A project report

on

Handwritten Mathematical Equation Detection and Extraction From Images

Submitted by :

Modem Praveen

17BCS035

Computer Science and Engineering

Under the Guidance of

Dr .Jagadish D N

Assistant Professor , Electronics and Communication Engineering

Dharwad

ज्ञानेन विकासः

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY - DHARWAD

Contents

List of figures

Acknowledgement

With a deep sense of gratitude, I express sincere thanks to my supervisor, **Dr. Jagadish** Assistant professor Electronics and communication Engineering at Indian Institute of Information Technology Dharwad, for providing me an opportunity to work on this project and giving me proper support and constant encouragement during my project work from initial to final level and I express my heartiest gratitude to my **friends** , **Dr.Arun Chauhan** and **Dr.Laxman Mahto** for their constant support .

Abstract

Text detection and recognition in images plays an important role in content analysis of images. Furthermore, handwritten character recognition is a difficult task in the area of image processing. When compared to these problems Handwritten Mathematical Equation Detection and Extraction from images is one of the most complicated problems posted in the area of computer vision research.

In this project we took mathematical symbols and digits (0-9) as a dataset and we did character recognition, segmentation , classification and evaluation of few basic math answers ..For that we recognised characters by converting image into binary and finding contours , segmented by basic python coding and finding coordinates and contours .Used Convolutional Neural Networks for classification. Finally we evaluated a few basic math answers.

Key words : Character recognition ,Segmentation, Classification , Contours , Convolutional Neural Networks.

Introduction

Scope of the project

As we know it is very difficult and time taking process to enter equations in a document. Some of the ways we do is we write them in a latex code and convert those codes into equation format or other way is we enter every math equation manually in a word document and there are several ways which take a long time and more effort. So instead of these efforts here we can use Neural Networks which takes handwritten equation images as an input and it extracts those equations as an output.And at the end based on extracted equations we evaluated basic math answers.

To achieve this we should follow below steps :-

1)Data collection
2)Data Pre-processing
3)Segmentation
4) label Classification/Prediction
5) Evaluation

Workflow

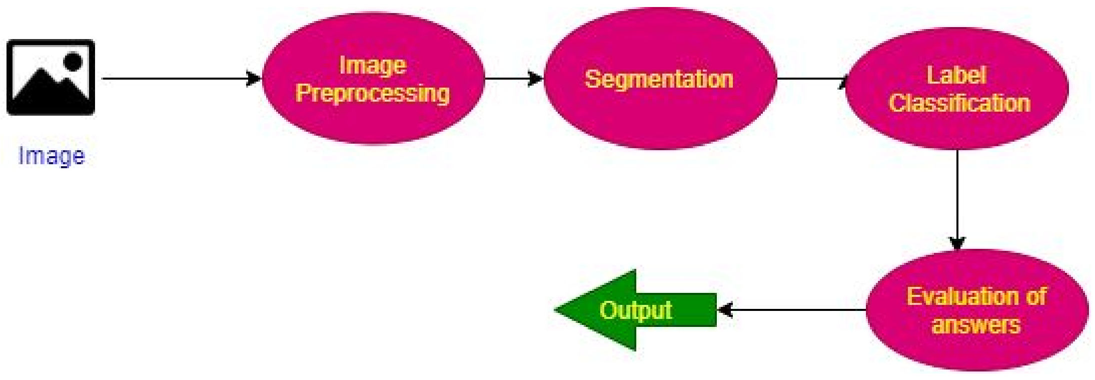


Figure 1. Workflow flowchart

Literature Review

This topic will contain the current trend in the Character Recognition, Classification and Extraction related information found in research papers.

- Several works have been done so far for handwritten character recognition . This project focuses on numerous techniques used for feature extraction and recognition. An effective and robust system for recognition of printed and handwritten mathematical characters have been proposed by Zanibbi et. al. (2002). Which evaluates an expression by using three successive passes i.e., the layout pass, lexical pass and operator tree. This tree manipulation which is used in each pass can be represented by tree transformation. Bage et.al.(2013) has proposed a recognition system for offline handwritten mathematical symbols. For feature extraction, the shape of the character is considered. The proposed system is based on relative study of feature extraction methods. Classification has been carried out using a neural network approach with a recognition rate of 90% . Ahmad-Montaser Awal et al(2010) discussed some issues related to the problem of online mathematical expression recognition. Ha. et. al. (1995, August) have proposed a system that understands mathematical expression from the printed document images. In the development of this system, they have adapted object oriented methodology to describe the data abstraction for the hierarchical structure of the mathematical expression is given in the form of the expression tree. Pradeep et. al. (2010) have proposed a diagonal feature extraction technique for the handwritten character by using feed-forward neural network algorithms. This technique uses diagonal, horizontal and vertical features for the classification purpose. A few papers are available online for recognition of mathematical expression using convolutional neural networks(CNN).In recent years, Convolutional Neural Network (CNN) has made a series of revolutionary research results in the fields of image classification and detection. The past few years a branch of artificial neural networks called deep learning has shown great potential in solving classification problems. The field of deep learning started gaining popularity in 2012 when Alexnet al.

demonstrated their deep network architecture named AlexNet for image classification which outperformed.

Data Collection

Data Collection is a fundamental need to proceed with the above mentioned workflow. Here to extract mathematical equations we collected few types of handwritten alphabets , handwritten digits and handwritten mathematical symbols. These datasets are downloaded from kaggel.

Source : https://www.kaggle.com/xainano/handwrittenmathsymbols

Example Images :

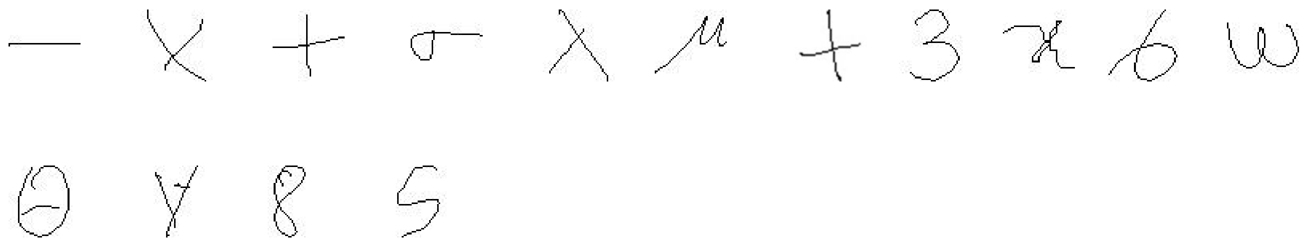


Figure 2. Math symbols,digits and alphabets images

Data Pre-processing

Pre-processing of the image is the procedure which encompasses changes and modifications to the image to make it suitable for our model to train and classify .

Here we followed below steps :

Changing Shape

we converted images in the dataset into the same dimensions (28x28). So that we can train model with all images and classify ,recognise them without getting any shape errors.

Furthermore , Changing dimensions done for input images which contain mathematical equations. This we can see in line segmentation. After line segmentation we took every line and separated characters. To classify that characters we changed the shape of them and gave them to the model .

Binary conversion

Firstly, Converted all input images into grayscale images and by fixing the threshold value we changed it as a binary image. From this binary image one can easily recognise the characters .

csv format conversion

Accessed the folder in which each folder has one specific character ,then flattened them and appended every symbol and made a dataset.

Segmentation

line segmentation

For line segmentation, we should first detect the line. So assumed that there is some gap between lines and there is some interaction between exponential characters and lines. From that assumptions extracted lines from images.

Example:

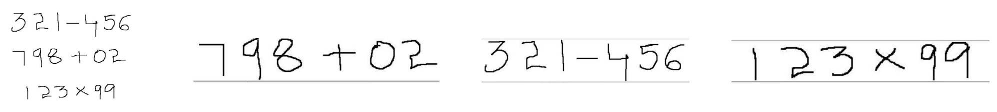


Figure 3 .Segmented Lines

From the above images we can see that those extracted lines are not in order . By comparing Y coordinates sorted them in order and gave them to character segmentation code.

Character segmentation

After detecting all the lines,we have to send the extracted line images to the text segmentation code.It will divide all characters in a line.

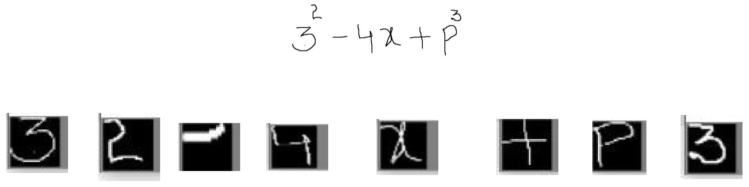
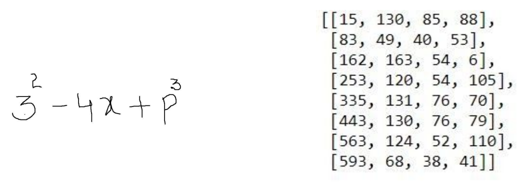


Figure 4. Segmented characters from a line

For us it is easy to say whether the given number is an exponent or not but for the model it's not that easy.So here by finding coordinates of sorted contours ,separated exponential terms.

Example :



```
[[15, 130, 85, 88],
 [83, 49, 40, 53],
 [162, 163, 54, 6],
 [253, 120, 54, 105],
 [335, 131, 76, 70],
 [443, 130, 76, 79],
 [563, 124, 52, 110],
 [593, 68, 38, 41]]
```

Figure 5. Coordinates of every character in a equation

The values above are X,Y,W,Z values. In Y values we can see 2 values are very low compared to others. So those are powers.As mentioned above we took the average of the Y column and from that threshold we separated all powers.Other character segmentations are done using basic python code.

Label prediction

To predict labels we trained Convolutional Neural Networks(CNN) using Keras . And the output of this CNN is given to the Dense layer .
As a base, Initialised the Sequential model and added all below layers to that model.

Layers

Layer 1 :

Firstly, added a conv2D layer with 30 filters and 5 x 5 kernel size and gave input shape as (1, 28,28) because we reshaped all images to 1x28x28 and converted them into binary images as mentioned in data preprocessing.Finally used the activation function Relu.

Why Relu ?

When Relu is used in   the training of a Neural Network then it would be faster compared to sigmoid . As we know that the derivative of Sigmoid is y(1-y) where y is sigmoid . So gradients will vanish easily.But in Relu there is no such problem.It is a zero centered activation function where values lie between 0 to infinity.

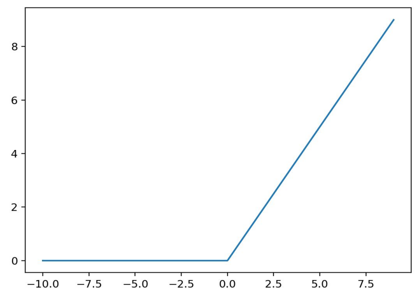


Figure 6. Relu Plot

Layer 2

Next layer we took Max Pooling layer 2D where pool size is (2,2).It is a pooling operation that selects the maximum element from the region of the feature map covered by the filter as mentioned in the below image.

Why max pooling ?

Pooling layer is not necessary but advisable to reduce the number of extracted features and to avoid overfitting.

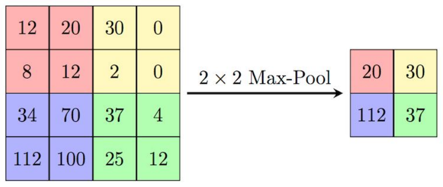


Figure 7. Max-pool

Layer 3

Next layer we gave conv2D and filter size as the previous layer output . After max pooling with 2x2 pool size 1st layers 30 became 15. And used Relu as an activation function , kernel size 3x3.

Layer 4

Used max pooling with pool size (2,2)

Layer 5

Used Dropout with probability 0.2 as 5th layer.

Why Dropout ?

It is one of the regularization techniques used to reduce overfitting .

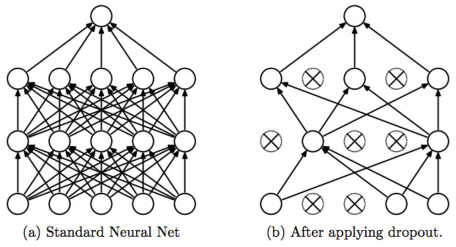
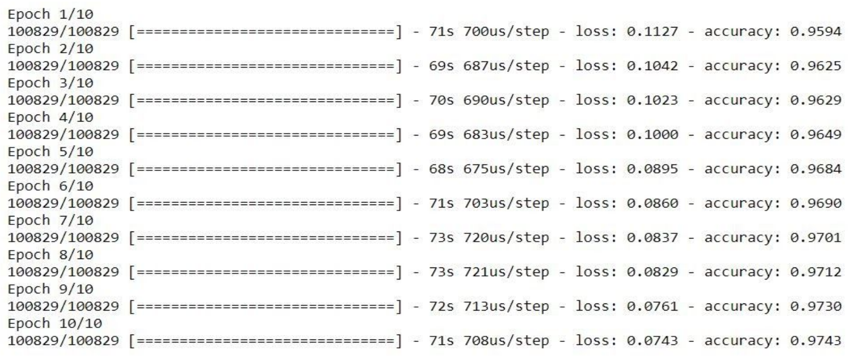


Figure 8. Dropout

Later layers

Later we used a flatten layer to flatten outputs and given to few dense layers.

Model training and accuracy .

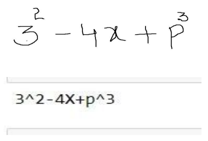


```
Epoch 1/10
100829/100829 [==============================] - 71s 700us/step - loss: 0.1127 - accuracy: 0.9594
Epoch 2/10
100829/100829 [==============================] - 69s 687us/step - loss: 0.1042 - accuracy: 0.9625
Epoch 3/10
100829/100829 [==============================] - 70s 690us/step - loss: 0.1023 - accuracy: 0.9629
Epoch 4/10
100829/100829 [==============================] - 69s 683us/step - loss: 0.1000 - accuracy: 0.9649
Epoch 5/10
100829/100829 [==============================] - 68s 675us/step - loss: 0.0895 - accuracy: 0.9684
Epoch 6/10
100829/100829 [==============================] - 71s 703us/step - loss: 0.0860 - accuracy: 0.9690
Epoch 7/10
100829/100829 [==============================] - 73s 720us/step - loss: 0.0837 - accuracy: 0.9701
Epoch 8/10
100829/100829 [==============================] - 73s 721us/step - loss: 0.0829 - accuracy: 0.9712
Epoch 9/10
100829/100829 [==============================] - 72s 713us/step - loss: 0.0761 - accuracy: 0.9730
Epoch 10/10
100829/100829 [==============================] - 71s 708us/step - loss: 0.0743 - accuracy: 0.9743
```

Here we got 97.43% accuracy for this model .

After training segmented images are given to model and it will classify them

Example :



```
3^2-4X+p^3
```

Figure 9.Classification

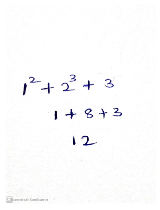
Evaluating answers

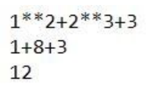
Here we are going to Evaluate whether the written answer is correct or wrong . To achieve this we should take the answer which we are going to evaluate then have to do the cleaning and segment it line by line and have to classify them . After all these things the eval function is used to all lines and appended outputs in a list.

Lastly we will comapre whether every value in the list is the same .If it is the same then the answer is correct, else it is wrong.

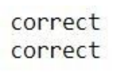
In the below example the first row is a question and the remaining rows are answers.So we compare the first row's output to every individual row's output . According to that we will conclude whether that answer is correct or wrong to that particular question.

Example :





```
1**2+2**3+3
1+8+3
12
```

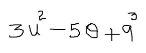


```
correct
correct
```

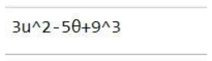
Figure 10. Evaluating answers

Results

Extraction

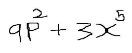


$3u^2-5\theta+9^3$

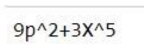


```
3u^2-5θ+9^3
```

Result 1



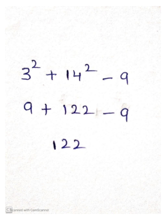
$9p^2+3x^5$



```
9p^2+3X^5
```

Result 2

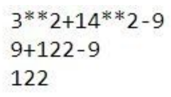
Evaluating answers



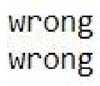
3^2+14^2-9

$9+122-9$

122

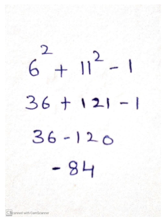


```
3**2+14**2-9
9+122-9
122
```



```
wrong
wrong
```

Result 3

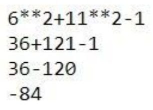


6^2+11^2-1

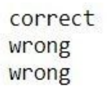
$36+121-1$

$36-120$

-84

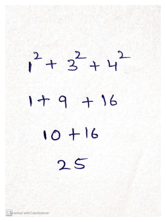


```
6**2+11**2-1
36+121-1
36-120
-84
```



```
correct
wrong
wrong
```

Result 4

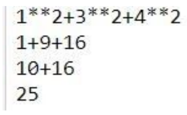


$1^2+3^2+4^2$

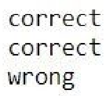
$1+9+16$

$10+16$

25



```
1**2+3**2+4**2
1+9+16
10+16
25
```



```
correct
correct
wrong
```

Result 5

Conclusion

In this Project we mainly focused on polynomial equations and evaluating basic math answers. To extract remaining math expressions like integration(upper limit and lower limit), fractions..etc the process is same but we need to add code in character segmentation part like we did for exponential terms(By comparing Y coordinates).This makes the process of writing equations easier instead of writing in latex code or writing in other documents. We used convolutional neural networks to classify characters , In preprocessing changed shapes of the images and converted them to binary images.Line segmentation and exponential segmentation is done by using finding contours and Y coordinates . Finally , Using “eval” function calculated every line and assigned whether the answer was correct or wrong .

Appendix

Codes

CNN Model

```
model = Sequential()
model.add(Conv2D(30, (5, 5), input_shape=(1 , 28, 28), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(15, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(24, activation='softmax'))
# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(np.array(l), cat, epochs=30, batch_size=200,shuffle=True,verbose=1)
```

CSV Conversion

```
def load_images_from_folder(folder):
    train_data=[]
    for filename in os.listdir(folder):
        img = cv2.imread(os.path.join(folder,filename),cv2.IMREAD_GRAYSCALE)
        img=~img
        if img is not None:
            ret,thresh=cv2.threshold(img,127,255,cv2.THRESH_BINARY)

00000000000000000000000000000000ctrs,ret=cv2.findContours(thresh,cv2.RETR_EXTERNA
L,cv2.CHAIN_APPROX_NONE)
            cnt=sorted(ctrs, key=lambda ctr: cv2.boundingRect(ctr)[0])
            w=int(28)
            h=int(28)
            maxi=0
            for c in cnt:
                x,y,w,h=cv2.boundingRect(c)
                maxi=max(w*h,maxi)
                if maxi==w*h:
                    x_max=x
                    y_max=y
                    w_max=w
                    h_max=h
            im_crop= thresh[y_max:y_max+h_max+10, x_max:x_max+w_max+10]
            im_resize = cv2.resize(im_crop,(28,28))
            im_resize=np.reshape(im_resize,(784,1))
            train_data.append(im_resize)
    return train_data
```

Extracting equation Code :

```
image = cv2.imread('1.JPG',cv2.IMREAD_GRAYSCALE)
if image is not None:
    #images.append(img)
    image=~image
    ret,thresh=cv2.threshold(image,127,255,cv2.THRESH_BINARY)

    ctrs,ret=cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    cnt=sorted(ctrs, key=lambda ctr: cv2.boundingRect(ctr)[0])
    w=int(28)
    h=int(28)
    train_data=[]
    #print(len(cnt))
    rects=[]
    for c in cnt :
        x,y,w,h= cv2.boundingRect(c)
#          print(x,y,w,h)
        rect=[x,y,w,h]
        rects.append(rect)
    #print(rects)
    bool_rect=[]
    for r in rects:
        l=[]
        for rec in rects:
            flag=0
            if rec!=r:
                if r[0]<(rec[0]+rec[2]+10) and rec[0]<(r[0]+r[2]+10) and r[1]<(rec[1]+rec[3]+10) and
rec[1]<(r[1]+r[3]+10):
                    flag=1
                l.append(flag)
            if rec==r:
                l.append(0)
        bool_rect.append(l)
    #print(bool_rect)
    dump_rect=[]
    for i in range(0,len(cnt)):
        for j in range(0,len(cnt)):
            if bool_rect[i][j]==1:
                area1=rects[i][2]*rects[i][3]
                area2=rects[j][2]*rects[j][3]
                if(area1==min(area1,area2)):
                    dump_rect.append(rects[i])
    #print(len(dump_rect))
    final_rect=[i for i in rects if i not in dump_rect]
    #print(final_rect)
    i=10
    q=[]
    for r in final_rect:
        x=r[0]
        y=r[1]
        w=r[2]
        h=r[3]
        print(x,y,w,h)
        k=[x,y,w,h]
        q.append(k)
        im_crop =thresh[y:y+h+10,x:x+w+10]


        im_resize = cv2.resize(im_crop,(28,28))
        cv2.imwrite('ROI_{}.jpeg'.format(i+100), im_resize)
        cv2.imshow("work",im_resize)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
        i+=1
        im_resize=np.reshape(im_resize,(1,28,28))
        train_data.append(im_resize)

summ=0
for i in q:
    summ+=i[1]

avg=summ/len(q)
b=[]
print(avg)

for i in q:
    if i[1]< avg*0.75:
        b.append(q.index(i))

s=''
for i in range(len(train_data)):
    train_data[i]=np.array(train_data[i])
    train_data[i]=train_data[i].reshape(1,1,28,28)
    result=loaded_model.predict_classes(train_data[i])
    if(result[0]==10):
        s=s+'-'
    if(result[0]==11):
        s=s+'('
    if(result[0]==12):
        s=s+')'
    if(result[0]==0):
        s=s+'0'
    if(result[0]==1):
        s=s+'1'
    if(result[0]==2):
        s=s+'2'
    if(result[0]==3):
        s=s+'3'
    if(result[0]==4):
        s=s+'4'
    if(result[0]==5):
        s=s+'5'
    if(result[0]==6):
        s=s+'6'
    if(result[0]==7):
        s=s+'7'
    if(result[0]==8):
        s=s+'8'
    if(result[0]==9):
        s=s+'9'
    if (result[0]==13):
        s=s+'+'
    if (result[0]==14):
        s=s+'∀'
    if (result[0]==15):
        s=s+'μ'
    if (result[0]==16):
        s=s+'p'
    if (result[0]==17):
        s=s+'σ'
    if (result[0]==18):
        s=s+'*'
    if (result[0]==19):
        s=s+'u'
    if (result[0]==20):
        s=s+'v'
    if (result[0]==21):
        s=s+'w'
    if (result[0]==22):
        s=s+'x'
    if (result[0]==23):
        s=s+'y'
print(s)

k=0
for i in b:
    s=s[0:i+k]+'^'+s[i+k:]
    k=k+1
    print(s)
print(s)
```

Evaluating answers Including Line segmentation

```
def extract_line(image, beta=0.7, alpha=0.002):
    img = image.copy()
    H,W = img.shape[:2]
    h5 = int(.02 * H)
    w5 = int(.02 * W)
    img[:h5,:] = [255,255,255]
    img[-h5:,:] = [255,255,255]
    img[:,:w5] = [255,255,255]
    img[:,-w5:] = [255,255,255]
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    th, threshed = cv2.threshold(gray, 127, 255,
cv2.THRESH_BINARY_INV|cv2.THRESH_OTSU)
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))
    dilation = cv2.dilate(threshed,kernel,iterations = 1)
    temp = dilation.copy()
    contours,hierarchy =
cv2.findContours(dilation,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE )
    areas = []
    for c in contours:
        areas.append([cv2.contourArea(c)**2])
    cont_thresh= alpha * max(areas)[0]
    mask = np.ones(dilation.shape[:2], dtype="uint8") * 255
    for c in contours:
        if( cv2.contourArea(c)**2 < cont_thresh):
            cv2.drawContours(mask, [c], -1, 0, -1)
    cleaned_img = cv2.bitwise_and(temp, temp, mask=mask)
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))
    dil_cleaned_img = cv2.dilate(cleaned_img,kernel,iterations = 10)
    cleaned_orig = cv2.erode(cleaned_img, kernel, iterations=1)
    hist = cv2.reduce(dil_cleaned_img,1, cv2.REDUCE_AVG).reshape(-1)
    th = 1
    H,W = img.shape[:2]
    uppers = np.array([y for y in range(H-1) if hist[y]<=th and hist[y+1]>th])
    lowers = np.array([y for y in range(H-1) if hist[y]>th and hist[y+1]<=th])
    diff_1 = np.array([j-i for i,j in zip(uppers,lowers)])
    diff_index_1 = np.array([True if j > beta*(np.mean(diff_1)-np.std(diff_1)) else False for j in
diff_1 ])
    uppers = uppers[diff_index_1]
    lowers = lowers[diff_index_1]
    uppers[1:] = [i-int(j)/3 for i,j in zip(uppers[1:], diff_1[1:])]
    lowers[:-1] = [i+int(j)/4 for i,j in zip(lowers[:-1], diff_1[:-1])]
    diff_2 = np.array([j-i for i,j in zip(uppers,lowers)])
    diff_index_2 = np.array([True]*len(uppers))
    for i,diff in enumerate(diff_2):
        if(i>0):
            if( (diff_2[i-1] < (diff/2)) and (( lowers[i-1]-uppers[i]) > ((lowers[i-1]-uppers[i-1])/5)) ):
                uppers[i] = uppers[i-1]
                diff_2[i] = diff_2[i]+diff_2[i-1]
                diff_index_2[i-1] = False
    diff_index = diff_index_2
    cleaned_orig_rec = cv2.cvtColor(cleaned_orig, cv2.COLOR_GRAY2BGR)

    return cleaned_orig, uppers[diff_index], lowers[diff_index]


def text_segment(Y1,Y2,X1,X2,box_num,line_name, dict_clean = dict_clean_img,\
             acc_thresh = 0.60, show = True):
    img = dict_clean[box_num][Y1:Y2,X1:X2].copy()
    L_H = Y2-Y1
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))
    dilation = cv2.dilate(img,kernel,iterations = 2)
    erosion = cv2.erode(dilation,kernel,iterations = 1)

    # Find the contours
    if(cv2.__version__ == '3.3.1'):
        xyz,contours,hierarchy =
cv2.findContours(erosion,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    else:
        contours,hierarchy =
cv2.findContours(erosion,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    areas = []
    for c in contours:
        areas.append([cv2.contourArea(c)**2])
    ct_th = 0.005 * max(areas)[0]
    cnts = []
    for c in contours:
        if( cv2.contourArea(c)**2 > ct_th):
            cnts.append(c)
    contours_sorted,bounding_boxes = sort_contours(cnts,method="left-to-right")
    char_locs = []

    img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)

    i = 0
    char_type =[]
    while i in range(0, len(contours_sorted)):
            x,y,w,h = bounding_boxes[i]
            exp = 0
            if i+1 != len(contours_sorted):
                x1,y1,w1,h1 = bounding_boxes[i+1]
                if abs(x-x1) < 10 and  (h1+h) < 70:
                    #print(h+h1)
                    minX = min(x,x1)
                    minY = min(y,y1)
                    maxX = max(x+w, x1+w1)
                    maxY = max(y+h, y1+h1)
                    x,y,x11,y11 = minX, minY, maxX, maxY

                    x,y,w,h = x,y,x11-x,y11-y
                    i = i+2
                    continue
            if(h<0.10*L_H and w<0.10*L_H):
                i=i+1
                continue
            char_locs.append([x-2,y+Y1-2,x+w+1,y+h+Y1+1,w*h])
            cv2.rectangle(img,(x,y),(x+w,y+h),(153,180,255),2)
            if i!=0:
```

```
            if y+h < (L_H*(1/2)) and y < bounding_boxes[i-1][1] and h < bounding_boxes[i-1][3]:
                exp = 1
                cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
        i = i+1
        char_type.append(exp)

    df_char = pd.DataFrame(char_locs)
    df_char.columns=['X1','Y1','X2','Y2','area']
    df_char['exp'] = char_type
    df_char['pred'] = df_char.apply(lambda c: predict(dict_clean[box_num],c['X1'],\
        c['Y1'],c['X2'], c['Y2'], acc_thresh=acc_thresh), axis=1 )
    df_char['pred_proba'] = df_char.apply(lambda c: predict(dict_clean[box_num],c['X1'],\
        c['Y1'],c['X2'], c['Y2'], acc_thresh=acc_thresh), axis=1 )
    df_char.apply(lambda c: cv2.putText(img, c['pred'], (c['X1']-10,35),
cv2.FONT_HERSHEY_SIMPLEX,
              1.5,(147,96,247), 3, cv2.LINE_AA), axis=1)
    df_char['line_name'] = line_name
    df_char['box_num'] = box_num
    return [box_num,line_name,df_char]

def sort_contours(cnts, method="left-to-right"):

    reverse = False
    i = 0
    if method == "right-to-left" or method == "bottom-to-top":
        reverse = True
    if method == "top-to-bottom" or method == "bottom-to-top":
        i = 1
    boundingBoxes = [cv2.boundingRect(c) for c in cnts]
    (cnts, boundingBoxes) = zip(*sorted(zip(cnts, boundingBoxes),
        key=lambda b:b[1][i], reverse=reverse))
    return (cnts, boundingBoxes)

def predict(img,x1,y1,x2,y2, acc_thresh = 0.60):
    gray = img[y1:y2, x1:x2]
    temp = gray.copy()
    kernel_temp = np.ones((3,3), np.uint8)
    temp_tmp = cv2.dilate(temp, kernel_temp, iterations=3)
    contours_tmp,hierarchy =
cv2.findContours(temp_tmp,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    if(len(contours_tmp) > 1):
        contours,hierarchy =
cv2.findContours(gray,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
        mask = np.ones(gray.shape[:2], dtype="uint8") * 0
        areas = [cv2.contourArea(c) for c in contours]
        max_index = np.argmax(areas)
        cnt=contours[max_index]
        cv2.drawContours(mask, [cnt], -1, 255, -1)
        gray = cv2.bitwise_and(temp, temp, mask=mask)

    grayN = process_img (gray, resize_flag = 0)

    classes = model.predict(grayN, batch_size=2)
    ind = np.argmax(classes)
    c = ['0','1','2','3','4','5','6','7','8','9','+','-','*','(',')']
    return c[ind]

def process_img (gray, resize_flag = 1, preproc = 0):
    gray = gray.copy()
    if (preproc == 0):
        gray = cv2.GaussianBlur(gray,(7,7),0)
    else :
        kernel = np.ones((3,3), np.uint8)
        gray = cv2.dilate(gray, kernel, iterations=1)
        gray = cv2.GaussianBlur(gray,(5,5),1)
        gray = cv2.dilate(gray, kernel, iterations=2)
        gray = cv2.erode(gray, kernel,iterations=2)
    while np.sum(gray[0]) == 0:
        gray = gray[1:]

    while np.sum(gray[:,0]) == 0:
        gray = np.delete(gray,0,1)

    while np.sum(gray[-1]) == 0:
        gray = gray[:-1]

    while np.sum(gray[:,-1]) == 0:
        gray = np.delete(gray,-1,1)

    rows,cols = gray.shape

    if(resize_flag) == 1:
        interpolation=cv2.INTER_AREA
    else:
        interpolation=cv2.INTER_CUBIC
    if rows > cols:
        factor = 20.0/rows
        rows = 20
        cols = int(round(cols*factor))
        gray = cv2.resize(gray, (cols,rows),interpolation=interpolation)
    else:
        factor = 20.0/cols
        cols = 20
        rows = int(round(rows*factor))
        gray = cv2.resize(gray, (cols, rows),interpolation=interpolation)
    colsPadding = (int(math.ceil((28-cols)/2.0)),int(math.floor((28-cols)/2.0)))
    rowsPadding = (int(math.ceil((28-rows)/2.0)),int(math.floor((28-rows)/2.0)))
    gray = np.lib.pad(gray,(rowsPadding,colsPadding),'constant')
    shiftx,shifty = getBestShift(gray)
    shifted = shift(gray,shiftx,shifty)
    grayS = shifted
    grayS = grayS.reshape(1,1,28,28)
    grayS = grayS/255

    return grayS

def getBestShift(img):
    cy,cx = ndimage.measurements.center_of_mass(img)
    rows,cols = img.shape
    shiftx = np.round(cols/2.0-cx).astype(int)
    shifty = np.round(rows/2.0-cy).astype(int)

    return shiftx,shifty

def shift(img,sx,sy):
    rows,cols = img.shape
    M = np.float32([[1,0,sx],[0,1,sy]])
    shifted = cv2.warpAffine(img,M,(cols,rows))
    return shifted
train_datagen = ImageDataGenerator(
    data_format='channels_first',
    zca_whitening = True,
    rotation_range = 0.2)

def evaluate(df,A,B,X,Y):
    '''Function to evaluate mathematical equation and give bool output
    Input: Dataframe
           Values
    Output:
        Boolean T/F
    '''
    #Evaluating Expression
    actual = A*X*X+(B*Y)

    pred = df["exp"].apply(lambda d: "**" if d==1 else "")
    pred = "".join(list(pred+df["pred"]))


    matchesN = re.findall('^\-\-', pred)
    if(len(matchesN) > 0):
        for s in matchesN:
             pred = pred.replace(s,'')



        #looking for broken 5's
    matches5 = re.findall(r'5\*\*-\D*', pred)
    if(len(matches5) > 0):
        for s in matches5:
            sn = s.split('5**-')
            snew = sn[0]+'5'+sn[1]
            pred = pred.replace(s,snew)



    matchesB_left = re.findall(r'\d\(\d', pred)
    matchesB_right = re.findall(r'\d\)\d', pred)

    if(len(matchesB_left) > 0 or len(matchesB_right) > 0):
        for s in matchesB_left:
            sn = s.split('(')
            snew = sn[0]+'*('+sn[1]
            pred = pred.replace(s,snew)

        for s in matchesB_right:
            sn = s.split(')')
            snew = sn[0]+')*'+sn[1]
            pred = pred.replace(s,snew)

    a.append(pred)

df_lines = pd.DataFrame()
r=1
box = img
H,W = box.shape[:2]
cleaned_orig,y1s,y2s = extract_line(box)
x1s = [0]*len(y1s)
x2s = [W]*len(y1s)
df = pd.DataFrame([y1s,y2s,x1s,x2s]).transpose()
df.columns = ['y1','y2','x1','x2']
df['box_num'] = r

df_lines= pd.concat([df_lines, df])

dict_clean_img.update({r:cleaned_orig})
dict_img.update({r:box})
df_lines['line_name'] = ['%d%d' %(df_lines.box_num.iloc[i],df_lines.index[i]) \
        for i in range(len(df_lines))]

list_chars = list(df_lines.apply(lambda row: text_segment(row['y1'],row['y2'],\
                row['x1'],row['x2'], row['box_num'],row['line_name']), axis=1))

tfback._get_available_gpus = _get_available_gpus
tfback._get_available_gpus()
a=[]
```

```
df_chars = pd.DataFrame(list_chars)
df_chars.columns = ['box_num', 'line_name', 'char_df']

box_nums = df_chars.box_num.unique()
char_area_list = []
df_chars['char_df'].apply(lambda d: char_area_list.append(list(d['area'])) )
gamma = 0
max_ar = max([max(i) for i in char_area_list])
ar_thresh = max_ar*gamma
df_chars['char_df'] = df_chars['char_df'].apply(lambda d: d[d.area > ar_thresh] )

for bn in box_nums:
   box_img = dict_clean_img[bn]
   box_img_1 = dict_img[bn]
   box_img = cv2.cvtColor(box_img, cv2.COLOR_GRAY2BGR)

   df = df_chars[df_chars.box_num == bn].copy()

   df_l = df_lines[df_lines["box_num"]==bn].copy()

   df['char_df'].apply(lambda d: d.apply(lambda c: cv2.rectangle(box_img, \
     (c['X1'],c['Y1']),(c['X2'], c['Y2']),(255*(c['exp']==1),180,0),2+(2*c['exp'])), axis=1 ) )

   df['line_status'] = df['char_df'].apply(lambda d:
evaluate(d[["pred","exp","pred_proba"]],A,B,X,Y))

   scale_percent = 200

b=[]
for i in a:
   b.append(eval(i))
for i in range(1,len(b)):
   if b[0] == b[i]:
      print("correct")
   else:
      print("wrong")
```

References

- https://www.researchgate.net/publication/331617158_Recognition_and_Solution_for_Handwritten_Equation_Using_Convolutional_Neural_Network
- https://www.geeksforgeeks.org/introduction-deep-learning/
- https://vtechworks.lib.vt.edu/bitstream/handle/10919/46724/Bruce_JR_T_2014.pdf;sequence=1
- https://www.kaggle.com/xainano/handwrittenmathsymbols