2019

Evaluation of Handwritten Mathematical Equations

DIVYAPRABHA MURUGESAN

Project Overview and Details

This section outlines the details related to the project background, proposed scope and perceived technical approach towards attaining the end objective of optical character recognition of mathematical equation using computer vision

1.1 Background

The overall idea behind this project is to develop an computer vision algorithm along with solution package for recognizing and digitizing steps of solving a mathematical equation written by freehand on a paper, validating the steps and final answer of the recognized handwritten lines by maintaining the context

1.2 Project Scope and Details

The following are the broad modules which will be catered to, towards the realization of the end objective of the project:

1. Workspace Detection using valid markers in the sheet
2. Detecting and localizing each single lines
3. Perform Optical Character Recognition in each detected line

Evaluating each line and providing feedback in terms of red/green bounding box drawn across it where green represents correct and red represents wrong answers.

1.3 Functional Scope

The primary functional blocks of the project are outlined in the block diagram below:

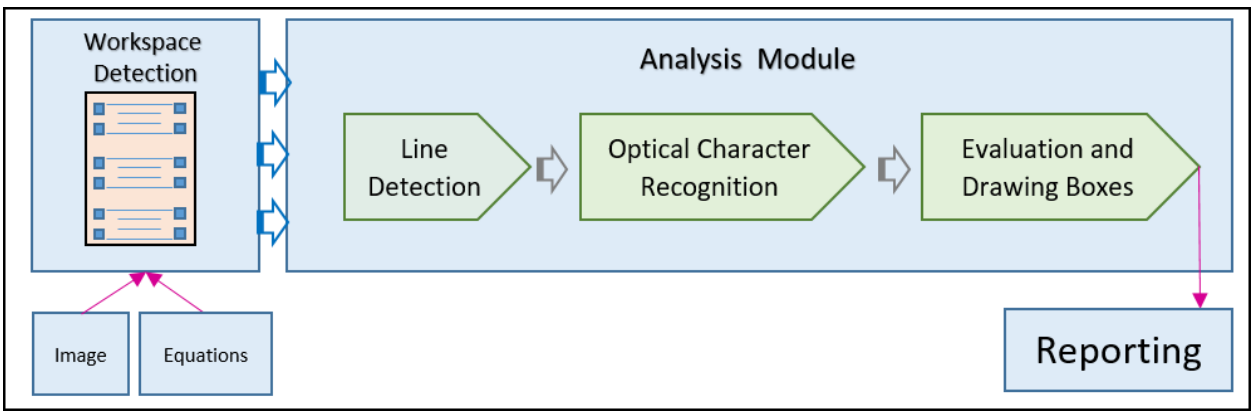

Figure 1: Workflow

As shown the overall solution can be divided into two parts, i.e ‘Workspace Detection’ module and ‘Analysis Module’. Workspace detection module is responsible for detecting multiple workspaces in a given sheet of paper using pre-defined markers as shown in image: 1 sheet. Analysis module is responsible for detecting and localizing characters in any given single workspace, and mathematically analysizing them and drawing red, green lines depending upon their correctness

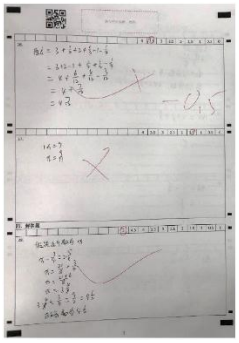

Image: 1 Sheet

2. Technical Scope

This section gives a detailed description about the data set used for analytics and the various models and techniques used, a comparison of those techniques and the proposed technique.

2.1 Data

Two open source datasets such as MNIST and Kaggle’s mathematical symbols are used for optical character recognition.

Total no of images	= 62,250
No of classes	= 15
Classes	= ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '+', '-', 'times','(',')']

2.1.1 MNIST

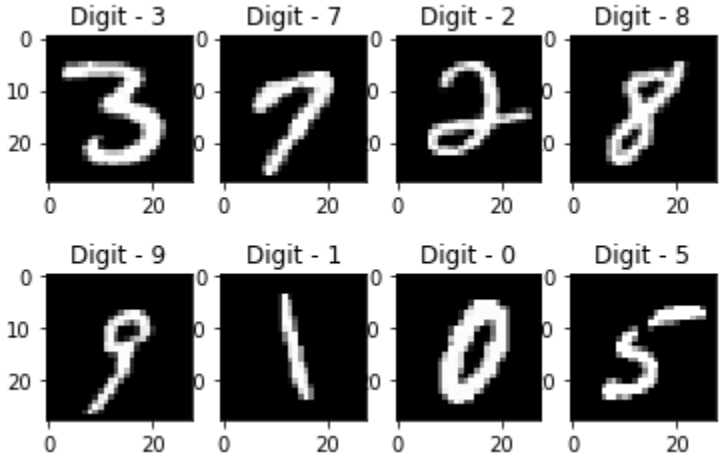

Figure 2: Sample images from MNIST

Samples provided from MNIST (Modified National Institute of Standards and Technology) dataset includes handwritten digits total of 70,000 images consisting of 60,000 examples in training set and 10,000 examples in testing set, both with labeled images from 10 digits (0 to 9) where the background is black and the digits are white. These images where normalized to fit a 20*20 pixel box without altering the aspect ratio and then centered in a 28 * 28 pixels by centre of mass.

2.1.2 Kaggle's Handwritten Mathematical symbols

This datasets include 82 symbols but only a few symbols such as "+", "-", "*", "(", ")" are selected. Each symbol contains at most 4000 examples Images has to be processed in the same way as MNIST before training. Steps involved in preprocessing are,

- Converting to binary image where the background is black and the symbols are in white
- Dilating by a 3 * 3 kernel
- Padding to 20 * 20 image while preserving the aspect ratio
- Padding to 28 * 28 image by centre of mass

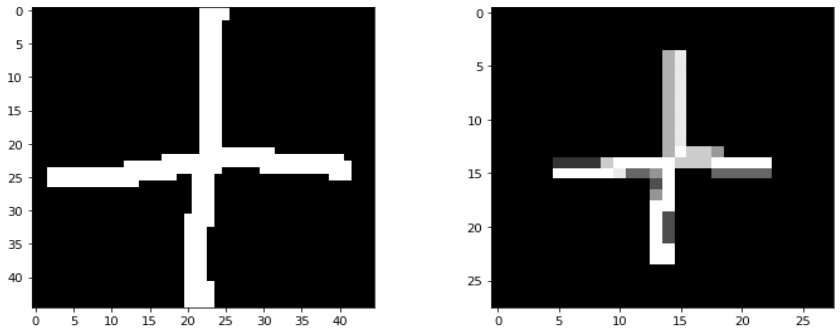

Figure 3: Crop and centered single math symbols

2.1.3 Training and Validation Split

The dataset is split into training and validation subsets in the ratio of 8:2 using tensorflow's image data generator.

```
train_datagen = ImageDataGenerator(rescale=1./255,

    data_format='channels_first',
    validation_split=0.2)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(28, 28),
    color_mode = "grayscale",
    batch_size=20,
    shuffle = True,
    classes = ['0','1','2','3','4','5','6','7','8','9','+','-','times','(',')'],
    class_mode="sparse",
    subset='training')

validation_generator = train_datagen.flow_from_directory(
    train_dir, # same directory as training data
    target_size=(28, 28),
    color_mode = "grayscale",
    batch_size=20,
    shuffle = True,
    classes = ['0','1','2','3','4','5','6','7','8','9','+','-','times','(',')'],
    class_mode="sparse",
    subset='validation')
```

Code snippet

2.2 Workspace Detection Module

Workspace detection model assumes that there are valid rectangular boxes in the given scanned worksheet. Steps involved in workspace detection are

- Finding closed object contours (Rectangular boxes)
- Sorting the contours (Top-to-Bottom) based on the coordinates
- Choosing the desired boxes based on the area.

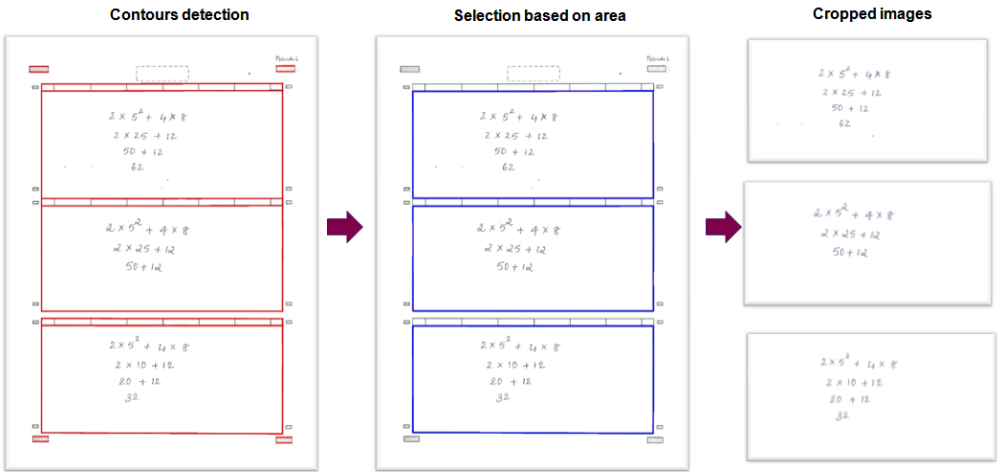

Figure 4: Workspace detection steps

2.3 Analysis Module Approaches

2.3.1 Line Detection using forward derivative

The approach for line detection assumes that there is a sufficient gap between lines and there is some intersection between exponential characters and line. The binary images of the detected workspaces are compressed in a single array to take forward derivative thereby detecting the coordinates of each line.

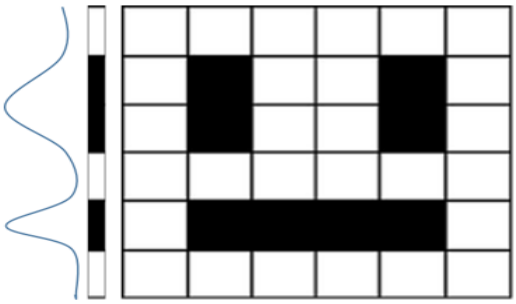

Figure 5 Line detection using forward derivative

2.3.2 Line Detection using OCRopus

OCRopus is a collection of document analysis programs, it performs the following steps

- Binarization
- Page-Layout Analysis
- Text-Line Recognition
- OCR

The default parameters and settings of OCRopus assume 300dpi binary black-on-white images. If images are scanned at a different resolution, the simplest thing to do is to

downscale/upscale them to 300dpi. The text line recognizer is fairly robust to different resolutions, but the layout analysis is quite resolution dependent.

Binarization

Steps involved in binarization are estimating skew angle, estimating thresholds and rescaling.

```
# perform binarization

!./ocropus-nlbin tests/image_3.jpg -o book
```

Page Layout Analysis

Page layout analysis removes horizontal and vertical lines. Performs text line finding, this identifies the tops and bottoms of text lines by computing gradients and performs some adaptive thresholding, those components are then used as seeds for the text-line recognition.

```
# perform page layout analysis

!./ocropus-gpageseg 'book/0001.bin.png'
```

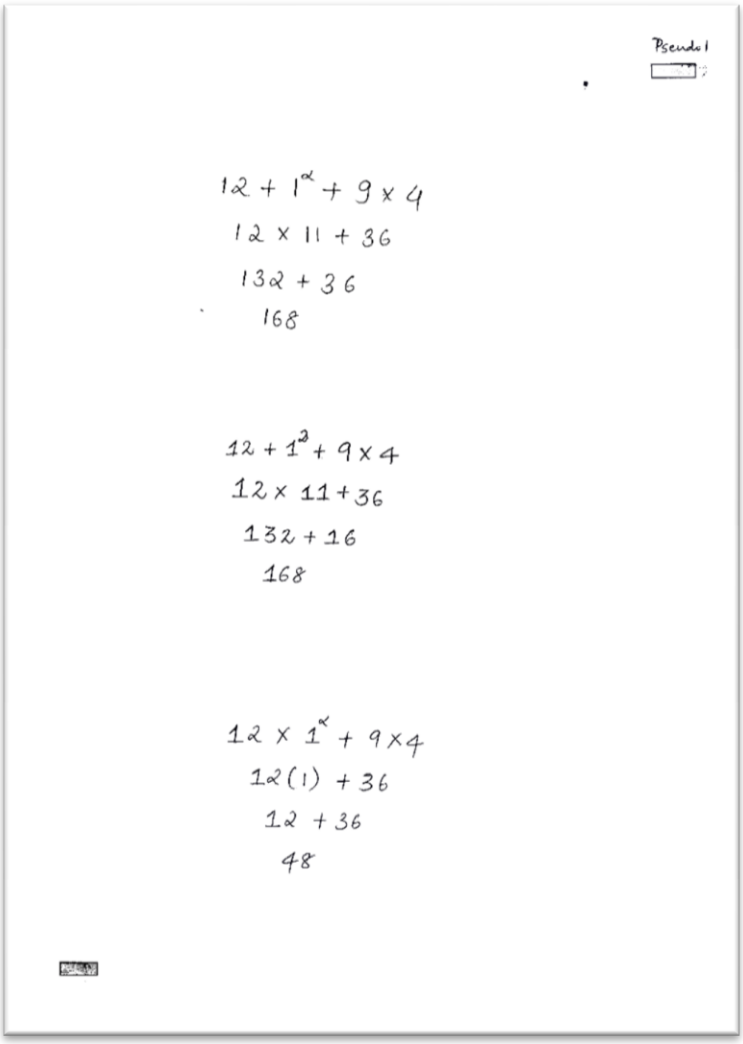

Figure 6 Page segmentation

Text line Recognition

Text line Recognition uses CLSTM. CLSTM is an implementation of the LSTM recurrent neural network model in C++, using Eigen library for numerical computations. After text line recognition each lines in the image are then extracted and saved as separate images with labels in '.txt' format.

Drawbacks in OCRopus

OCRopus cannot detect all-caps text, some special symbols (including "?"), typewriter fonts, and subscripts/superscripts.

2.3 Character Recognition Approaches

2.3.1 Deep Columnar Convolutional Neural Network

The proposed model is a single deep and wide neural network architecture that offers near state--of-the-art performance like ensemble models on various image classification challenges, such as MNIST, CIFAR-10 and CIFAR-100 datasets.

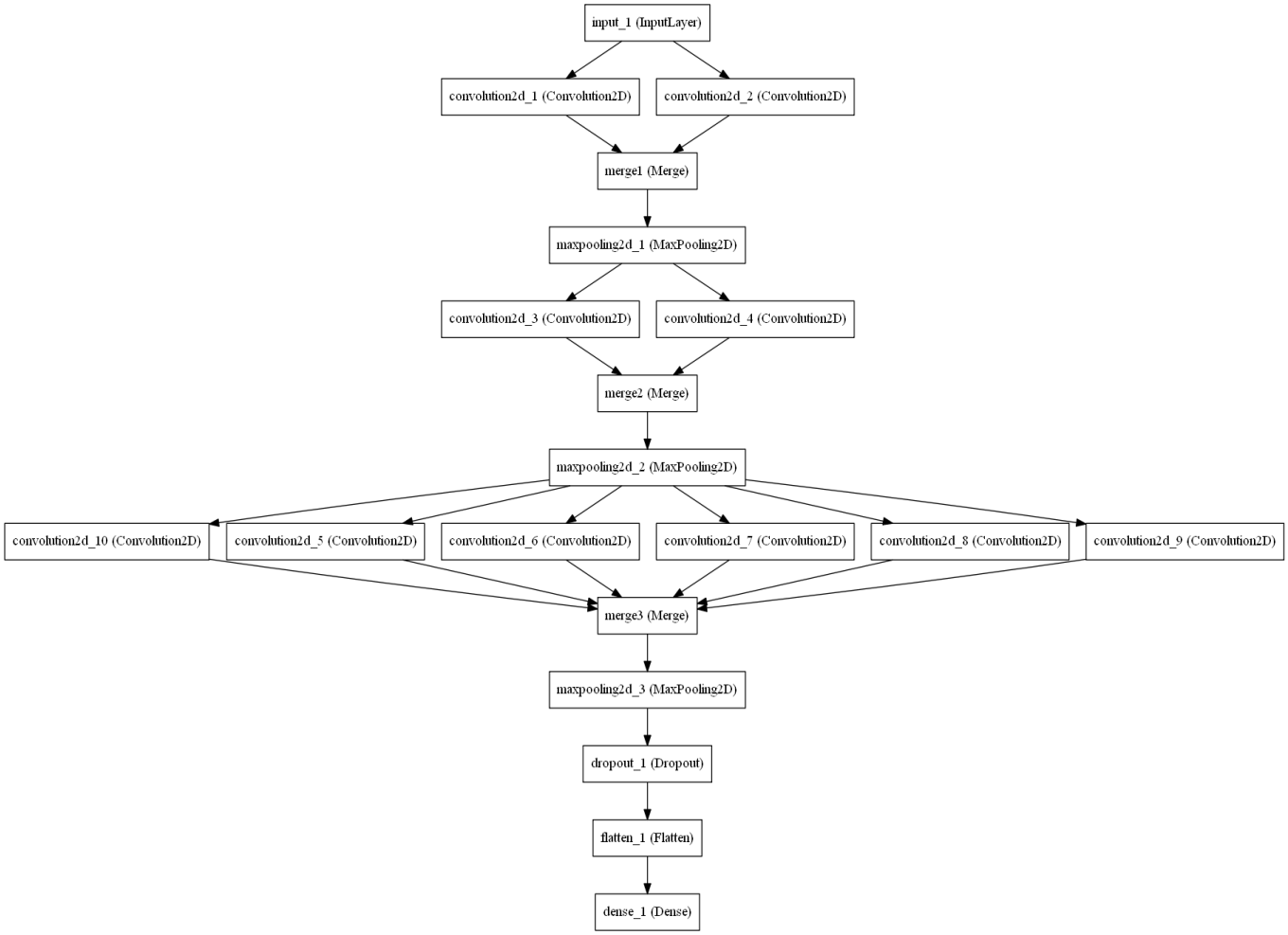

Figure 7: DCCNN Architecture

The important features of DCCNN is its

Wide architecture the proposed model utilizes a large number of maps per layer, stacked in both horizontal (fork) and vertical (merge) layers. These vertical layers allows the model to see two versions of input at the same time thereby preventing the network from loss of information.

Pooling via Convolutional Subsampling using maxpooling reduces the dimensions of the image thereby reducing the no of parameters but at the cost of losing some important features. Instead some of the maxpooling layers are replaced by convolutional layers with increased strides

Variable Kernel size All forked convolutional layers which accept a preprocessed image possess a 5x5 kernel. This improves the convergence speed and the accuracy of the network. All middle tier fork layers utilize either a 4x4 or 3x3 kernel, while simultaneously increasing the number of maps. The final tier fork and merge layer use a 3x3 kernel to improve performance, and often have the largest map size.

Data preparation for DCCNN:

Image pre-processing is performed prior to prediction for the extracted characters from scanned worksheet to match the training dataset.

- Binarizing the images
- Removing unnecessary noises
- Dilated and eroded by a 3 * 3 kernel
- Applied Gaussian Blur with sigma equals to 1
- Removing rows and columns where all the pixels are black
- After making the aspect ratio same padded to a 20 * 20 image
- Padded to 28 * 28 by center of mass

Training:

- **Activation Function** Softmax is used in the final layer, all other layers contain ReLu activation function.
- **Optimizers:**
  - AdaDelta optimizer is an adaptive learning rate method which requires no manual tuning of a learning rate performed well compared to other optimizers. The parameters used are,
    - **Learning Rate** = 1.0
    - **Rho** = 0.95 (decay factor)
- Batch Normalization is used to overcome vanishing gradient problem
- Dropout of 50 % is used before the final dense layer
- Model is trained for only 10 epochs with accuracy up to 96 %
- **Augmentation** only slightly improved accuracy in this case (Random rotations, width shift, height shift)
  - **Rotation degree** = 20
  - **Width shift** = 20% of image width
  - **Height shift** = 20% of image height

2.3.2 Tesseract

Tesseract 4 adds a new neural net (LSTM) based OCR engine which is focused on line recognition. It also has a unique configuration option for detecting equation region in the document.

Drawbacks in tesseract:

- Works best on clean and computer generated images.
- Incorrect detection of numbers.
- Unable to recognize the exponents of the base.
- Operators are confused between numbers.

- Unable to recognize decimal points.

We may need to retrain the tesseract on handwritten mathematical expressions to improve accuracy.

Training Tesseract:

Method

- Split the provided datasets into train / test datasets
- Run tesseract to establish base line performance of tesseract without any re-training or fine-tuning on the test datasets
- Parameters to be tuned –oem (OCREngine Mode), --psm, --lang (language)

Data preparation

- Tesseract accepts images in .tif format, and the ground truth values of each image has to prepared manually stored in groundtruth.csv file.
- Need to prepare a moderate size of train data that can demonstrate the re-train / fine-tune efficacy
- Need a boxfile which contains bounding box coordinates of single glyph or whole text line of each images

References

GitHub page for DCCNN – Contains the model architecture suitable for building tensorflow models also contains .h5 file for weights but there is a mis-match between the model architecture and the weights because of channels-first based building of model design in the architecture.

MNIST as jpgs - This page contains MNIST images in jpg format

Handwritten mathematical symbols dataset – Kaggle's dataset on handwritten mathematical symbols, extracted and modified from CROHME dataset

CROHME dataset - This dataset provides more than 11,000 expressions handwritten by hundreds of writers from different countries, merging the data sets from 4 CROHME competitions. Writers were asked to copy printed expressions from a corpus of expressions. The corpus has been designed to cover the diversity proposed by the different tasks and chosen from an existing math corpus and from expressions embedded in Wikipedia pages

HASY dataset – This dataset contains 150,000 instances of 369 handwritten symbols similar to MNIST

Recognition of Handwritten Textual Annotations using Tesseract Open Source OCR Engine – Paper on training tesseract for handwritten texts, (Note – mentioned that there will be 1 error in every 10 predictions)

https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract-4.00 - Training tesseract 4 detailed instructions

Multi-Scale Attention with Dense Encoder for Handwritten Mathematical Expression Recognition- Paper on Offline math formula recognition. Validated on CROHME dataset.

Image-to-Markup Generation with Coarse-to-Fine Attention – This paper presents attention based encoder-decoder model to convert images to Latex. Validated on CROHME dataset.

https://github.com/handong1587/handong1587.github.io/blob/master/_posts/deep_learning/2015-10-09-ocr.md

https://github.com/pannous/caffe-ocr/tree/master/training_images

https://github.com/pannous/tensorflow-ocr

https://github.com/quasiris/EAST

https://github.com/tmbdev/ocropy

danvk.org/2015/01/09/extracting-text-from-an-image-using-ocropus.html

https://github.com/chongyangtao/DeepHCCR

https://github.com/chongyangtao/Awesome-Scene-Text-Recognition

https://www.azoft.com/blog/ocr-receipt-recognition/