RV Educational Institutions ®
RV College of Engineering ®

Autonomous Institution Affiliated to Visvesvaraya Technological University, Belagavi

Approved by AICTE, New Delhi, Accredited By NAAC, Bengaluru And NBA, New Delhi

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

"AUTO GRADATION OF HANDWRITTEN MATHEMATICAL ANSWER SHEETS"

Minor Project (18IS64)

Submitted By

Adamyaa D N (1RV18IS002)

Ananya G M (1RV18IS006)

Varshini P (1RV18IS058)

Under the Guidance of

S. G. Raghavendra Prasad

Assistant Professor, ISE

In partial fulfillment for the award of the degree
of
Bachelor of Engineering
in
INFORMATION SCIENCE AND ENGINEERING
2021

CHAPTER-1

INTRODUCTION

CHAPTER 1
INTRODUCTION

Grading is an essential part of education. Assessing each answer sheet manually, offering fair, unbiased and valid grades is difficult most of the time. Although the printed text recognition is considered as a clarified issue, handwritten text recognition remains as a demanding task.

The proposed system attempts to develop a computer vision algorithm along with a 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.

The overall idea behind this project is to develop a computer vision algorithm along with a 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.

Optical Character Recognition in the field of Natural Language Processing is one of the budding technologies in the field of information science today. Automating frequent sequences of actions in everyday life using various machine algorithms, hence an amazing leap of advancements can be observed in the field of data science.Computer Vision and Convolutional Neural Networks stand as the fundamental pillars of Artificial Intelligence and Machine Learning.

The following are the broad modules that 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 a red/green bounding box drawn across it where green represents correct and red represents wrong answers.

CHAPTER-2

LITERATURE SURVEY

CHAPTER 2

LITERATURE SURVEY

Although the printed text recognition is considered a clarified issue these days, handwritten text recognition remains a demanding task, mainly due to the huge variation in handwriting among certain people including the size, orientation, thickness, format, and dimension of each written letter or digit.

A parallel study of different approaches to each of the modules as described below has been conducted. For the analysis of the workspace, detection of lines can be done either with the approach of forward derivative or using OCRopus. Similarly, the recognition of characters can be done using the Deep Columnar Convolutional Neural Network model trained on the MNIST dataset or using Tesseract, both of whose accuracy needs to be measured for our proposed model. Research papers concerning contour detection, binarization, extraction and recognition of characters, and other mathematical symbols have been studied and the methods mentioned will be considered in the implementation of the project.

Different universities have taken different approaches to achieve the objective. The most commendable results were obtained from the authors of Automatically Solving Number Word Problems by Semantic Parsing and Reasoning who designed a language called Dolphin Language for semantic representation. The implementation of Context Free Parsing for semantic parsing along with KAZB and Basic Sum methods proved to be very beneficial by giving about 95% accuracy.

The following table consolidates the findings of different papers published in the field.

TITLE	AUTHOR & YEAR OF PUBLICATION	OBJECTIVE	DATA	METHODOLOGY	CONCLUSION
Automated Grading for Handwritten Answer Sheets using CNN	Emran Shaikh, Ayisha Manzoor, Ghazanfar Latif & Iman Ahmed Mohiuddin. October 2019.	The objective was to automatically grade handwritten answer sheets.	250 students were given answer sheets with 20 questions each, obtained a total of 5080 segmented images.	Model 1 consisted of 4 CNN layers and 3 dense layers. Model 2 consisted of 4 CNN layers and 4 dense layers.	Model 1 produced 92.866 and model 2 produced 92.322 accuracy.
Automatically Solving Number Word Problems by Semantic Parsing and Reasoning	Shuming Shi , Yuehui Wang , Chin-Yew Lin , Xiaojiang Liu and Yong Rui.	The objective was to solve number word problems automatically and produce accurate answers	The problems set contained 1,878 math number word problems, collected from two web sites: algebra.com6 and answers.yahoo .com7	DOL language was used for semantic representation. CFG was used as a semantic parser. KAZB and Basicsum baseline methods were implemented.	The approach achieves a particularly high precision of 95%. That means once an answer is provided by our approach, it has a very high probability of being correct.

Deep Columnar Convolutional Neural Network	Somshubra Majumdar, Ishaan Jain. July 2016.	The objective was to observe the performance of DCNN model and compare it with the performance of other top models.	MNIST dataset - It is a dataset of 60,000 small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9.	Deep Convolutional Neural Networks are used. Adadelta learning algorithm used as optimizer. Works on MNIST dataset.	DCNN method has near state of the art performance using only a single model and obtains slightly better results.

Table 2.1 : Literature Survey

CHAPTER-3
IMPLEMENTATION

CHAPTER 3
IMPLEMENTATION

3.1 HARDWARE AND SOFTWARE REQUIREMENTS

1. Ubuntu 16.04 and above, Windows 10, MacOS
2. CPU Freq. 2.30GHz
3. Google Colaboratory

3.2 Dataset

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','(',')']

3.2.1 MNIST Dataset

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.

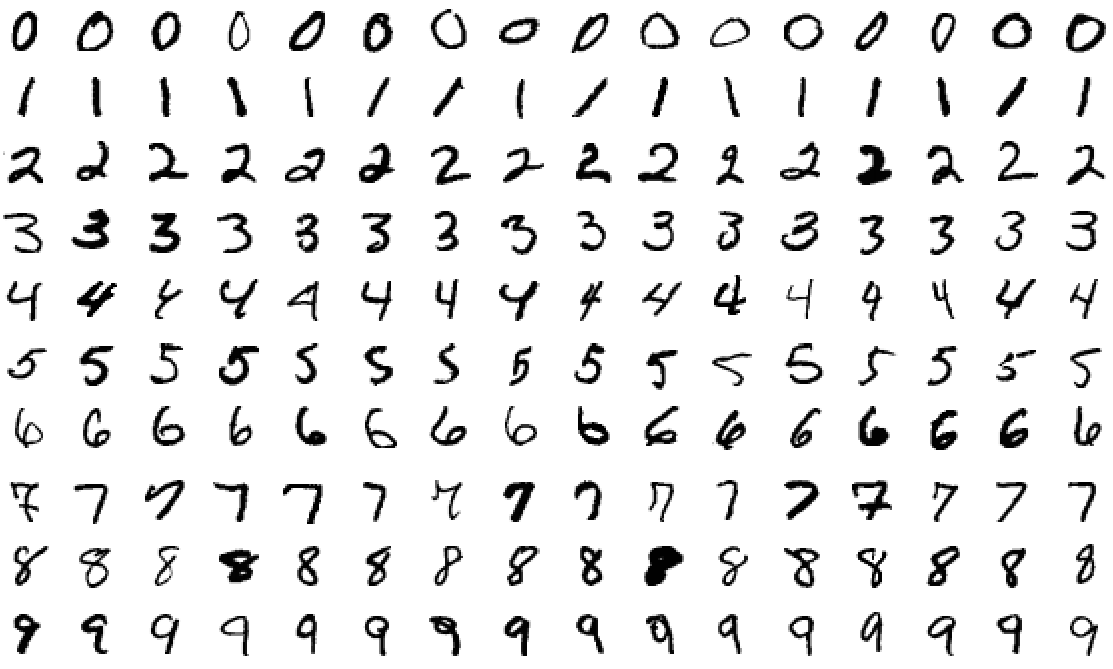


Fig 3.1 : MNIST Dataset

3.2.2 Kaggle's handwritten mathematical worksheets

This datasets include 82 symbols but only a few symbols such as "+", "-", "*", "(", ")" are selected. Each symbol contains at most 4000 examples. Images have 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

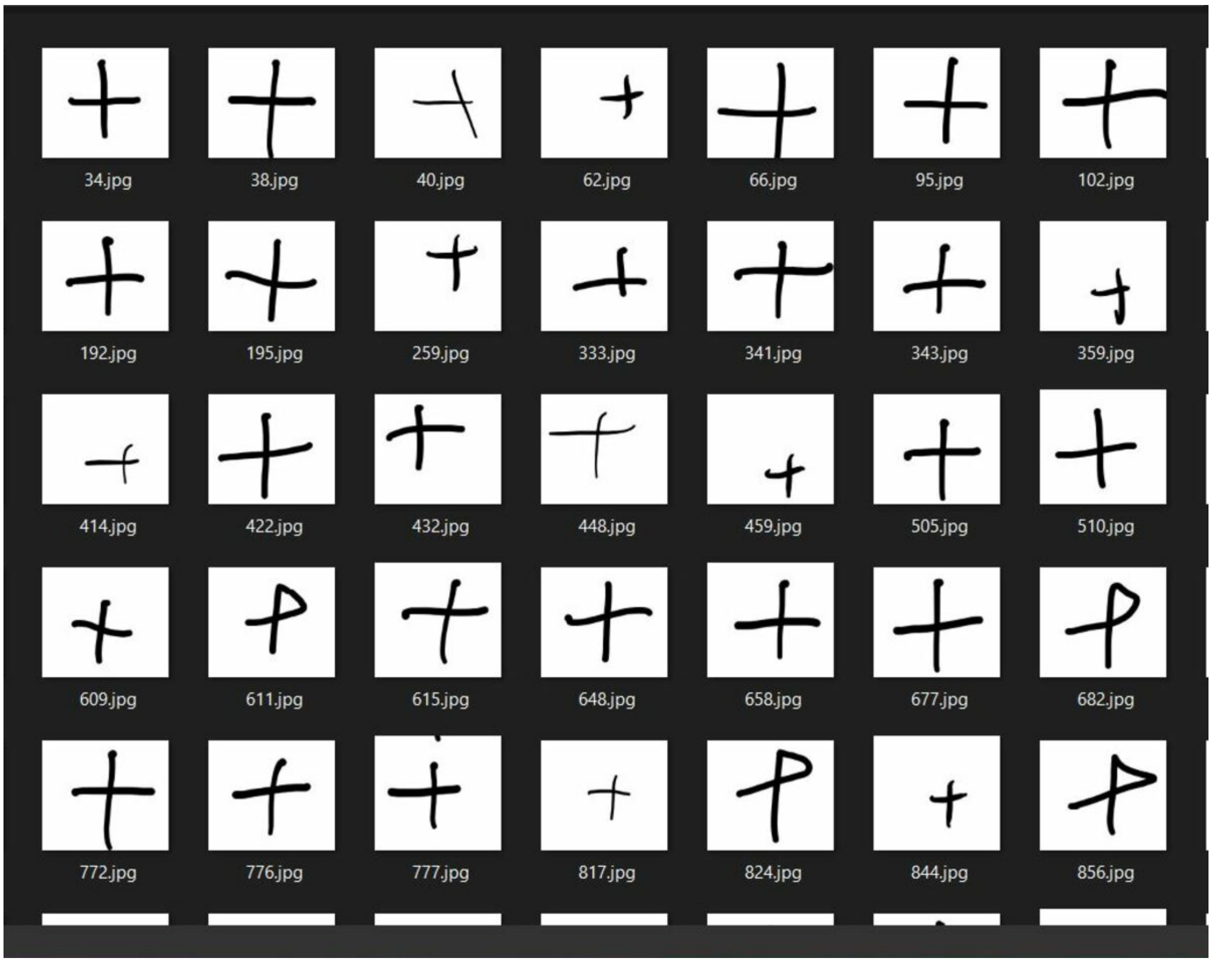
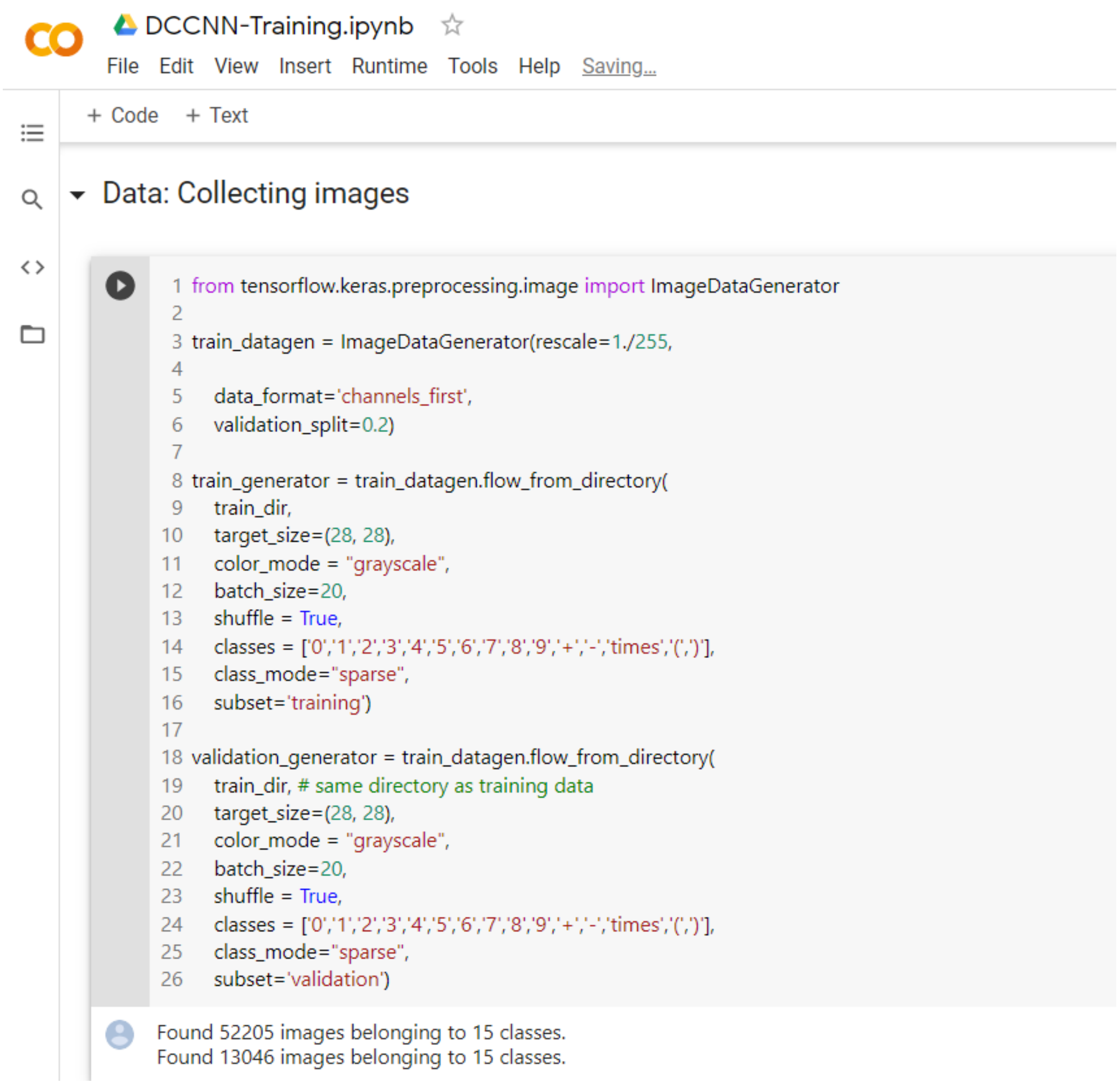


Fig 3.2 : Kaggle Dataset

3.2.3 Model Training

The dataset is split into training and validation subsets in the ratio of 8:2 using tensorflow's image data generator. The model is trained with 52205 images split into 15 classes, validated with 13046 images again split into 15 classes. The model consists of 3 layers of neural networks. The first layer consists of 2 fork layers, 1 merge layer and 1 maxpool layer with 1 filter layer and RELu as the activation function. The second layer is similar to the first layer but has 2 filter layers in both the fork layers unlike layer 1. The third layer consists of 6 fork layers and one merge layer. The dropout value is set to be 0.5. This is passed through a flatten layer with activation function softmax.



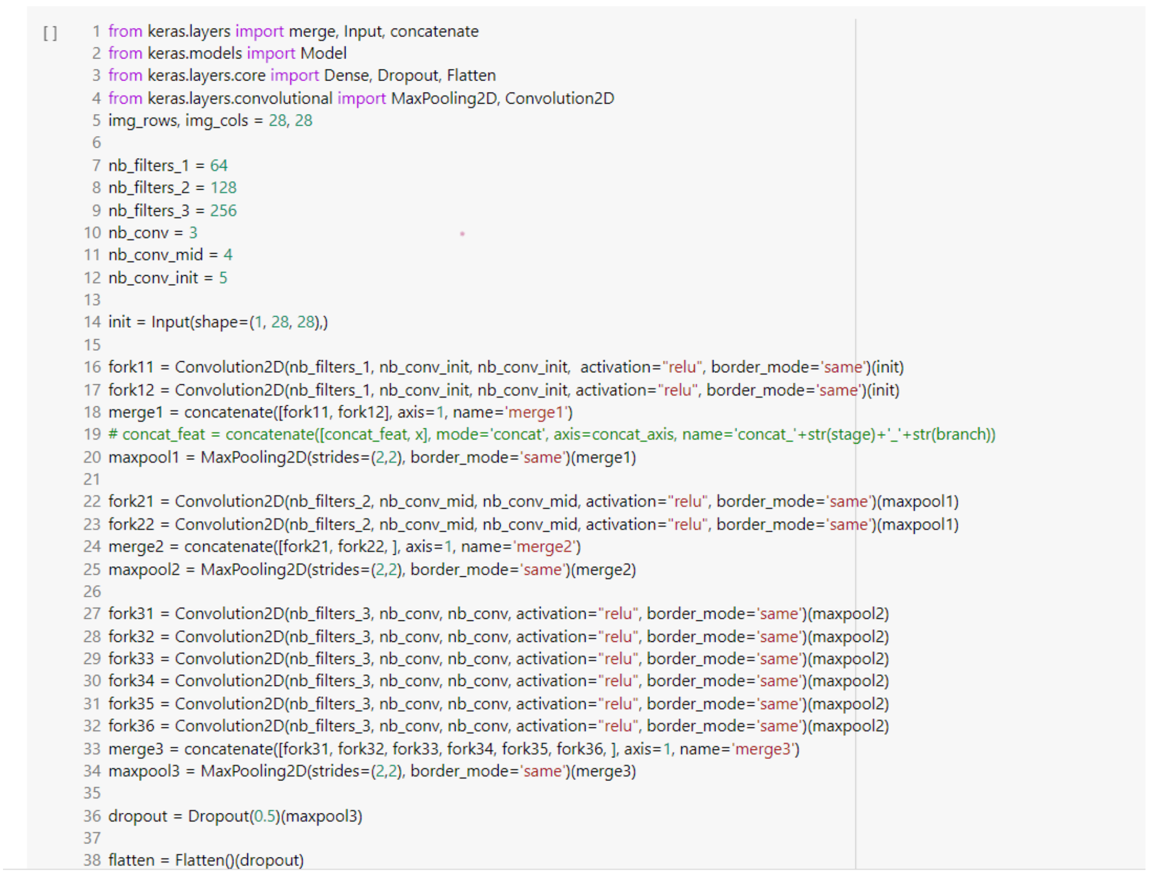
Data: Collecting images

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

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')
```

```
Found 52205 images belonging to 15 classes.
Found 13046 images belonging to 15 classes.
```

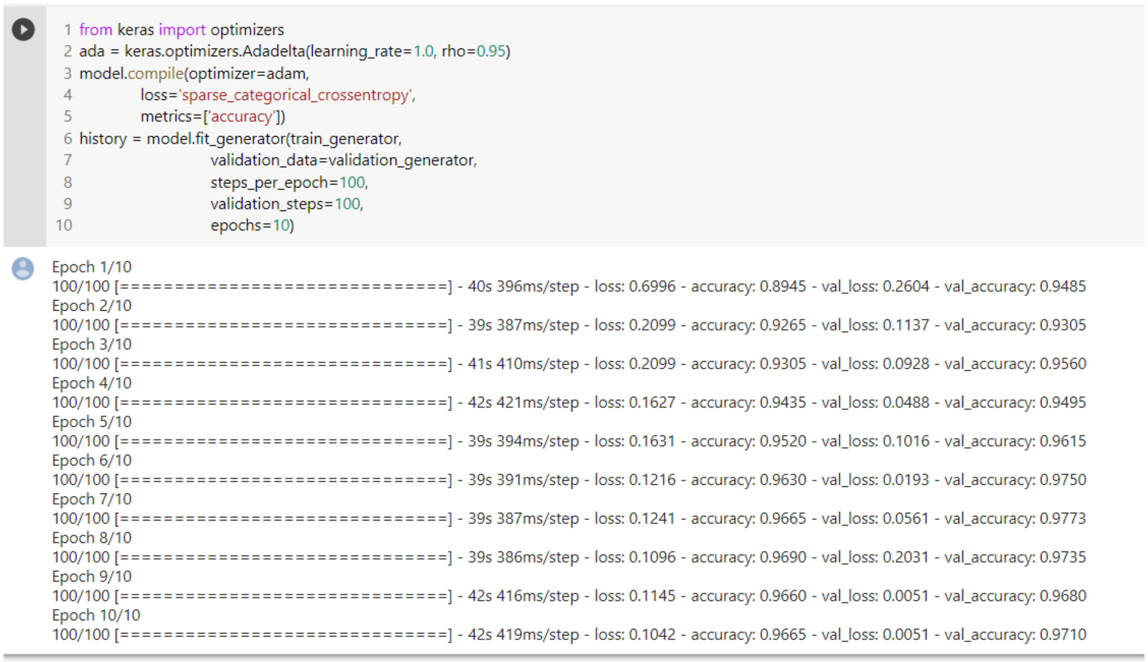
Fig 3.3 : Train and Test Split

Deep Learning Model : DCCNN



```
[ ]
from keras.layers import merge, Input, concatenate
from keras.models import Model
from keras.layers.core import Dense, Dropout, Flatten
from keras.layers.convolutional import MaxPooling2D, Convolution2D
img_rows, img_cols = 28, 28

nb_filters_1 = 64
nb_filters_2 = 128
nb_filters_3 = 256
nb_conv = 3
nb_conv_mid = 4
nb_conv_init = 5

init = Input(shape=(1, 28, 28),)

fork11 = Convolution2D(nb_filters_1, nb_conv_init, nb_conv_init,  activation="relu", border_mode='same')(init)
fork12 = Convolution2D(nb_filters_1, nb_conv_init, nb_conv_init, activation="relu", border_mode='same')(init)
merge1 = concatenate([fork11, fork12], axis=1, name='merge1')
# concat_feat = concatenate([concat_feat, x], mode='concat', axis=concat_axis, name='concat_'+str(stage)+'_'+str(branch))
maxpool1 = MaxPooling2D(strides=(2,2), border_mode='same')(merge1)

fork21 = Convolution2D(nb_filters_2, nb_conv_mid, nb_conv_mid, activation="relu", border_mode='same')(maxpool1)
fork22 = Convolution2D(nb_filters_2, nb_conv_mid, nb_conv_mid, activation="relu", border_mode='same')(maxpool1)
merge2 = concatenate([fork21, fork22, ], axis=1, name='merge2')
maxpool2 = MaxPooling2D(strides=(2,2), border_mode='same')(merge2)

fork31 = Convolution2D(nb_filters_3, nb_conv, nb_conv, activation="relu", border_mode='same')(maxpool2)
fork32 = Convolution2D(nb_filters_3, nb_conv, nb_conv, activation="relu", border_mode='same')(maxpool2)
fork33 = Convolution2D(nb_filters_3, nb_conv, nb_conv, activation="relu", border_mode='same')(maxpool2)
fork34 = Convolution2D(nb_filters_3, nb_conv, nb_conv, activation="relu", border_mode='same')(maxpool2)
fork35 = Convolution2D(nb_filters_3, nb_conv, nb_conv, activation="relu", border_mode='same')(maxpool2)
fork36 = Convolution2D(nb_filters_3, nb_conv, nb_conv, activation="relu", border_mode='same')(maxpool2)
merge3 = concatenate([fork31, fork32, fork33, fork34, fork35, fork36, ], axis=1, name='merge3')
maxpool3 = MaxPooling2D(strides=(2,2), border_mode='same')(merge3)

dropout = Dropout(0.5)(maxpool3)

flatten = Flatten()(dropout)
```

Fig 3.4 : DCCNN Layers



```
from keras import optimizers
ada = keras.optimizers.Adadelta(learning_rate=1.0, rho=0.95)
model.compile(optimizer=adam,
          loss='sparse_categorical_crossentropy',
          metrics=['accuracy'])
history = model.fit_generator(train_generator,
                    validation_data=validation_generator,
                    steps_per_epoch=100,
                    validation_steps=100,
                    epochs=10)
```

```
Epoch 1/10
100/100 [==============================] - 40s 396ms/step - loss: 0.6996 - accuracy: 0.8945 - val_loss: 0.2604 - val_accuracy: 0.9485
Epoch 2/10
100/100 [==============================] - 39s 387ms/step - loss: 0.2099 - accuracy: 0.9265 - val_loss: 0.1137 - val_accuracy: 0.9305
Epoch 3/10
100/100 [==============================] - 41s 410ms/step - loss: 0.2099 - accuracy: 0.9305 - val_loss: 0.0928 - val_accuracy: 0.9560
Epoch 4/10
100/100 [==============================] - 42s 421ms/step - loss: 0.1627 - accuracy: 0.9435 - val_loss: 0.0488 - val_accuracy: 0.9495
Epoch 5/10
100/100 [==============================] - 39s 394ms/step - loss: 0.1631 - accuracy: 0.9520 - val_loss: 0.1016 - val_accuracy: 0.9615
Epoch 6/10
100/100 [==============================] - 39s 391ms/step - loss: 0.1216 - accuracy: 0.9630 - val_loss: 0.0193 - val_accuracy: 0.9750
Epoch 7/10
100/100 [==============================] - 39s 387ms/step - loss: 0.1241 - accuracy: 0.9665 - val_loss: 0.0561 - val_accuracy: 0.9773
Epoch 8/10
100/100 [==============================] - 39s 386ms/step - loss: 0.1096 - accuracy: 0.9690 - val_loss: 0.2031 - val_accuracy: 0.9735
Epoch 9/10
100/100 [==============================] - 42s 416ms/step - loss: 0.1145 - accuracy: 0.9660 - val_loss: 0.0051 - val_accuracy: 0.9680
Epoch 10/10
100/100 [==============================] - 42s 419ms/step - loss: 0.1042 - accuracy: 0.9665 - val_loss: 0.0051 - val_accuracy: 0.9710
```

Validation Accuracy (10th Epoch) : ~97%

Fig 3.5 : Accuracy of the model

3.3 Workspace Detection

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.

3.4 Analysis Module

3.4.1 Line Detection using forward detective

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

3.4.2 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.

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.

Text line Recognition

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

3.5 Character Recognition

3.5.1 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.

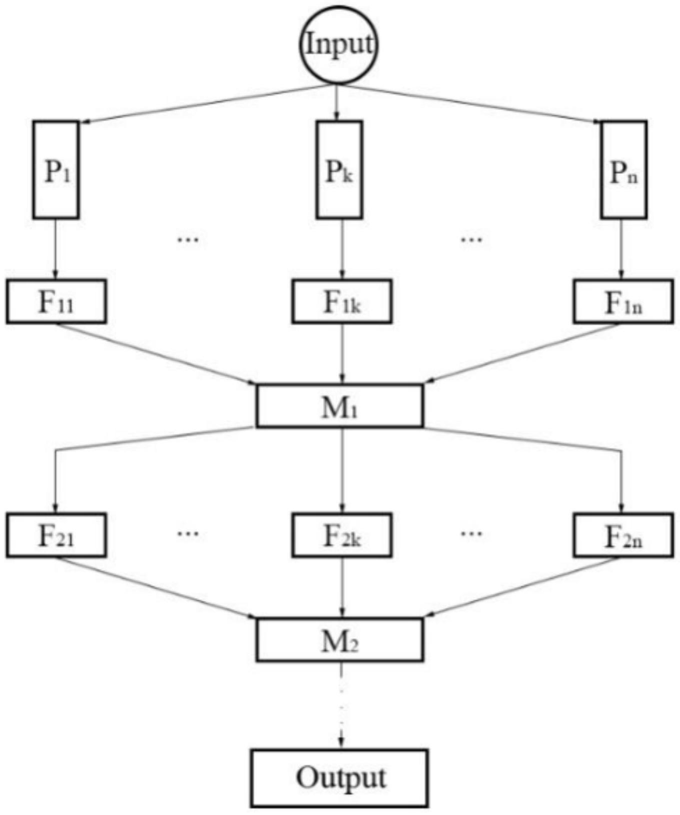


Fig 3.6 : DCCNN Model

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 allow 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 max pooling reduces the dimensions of the image thereby reducing the number of parameters but at the cost of losing some important features. Instead some of the max pooling 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 the 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 the ReLu activation function.
- Optimizers: o 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

3.6 Result

- The result of the equation is indicated by colored bounding boxes

  Red bounding boxes - Incorrect answer

  Green bounding box - Correct answer

  Blue bounding box - Undetermined (Needs manual evaluation)

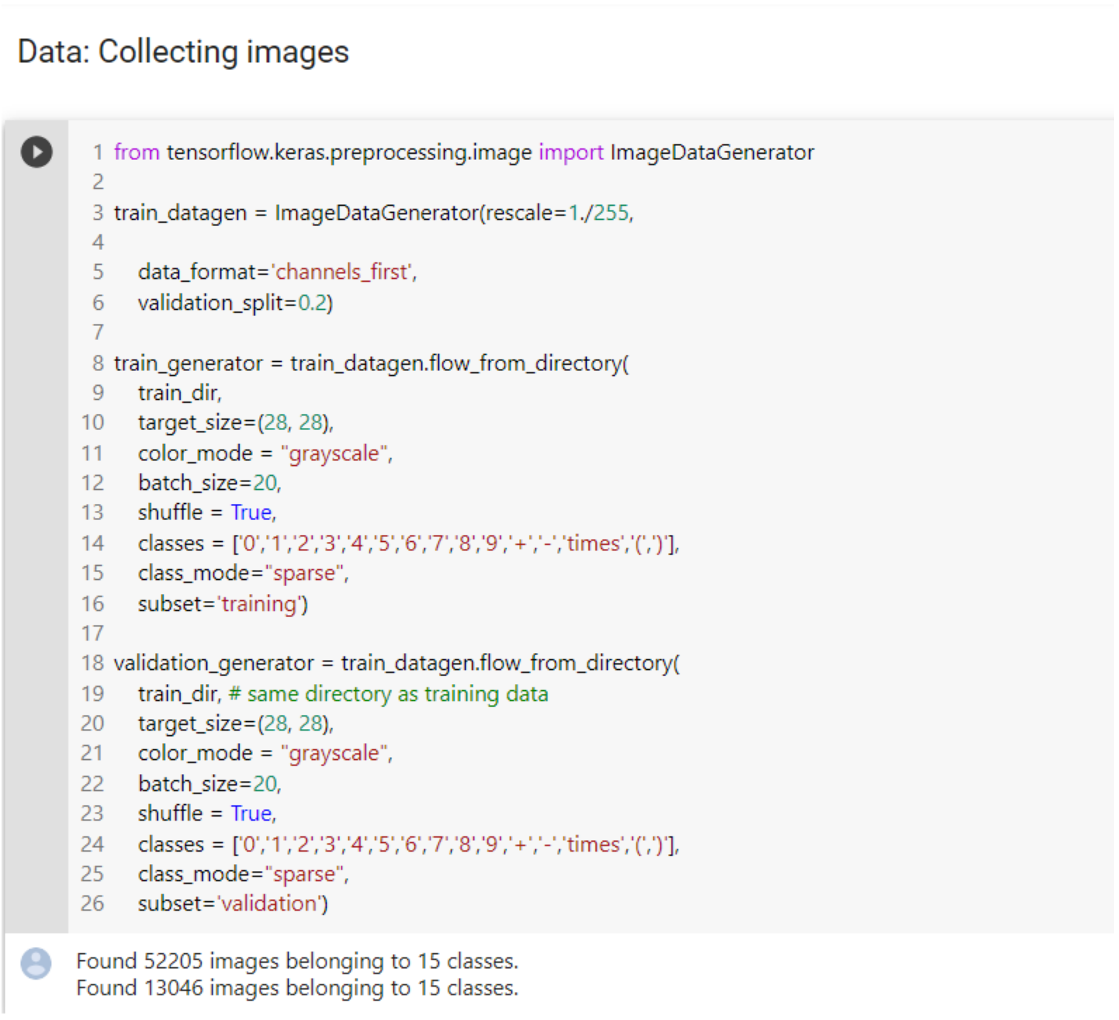
CHAPTER-4

TESTING

CHAPTER 4

TESTING

- **Test Suite 1 : DCCNN Training**
    - **Training and Validation Split**



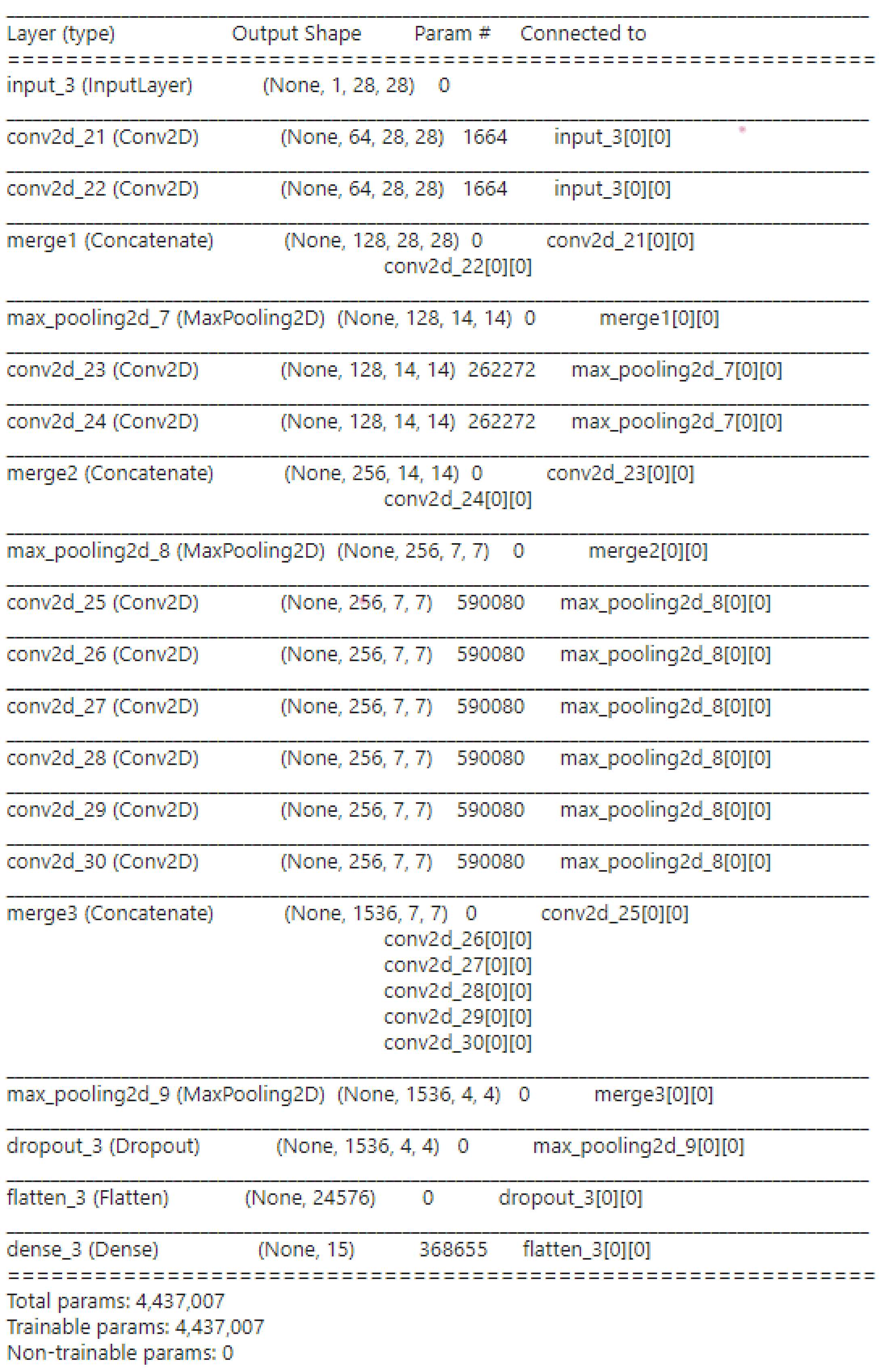
Data: Collecting images

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

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')
```

```
Found 52205 images belonging to 15 classes.
Found 13046 images belonging to 15 classes.
```

Fig 4.1 : Training and Testing split

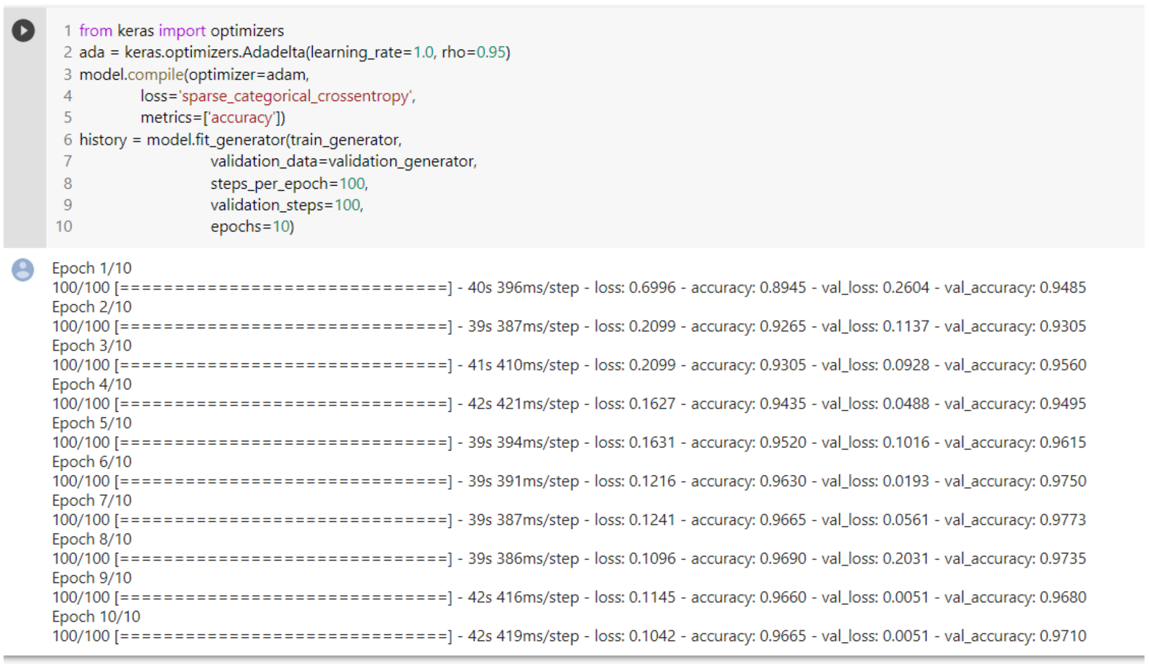
- **Parameter Training**



```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_3 (InputLayer)             (None, 1, 28, 28)     0
____________________________________________________________________________________________________
conv2d_21 (Conv2D)               (None, 64, 28, 28)    1664        input_3[0][0]
____________________________________________________________________________________________________
conv2d_22 (Conv2D)               (None, 64, 28, 28)    1664        input_3[0][0]
____________________________________________________________________________________________________
merge1 (Concatenate)             (None, 128, 28, 28)   0           conv2d_21[0][0]
                                                                   conv2d_22[0][0]
____________________________________________________________________________________________________
max_pooling2d_7 (MaxPooling2D)   (None, 128, 14, 14)   0           merge1[0][0]
____________________________________________________________________________________________________
conv2d_23 (Conv2D)               (None, 128, 14, 14)   262272      max_pooling2d_7[0][0]
____________________________________________________________________________________________________
conv2d_24 (Conv2D)               (None, 128, 14, 14)   262272      max_pooling2d_7[0][0]
____________________________________________________________________________________________________
merge2 (Concatenate)             (None, 256, 14, 14)   0           conv2d_23[0][0]
                                                                   conv2d_24[0][0]
____________________________________________________________________________________________________
max_pooling2d_8 (MaxPooling2D)   (None, 256, 7, 7)     0           merge2[0][0]
____________________________________________________________________________________________________
conv2d_25 (Conv2D)               (None, 256, 7, 7)     590080      max_pooling2d_8[0][0]
____________________________________________________________________________________________________
conv2d_26 (Conv2D)               (None, 256, 7, 7)     590080      max_pooling2d_8[0][0]
____________________________________________________________________________________________________
conv2d_27 (Conv2D)               (None, 256, 7, 7)     590080      max_pooling2d_8[0][0]
____________________________________________________________________________________________________
conv2d_28 (Conv2D)               (None, 256, 7, 7)     590080      max_pooling2d_8[0][0]
____________________________________________________________________________________________________
conv2d_29 (Conv2D)               (None, 256, 7, 7)     590080      max_pooling2d_8[0][0]
____________________________________________________________________________________________________
conv2d_30 (Conv2D)               (None, 256, 7, 7)     590080      max_pooling2d_8[0][0]
____________________________________________________________________________________________________
merge3 (Concatenate)             (None, 1536, 7, 7)    0           conv2d_25[0][0]
                                                                   conv2d_26[0][0]
                                                                   conv2d_27[0][0]
                                                                   conv2d_28[0][0]
                                                                   conv2d_29[0][0]
                                                                   conv2d_30[0][0]
____________________________________________________________________________________________________
max_pooling2d_9 (MaxPooling2D)   (None, 1536, 4, 4)    0           merge3[0][0]
____________________________________________________________________________________________________
dropout_3 (Dropout)              (None, 1536, 4, 4)    0           max_pooling2d_9[0][0]
____________________________________________________________________________________________________
flatten_3 (Flatten)              (None, 24576)         0           dropout_3[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 15)            368655      flatten_3[0][0]
====================================================================================================
Total params: 4,437,007
Trainable params: 4,437,007
Non-trainable params: 0
```

Fig 4.2 : Parameter Training

- **Accuracy**



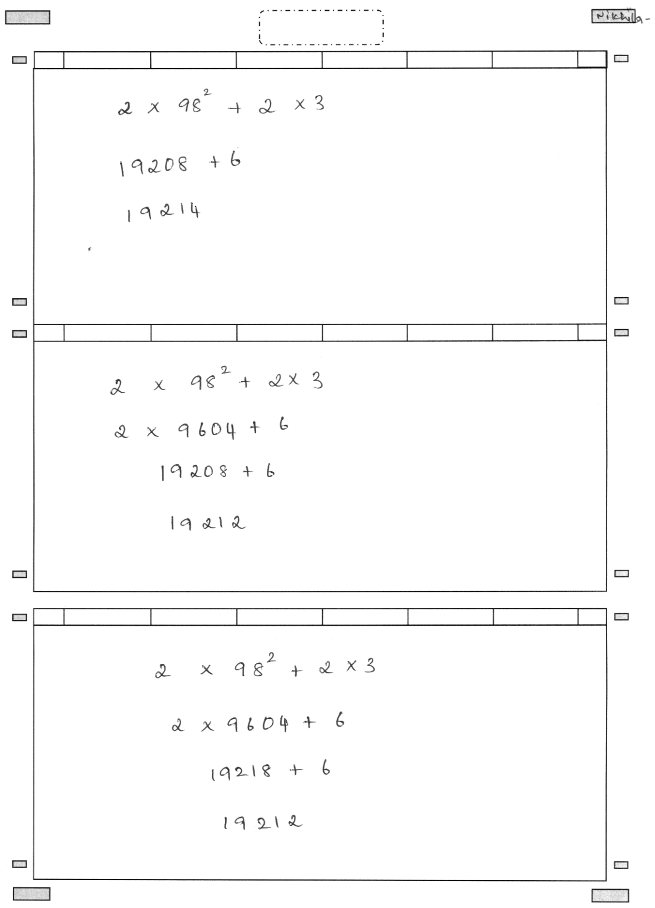
```
from keras import optimizers
ada = keras.optimizers.Adadelta(learning_rate=1.0, rho=0.95)
model.compile(optimizer=adam,
         loss='sparse_categorical_crossentropy',
         metrics=['accuracy'])
history = model.fit_generator(train_generator,
                   validation_data=validation_generator,
                   steps_per_epoch=100,
                   validation_steps=100,
                   epochs=10)
```

```
Epoch 1/10
100/100 [==============================] - 40s 396ms/step - loss: 0.6996 - accuracy: 0.8945 - val_loss: 0.2604 - val_accuracy: 0.9485
Epoch 2/10
100/100 [==============================] - 39s 387ms/step - loss: 0.2099 - accuracy: 0.9265 - val_loss: 0.1137 - val_accuracy: 0.9305
Epoch 3/10
100/100 [==============================] - 41s 410ms/step - loss: 0.2099 - accuracy: 0.9305 - val_loss: 0.0928 - val_accuracy: 0.9560
Epoch 4/10
100/100 [==============================] - 42s 421ms/step - loss: 0.1627 - accuracy: 0.9435 - val_loss: 0.0488 - val_accuracy: 0.9495
Epoch 5/10
100/100 [==============================] - 39s 394ms/step - loss: 0.1631 - accuracy: 0.9520 - val_loss: 0.1016 - val_accuracy: 0.9615
Epoch 6/10
100/100 [==============================] - 39s 391ms/step - loss: 0.1216 - accuracy: 0.9630 - val_loss: 0.0193 - val_accuracy: 0.9750
Epoch 7/10
100/100 [==============================] - 39s 387ms/step - loss: 0.1241 - accuracy: 0.9665 - val_loss: 0.0561 - val_accuracy: 0.9773
Epoch 8/10
100/100 [==============================] - 39s 386ms/step - loss: 0.1096 - accuracy: 0.9690 - val_loss: 0.2031 - val_accuracy: 0.9735
Epoch 9/10
100/100 [==============================] - 42s 416ms/step - loss: 0.1145 - accuracy: 0.9660 - val_loss: 0.0051 - val_accuracy: 0.9680
Epoch 10/10
100/100 [==============================] - 42s 419ms/step - loss: 0.1042 - accuracy: 0.9665 - val_loss: 0.0051 - val_accuracy: 0.9710
```

Validation Accuracy (10th Epoch) : ~97%

Fig 4.3 : Accuracy of the model

- **Test Suite 2 : Workspace Detection**



Nikhila-

$2 \times 98^2 + 2 \times 3$

$19208 + 6$

19214

$2 \times 98^2 + 2 \times 3$

$2 \times 9604 + 6$

$19208 + 6$

19212

$2 \times 98^2 + 2 \times 3$

$2 \times 9604 + 6$

$19218 + 6$

19212

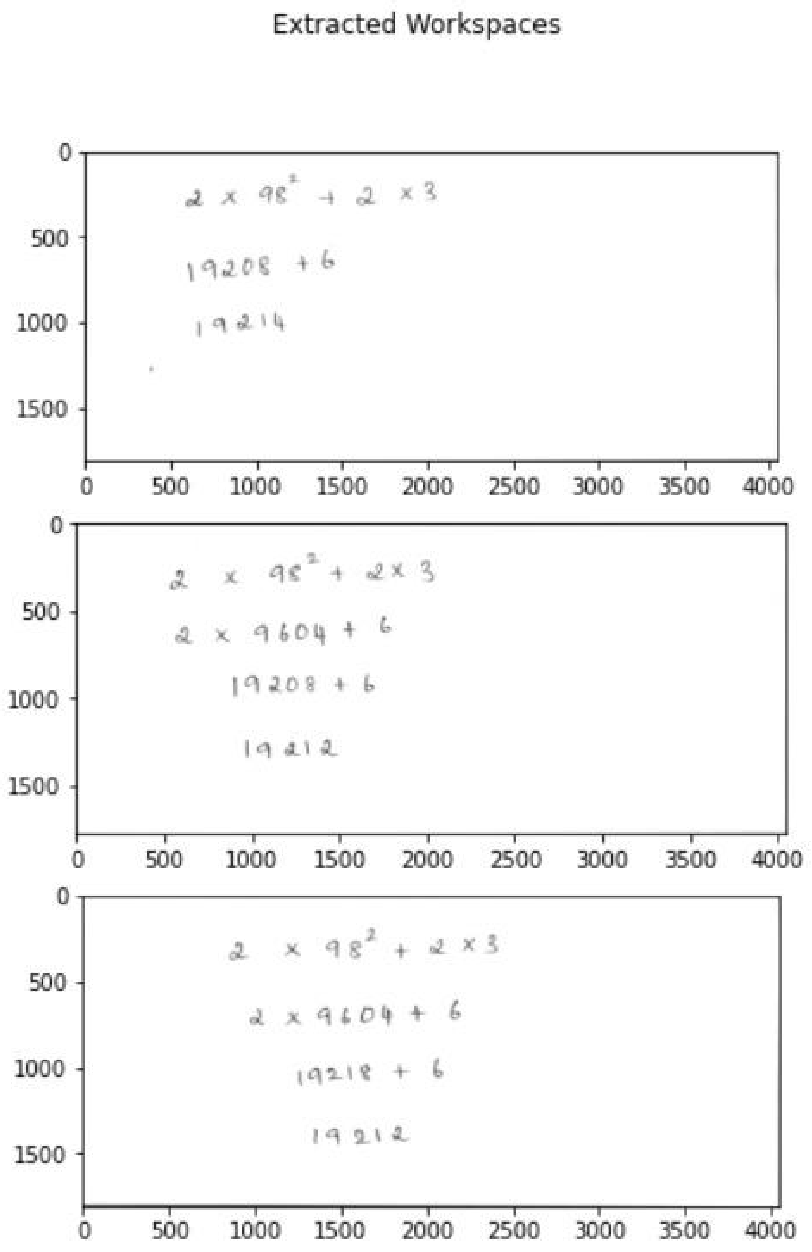


Fig 4.4 : Rectangle Formation

- **Test Suite 3 : Line Detection**

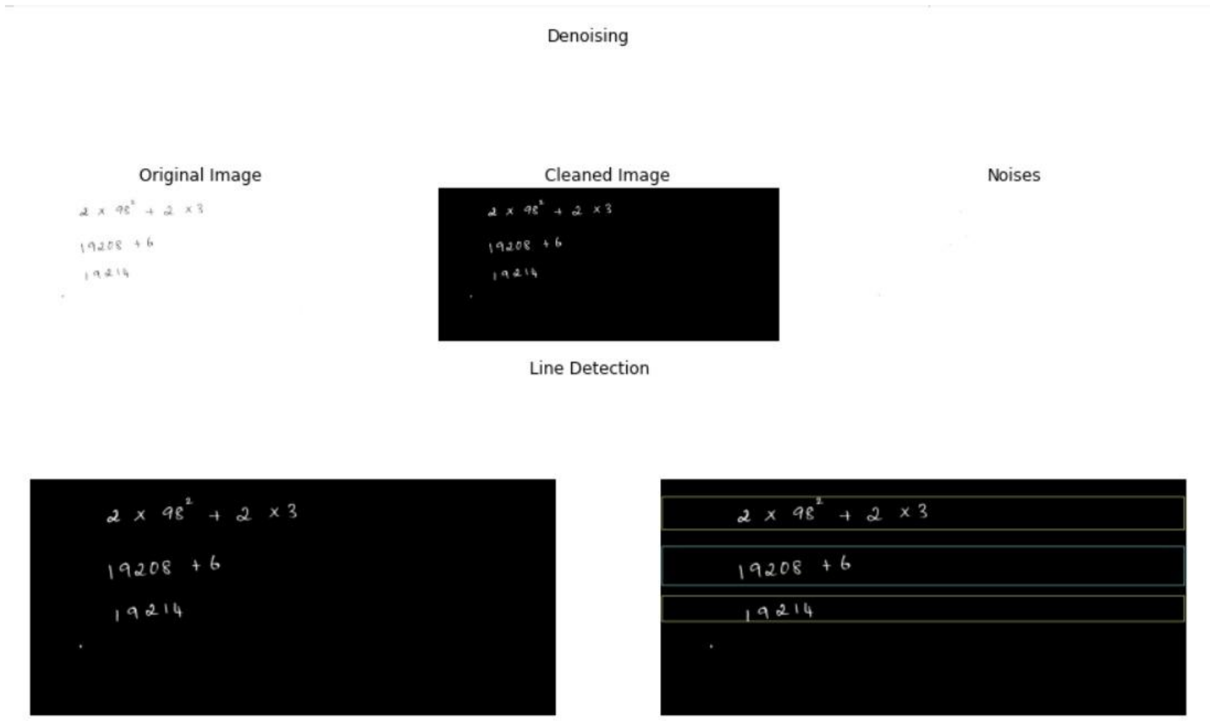


Fig 4.5 : Denoising and Line Detection Stage 1

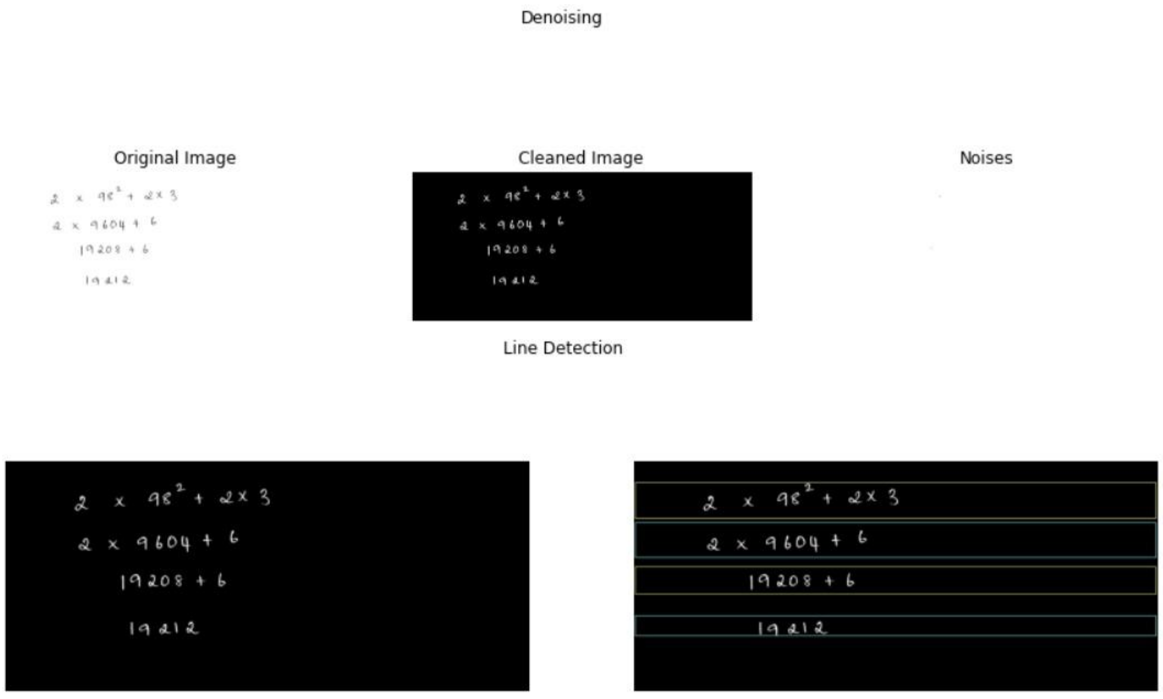


Fig 4.6 : Denoising and Line Detection Stage 2

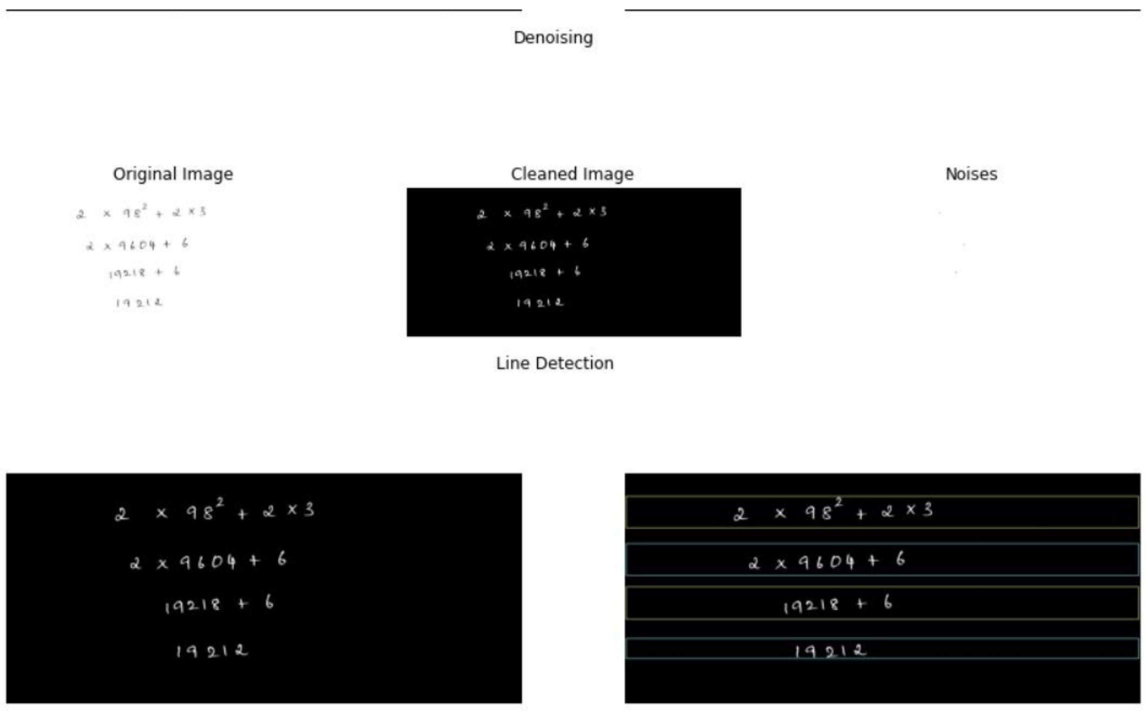


Fig 4.7 : Denoising and Line Detection Stage3

- **Test Suite 4 : Character Segmentation**

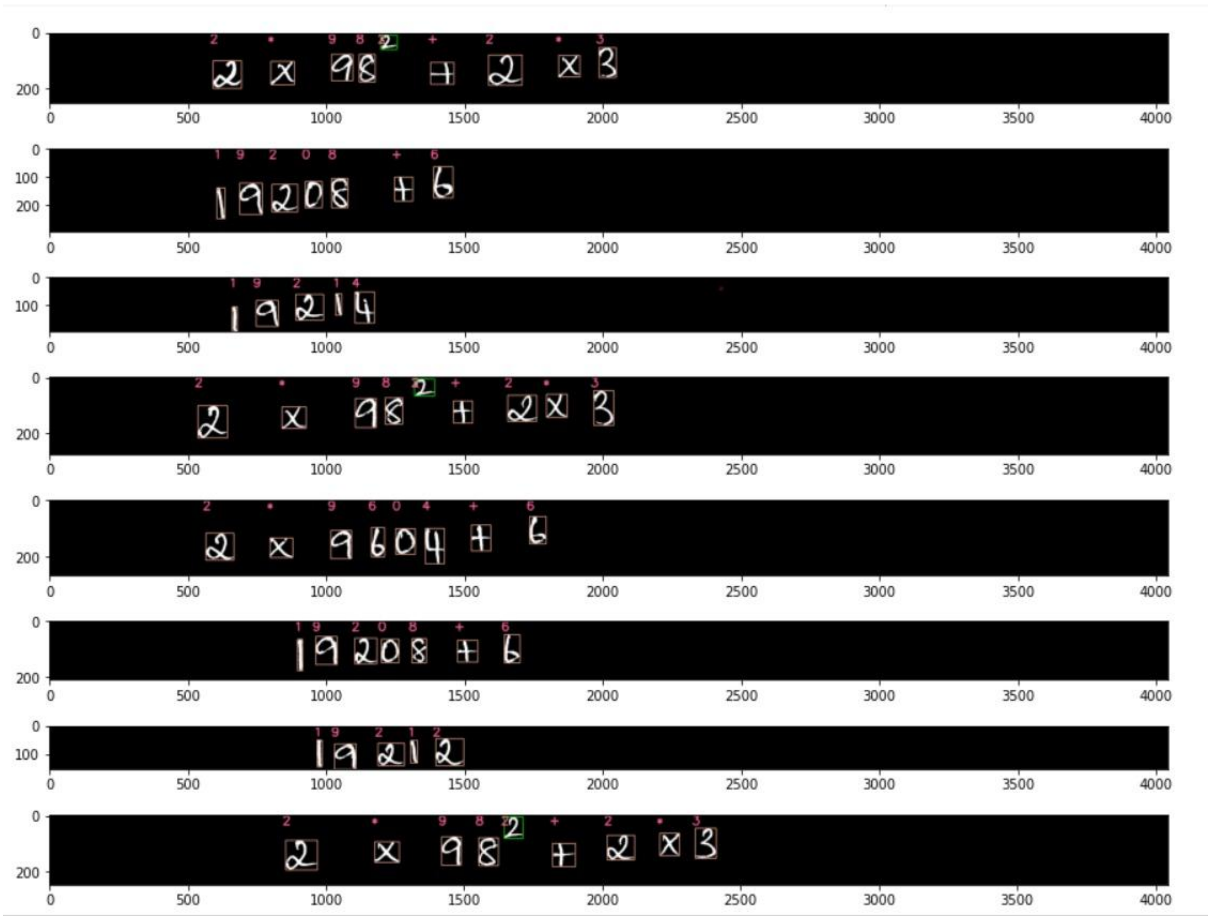


Fig 4.8 : Character Segmentation

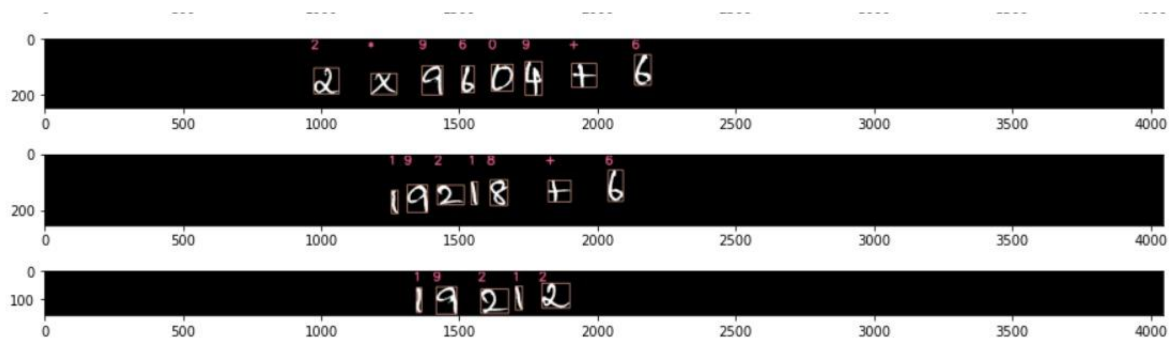
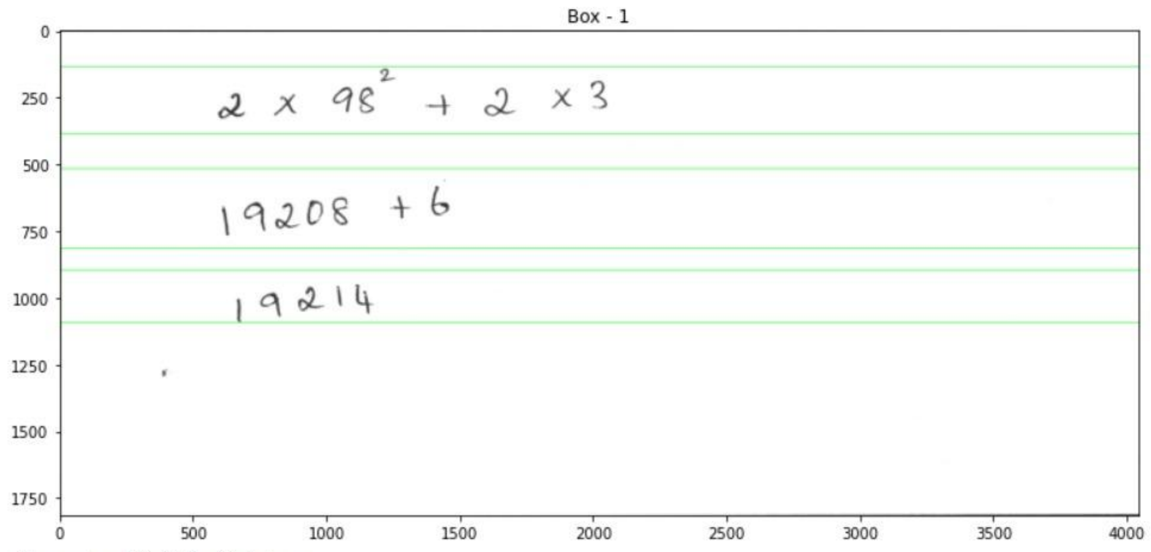


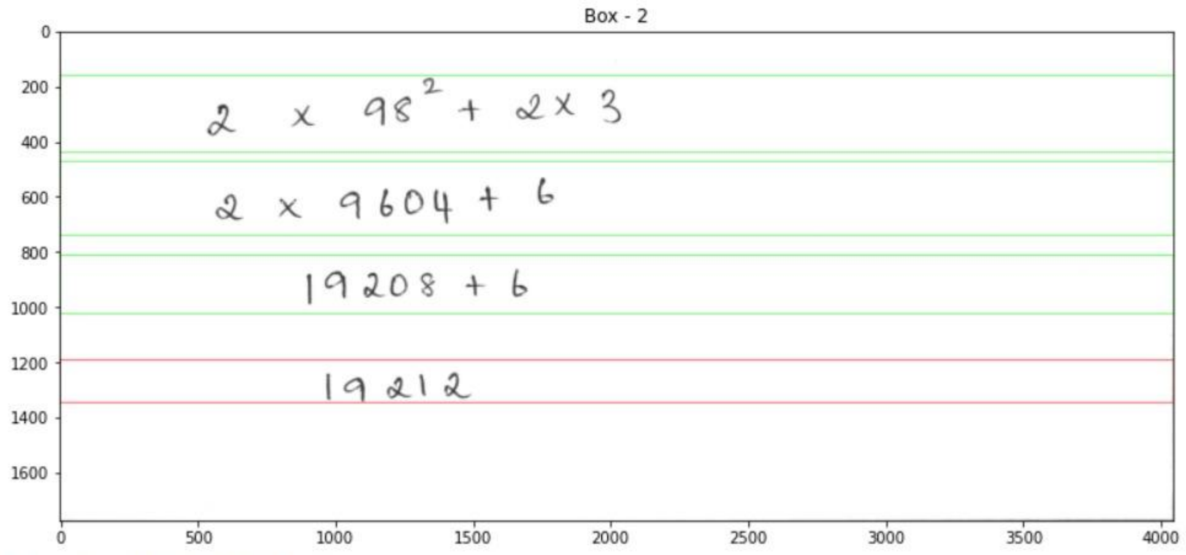
Fig 4.8 : Character Segmentation

- **Test Suite 5 : Equation Evaluation**

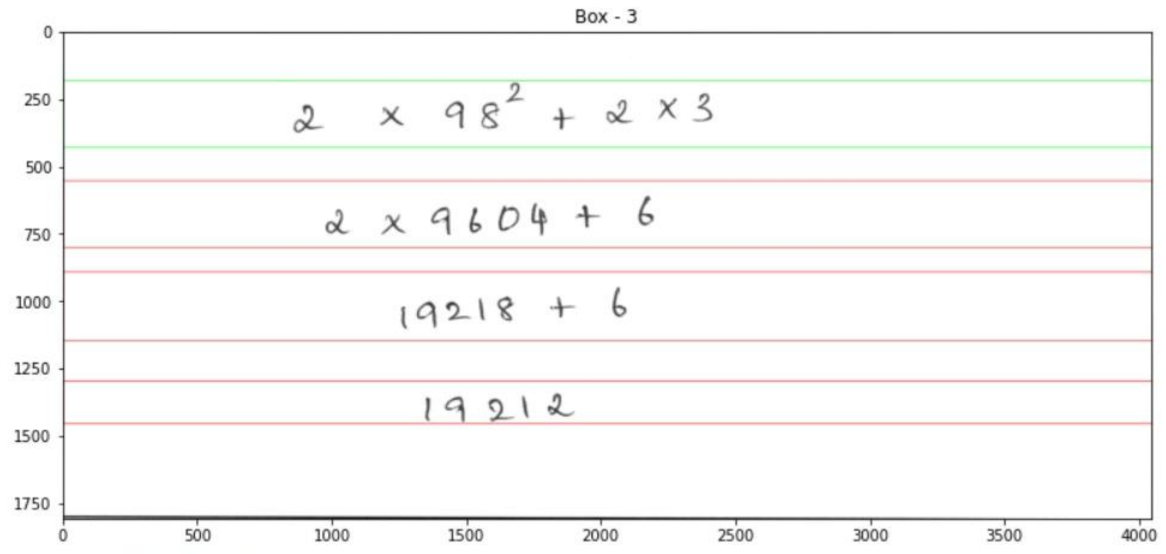
```
BOX 1
2*98**2+2*3
19208+6
19214
BOX 2
2*98**2+2*3
2*9604+6
19208+6
19212
BOX 3
2*98**2+2*3
2*9609+6
19218+6
19212
```



```
<Figure size 432x288 with 0 Axes>
```


Box - 2
$2 \times 98^2 + 2 \times 3$
$2 \times 9604 + 6$
$19208 + 6$
19212

<Figure size 432x288 with 0 Axes>


Box - 3
$2 \times 98^2 + 2 \times 3$
$2 \times 9604 + 6$
$19218 + 6$
19212

<Figure size 432x288 with 0 Axes>

CHAPTER-5

CONCLUSION AND FUTURE SCOPE

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

Conclusion

- Educational institutions such as universities, schools and various learning centres are an integral part of the society.
- Examinations and grading hold a significant role in the functioning of these institutions.
- Although grading is typically seen as a manual job, it can be observed that a much higher efficiency can be achieved by automating the process.
- The automated grading system reduces the amount of human errors, increases speed and thereby produces highly reliable output.

Future Scope

- The application can be trained with an even bigger dataset and obtain a higher accuracy.
- The domain can be extended to evaluation of answer sheets of different subjects.
- The domain can be extended to evaluation of answer sheets written in different languages.
- The application can have an easy to use front end for users to make use of the application efficiently.

REFERENCES

- 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.
- Recognition of Handwritten Textual Annotations using Tesseract Open Source OCR Engine – Paper on training tesseract for handwritten texts
- 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