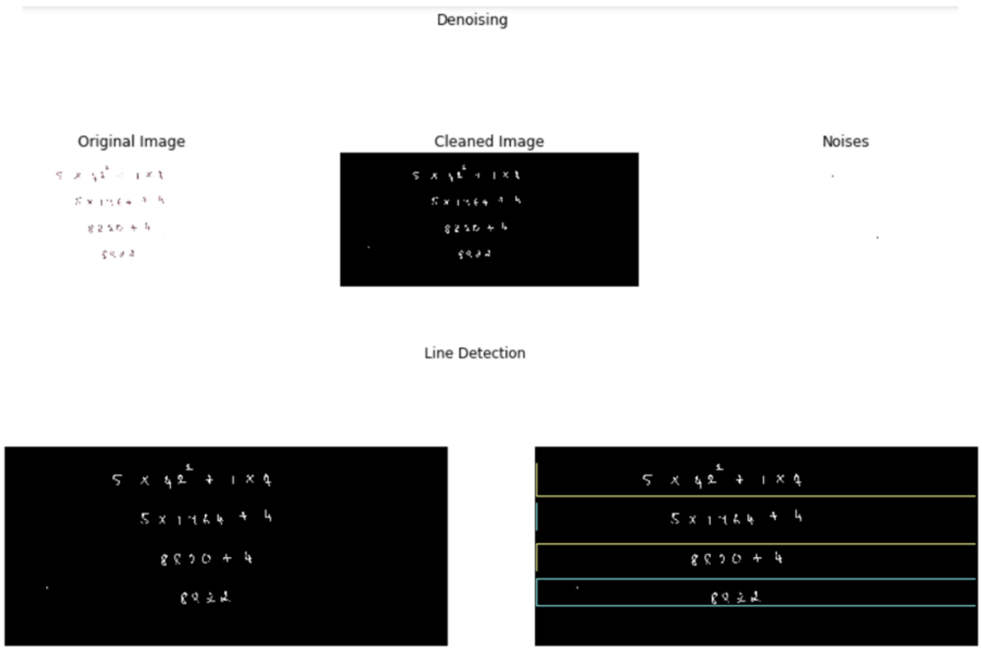


Figure 2.11: Line detection from a work-space located in previous section

Character Segmentation

After identifying all of the lines, we must pass the extracted line pictures to the text segment function, which will segment the characters using openCV’s findcontours() function and sort them using the function sort_contours() mentioned before, with the method set to left-to-right. It’s a simple task for humans to determine if a given number is an exponent or not, but it’s not so simple for the model. Assuming that the exponents are at least half of the line, we may draw a baseline in the image’s center and consider every character that is above it to be an exponent.

Optical Character recognition of each segmented character is performed using the DCCNN model trained earlier on MNIST digits and Kaggle’s symbols datasets to get the predictions of the these character class.

Each character is rounded in a box and its predictions are shown above it in pink font, the exponents are differentiated from the rest of the character by bounding it in a green box.

Evaluation and box creation

The **eval** method in Python allows us to evaluate arbitrary Python expressions from a text or compiled code input. When we need to dynamically evaluate Python expressions from any input, whether it’s a string or a built code object, this method comes in useful. So, it is used to solve any arithmetic equation identified from each extracted lines. The

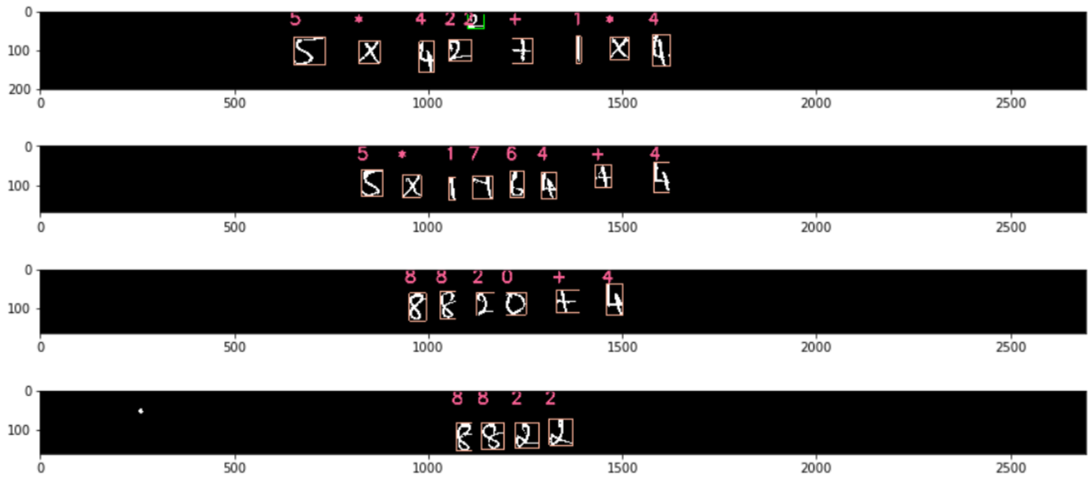


Figure 2.12: Character segmentation from each line detected and prediction of each character

built-in Python eval function may be used to dynamically evaluate expressions from a string or compiled code input. When we give eval a text, it parses it, converts it to bytecode, and evaluates it as a Python expression. When we call eval with a built code object, however, the method just executes the evaluation step, which is useful if you run eval several times with the same input. During the assessment process:

Each equation is solved and the answer is stored for verification in the next steps. Then each handwritten line identified is calculated and the results are then compared to the saved answer. Finally If the equation in a particular line is correctly solved,then a green bounding box is drawn;else if it is incorrect, then red box is draw. Blue bounding box will indicate that it is unidentifiable and a manual calculation is required to check the solution.

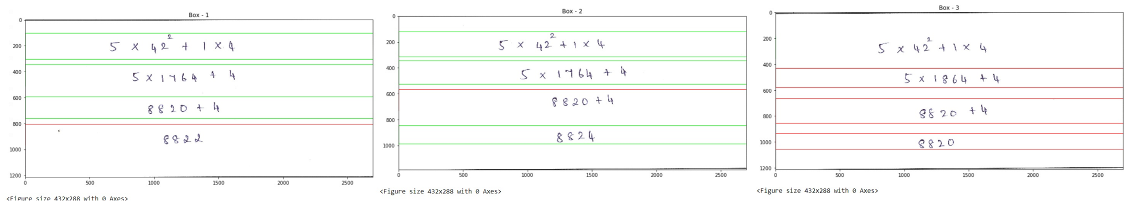


Figure 2.13: Final results from evaluation module

The above evaluation and green box across each line indicates that calculation done in box 2 for each line is correct. Whereas the final answer received in box1 and box3 is incorrect as indicated by red box around them.

Chapter 3

Conclusion

The system first took scanned worksheets with handwritten equations and sent it to the work-space detection module, where all the different possible rectangular work-spaces where detected. These detected work-spaces are then fed into the line extraction module, where each individual line from each of the work-space is detected and sent to next module for character segmentation. Here each character from the line is extracted and predicted using the Deep Columnar Neural Network(DCCNN) model.In the final stage the evaluation system will calculate the each line and then draw boxes around the line in green/red colour to indicate whether the solution provided is correct or not.

Future Work

1. Currently the system has limited scope of solving the simple arithmetic equations, expanding the functionalities of the mathematical operations used to advanced mathematical operations like differentiation, integration etc.

2. Right now the Function takes input in form of scanned images of worksheets stored in system. Integrating live capturing of images through mobile camera and showing the results on device.

3. Handwritten texts are currently used to train neural network model which is used for prediction of isolated texts from each line of equation, this text recognition feature can be extended for users of tablets using stylus pen i.e digital hand written texts.

Bibliography

1. Deep Columnar Convolutional Neural Network, Somshubra Majumdar Ishaan Jain D. J. Sanghvi College of Engineering, Mumbai, India. International Journal of Computer Applications (0975 – 8887) Volume 145 – No.12, July 2016 : Link

2. Recognition of Online Handwritten Mathematical Expressions Using Convolutional Neural Networks, Catherine Lu, Karanveer Mohan, Computer Science, Stanford University : Link

3. Neural network models : DCCNN Models

4. Kaggle's symbol data: Source

5. MNIST digit's data: Source

6. OpenCV : Computer-Vision-1

7. OpenCV : Computer-Vision-2

8. OpenCV : Computer-Vision-3

9. OpenCV : Computer-Vision-4

10. Stanford Edu : Automatic Equation Solver and plotter

11. Awesome-Scene-Text-Recognition : Text Recognition

12. recognize-your-handwritten-numbers : Handwritten Numbers Recognition

13. DeepHCCR : Deep HCCR