IMAGE EQUATION SOLVER

Pratap Roy Choudhury and Raj Bhowmik

Indiana University Bloomington

Usable Artificial Intelligence, Spring 2022

1. PROBLEM STATEMENT

The problem to be addressed with this study is to illustrate the approaches of solving arithmetic equations from images. The challenge is aimed at leveraging the use of deep learning and image processing techniques to recognize a wide array of handwritten as well as digitally found mathematical equations.

2. DATA REPOSITORY

The dataset has been collected from Kaggle and Google images searched manually. Data are also collected from various repositories available on the internet. The datatype is a combination of jpeg and png i.e., and image format.

2.1. DATASET COLLECTION FOR TRAINING THE CLASSIFIER

From the visual Inspection of the images, it is very clear that there are two main categories into which the images of arithmetic expressions can be classified:

- **Handwritten Characters**
- **Digital Font Characters**

Hence the training dataset should have images that cover both handwritten and digital fonts. The following are the dataset collections used to train ensemble.

- **MNIST** Dataset for Handwritten Digits Recognition - directly imported in the script
- English Font Dataset to Recognize Digital Fonts for Characters (0-9)
  - Source: http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/
- Handwritten Math Symbols and Digits Dataset for Operator Recognition (+, -, /, %, *).
  - Source: https://www.kaggle.com/clarencezhao/handwritten-math-symbol-dataset

Dataset Overview:

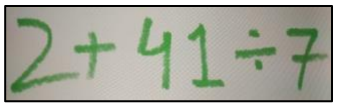
- **MNIST**: 60000 training images covering digits (0-9) and 10000 validation Images
- **English Font Dataset:** 6000 Image for training covering fonts of digits (0-9) and 4000 validation images.
- **Math Symbols:** 2000 Images for training covering (*, /, +, -). modulo sign (%) was not present in the dataset. It has been explicitly created by rotating "/" symbol by 45 degrees.

For the better understanding of the available dataset and the structure of the data repository, the entire collection of the dataset is shared in the following google drive link (here).

The datasets are structured in the following ways based on their applications.

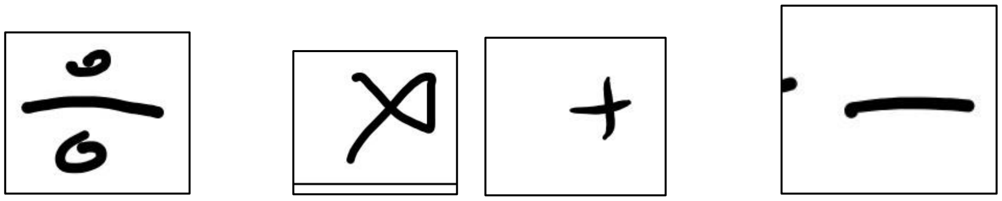
2.2. Target: The target dataset is a collection of mathematical equations. 64 equation images are collected and created as of now which will be used to solve the equation after detecting correctly.

Example image data:



2.3. SymbolSample: This folder contains image data of arithmetic operators. This includes addition, multiplication, division, and subtraction symbols. We have collated the Images for these operators in their multiple handwritten formats.

Example image data:



2.4. FontSample: This folder contains the image dataset of the digital font sample of our numbers, which is needed to identify a digit with any font variation and to perform mathematical calculations.

Example image data:

3. DATA INTERPRETATION & ARGUMENT

The Following are the initial observations based on visual inspection of the images displayed above

I. There are multiple categories into which one can place each image above.
II. The background of the images is not always Monochromatic.
III. We observe noisy background.
IV. We have images with Non-Uniform Illumination problem.
V. Digits and Operators (Foreground Objects) are not Monochromatic and have lot of variations in the way they are represented i.e., Handwritten vs Digital Format (Font Styles) / Thin Characters vs Thick Characters.

4. DATA MANAGEMENT

4.1. DATA CLEANING AND PREPROCESSING

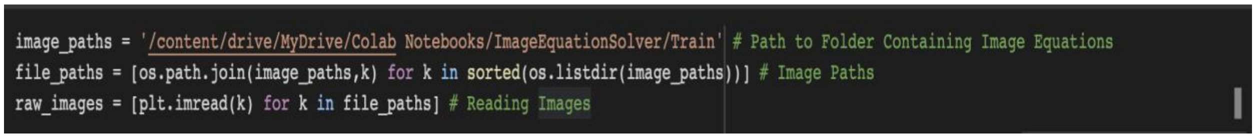
Based on the above observations, we proposed the below approaches to clean the noisy data and pre-process each character from the image for training purpose. Each of the steps are elaborately explained in the data Analysis section.

4.1.1. Denoising and Preprocessing: Denoising and creating multiple blurred versions of image to handle different types of noises that are present in the source image and pass them for further implementation. This is the first stage in the solution pipeline where an image is denoised with FastN1MeanDenoising and multiple blurred versions of the image are created to handle impulse/non-impulse noises. Multiple versions are needed for accurate character segmentation and identification.

4.1.2. Character Segmentation: Segmenting individual characters (digits and operators) from the image using image processing/deep learning approaches. Character Localization/Segmentation refers to finding bounding box of the individual digits/operators in an image. the accuracy of extraction is highly dependent on the quality of segmentation (localization).

4.2. STORING/READING THE DATA

All the datasets are stored in their respective folders: Target, FontSample, SymbolSample in a dedicated directory structure. The data has been imported in a python list data structure. Below is a snapshot of reading the train data:



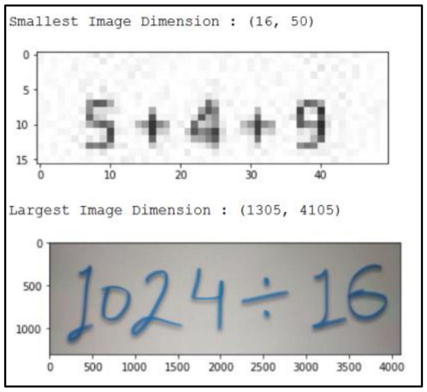
```
image_paths = '/content/drive/MyDrive/Colab Notebooks/ImageEquationSolver/Train' # Path to Folder Containing Image Equations
file_paths = [os.path.join(image_paths,k) for k in sorted(os.listdir(image_paths))] # Image Paths
raw_images = [plt.imread(k) for k in file_paths] # Reading Images
```

4.3. STATISTICAL ANALYSIS

The dataset has a variation in their dimension because of its digital and handwritten structure. Also, digits have the orientation problems. i.e., all the digits and operators are not written in a straight line.

Also, just by visual inspection one can easily figure out that they have different Sizes.

From our quick analysis in the training data, we found that our lowest image dimension is (16,50) and the largest image dimension is (1305,4105). Below is a snapshot of the same.



```
Smallest Image Dimension : (16, 50)
```

```
Largest Image Dimension : (1305, 4105)
```

Figure: ***Lowest and Highest dimensional image equation present in the dataset***

Further, we have converted the images to Grayscale, to check the distribution of the data, using the OpenCV libraries of Python.

5. VISUALIZATION

Gaussian and other noises in images can be inferred from distribution of gray levels centered around certain pixel value in few of the images. Below is a partial snapshot of our data visualization.

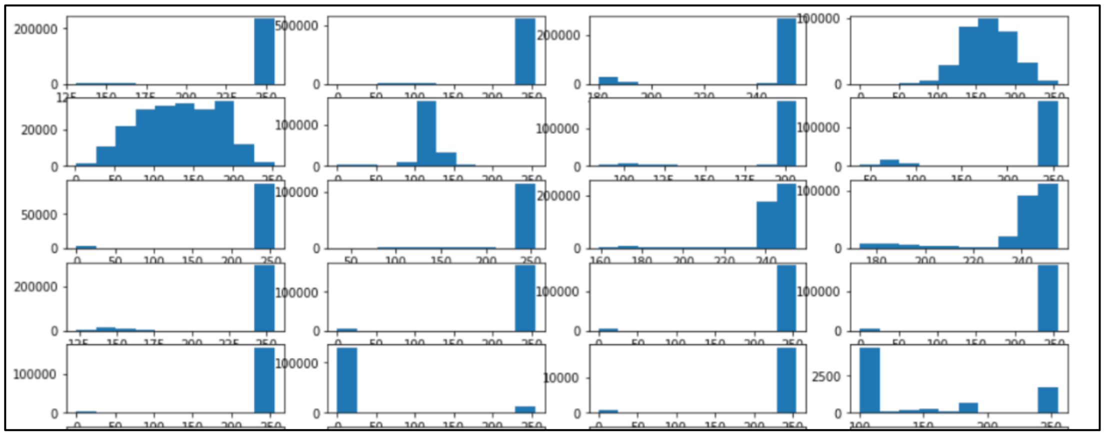


Figure: ***Histogram distribution of the grayscale version of the raw images***

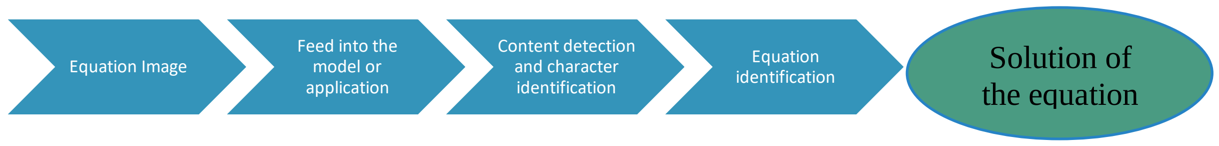
6. RESEARCH AND DESIGN

6.1. STAKE HOLDER ANALYSIS

- **Who?:** This data could be used by anybody in the mathematical field. The user groups are especially the educational institutions – Colleges, schools. Also, this can be implemented as a use-case for educational companies like coursera, BYJU's, Unacademi etc. This model can be used by anybody irrespective of gender or age. The images could be fed into out model and corresponding result will be provided.
- **How?:** Imagine a book with equations being solved by someone and they don't have the answers to it. Our model could be used to solved just by feeding the model a picture of the equation. Instead of manually solving the questions, they can just insert the image and get the result with a fraction of second using Deep Learning technology of our model.

6.2. CONTEXT

- **Use case:** The application is designed to have any digital image of arithmetic equation as an input to generate the solution for the same. The stakeholder doesn't require to engage with any of the data which are used to train our classification models such as all possible number fonts, mathematical symbols, number sets etc. The user just needs to have an image of the arithmetic equation.
- **Workflow:**


Equation Image
Feed into the model or application
Content detection and character identification
Equation identification
Solution of the equation

- **Information & Insights:** The objective is to develop an application based on image detection. The internal process is to train Neural Network based classifier on all possible combinations of numbers and operator's dataset and the best trained model will be used as the Blackbox in the application. Any user will provide an image of equation and receive the detected equation and its solution in some fraction of seconds as an output. In this way, it can help any person from the academic background or mathematical practitioner to quickly solve an equation without spending much time manually or to validate the solution.

6.3. ENHANCEMENT OF OPTICAL CHARACTER RECOGNITION AND DESIGN PLAN

Image Arithmetic Equation Solver can be viewed as a subset of Optical Character Recognition (OCR) Task. There are many off the shelf OCR solutions that are available as packages and API's that are designed to perform accurate extraction of text from document images and natural scenes.

However, it is very challenging task for any state-of-the-art OCR Solutions to extract text and digits with great accuracy from all the possible types of images with textual/digital content.

There are several factors that contribute to poor performance of these OCR solutions. The following are the significant factors that affect performance of OCR:

- Low Resolution of Images
- Large Noise in the Image
- Text/Content on Non-Monochromatic Background such as Natural Scenes.
- Ambiguous Characters (Often Problem in the case of Handwritten Text Recognition)
- Non-Uniform Illumination
- Cursive & very closely spaced Characters often creates big challenges in Segmenting and Identifying Individual Characters

Hence, based on the type of images and specific task at hand, we need to customize the existing solutions or develop custom solutions that are tailored to solve specific problem. Below are the proposed design steps that we are going to follow and implement our customize solution to solve an arithmetic equation given as an image.

- Formally Defining Image Equation Solver Problem Statement (Defined at the beginning)
- Initial Exploration & Observations (Done above)
- Identifying required Image Processing and Deep Learning Components
- Implementing Components and Building Blocks
- Encapsulating all the Components into Single Solution Pipeline

6.4. DATA ETHICS

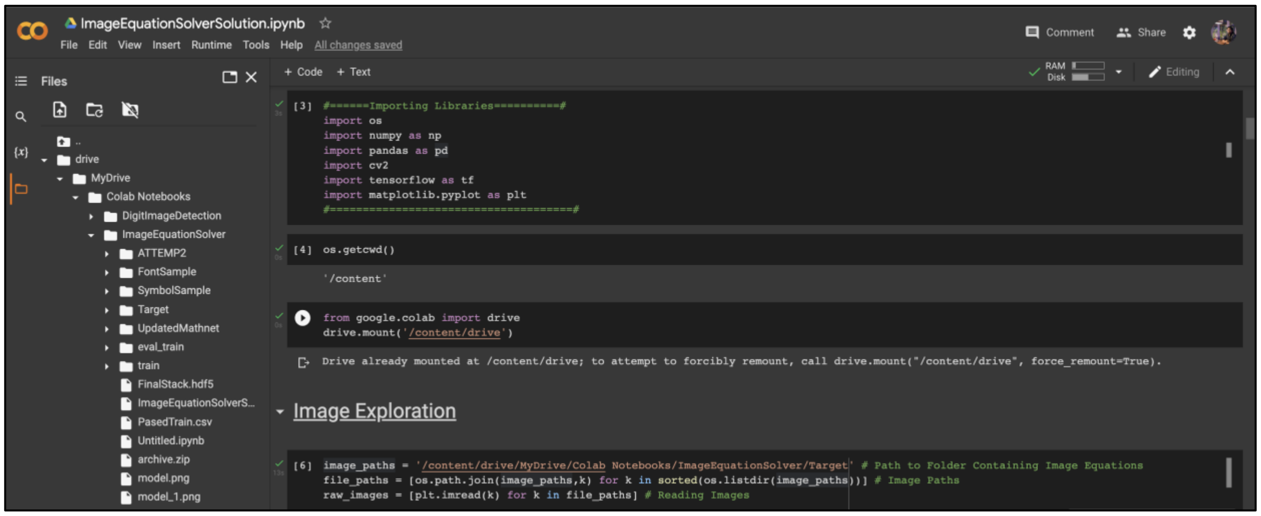
The data used is ethical and without any negative consequences to the society. It's a big step towards education by solving the mathematics simply by inserting a picture.

- **Value:** Our goal is to help students or any educational institution curb manual mistakes or get a faster solution to a particular problem. In the coming days, for higher computational easiness, this method can simply the process of solving mathematical equations just by feeding equation image or scan the equation from paper and by feeding into any application holding this model, can accurately detect the text information and detect the actual equation and it can solve with the help of math libraries to generate the solution as an output.

- **Unintended Consequences:** The only negative side, just like any other product, is student can feed pictures of a question and get the result without learning the concepts of mathematics, which can be a malpractice. So, getting past that, we aim to bring a positive change to the community.

7. ENVIRONMENT SELECTION

We are implementing our project in Google Colab due to GPU restriction in our current owned computers. As we are planning to use TensorFlow and Keras python libraries to train our model, so we are implementing our code in Google Collab itself.



```
[3] #======Importing Libraries==========#
    import os
    import numpy as np
    import pandas as pd
    import cv2
    import tensorflow as tf
    import matplotlib.pyplot as plt
    #====================================#

[4] os.getcwd()

    '/content'

    from google.colab import drive
    drive.mount('/content/drive')

    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

[6] image_paths = '/content/drive/MyDrive/Colab Notebooks/ImageEquationSolver/Target' # Path to Folder Containing Image Equations
    file_paths = [os.path.join(image_paths,k) for k in sorted(os.listdir(image_paths))] # Image Paths
    raw_images = [plt.imread(k) for k in file_paths] # Reading Images
```

Figure: ***Google colab environment for TensorFlow and data folder management***

8. SOLUTION ARCHITECTURE

Based on the above observations the task of Image Equation Solver can be broken down into several tasks. The following are the subtasks that form our wish list and would be stitched into solution pipeline

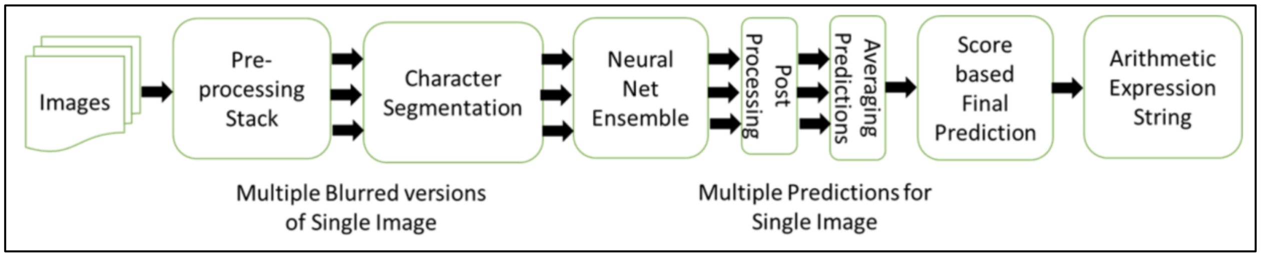

Images
Pre-processing Stack
Character Segmentation
Neural Net Ensemble
Post Processing
Averaging Predictions
Score based Final Prediction
Arithmetic Expression String
Multiple Blurred versions of Single Image
Multiple Predictions for Single Image

Figure: ***Block Diagram of the Solution Pipeline at a high level***

The above figure illustrates the components and building blocks of the solution pipeline. Each of the function blocks are discussed below in the subsequent sections.

9. SOLUTION PIPELINE DEVELOPMENT

The Following is the approach that has been adopted to solve the problem according to the solution architecture.

I. **Denoising and Preprocessing:** Denoising and creating multiple blurred versions of image to handle different types of noises that are present in the source image and pass them through the pipeline.

II. **Character Segmentation & Character Identification/Detection:** Segmenting individual characters (Digits and Operators) from the image using image processing/Deep Learning approaches.

III. **Image Classifier Training:** Training Multiple CNN Classifiers to classify a character into a digit or Operator.

IV. **Post Processing:** Performing post processing to correct potential misclassifications by model.

V. **Averaging Predictions:** Averaging the predictions of multiple CNNs to classify a character.

VI. **Result Aggregation and Final Prediction:** The above process is repeated for all the blurred versions of the image and the result with maximum detected characters or with maximum total predicted probability scores across characters is chosen as final prediction.

9.1. DENOISING AND PREPROCESSING

This is the first stage in the solution pipeline where an image is denoised with **fastN1MeanDenoising** and multiple blurred versions of the image are created to handle impulse/non-impulse noises. Multiple versions are needed for accurate character segmentation and identification. We have used the function **cv2.fastNlMeansDenoisingColored()** which is the implementation of Non-local Means Denoising algorithm [1]. This function is to remove noise from colored images.

We are using median blur function for blurred version of the image. The **mediaBlur** is a non-linear filtering technique. As clear from the name, this takes a median of all the pixels under the kernel area and replaces the central element in with this median value. This is quite effective reducing a certain type of noise (like salt-and-pepper noise) with considerably less edge blurring as compared to other linear filters of the same size [2]. Because we are taking a median, the output image will have no new pixel values other than that in the input image.

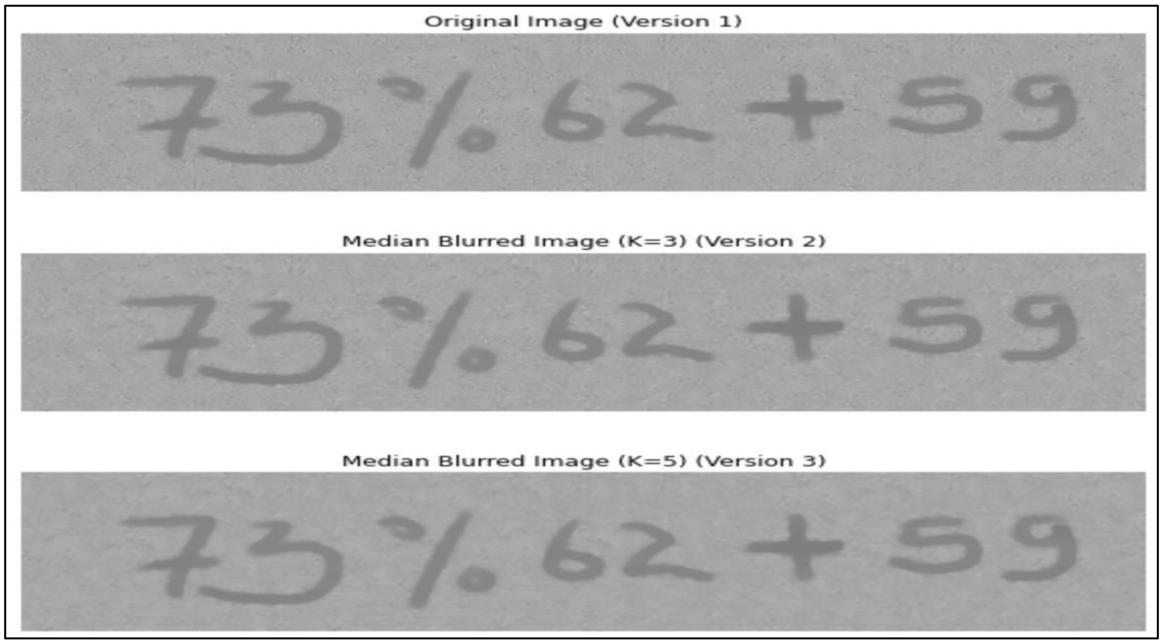

Original Image (Version 1)
Median Blurred Image (K=3) (Version 2)
Median Blurred Image (K=5) (Version 3)

Figure: ***Illustration of one of the raw images after denoised and blurred using medianBlur technique with kernel size 3 and 5***

9.2. CHARACTER LOCALIZATION (SEGMENTATION)

Character Localization/Segmentation finds the bounding box of the individual Digits/Operators in an image. The accuracy of the extraction is highly dependent on the quality of segmentation (localization)

As observed from above images, we can see that there are quite an array of scale, color, and noise. We leverage the already built pre-trained Deep Learning model that can accurately detect bounding box of a character in each image.

For the character identification, we identify each character which would help us in calculating the equation. It is followed by post processing steps. This step is to correct potential misclassifications made by the model. So, to avoid any misclassification which could lead to bad accuracy or classifying a character wrong which would lead to wrong calculation of our equation. The next step is exception handling along with building the final pipeline. It should be noted that not all the Images that are provided can be parsed and extract valid output. There are cases where there would be misclassification or incorrected character localization that could lead to inconsistent results. To handle all such cases where the parsing was not successful or has resulted in an **Invalid Arithmetic Expression,** we will use the **EASYOCR** to parse the Results. This way we have a **combination** of both custom logic and state of the ART OCR solution that can help us to minimize error and improve extraction accuracy at a character level.

9.2.1. CRAFT TEXT DETECTION FOR CHARACTER LOCALIZATION

Character Region Awareness for Text Detection (CRAFT) is a pretrained Deep Leaning model that is trained to detect text in natural scenes.

The main power of CRAFT model is the fact that not only it can accurately detect text in a natural scene image but also can detect the individual character in the image and an affinity score that would link characters to form words. The affinity score estimation of CRAFT makes CRAFT model detect continuous text even on a curved contour.

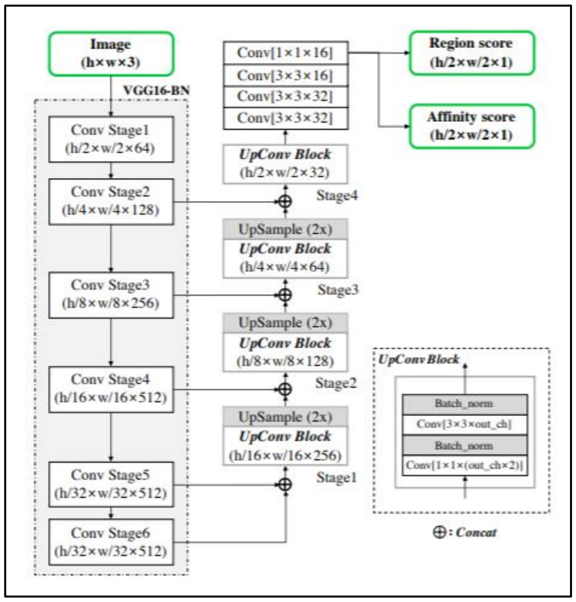


Image (h×w×3)
VGG16-BN
Conv Stage1 (h/2×w/2×64)
Conv Stage2 (h/4×w/4×128)
Conv Stage3 (h/8×w/8×256)
Conv Stage4 (h/16×w/16×512)
Conv Stage5 (h/32×w/32×512)
Conv Stage6 (h/32×w/32×512)
Conv[1×1×16]
Conv[3×3×16]
Conv[3×3×32]
Conv[3×3×32]
UpConv Block (h/2×w/2×32)
Stage4
UpSample (2x)
UpConv Block (h/4×w/4×64)
Stage3
UpSample (2x)
UpConv Block (h/8×w/8×128)
Stage2
UpSample (2x)
UpConv Block (h/16×w/16×256)
Stage1
Region score (h/2×w/2×1)
Affinity score (h/2×w/2×1)
UpConv Block
Batch_norm
Conv[3×3×out_ch]
Batch_norm
Conv[1×1×(out_ch×2)]
⊕: Concat

Figure: *CRAFT Architecture* [4]

CRAFT adopts a fully convolutional network architecture based on VGG-16 as its backbone. In simple words, VGG16 is essentially the feature extracting architecture that is used to encode the network's input into a certain feature representation. The decoding segment of the CRAFT network is like UNet. It has skip connections that aggregate low-level features. The Following Section illustrates CRAFT based Digit and Operator Localization when the denoised image is passed as input to CRAFT.

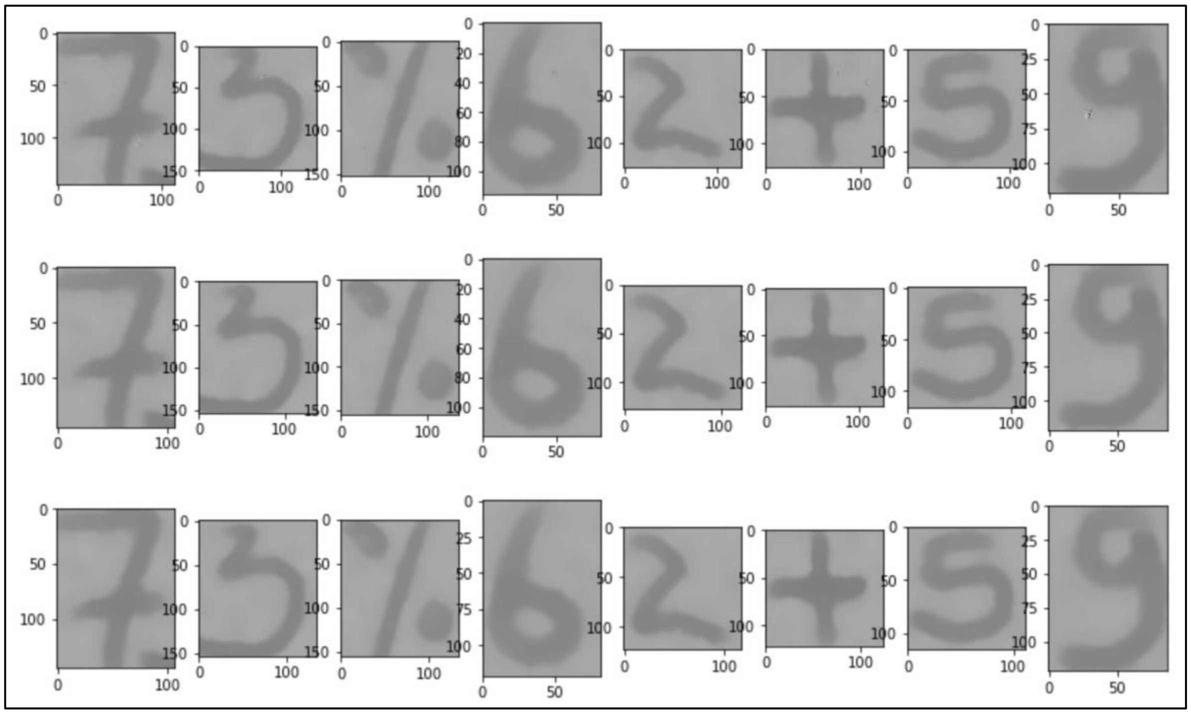


Figure: *Illustration of CRAFT based Digit and Operator Localization on denoised image*

As shown in the above figure craft can accurately draw bounding boxes over individual characters of the image.

We deliberately set link score threshold to **np.inf** in the process so that characters and not allowed to merge to form words. This way we get bounding boxes at character level.

Clearly from above image one can spot that when the noise is high the last 2 characters **"-9"** got merged. However, we observe that by blurring Image with Median Blurring (k=5) **"-"** and **"9"** got segmented accurately.

The same can be applied on raw images where the background is clear and white without any noise.

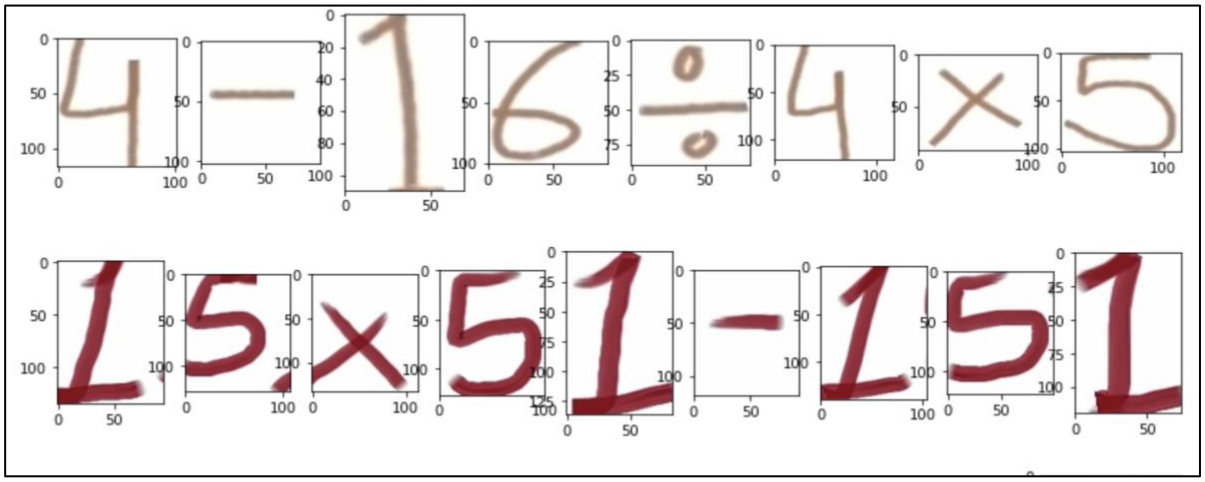


Figure: ***Illustration of CRAFT based Digit and Operator Localization on raw image with clear background***

9.3. NEURAL NETWORK ENSEMBLE FOR CHARACTER RECOGNITION

Once the characters are localized with decent accuracy, then the most important step in the pipeline is to create digital output (Textual Representation) for each character image. This task can be framed as an image classification problem in which we train a network that can identify the Digit/Operator given its localized image.

Neural Networks especially CNNs are prone to overfitting, and it would be quite difficult for the network to generalize on unseen samples.

We have adopted Ensemble of CNNs to predict the character and we average the predicted probability for each character across the CNN Ensemble to get a final probability estimate for each character so that the score for error/misclassification is reduced.

We choose a heterogenous ensemble of **EfficientNet B1-B5** to predict Characters and Average the Probabilities across the Ensembles to Achieve the Classification Result.

The Ensemble Neural Network that has **8** individual and independently trained EfficientNet based Neural Networks to perform character recognition.

The following are the details of the specific training configuration for each CNN in the Ensemble.

I. **EfficientNetB4_English_Font_UnWeighted:** Neural Network trained on EfficientNEtB4 with English font, MNIST and Math Symbol dataset without any class weights for Character Recognition.

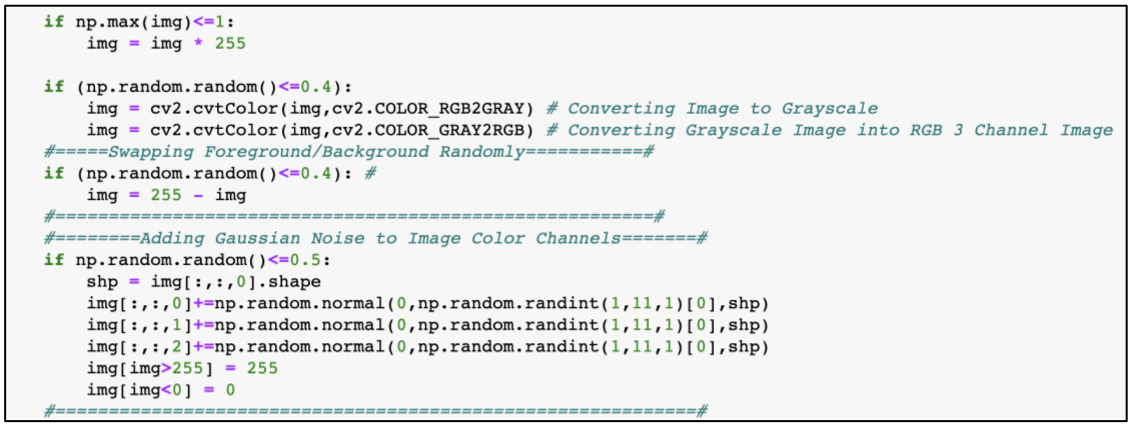
II. **EfficientNetB4_English_Font_Weighted_2x:** Neural Network trained on EfficientNEtB4 with English font, MNIST and Math Symbol dataset with Math Symbol Classes weighted by a factor of **2** in the Loss Function.

III. **EfficientNetB4_English_Font_Weighted_3x:** Neural Network Trained on EfficientNEtB4 with English font, MNIST and Math Symbol dataset with Math Symbol Classes weighted by a factor of **3** in the Loss Function.

IV. **EfficientNetB1_MNIST_Weighted_3x to EfficientNetB5_MNIST_Weighted_3x:** 5 Neural Networks based on EfficientNetB1 to EfficientNetB5 are trained on MNIST Data and math symbol data with symbol classes weighted by a Factor of **3** in the Loss Function as symbol classes are underrepresented as Compared to Digit Classes from MNIST.

9.3.1. IMAGE AUGMENTATION

We randomly perform Width/Height transformation and add Gaussian Noise to individual color channels of the image to introduce color distortion and randomly interchanging background with foreground. Images are loaded for training using Image Data Generator.

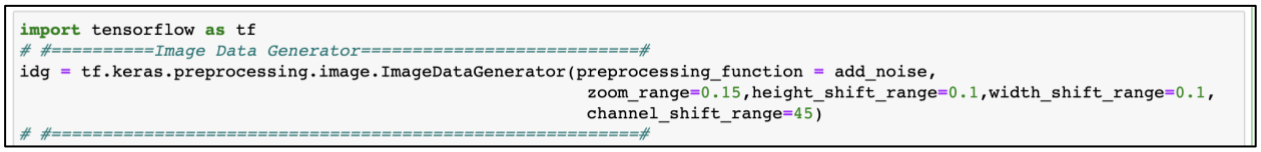


```
if np.max(img)<=1:
    img = img * 255

if (np.random.random()<=0.4):
    img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY) # Converting Image to Grayscale
    img = cv2.cvtColor(img,cv2.COLOR_GRAY2RGB) # Converting Grayscale Image into RGB 3 Channel Image
#=====Swapping Foreground/Background Randomly===========#
if (np.random.random()<=0.4): #
    img = 255 - img
#=========================================================#
#========Adding Gaussian Noise to Image Color Channels=======#
if np.random.random()<=0.5:
    shp = img[:,:,0].shape
    img[:,:,0]+=np.random.normal(0,np.random.randint(1,11,1)[0],shp)
    img[:,:,1]+=np.random.normal(0,np.random.randint(1,11,1)[0],shp)
    img[:,:,2]+=np.random.normal(0,np.random.randint(1,11,1)[0],shp)
    img[img>255] = 255
    img[img<0] = 0
#============================================================#
```

9.3.2. IMAGE DATAGENERATOR

Here, the augmented image is passed through the **ImageDataGenerator()** function. It generates batches of tensor image data with real-time data augmentation. With this instance, the training and validation datasets are trained before passing through the classification network.



```
import tensorflow as tf
# #==========Image Data Generator==========================#
idg = tf.keras.preprocessing.image.ImageDataGenerator(preprocessing_function = add_noise,
                                                       zoom_range=0.15,height_shift_range=0.1,width_shift_range=0.1,
                                                       channel_shift_range=45)
# #========================================================#
```

9.3.3. EFFICIENTNET BASED FEEDFORWARD CLASSIFICATION NETWORK

A method is created to retrieve Classification Feedforward Head for a Given Base EfficientNetwork.

Parameters:

- **mdl**: (tf.keras.Model) Backbone Network(EfficientNet B0 to B7)
- **Returns**: MODEL : (tf.keras.Model) Custom EfficientNet with Classification Feedforward Head

The number of hidden layers and the hyper-parameters and the output layer with the number of class activation function names are defined within this method.

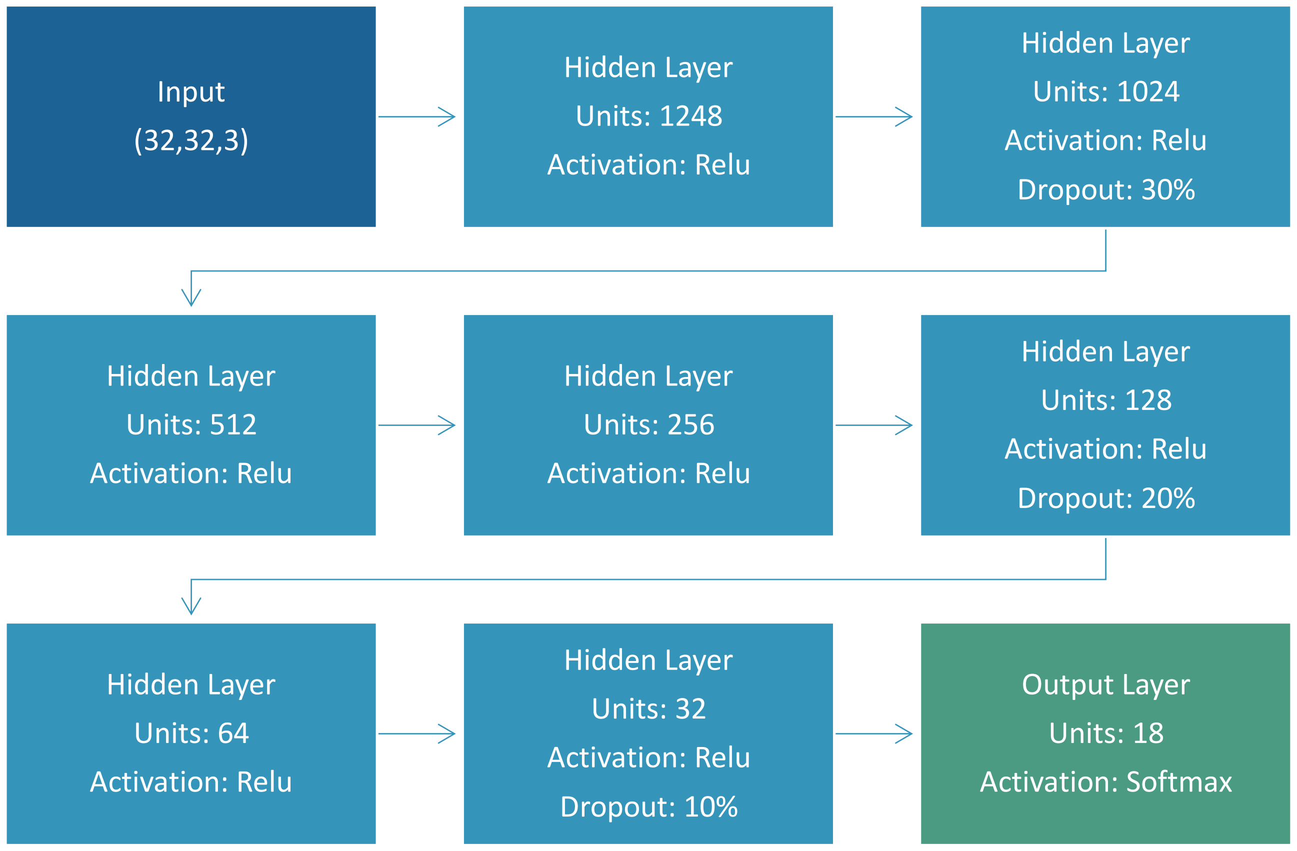


Figure: ***EfficientNet based Neural Network classifier layers with hyperparameters***

9.3.4. TRAINING NETWORKS

For our model, we are initially creating 3 (I, II & III) EfficientnetB4 based Feedforward Networks for Classification - one unweighted, the other two with weighted loss by factor 2 and 3 of the symbols function.

- **EfficientNetB4_English_Font_UnWeighted**
- **EfficientNetB4_English_Font_Weighted_2x**
- **EfficientNetB4_English_Font_Weighted_3x**

(Models train on English Font Images of (0-9), MNIST and Math Symbols Dataset)

The common training parameters are given below.

- **Image Input Shape:** (32,32,3)
- **Number of Epochs of Training:** 10
- **Optimizer:** Stochastic Gradient Descent (SGD)

There were only 2000 images for +, -, /, * for training which is way less than Digits available for Training in MNIST which is around 60,000.

Similarly, 5 more model stacks are generated using EfficientNetB1 to EfficientNetB5 on MNIST set with weighted factor of 3. The models are trained with 10 epochs and 100 steps per epoch.

- **EfficientNetB1_MNIST_Weighted_3x**
- **EfficientNetB2_MNIST_Weighted_3x**
- **EfficientNetB3_MNIST_Weighted_3x**
- **EfficientNetB4_MNIST_Weighted_3x**
- **EfficientNetB5_MNIST_Weighted_3x**

10. STACKED MODEL ARCHITECTURE

Three models that has been trained viz **- EFFB1.hdf5** for the original unweighted B4 model, **EFFB2.hdf5** and **EFFB3.hdf5** models with weighted loss functions by factors 2 and 3, are stacked together to create a pipeline of models which will be applied to the target images.

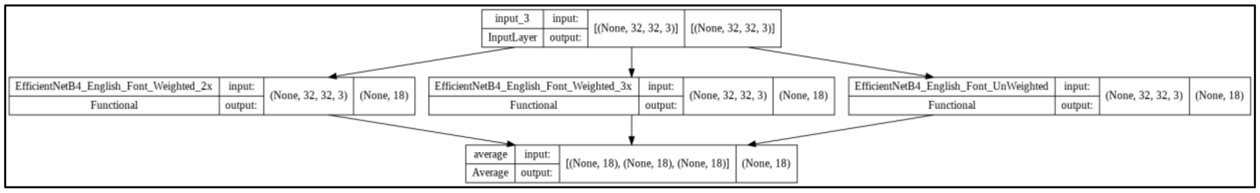


Figure: ***Neural Network based trained classification 3 model stack***

Above is the part of the entire 8-model stack which has the other 5 model components which are stored as Mathnet_B1.hdf5, Mathnet_B2.hdf5, Mathnet_B3.hdf5, Mathnet_B4.hdf5 and Mathnet_B5.hdf5. These stacks are following the above pipeline on the model stacks.

Note: All the datasets, trained neural network models (8 models), final model stack etc. are shared in the goggle drive folder (here)

11. POST PROCESSING METHODS

Post processing methods are required as a part of any OCR solutions that can help to correct any misclassifications and fine-tune the extraction results.

One post processing task which is performed here is to handle misclassification od "/" as 1. During training the model, since MNIST dataset has many 1s that look like forward slash "/", it is expected that classifier may predict / as 1. Following is the post processing approach.

- Identify Characters predicted as 1 by Neural Network Ensemble
- Perform ForeGround Separation on Corresponding Character Image using KMeans Clustering
- Contour detection on the Foreground Segmented Image after fitting an ellipse to get Inclination of the Character
- If the Angle of Inclination is between 12 to 90 degrees, then 1 is replaced with /

Note: The above post processing method is invoked if a character is predicted as 1 by the network ensemble and the character to the left and right are predicted as Number by the network. It is a logical way of applying correction as it cannot have a valid mathematical expression where an operator has another operator in its immediate neighborhood.

Below is the illustration of the post processing where the actual "/" symbol is processed by Foreground separation and using the K-means clustering, it is detected correctly as "/".

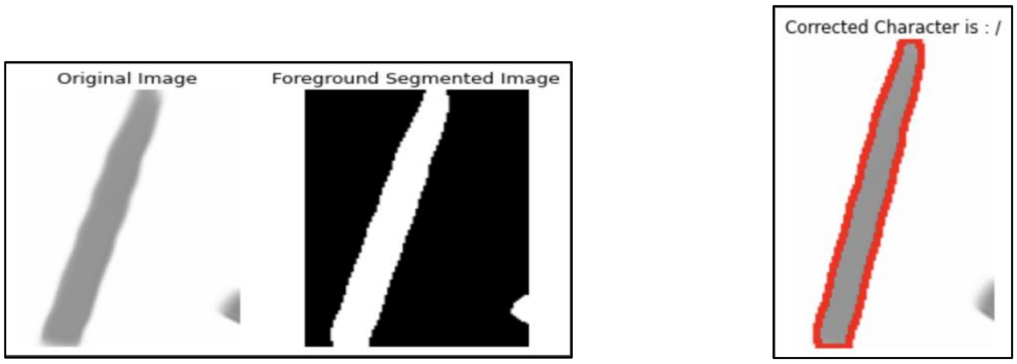


Figure: ***ForeGround separated and clustered character recognition***

12. MODEL EVALUATION ON TRAINING IMAGE

As mentioned in the original model pipeline, the stacked model is applied on a processed raw image and extraction is done on three different variations of that input image - one is the original image, and other two are median blurred with weighted factors of 3 and 5.

The correct result is chosen that has either the maximized number of detected characters or has maximized total probability of overall characters detected.

The probability score is calculated based on the character identification by each of the stacked models and then averaging the probabilities of each character detected.

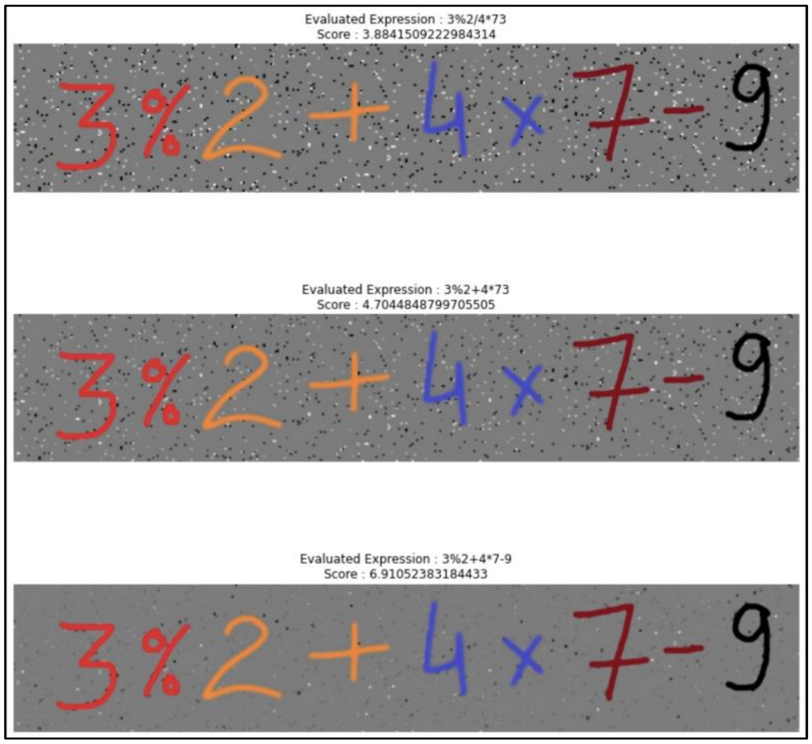

Evaluated Expression : 3%2/4*73
Score : 3.8841509222984314
Evaluated Expression : 3%2+4*73
Score : 4.7044848799705505
Evaluated Expression : 3%2+4*7-9
Score : 6.91052383184433

Figure: ***Probabilistic score-based result selection***

Averaging Prediction

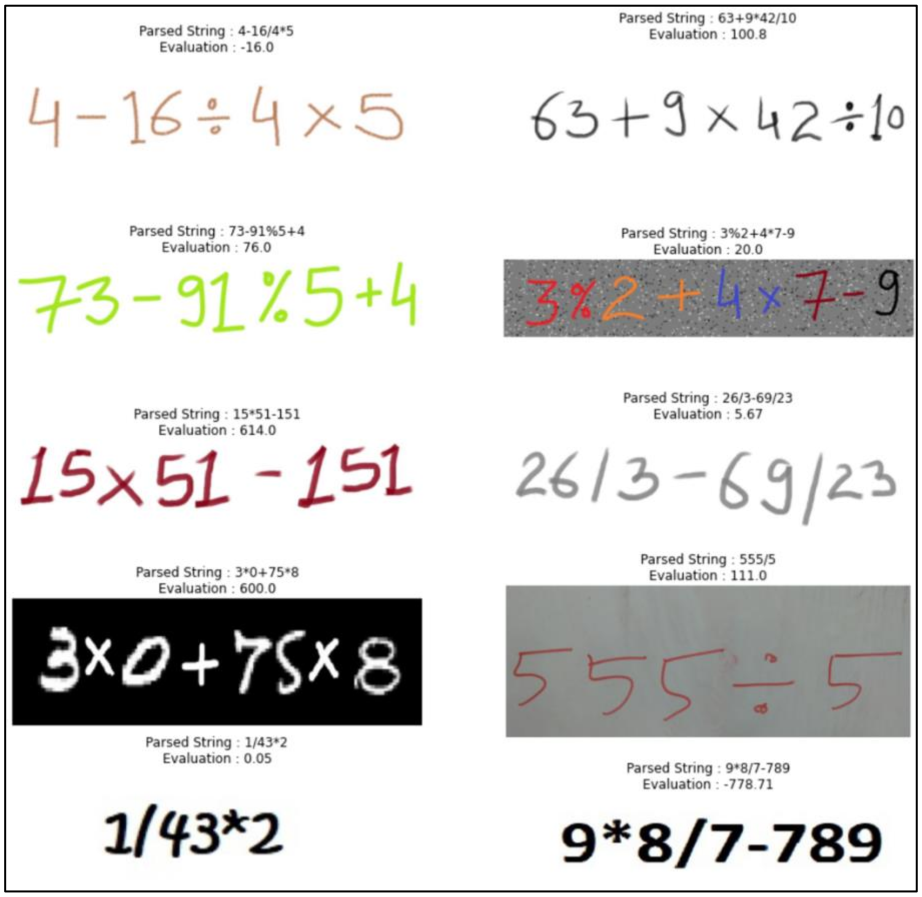
As shown above, the noisy version of the image has some misclassifications and due to improper character segmentations, results are not good. However, it has been observed that the median blurred versions with K=5, the ensemble is able to correctly identify the expression.

The final predicted expression from multiple versions of image is selected based on the following scoring criterion:

- The image which results in maximum number of characters predicted (i.e., length of predicted string is maximum) is chosen as the final predicted expression.
- If multiple versions of image result in same number of predicted characters, then score is calculated as sum of predicted probabilities of each character in the evaluated string and the expression with maximum score is selected as final expression.

13. SCORE BASED FINAL PREDICTION

Once the averaging scoring technique is implemented, the raw images are passed as input and goes through all the cleaning and preprocessing steps and the three versions of each image are used to apply the model for prediction. Below is the illustration of a set of images that are predicted using the entire pipeline developed.



```
Parsed String : 4-16/4*5
Evaluation : -16.0

Parsed String : 63+9*42/10
Evaluation : 100.8

Parsed String : 73-91%5+4
Evaluation : 76.0

Parsed String : 3%2+4*7-9
Evaluation : 20.0

Parsed String : 15*51-151
Evaluation : 614.0

Parsed String : 26/3-69/23
Evaluation : 5.67

Parsed String : 3*0+75*8
Evaluation : 600.0

Parsed String : 555/5
Evaluation : 111.0

Parsed String : 1/43*2
Evaluation : 0.05

Parsed String : 9*8/7-789
Evaluation : -778.71
```

Figure: ***Expression extraction and equation solution on sample test images***

14. DESIGN INTERVENTION

The goal is to detect any image based arithmetic equation and generate the output of the equation.

- **Target behavior:** The target behavior is to correctly output the solution of a digitally curated mathematical equation.
- **Expected behavior:** The expected behavior of our intervention is similar to the targeted behavior. For a single image being passed to the model, the output would be given in a string format. And, when bulk images are passed to the model, the output would be stored in a csv or excel file with each input as an index and corresponding column as a result.
- **Form of delivery:** The deliverable is in the form of an image if it's only one or two image equation with the correct expression detected and the mathematical solution displayed on the IDE. If there are bulk image set, the results are stored in a csv or excel file as the equation solution for every image index number.
- **Effect measurement:** The training model accuracy measurement is proposed as to have the actual solutions of each image in a csv or excel file with the image index and the model output will be compared to each of those image's actual solution to get the model accuracy on train image set.

15. LIMITATIONS & EXCEPTION HANDLING

The above results are the example of where the custom image equation solver pipeline can predict the characters with very high accuracy.

It should be noted that not all images can be parsed and extracted as valid expressions that are provided or generated whether by digital method or manually which can have many other writing variations which are out of the training scope. These are the cases where there would be misclassification or incorrect character localization that could lead to inconsistent results.

To handle all such cases where the parsing was not successful or has resulted in an Invalid Arithmetic Expression, the EASYOCR will be used to parse the results. This way it can have a combination of both custom logic and state of the art OCR solution that can help to minimize error and improve the extraction accuracy at a character level.

REFERENCES

[1] OpenCV denoising. https://docs.opencv.org/3.4/d1/d79/group__photo__denoise.html

[2] Smoothing Filters. https://theailearner.com/tag/cv2-medianblur/

[3] CRAFT Text Detector. https://pypi.org/project/craft-text-detector/

[4] Youngmin Baek, Bado Lee, Dongyoon Han, Sangdoo Yun, and Hwalsuk Lee. Clova AI Research, NAVER Corp. "Character Region Awareness for Text Detection". 3 April 2019. https://arxiv.org/pdf/1904.01941.pdf

[5] Mingxing Tan, Quoc V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". 11 September 2020. https://arxiv.org/pdf/1905.11946.pdf

[6] TensorFlow EfficientNet.
https://www.tensorflow.org/api_docs/python/tf/keras/applications/efficientnet/EfficientNetB4