Competition Overview and Postmortem

Monday, September 14, 2020 5:04 PM

- [NOTE 9/13/20: To be addended after competition winners publish their code and I can review it. At this point I've only seen the one solution from the end of competition webinar from leading teams.]

Competition Overview

- This was a really fun competition which required creating a pipeline of pretty doable tasks. The challenge was to solve a sudoku puzzle given an image of the puzzle that was randomly colored, rotated, and the grid lines and digits randomly varied even within a single puzzle. https://www.aicrowd.com/challenges/ai-for-good-ai-blitz-3/problems/sudoku
- I made a nearly perfect solver solving 97.4% of the puzzles correctly, which unfortunately put me in 13th place (a bunch of people solved 100% of the puzzles correctly). All of the errors in my model came from the digit recognizer which had an accuracy > 99.9%, but because each puzzle had 81 digits (counting blank as a digit) those very few errors could throw the whole thing off.
- If I had had more time I'm sure I could have gotten the digit recognizer to work 100% of the time using one or a combination of:
    - transfer learning from the images of the digits in the training puzzles
    - just simply improving my homemade digit recognizer
    - incorporating an already produced OCR tool like pytesseract
    - Using the rules of the Sudoku puzzle to inform the digit classification
    - stacking together a few different digit classifiers
- I'm very curious to see if the competition winners did anything drastically different than me to get 100% perfect digit recognition

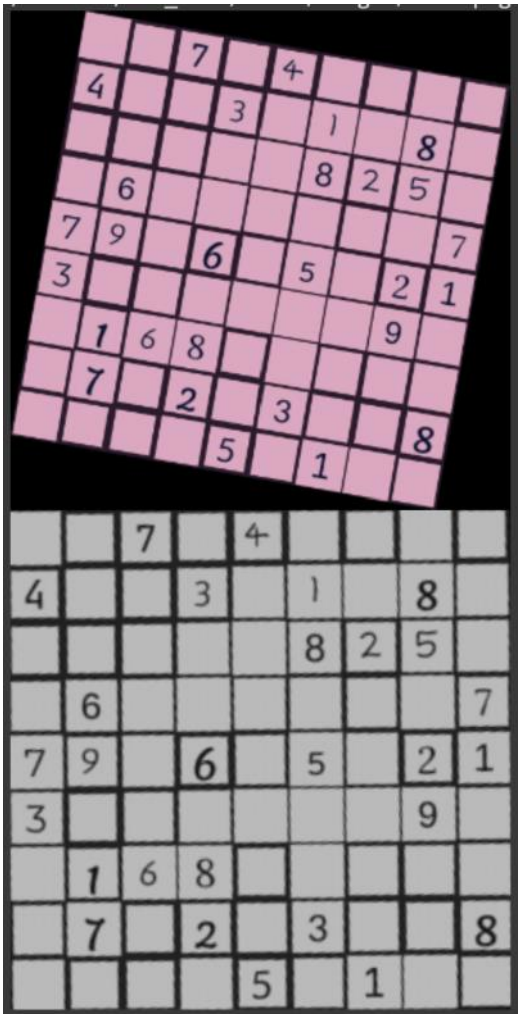
Model Overview

- For this competition my solution pipeline consisted of basically three steps
    1. digit extraction
    2. digit classification
    3. puzzle solving
- For digit extraction I came up with a 100% effective method for this particular dataset. I kept it very simple and specific to this dataset, as written this solution isn't generalizable to any image of a sudoku puzzle, it was built for and only works for these images.
    - I think it's an important skill to know when you should build something generalizable and when you should build something specific. In this highly time constrained competition this specific solution (which worked 100% of the time) was the right approach
- The code below takes a grayscale image and deroates the image by deteriming the angle the puzzle is rotated by fitting a square to the binarized image and then using a simple geometric perspective transformation to return the derotated image at the original image size

```
def sqdif(bingray,sq,ang):
    rsq=imutils.resize(imutils.rotate_bound(sq,ang),width=bingray.shape[0],height=bingray.shape[0])

    return np.sum(np.abs(bingray-rsq))
```

```
def derotate(img):
    sq=np.ones(img.shape)*255
    imsz = img.shape[0]

    # binarize by blurring then using Otsu's method
    # blur=cv2.GaussianBlur(gray,(1,1),0)
    _,bingray = cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

    rotang=minimize_scalar(lambda ang: sqdif(bingray,sq,ang),
                           bounds=[-45,45], method='Bounded').x

    if rotang<0:
        triang=np.tan(np.pi/4+rotang*np.pi/180)
        act_x=imsz/2*(1+triang)
        act_y=imsz/2*(1-triang)

        pts1 = np.float32([[0,act_y],[act_x,0],[imsz-act_x,imsz],[imsz,imsz-act_y]])

    else:
        triang=np.tan(np.pi/4-rotang*np.pi/180)
        act_x=imsz/2*(1-triang)
        act_y=imsz/2*(1+triang)

        pts1 = np.float32([[act_x,0],[imsz,imsz-act_y],[0,act_y],[imsz-act_x,imsz]])

    pts2 = np.float32([[0,0],[imsz,0],[0,imsz],[imsz,imsz]])
    M = cv2.getPerspectiveTransform(pts1,pts2)

    return cv2.warpPerspective(img,M,(imsz,imsz))
```

- Digit extraction is then as simple as breaking the grid into 9x9 squares

- 
- 

- For the digit recognition I tried a number of different approaches but wound up creating my own training data using PIL and training a CNN to classify those images
    - To make my own training data I found a number of fonts which best matched the ones used in the image and randomly changing a bunch of parameters of the image (lines, brightness, etc)

```
offmn=-3
# Line max offset +1
offmx=4
# Line thickness min
lnthkmn=1
# Line thickness max +1
lnthkmx=4

eveimg=np.ones((imsz,imsz))
imgname=0
for num in ['1','2','3','4','5','6','7','8','9','blank']:
    t_start = time.time()
    for fontfile, pos in [('arial.ttf',(10,3)),('cambria.ttc',(10,2)),('corbel.ttf',(10,4)),
                    ('mvboli.ttf',(8,-2)),('segoepr.ttf',(7,-5)),('pala.ttf',(10,7)),
                        ('aldrich.ttf',(11,6)),
                        ('C:/Users/isaac/AppData/Local/Microsoft/Windows/Fonts/lemonada.ttf',(9,-4))]:
        # create font object with the font file and specify size
        font=ImageFont.truetype(fontfile,24)

        for _ in range(imgs_per_font):
            # increment the image name counter (global to all numbers and fonts)
            imgname+=1

            # convert to PIL
            textimg=Image.fromarray(np.uint8(
                eveimg.copy()*np.random.randint(low=150,high=256)))
            # creat a draw instance
            draw = ImageDraw.Draw(textimg)

            # draw the message on the background
            if num!='blank':
                draw.text(pos, num, fill='rgb(0,0,0)', font=font)

            # draw the lines
            # left side line
            if np.random.rand()<linprob:
                ofst=np.random.randint(low=offmn, high=offmx)
                thkn=np.random.randint(low=lnthkmn, high=lnthkmx)
                draw.line([(0+ofst,0),(0+ofst,imsz)],width=thkn)
            # right side line
            if np.random.rand()<linprob:
                ofst=np.random.randint(low=offmn, high=offmx)
```

- I then trained a CNN using fastai to classify the images
- To solve the puzzle I used a deterministic algorithm from Peter Norvig https://norvig.com/sudoku.html
-
- The trick in the whole pipeline was that when the digit classifier made a mistake the sudoku solver would error out and I used this as an impetus to rerun the digit classification with more tta using the following setup which tries 5 times with increasing tta before giving up

```
n_tta=10
attempt_cntr=0
failedsolve=True
while failedsolve:
    attempt_cntr+=1

    tst_dl=learn.dls.test_dl(test_imgs)

    learn.epoch=0
    preds = learn.tta(dl=tst_dl,n=n_tta)[0]

    ### Attempt straightforward solution ###
    curgrid=pd.Series(torch.argmax(preds,dim=1).numpy()).map(labeldict)
    curgrid=''.join(curgrid)

    try:
        solved=solve(curgrid)
        solved_grid=''.join(solved.values())

        # failedsolve = False
        return solved_grid
    except:
        print('Solving failed for '+impath)
        n_tta+=2

    if attempt_cntr>=5:
        print('Giving up on '+impath)
        return 'sudontku'
```

- This worked really well, but not perfectly. On the whole training set I think less than one percent of the puzzles failed ('sudontku') the other errors came from getting unlucky where a mislabeled digit still worked and a puzzle solution was able to be found

Postmortem

- I ran out of time to work on this. I had a lot of other ideas I wanted to try out to deal with the cases where the sudoku solver errored. I also had a lot of ideas on how to make the digit classifier work 100% of the time, but because mine was working so well I wasn't prioritizing that anyways.
- I'm looking forward to seeing how the other participants built their digit classifiers. The only thing I'm aware of other doing was transfer learning using the images of the digits in the training puzzles