CVIT Workshop Day 2 ~~day 1~~ session 1

Document Images

skew correction

skew makes difficult to read & understand text

compute the skew angle/ distortion angle, and then correct it

skew refers to the rotation of the document from its correct orientation

https://tinyurl.com/24t6szta Colab Notebook for Skew Correct

directly connect Colab to Github

create a folder named Day 2

the notebook would be updated on Github whenever you ma a ~~Git~~ push.

~~desri~~

the libraries involved are skimage.transform

hough_line

hough_line_peaks

probabilistic_hough_line

rotate

canny

skimage.io imread (can also use cv2, but it reads images in BGR

skimage.color rgb2gray

suggestion - read the documentations for the CV2 and skimage libraries

```
from google.colab.patches import cv2_imshow
```

Steps

first mount the Google Colab Notebook to drive, and set up the environment to

a code nuance

every ~~to~~ image may not have just 3 channels

~~the~~ given image just has 2 channels and if we use

```
gray = rgb2gray(image[:,:,:3])
```

it would throw an error because the image just has 2 channels. (~~and~~

rgb2gray only works for 3 color channels

~~so if there are only two color channels~~

there are usually four color channels, ~~there also~~ the fourth one being alpha channel. rgb to gray only works on RGB channels. Hence we use this; to use only the first 3 channels.

~~Thus~~ If the no. of channels is <=2, then the gray is simply the input image itself. Thus we would have

```
if len(image.shape) > 2:
    gray = rgb2gray(image[:,:,:3])
else: gray = image
```

Edge Detection - Canny

remember Sobel filters?

Canny Edge Detection is similar with extra preprocessing and refinement steps like noise reduction, non maximal suppression, etc.

if the edges are very fine, very sharp then we reduce the size of the filter, and vice versa.

Function signature:- canny(image, sigma=1.0, low_threshold=None, high_threshold=None, mask=None, use_quantiles=False, mode='constant', cval=0.0)

Canny Edge Detection - OpenCV Docs

multi-stage algorithm

1) Noise Reduction

2) Finding intensity gradient of the image

$$\text{Edge\_Gradient } (G) = \sqrt{G_x^2 + G_y^2}$$

$$\text{Angle } (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

the notebook is written in skimages, but mostly CV2 is used for most purposes, the canny can also be ~~implemented~~ run in CV2 instead of skimages

3) non-maximum suppression

4) hysteresis thresholding

Noise Reduction

remove noise with a 5×5 Gaussian filter

Intensity Gradient

non maximum suppression means suppressing values other than the maximum values. So suppose we have 3 ~~values, or~~ high values. ~~...~~ so we might select the highest one and neglect the rest.

Hough Transform

details ~~...~~ will be dealt with in D.I.P course.

used to detect geometrical ~~sh~~ shapes —— circles, triangles

how? parametric space

accumulator

in any kind of transform — first we define it mathematically

for eg. circle needs three values —— (x, y, R)

centre coordinates

for a line we need a slope and a point ~~...~~

in image processing origin lies on the top-left corner

to detect lines, we calculated the edges. we take a point, and see if a line passes through the point

np.linspace is used

```
angles = np.deg2rad( np.arange (0.1, 180.0)
```

for any structure. If a lot of lines are passing through it, then it is a point of a structure

h — accumulator

hough space is (r, θ)

r → dist, θ → slope

every (r, θ) is a line, which might be passing through a lot of pts. say 60, then h is 60, because the line got 60 pts

transform to Huff space to because in the Cartesian space the complexity to identify line would be 180 × (no. of points possible)

If the image size increase, the search space for the Hough space does not increase.

Hence computation becomes much easier

h.shape = (2001, 180)

2001 → distance from origin

h is a matrix of this size. Each value corresponds to the no. of points having that particular (r & θ).

We want to know those parts with the maximum points.

We had given the resolution for theta (during linspace) but we did not give the resolution for r.

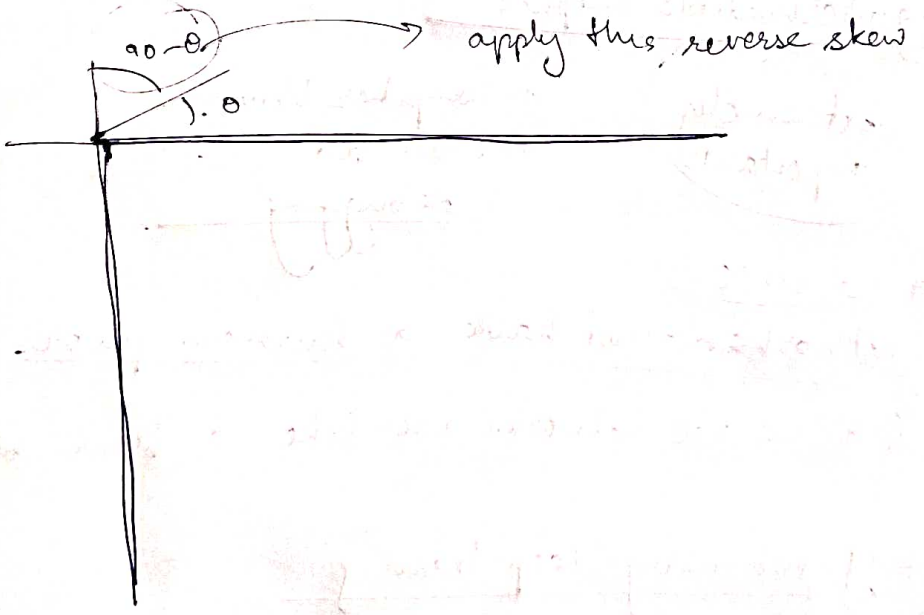
accumulator value
how many points lie on that line

hough_line_peaks
takes as input the hough_line_space

accum, angles, dists = hough_line_peaks (h, theta, d)

list (zip(* hough_line_peaks (h, theta, d)))

still there are outliers.
we use the property that maximum lines obtained would be correctly oriented.



affine transformation = translation + rotation + scaling

translate

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

2D rotate

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

shear

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

rotation is a combination of two shears

cv2. getRotationMatrix2D (center, skew_angle, 1.0)

cubic interpolation done in the image while (rotation + scaling) need interpolation

show_intern = False

shows all intermediate steps for debugging.

Visualizing intermediate outputs

(extremely important)

in Computer Vision one can do visual debugging.

General advice

neural networks do not have a human proof

overall a neural networks are like a black box

advantages of not using deep learning

- too powerful.
- compute intensive
- huge amts of data

second notebook

https://tinyurl/bddrucyb

OCR

word level segmentation } both done in grayscale only
line level segmentation }

done in CV2.

Steps

gray = cv2.cvtColor (image, cv2.COLOR_BGR2GRAY)

def binarize_and_word_detect_contours

load & convert to grayscale

denoise the image

1) denoise ~~the~~ with Gaussian blur, kernel 5x5

2) advanced denoising technique

cv2.fastNlMeansDenoising (blurred, None, h=10,
templateWindowSize

denoising necessary to segregate words, because word identification requires detection of a continuous white space.

If we did not denoise, then multiple words could be detected as one.

thresholding

cv2.threshold (denoised, 127, 255, cv2.THRESH_BINARY)

make the text thicker

we apply a dilation operation

there are white spaces within continuous text, to remove these we use dilation operation

idea is to use morphological operations (dilation/erosion)

dilated_image = cv2.dilate(binary_image, text_kernel, iterations)

return dilated_image

function

getStructuringElement ——— defines a kernel

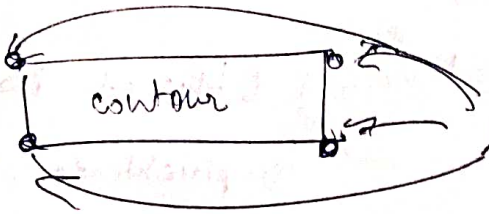
cv2.getStructuringElement(cv2.MORPH_RECT, (H, W))

we will use a rectangular kernel here because this is being done for words which are rectangular roughly

contours = cv2.findContours(wordTextDocument, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

(what is returned?)

RETR_EXTERNAL retrieves only the external points of each contour



returns only these

CHAIN_APPROX_SIMPLE: only the four corners are computed

(what is computed?)

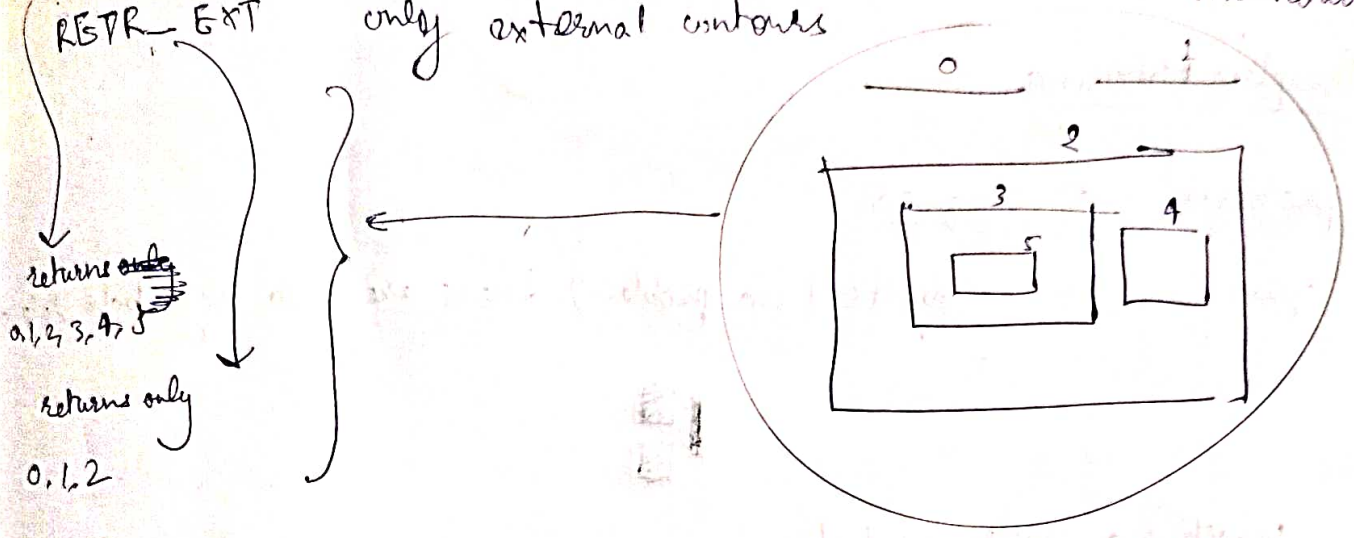
drawContours

(draw these contours)

Contours Hierarchy

RETR_LIST returns with no hierarchy, but all internal contours also returned

RETR_EXT only external contours



now we have got all the words segmented from each other.

Suppose we wanted EXACT boundaries of the words then after having treated these word blobs. we dilate with a bigger kernel to get only the word blob and then do fine contouring.

. Now, we can pass each word separately into the deep learning model. word level OCR

for line segmentation we would use a kernel with a larger width.

Then,

Image → inverse operator → dilation with a larger (wider) kernel → erosion kernel size

Erosion & dilation can be used as noise removal techniques also.

https://tinyurl.com/3t9ptvmf

CVIT Workshop Day 2

session 2

~~extracting a~~

Template Matching

problem with cheques

apart from the ~~no~~ digits (no problem) there are also symbols li

which is a single symbol.

partwise matching — break an image into smaller parts, template match each of these individual parts

we will first have a reference image containing all characters & digi in the correct order. This will be a clear image

Steps

1) extracting the bounding boxes from the reference image

cv2.findContours returns two values, the contours & the hier

~~either we can~~

```
refCnts = cv2.findContours(ref.copy(), cv2.RETR_EXTERNAL
                           cv2.CHAIN_APPROX_SIMPL
refCnts = imutils.grab_contours(refCnts)
refCnts = contours.sort_contours(refCnts, method = "left-to-r
```

def extract_digits_and_symbols

iterate over the contours.

rois = []

regions of interest

the region inside the bounding box

the four corners are enough to get the region of interest.

two other params:

min height, min width

use the fact that the height & width of the "other" ~~charact~~ symbols are much smaller than the digits. ~~to~~

On each contour, we use

(cX, cY, cW, cH) = cv2.boundingRect(c) } gives the origin coordinates, the height, the width of the bounding box of the contours

~~to obtain the~~

if cW >= minW and cH >= minH :

it is a digit

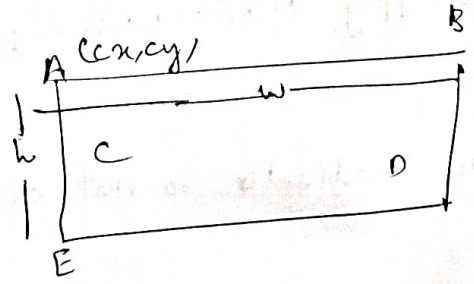
roi = image[cY : cY + cH, cX : cX + cW]

rois.append(roi)

Note that we had resized the image in the beginning so that we can set the hyperparameters such as minW & minH.

more detailed explanation

suppose we have 6 contour points:



we would be ~~given~~ returned cx, cy, cw, ch using cv2.boundingRect

Now, once we have segregated the digits, we need to segregate the characters which have more than one continuous regions.

Note that each of them is just 3 regions large. So just ~~can~~ combine 3 reg at a time when you come to characters.

Now we map the rois to the names of the digits/characters represented by them.

Now, we again perform a resize so that template matching is possible.

Now coming ~~to~~ to the Query Image

we will first crop the bottom part of the cheque because that contains the ~~num~~ code.

first convert to grayscale

then perform ~~morphological operations~~ morphology operation:

blackhat = cv2.morphologyEx(gray, cv2.MORPH_BLACKHAT, rectKer

finds the dark regions against a light.

note that we could also have performed a binary inversion. but morphologyoperation is better in most cases.

~~to compute the~~

Now we need to extract the bounding boxes of the query image.

first we can divide the whole cheque no. into three words/groups

~~first~~ so we would need to apply dilation (usual procedure)

BUT

first we apply a horizontal gradient, so that all the horizontal

So we apply a Sobel filter to remove external edges; which includes watermark

the highest values obtained by this are made 255. Others are made 0.

Closing (dilation followed by erosion)

closing small holes inside foreground images

cv2.morphologyEx(gradX, cv2.MORPH_CLOSE, rectKernel)

thresholding

move the whites to 255, rest to 0

skimage.clear_border

this removes the "external borders" in the image

does not remove the lower boundary because it is not continuous,

thus it wants to detect a continuous boundary

(Note that these operations are best performed on binary images with only 0 & 1)

To separate the rest of the contours from those corresponding to the characters/digits, we use the width/height thresholding

(the contours corresponding to the edges have much smaller contour size than the characters/digits)

Now we need to get the content inside the bounding boxes around the bunches of characters/digits that we just formed

We again draw contours and bounding boxes around each character/digit

We will now template match with our stored information from the reference image

result = cv2.matchTemplate

cv2.TM_CCOEF = correlation coefficient

cv2.TM_CCORR = cross correlation

$$R(x,y) = \sum_{x',y'} \left(T'(x',y') \cdot I'(x+x', y+y') \right)$$

normalized by the size of the kernel

$$R(x,y) = \sum_{x',y'} \left(T(x',y') \cdot I(x+x', y+y') \right)$$

T → template

I → Image

output of ~~match~~ template matching will be another images.

If we are doing difference matching, then a black dot/region the matching image would correspond to a good match.

~~min_loc, max_loc~~

~~cv2.minMaxLoc(res)~~

cv2.minMaxLoc(result)

~~more~~

However template matching is not ideal for real life images - matching.

In these cases it would be best to train your own Hoh + Linear SVM Classifier or a CNN.

Prof. Ravi Kiran S Presentation

less availability of data

aspect ratio extremely high

need a method which does not require resizing.
because manuscripts come in different sizes
just to segment the lines

solutions

1) increase ~~max~~ each image to max width & max height
problems due to huge compute required
image graining problem

* 2) breaking horizontally (patching)

seem understanding

find all minimum energy path calculation and remove them from the image

take away :— the mindset, the thought process
and not quite so much the technique