نویسه خوان

محمدرضا صاحب زاده

۲۷ مرداد ۱۴۰۳

فهرست مطالب

پروژه اینست که ما باید با دو روش مدلی را تعریف کنیم که بتوان اعداد تصویر را حدس زد اولین روش اینگونه هست که باید با روش های پردازش تصویر عکس داده شده را تبدیل به اعداد جدا کرده و در آخر کل عدد را پیش بینی کنیم در روش دوم مدل بازگشتی با تابع هزینه ctc باشد را ارائه کنیم

۱.۰ روش جداکردن کاراکتر ها

۱.۱.۰ مدل

برای روش اول ، مدل با دیتاست mnist آموزش داده شد تا فرایند پیش بینی با این مدل انجام شود چون که پیش پردازش برای جدا کردن کاراکتر ها با این همه سعی کامل انجام نمیشد و تازه تعداد دیتاست داده شده هم در برابر شصت هزار عکس دیتاست mnist عدد کمی هست پس برای بهتر بودن نتیجه از مدلی که با دیتاست mnist آموزش داده ایم استفاده میکنیم

ابتدا مدل خود را امتحان میکنیم که برای دیتاست mnist یک مدل ساده با عمق کم کافی هست و دقت خوبی به ما میدهد

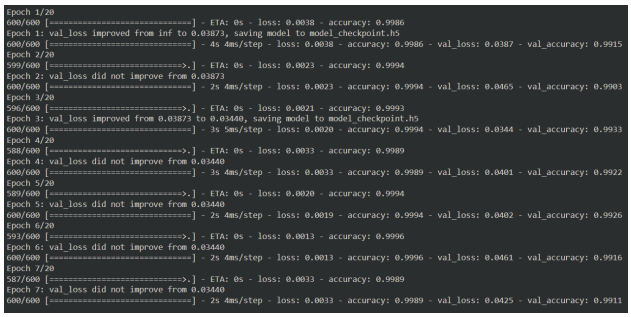
این مدل ساده هست ولی برای ورودی دادن به آن عکس باید بیست و هشت در بیست و هشت باشد

```python
model = keras.Sequential([
    keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),
    keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```

شکل ۱:

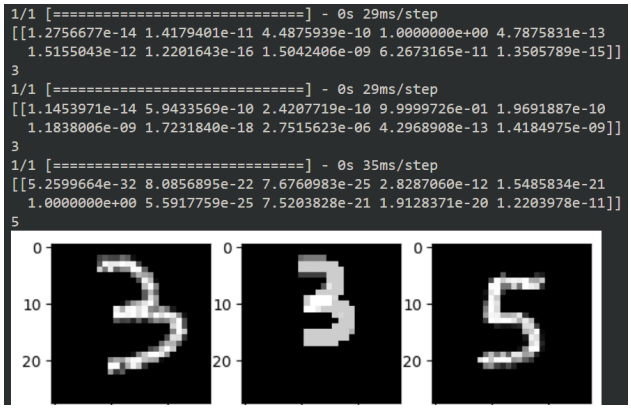
همانطور که در نتیجه آموزش میبینیم در داده تست ما به دقت نود و نه درصد رسیدیم

به دلیل به کار بردن callback وقتی میبیند پیشرفتی در لاس ایجاد نمیشود آموزش قطع میشود تا از overfit شدن جلوگیری کند



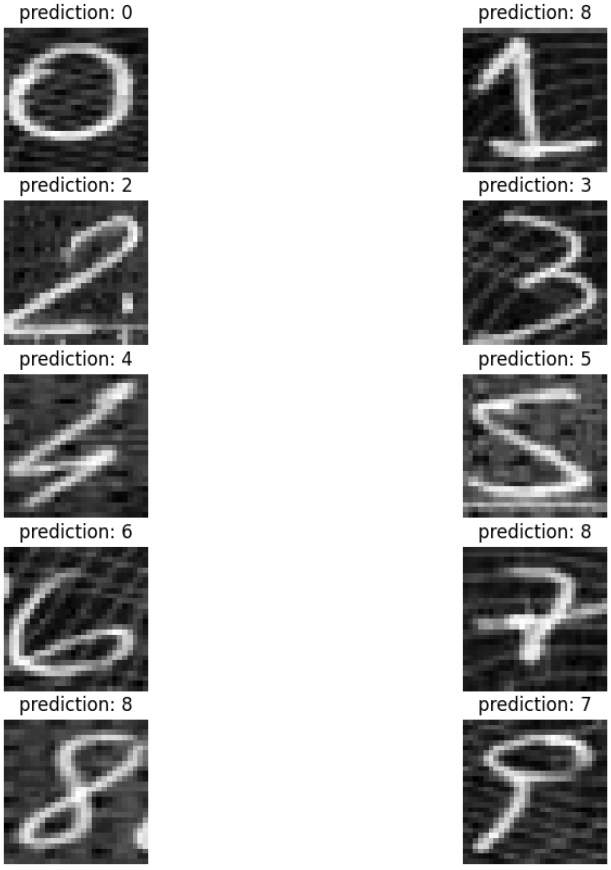
شکل ۲:

حال روی سه داده ای که خودم تعریف کردم مدل را تست میکنم که نتیجه خوبی میدهد



شکل ۳:

سپس روی دیتاستی که در پی دی اف بود اعداد را کراپ کردم تا ببینم بدون گذشتن از فیلتر چه دقتی دارد که بدک نیست



شکل ۴:

```
1/1 [==============================] - 0s 22ms/step
[[9.9699378e-01 2.1762775e-05 7.1253080e-04 5.9441564e-04 6.2477170e-06
  3.6169081e-06 7.3058018e-04 1.5810899e-06 3.4861060e-04 5.8684032e-04]]
0
1/1 [==============================] - 0s 22ms/step
[[1.4232136e-01 3.5081615e-03 2.8791828e-02 1.3837880e-04 7.9808684e-05
  2.0243637e-05 7.7548749e-03 1.3295372e-07 8.1726551e-01 1.1962401e-04]]
8
1/1 [==============================] - 0s 21ms/step
[[3.7234574e-06 5.9897768e-05 5.9422803e-01 2.0622085e-05 8.1482867e-04
  1.5843721e-06 4.1890717e-06 2.0841157e-06 4.0383616e-01 1.0288745e-03]]
2
1/1 [==============================] - 0s 21ms/step
[[5.7414804e-06 2.8661595e-04 8.1916889e-03 9.8484045e-01 4.7413858e-05
  9.6513621e-05 6.5117500e-05 7.4566866e-05 6.3103642e-03 8.1641148e-05]]
3
1/1 [==============================] - 0s 20ms/step
[[0.23213255 0.03366669 0.01186333 0.00950201 0.47224894 0.00976382
  0.1685397  0.02483322 0.01223461 0.02521515]]
4
1/1 [==============================] - 0s 20ms/step
[[1.7205646e-03 1.0413720e-02 1.8846196e-03 1.5823497e-01 2.3741004e-04
  5.7906556e-01 1.3926404e-02 7.7099615e-04 2.2781681e-01 5.9289602e-03]]
5
1/1 [==============================] - 0s 22ms/step
[[1.9748130e-01 2.1541331e-02 2.4204212e-01 3.0586952e-02 1.1411469e-02
  1.5819371e-03 4.0999198e-01 1.3348098e-04 7.8337051e-02 6.8923379e-03]]
6
1/1 [==============================] - 0s 21ms/step
[[1.15956445e-05 1.87244024e-02 1.24628767e-01 1.80659518e-02
  2.48958915e-02 1.42338424e-04 6.60499936e-05 5.93268275e-02
  6.21505678e-01 1.32632494e-01]]
8
1/1 [==============================] - 0s 21ms/step
[[5.6064573e-06 3.9374342e-04 8.8762090e-04 4.1205971e-05 2.4775864e-04
  2.1451928e-04 1.8105263e-04 9.1668990e-06 9.9798715e-01 3.2172171e-05]]
8
1/1 [==============================] - 0s 21ms/step
[[3.4316257e-05 4.1410205e-04 1.4336016e-04 2.5805591e-06 2.7883955e-04
  1.0088292e-03 3.5342655e-05 9.1647315e-01 8.0955833e-02 6.5368175e-04]]
7
```

شکل ۵:

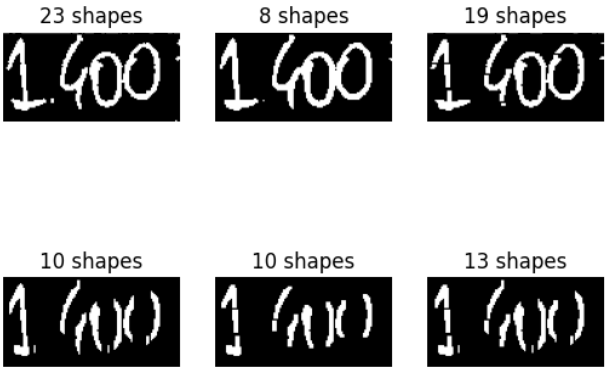
همانطور که میدانیم فرقی که بین دیتاست mnist و دیتاست داده شده اینست که پس زمینه mnist کاملا سیاه هست ولی پس زمینه دیتاست داده شده نویزی هست

filtering ۲.۱.۰

ایده اول برای جدا کردن کاراکتر ها که زدم این بود که از عملگر های باز و بسته مورفولوژی استفاده کردم تا نویز را کم کنم سپس با یک ترشولد عکس را باینری کردم و با تابع contour find ، تعداد شی های به هم پیوسته را پیدا کردم

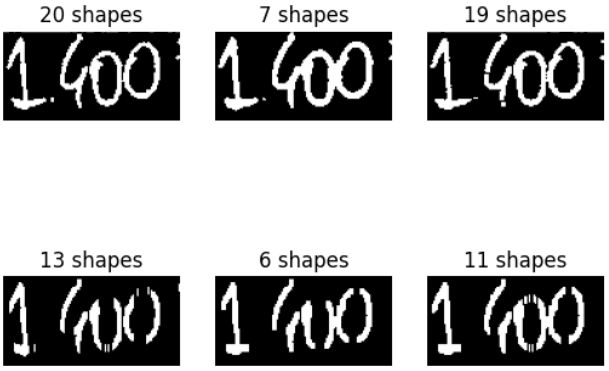
همانطور که میبینیم با همان عکس معمولی نتیجه بهتری در قسمت پیدا کردن کانتور شکل ۷ میدهد

شکل ۶:



شکل ۷:

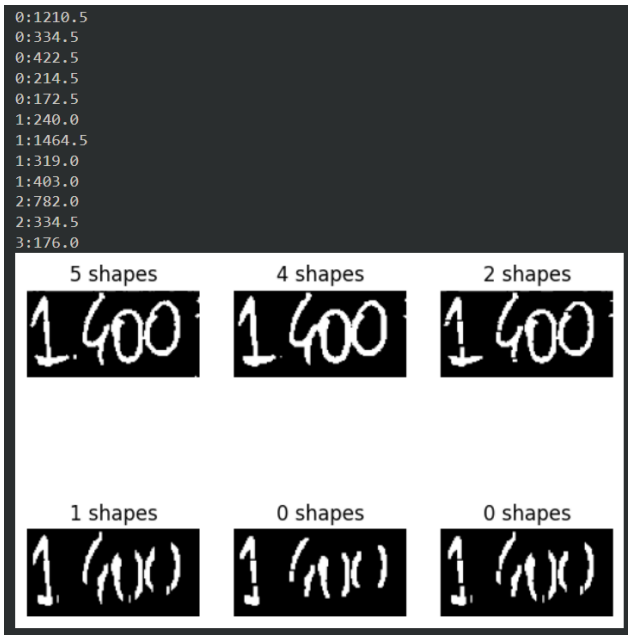
باینری کردن از طریق اوتسو هم امتحان کردم



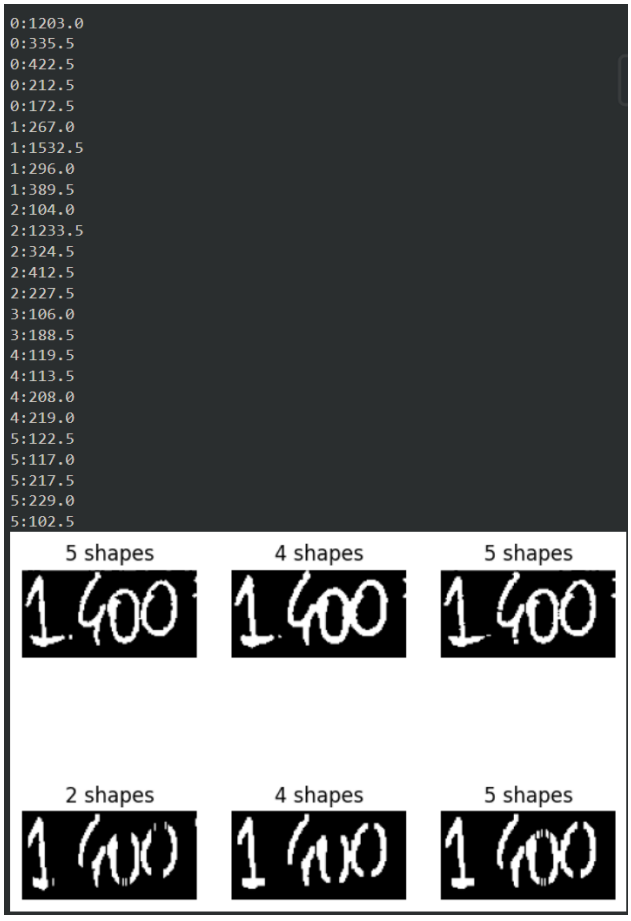
شکل ۸:

فرق خاصی نداشت

پس شی هایی که از مساحت خاصی به بالا هستند را استفاده میکنیم (تا از شی هایی که کوچک و مطمعنا عدد نیستند صرف نظر کنیم)



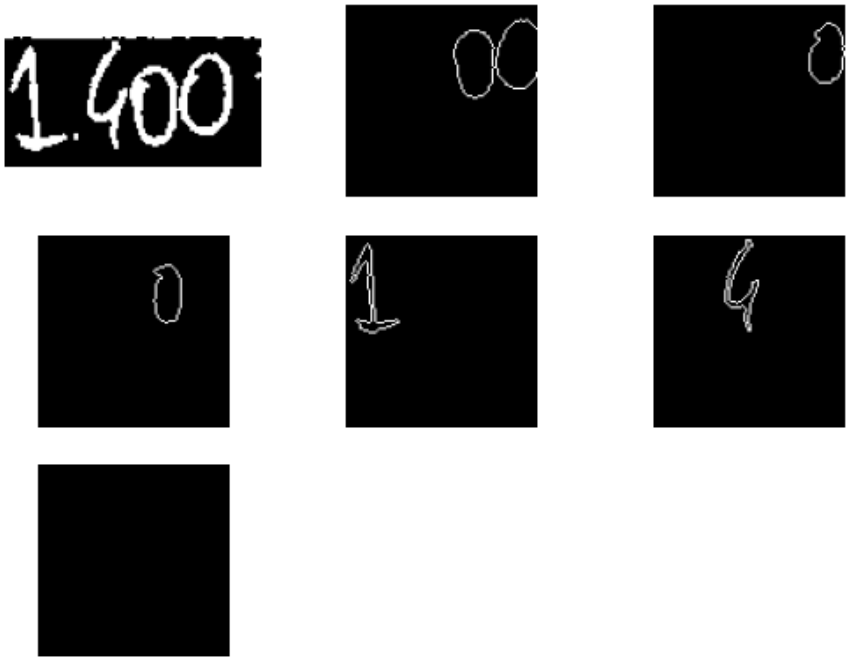
شکل ۹:
ترشولد ساده



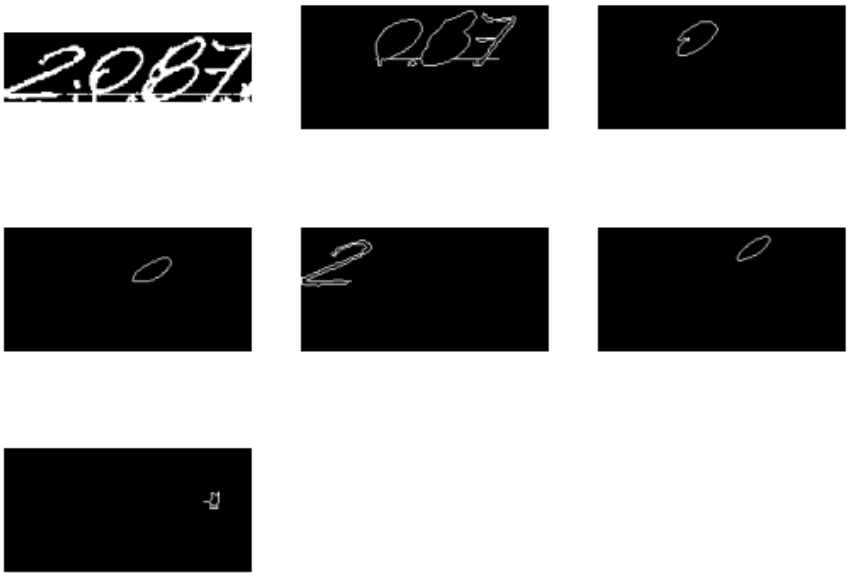
شکل ۱۰:
اوتسو

پس درنتیجه فقط از همان عکس های خودش استفاده میکنیم و قبل از ترشولد از مورفولوژی استفاده نمی‌کنیم

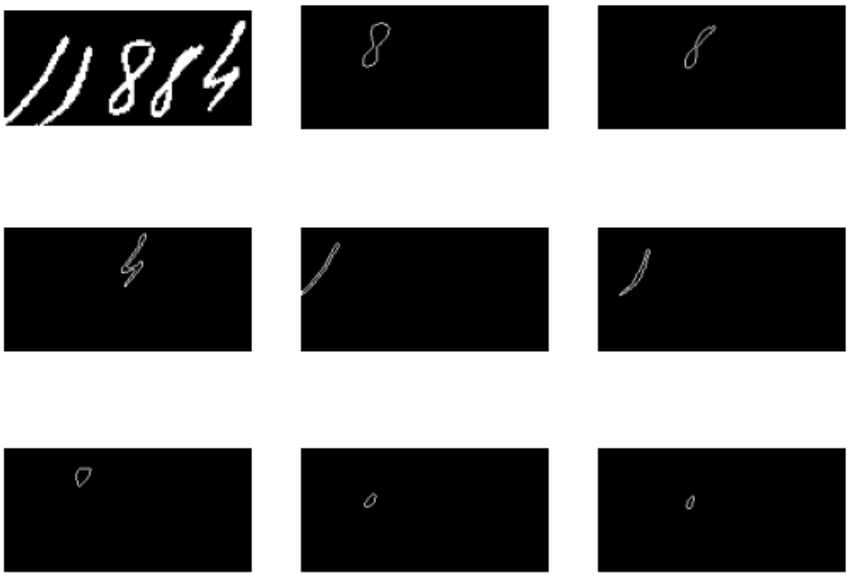
حال روی چند مثال که امتحان کردم متوجه شدم contour find لاوه بر پیکس های به هم چسبیده ، منحنی هایی هم که تشکیل یک منحنی بسته را میدهد هم پیدا میکند (طبق تعریفش در سایت opencv) پس این کاری اضافی هست و باعث مشکل در روند کار میشود چون ما عدد کامل را می‌خواهیم نه اینکه برای مثال هشت را به دو صفر تبدیل میکند و البته خود هشت را میابد ولی روند حذف اشکال اضافی سخت میشود پس از تابع دیگری استفاده میکنیم



شکل ۱۱:



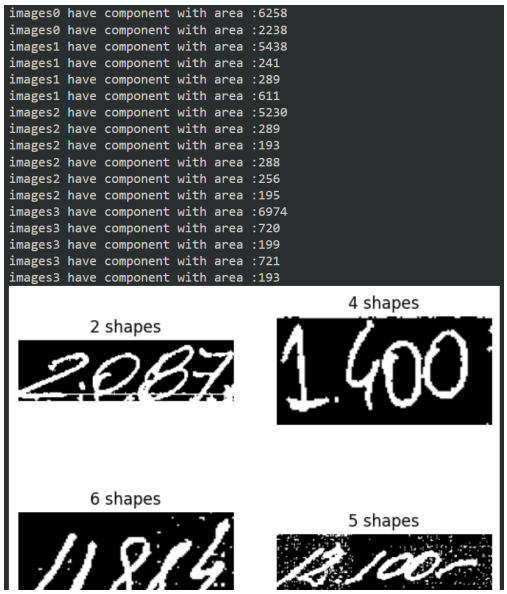
شکل ۱۲:



شکل ۱۳:

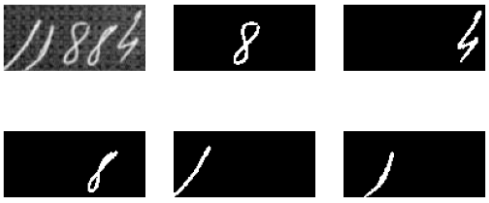
گذشته از این ، اگر خطی داشته باشیم(نویز) که از همه اعداد گذشته باشد، همه را یک کامپوننت تشکیل میدهد چون در واقع همه به همدیگر متصل هستند و این کار را مشکل میکند

این تابع connectedcomponent ود که نتیجه دلخواه ما را میدهد



شکل ۱۴:

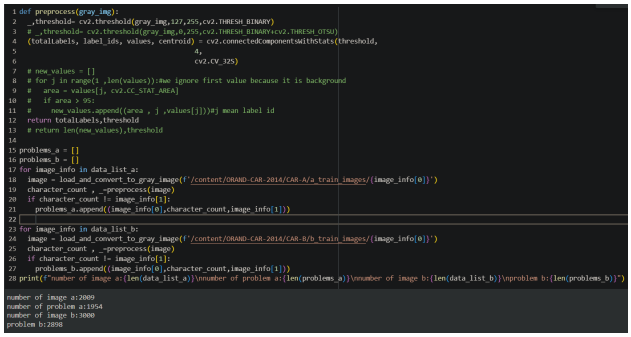
شکل ۱۵:



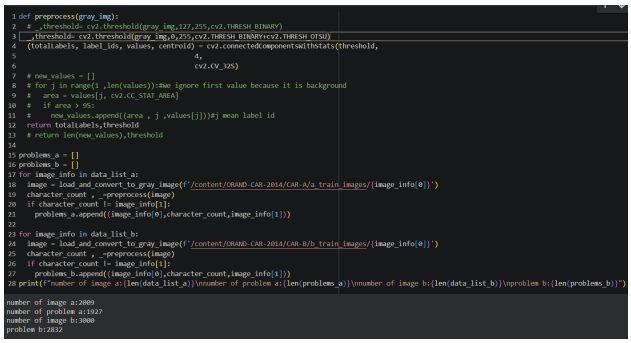
شکل ۱۶:

همانطور که می‌بینید جواب های نسبتا معقولانه تری در این روش گرفتیم

حال میخواهیم آزمایش کنیم که ببینیم این پیش پردازش ما چقدر خوب تعداد کاراکتر ها را حدس می‌زند و عکس هایی را که درست حدس نمی‌زند چه مشکلی دارد تا پیش پردازش را اصلاح کنیم



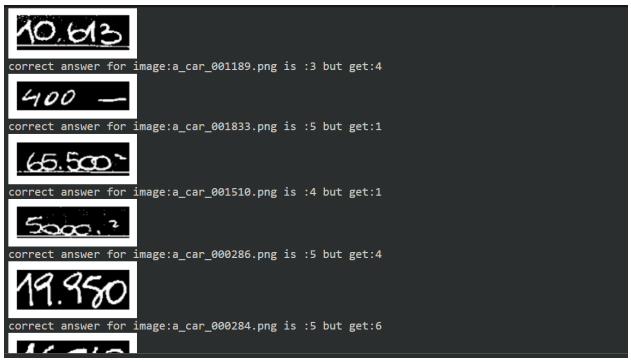
شکل ۱۷:
با ترشولد ساده



شکل ۱۸:
با اوتسو

شکل ۱۹:
فیلتر کردن مساحت شی ها

حال به صورت رندم چند عکس را که اشتباه حدس زده شده اند را میاوریم تا ایده بعدی برای بهبود را پیدا کنیم



شکل ۲۰:

خط افقی که پایین بعضی عکس ها هست اگر با عملگر باز توسط یک عنصر عمودی به ارتفاع ۳ حذف شوند،نتیجه ای بهتر حاصل میشود

ولی بازهم خط های افقی در بعضی عکس ها دیده میشود که اگر ارتفاع عنصر ساختاری در سایش را بیشتر کنیم با افزایش جواب های غلط روبرو میشویم و بدی این روش اینست که در بعضی عکس ها به دلیل عملگر های مورفولوژی یک عدد دارای گسستگی میشد و احتمالا در تشخیص مدل کار سخت تر میشد

```
number of image a:2009
number of problem a:1092
number of image b:3000
problem b:2390
```

شکل ۲۱:

سپس سعی کردم خط های صاف را تشخیص دهم که با تابع های مختلف این امکان پذیر نبود

```
area = values[j, cv2.CC_STAT_AREA]
x, y, w, h, area = values[j]
if area > 95:
  calculated_area = w * h
  if calculated_area >= ((1-area_threshold)*area) and calculated_area <= ((1+area_threshold)*area):
    new_values.append((area , j ,values[j]))#j mean label id
```

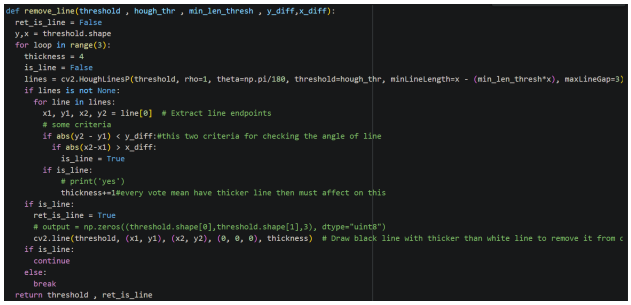
شکل ۲۲:

مشکل تابع findcontour و connectedcomponent این بود که اگر خط افقی بشود قبول نمی‌کنند و در کل ما احتیاجی به خط افقی نداریم ولی خب بعضی اوقات خط ها به اعداد میچسبند و در نتیجه خط به تنهایی قابل تشخیص نیست

پس باید از الگوریتم تشخیص خط استفاده کنیم تا فقط برای ما خط مشخصی را پیدا کند

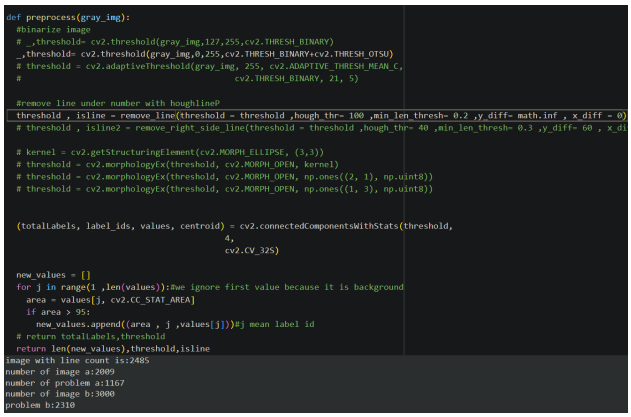
این ایده این بود که از طریق HoughlineP خط های صاف طولانی که در بعضی از تصاویر در زیر اعداد دیده میشد و مینیمم طول آن خط هم بر اساس عرض تصویر مشخص میشد (که اگر در تصویر عدد یک داشتیم ، یک حذف نشود) که عدد نیستند را شناسایی کنم و حذف کنم چون بقیه اعداد را به هم میچسبانند و باعث پردازش اشتباه میشود و باعث میشود که تعداد شی های ما را کمتر تشخیص دهد

در ضمن بدون عملگر مورفولوژی انجام شده است



```
def remove_line(threshold , hough_thr , min_len_thresh , y_diff,x_diff):
  ret_is_line = False
  y,x = threshold.shape
  for loop in range(3):
    thickness = 4
    is_line = False
    lines = cv2.HoughLinesP(threshold, rho=1, theta=np.pi/180, threshold=hough_thr, minLineLength=x - (min_len_thresh*x), maxLineGap=3)
    if lines is not None:
      for line in lines:
        x1, y1, x2, y2 = line[0]  # Extract line endpoints
        # some criteria
        if abs(y2 - y1) < y_diff:#this two criteria for checking the angle of line
          if abs(x2-x1) > x_diff:
            is_line = True
        if is_line:
            # print('yes')
            thickness+=1#every vote mean have thicker line then must affect on this
    if is_line:
      ret_is_line = True
      # output = np.zeros((threshold.shape[0],threshold.shape[1],3), dtype="uint8")
      cv2.line(threshold, (x1, y1), (x2, y2), (0, 0, 0), thickness)  # Draw black line with thicker than white line to remove it from c
    if is_line:
      continue
    else:
      break
  return threshold , ret_is_line
```

شکل ۲۳:



```
def preprocess(gray_img):
  #binarize image
  # _,threshold= cv2.threshold(gray_img,127,255,cv2.THRESH_BINARY)
  _,threshold= cv2.threshold(gray_img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
  # threshold = cv2.adaptiveThreshold(gray_img, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
  #                                   cv2.THRESH_BINARY, 21, 5)

  #remove line under number with houghlineP
  threshold , isline = remove_line(threshold = threshold ,hough_thr= 100 ,min_len_thresh= 0.2 ,y_diff= math.inf , x_diff = 0)
  # threshold , isline2 = remove_right_side_line(threshold = threshold ,hough_thr= 40 ,min_len_thresh= 0.3 ,y_diff= 60 , x_di

  # kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3,3))
  # threshold = cv2.morphologyEx(threshold, cv2.MORPH_OPEN, kernel)
  # threshold = cv2.morphologyEx(threshold, cv2.MORPH_OPEN, np.ones((2, 1), np.uint8))
  # threshold = cv2.morphologyEx(threshold, cv2.MORPH_OPEN, np.ones((1, 3), np.uint8))


  (totalLabels, label_ids, values, centroid) = cv2.connectedComponentsWithStats(threshold,
                                                4,
                                                cv2.CV_32S)

  new_values = []
  for j in range(1 ,len(values)):#we ignore first value because it is background
    area = values[j, cv2.CC_STAT_AREA]
    if area > 95:
      new_values.append((area , j ,values[j]))#j mean label id
  # return totallabels,threshold
  return len(new_values),threshold,isline
image with line count is:2485
number of image a:2009
number of problem a:1167
number of image b:3000
problem b:2310
```

شکل ۲۴:

خط هایی هم معمولا در سمت راست تصویر دیده میشد که آن ها هم باعث تشخیص اشتباه میشد

شکل ۲۵:

که کدی زده شد که فقط سمت راست تصاویر با تشخیص خط اگر خطی با طول کوتاه تری پیدا کرد ، حذف کند

```python
def remove_right_side_line(threshold ,hough_thr,min_len_thresh, y_diff,x_diff ):
  height, width = threshold.shape

  # Split the threshold into left and right halves
  left_side = threshold[:, :(55*width) //100 ]
  right_side = threshold[:, (55*width) // 100:]
  right_side , isline = remove_line(right_side , hough_thr , min_len_thresh , y_diff , x_diff)
  threshold = np.hstack((left_side, right_side))
  return threshold , isline
```

شکل ۲۶:

```python
def preprocess(gray_img):
  #binarize image
  # _,threshold= cv2.threshold(gray_img,127,255,cv2.THRESH_BINARY)
  _,threshold= cv2.threshold(gray_img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
  # threshold = cv2.adaptiveThreshold(gray_img, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
  #                                   cv2.THRESH_BINARY, 21, 5)

  #remove line under number with houghlineP
  threshold , isline = remove_line(threshold = threshold ,hough_thr= 100 ,min_len_thresh= 0.2 ,y_diff= math.inf , x_diff = 0)
  threshold , isline2 = remove_right_side_line(threshold = threshold ,hough_thr= 30 ,min_len_thresh= 0.2 ,y_diff= 60 , x_diff = 40)

  # kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3,3))
  # threshold = cv2.morphologyEx(threshold, cv2.MORPH_OPEN, kernel)
  # threshold = cv2.morphologyEx(threshold, cv2.MORPH_OPEN, np.ones((2, 1), np.uint8))
  # threshold = cv2.morphologyEx(threshold, cv2.MORPH_OPEN, np.ones((1, 3), np.uint8))


  (totalLabels, label_ids, values, centroid) = cv2.connectedComponentsWithStats(threshold,
                                          4,
                                          cv2.CV_32S)

  new_values = []
  for j in range(1 ,len(values)):#we ignore first value because it is background
    area = values[j, cv2.CC_STAT_AREA]
    if area > 95:
      new_values.append((area , j ,values[j]))#j mean label id
  # return totalLabels,threshold
  return len(new_values),threshold,isline2
```

```
image with right side line count is:689
number of image a:2089
number of problem a:1150
number of image b:3000
problem b:2300
```

شکل ۲۷:

دیدیم که فرق زیادی در ارور ها ایجاد نشد

در دیتاست تعدادی عکس پیدا کردم که دارای نویز های متناوب بود که خوب نویزهایش حذف نمیشد پس تصمیم گرفتم با استفاده از تبدیل فوریه ، نویز هارا حذف کنم

```python
def delete_noise_with_fourier(img):
  '''
  transform to frequency space
  shift for better visualization and noise deletion
  just keep center(circle with specific radius)
  reconstruc image with phase and magnitude
  '''
  # Convert image to float32 for better precision
  img_float32 = np.float32(img)

  # Compute 2D FFT
  dft = np.fft.fft2(img_float32)

  # Shift zero-frequency to center
  dft_shifted = np.fft.fftshift(dft)

  # Calculate magnitude (amplitude)
  old_magnitude_spectrum = np.abs(dft_shifted)

  new_magnitude_spectrum = fill_black_out_of_circle(old_magnitude_spectrum)


  # Calculate phase spectrum (arctan of real and imaginary components)
  phase_spectrum = np.angle(dft_shifted)

  # Reconstruct DFT from magnitude and phase (assuming real-valued image)
  reconstructed_dft = new_magnitude_spectrum * np.exp(1j * phase_spectrum)

  # Perform inverse FFT to reconstruct image
  dft_inverse = np.fft.ifft2(reconstructed_dft)

  # Shift zero-frequency back to top-left corner (optional)
```

شکل ۲۸:

```python
def preprocess(gray_img):
  #remove alternate noise with fourier transform
  # img = cv2.imread('saffrun.jpg',cv2.IMREAD_GRAYSCALE)
  denoise_img = delete_noise_with_fourier(gray_img)
  gray_img = denoise_img.astype(np.uint8)
  # cv2.imwrite('denoise_img.jpg',denoise_img)

  #binarize image
  # _,threshold= cv2.threshold(gray_img,127,255,cv2.THRESH_BINARY)
  _,threshold= cv2.threshold(gray_img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
  # threshold = cv2.adaptiveThreshold(gray_img, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
  #                                   cv2.THRESH_BINARY, 21, 5)

  #remove line under number with houghlineP
  threshold , isline = remove_line(threshold = threshold ,hough_thr= 100 ,min_len_thresh= 0.2 ,y_diff=20 , x_diff = 0)
  # threshold , isline2 = remove_right_side_line(threshold = threshold ,hough_thr= 30 ,min_len_thresh= 0.2 ,y_diff= 60

  # kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3,3))
  # threshold = cv2.morphologyEx(threshold, cv2.MORPH_OPEN, kernel)
  # threshold = cv2.morphologyEx(threshold, cv2.MORPH_OPEN, np.ones((2, 1), np.uint8))
  # threshold = cv2.morphologyEx(threshold, cv2.MORPH_OPEN, np.ones((1, 3), np.uint8))


  (totalLabels, label_ids, values, centroid) = cv2.connectedComponentsWithStats(threshold,
                                          4,
                                          cv2.CV_32S)

  new_values = []
  for j in range(1 ,len(values)):#we ignore first value because it is background
    area = values[j, cv2.CC_STAT_AREA]
    if area > 95:
      new_values.append((area , j ,values[j]))#j mean label id
```

```
image with right side line count is:2512
number of image a:2009
number of problem a:1194
number of image b:3000
problem b:2336
```

شکل ۲۹:

همانطور که میبینیم فرقی نکرد و با عوض کردن پارامتر های مربوط فقط بدتر میشود پس حدس من اشتباه بود

حال برای تست کردن تا این جای کار بر روی دیتایی که حداقل در پیش پردازش توانسته تعداد اعداد را به درستی پیش بینی کند آزمایش میکنیم

```python
# predict_my_input('/content/3_object.png')
# image_path_and_its_number_a
# image_path_and_its_number_b
correct_preprocess = []
for image_info in image_path_and_its_number_a:
  image = load_and_convert_to_gray_image(f'/content/ORAND-CAR-2014/CAR-A/a_train_images/{image_info[0]}')
  character_count , th ,isline=preprocess(image.copy())

  if character_count == len(image_info[1]):
    # correct_preprocess.append((image_info[0],character_count,image_info[1]))
    correct_preprocess.append((image_info[0],th,image_info[1]))


for image_info in image_path_and_its_number_b:
  image = load_and_convert_to_gray_image(f'/content/ORAND-CAR-2014/CAR-B/b_train_images/{image_info[0]}')
  character_count , th ,isline=preprocess(image)

  if character_count == len(image_info[1]):
    correct_preprocess.append((image_info[0],th,image_info[1]))

    # print(f"correct answer for image:{correct_process[0]} is :{correct_process[2]} but get:{result}")
# print(f"number of image a:{len(data_list_a)}\nnumber of problem a:{len(problems_a)}")
# print(f"number of image b:{len(data_list_b)}\nproblem b:{len(problems_b)}")
```

شکل ۳۰:

و این دیتاهای صحیح را یکی یکی میگیریم و کاراکتر ها را جدا میکنیم و هرکدام را در وسط صفحه قرار میدهیم چون وقتی داشتم نگاه میکردم ، اگر عدد در وسط صفحه باشد حدسش درست هست وگرنه غلط می‌شود سپس ریسایز به عرض و ارتفاع ۲۸ انجام میشود و سپس پیش بینی با مدل انجام میشود و کاراکتر ها را به هم متصل و نتیجه معلوم میشود

سپس با نتیجه خیلی بد مواجه شدم در صورتی که عکس ها خیلی تمیز و واضح بودند(تقریبا هیچکدام را درست پیش بینی نکرده بود)

شکل ۳۱:
نمونه ای از عکس تمیز

وقتی بیشتر دقت کردم ، متوجه شدم که ریسایز ، عکس ها را به شدت بد کیفیت میکند

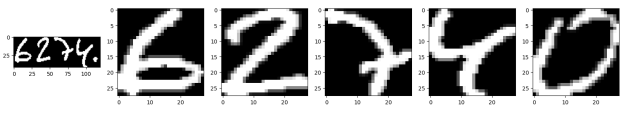
شکل ۳۲:

شکل ۳۳:

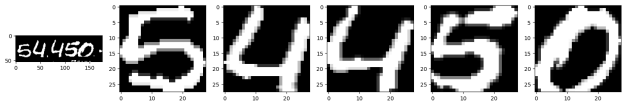
عکس ها به این دلیل ناواضح بودند که من هر عدد را روی عکسی با عرض و طول بزرگ می گذاشتم و در هنگام ریسایز به عرض و طول ۲۸ ، خب داده ای زیاد از دست می‌رفت

اول برای واضح شدن از ریسایز بایکیوبیک استفاده کردم که فرقی نداشت

سپس سعی کردم آن را در قالبی بزرگ نزارم و همان عدد را با هر سایزی که تشخیص داده بود ، به عرض و طول ۲۸ ریسایز میکردم ، در این صورت کیفیت خیلی بهتر شد



شکل ۳۴:



شکل ۳۵:

```python
list_of_errors = []
errors = 0
print(len(correct_preprocess))
# x = 10
for i in correct_preprocess:
  # if x < 0:
  #   break
  # x -= 1
# i = correct_preprocess[0]
  (totalLabels, label_ids, values, centroid) = cv2.connectedComponentsWithStats(i[1],
                                                4,
                                                cv2.CV_32S)
  # plt.imshow(i[1], cmap='gray')
  # plt.show()
  result,is_error = extract_each_character(i[1])
  if is_error:
    print(f"image from {i[0]} which must be {i[2]} and predict {result} get error")
  # print(f"result:{result}\nmust be:{i[2]}")
  if result != i[2]:
    list_of_errors.append((i[1] ,i[0],result,i[2]))
    errors += 1

print(f"total image : {len(correct_preprocess)}\nnumber of errors:{errors}")

```

```
1532
total image : 1532
number of errors:1407
```

شکل ۳۶:

تقریبا صد عدد را تشخیص داد ولی باز هم خوب نیست

سپس سعی کردم اعداد را طوری در عکس بگذارم از اطراف هم کمی ناحیه سیاه باشد چون مدل اینگونه آموزش دیده

شکل ۳۷:

شکل ۳۸:

```
list_of_errors = []
errors = 0
print(len(correct_preprocess))
# x = 10
for i in correct_preprocess:
  # if x < 0:
  #   break
  # x -= 1
# i = correct_preprocess[0]
  (totalLabels, label_ids, values, centroid) = cv2.connectedComponentsWithStats(i[1],
                                        4,
                                        cv2.CV_32S)
  # plt.imshow(i[1], cmap='gray')
  # plt.show()
  result,is_error = extract_each_character(i[1])
  if is_error:
    print(f"image from {i[0]} which must be {i[2]} and predict {result} get error")
  # print(f"result:{result}\nmust be:{i[2]}")
  if result != i[2]:
    list_of_errors.append((i[1] ,i[0],result,i[2]))
    errors += 1

print(f"total image : {len(correct_preprocess)}\nnumber of errors:{errors}")

```

```
1532
total image : 1532
number of errors:1023
```

شکل ۳۹:

تا به حال از ۵۰۰۹ عکس توانستیم ۵۰۷ تای آن را به درستی تشخیص دهیم یعنی دقت ده درصد

حال عملگر باز مورفولوژی هم اضافه میکنیم و دوباره اجرا میگیریم

```
list_of_errors = []
errors = 0
print(len(correct_preprocess))
# x = 10
for i in correct_preprocess:
  # if x < 0:
  #   break
  # x -= 1
# i = correct_preprocess[0]
  (totalLabels, label_ids, values, centroid) = cv2.connectedComponentsWithStats(i[1],
                                        4,
                                        cv2.CV_32S)
  # plt.imshow(i[1], cmap='gray')
  # plt.show()
  result,is_error = extract_each_character(i[1])
  if is_error:
    print(f"image from {i[0]} which must be {i[2]} and predict {result} get error")
  # print(f"result:{result}\nmust be:{i[2]}")
  if result != i[2]:
    list_of_errors.append((i[1] ,i[0],result,i[2]))
    errors += 1

print(f"total image : {len(correct_preprocess)}\nnumber of errors:{errors}")
print(f"accuracy:{len(correct_preprocess) - errors}/{5009} = {(len(correct_preprocess) - errors)/5009}")

```

```
1687
total image : 1687
number of errors:1144
accuracy:543/5009 = 0.10840487123178279
```

شکل ۴۰:

میبینیم که فرق خاصی در دقت انجام نشد

حال بدون مورفولوژی و حذف کردن خط های زیر اعداد تست میکنیم

شکل ۴۱:

در آخر هم از فاصله لونشتین هم استفاده کردم و کل دیتاست را حدس زدم تا اطلاعات بهتری از ارزیابی مدل داشته باشم

۲.۰ روش مدل بازگشتی

من برای حل این قسمت از کدی که در سایت kaggle بود استفاده کردم و البته آن را تغییر دادم چون مثل مسئله ما نبود کدی که در سایت بود از این قرار بود که برای کل حروف چه عدد چه حرف الفبا طراحی شده بود ولی ما فقط برای اعداد می‌خواستیم

تغییرات از این قرار بود که باید ماکسیمم طول خروجی را پیدا میکردم و همینطور حروف الفبا را حذف میکردم و فقط اعداد را باقی میزاشتم و ...

کد در این آدرس هست

https://www.kaggle.com/code/samfc۱۰/handwriting-recognition-using-crnn-in-keras

Preparing the labels for CTC Loss ۱.۲.۰

در این قسمت ،اول alphabet را که شامل حروف هست را تغییر میدهیم و فقط اعداد را می‌گذاریم

سپس max_str_len را به هشت تغییر میدهیم چون ماکسیمم طول خروجی هشت هست

(البته مدل های بازگشتی این خاصیت را دارند که خودشان متوجه شوند خروجی چه طولی دارد ولی من در این نسخه این ویژگی را اضافه نکردم)

سپس در ادامه ماجرا اطلاعات مورد نیاز برای محاسبه تابع کاهشی ctc جمع آوری میشود

Building our model ۲.۲.۰

```
input_data = Input(shape=(width_reshape, height_reshape, 1), name='input')

inner = Conv2D(32, (3, 3), padding='same', name='conv1', kernel_initializer='he_normal')(input_data)
inner = BatchNormalization()(inner)
inner = Activation('relu')(inner)
inner = MaxPooling2D(pool_size=(2, 2), name='max1')(inner)

inner = Conv2D(64, (3, 3), padding='same', name='conv2', kernel_initializer='he_normal')(inner)
inner = BatchNormalization()(inner)
inner = Activation('relu')(inner)
inner = MaxPooling2D(pool_size=(2, 2), name='max2')(inner)
inner = Dropout(0.3)(inner)

inner = Conv2D(128, (3, 3), padding='same', name='conv3', kernel_initializer='he_normal')(inner)
inner = BatchNormalization()(inner)
inner = Activation('relu')(inner)
inner = MaxPooling2D(pool_size=(1, 2), name='max3')(inner)
inner = Dropout(0.3)(inner)

# CNN to RNN
inner = Reshape(target_shape=((64, 1024)), name='reshape')(inner)#*****************
inner = Dense(64, activation='relu', kernel_initializer='he_normal', name='dense1')(inner)

## RNN
inner = Bidirectional(LSTM(256, return_sequences=True), name = 'lstm1')(inner)
inner = Bidirectional(LSTM(256, return_sequences=True), name = 'lstm2')(inner)

## OUTPUT
inner = Dense(num_of_characters, kernel_initializer='he_normal',name='dense2')(inner)
y_pred = Activation('softmax', name='softmax')(inner)

model = Model(inputs=input_data, outputs=y_pred)
```

شکل ۴۲:

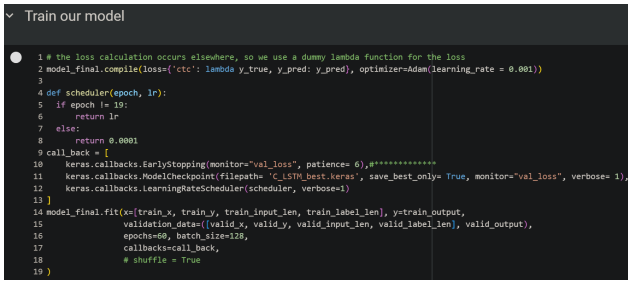
این مدل تلفیقی از شبکه کانولوشنی و بازگشتی هست

به طور خلاصه به این صورت که شبکه کانولوشنی اطلاعاتی را استخراج و مدل بازگشتی از آن استفاده میکند

که توضیحات دقیق تر در این سایت هست

https://theailearner.com/۲۰۲۱/۰۳/۱۰/ctc-problem-statement/

Train our model ۳.۲.۰



Train our model

```
# the loss calculation occurs elsewhere, so we use a dummy lambda function for the loss
model_final.compile(loss={'ctc': lambda y_true, y_pred: y_pred}, optimizer=Adam(learning_rate = 0.001))

def scheduler(epoch, lr):
  if epoch != 19:
      return lr
  else:
      return 0.0001
call_back = [
    keras.callbacks.EarlyStopping(monitor="val_loss", patience= 6),#***********
    keras.callbacks.ModelCheckpoint(filepath= 'C_LSTM_best.keras', save_best_only= True, monitor="val_loss", verbose= 1),
    keras.callbacks.LearningRateScheduler(scheduler, verbose=1)
]
model_final.fit(x=[train_x, train_y, train_input_len, train_label_len], y=train_output,
                validation_data=([valid_x, valid_y, valid_input_len, valid_label_len], valid_output),
                epochs=60, batch_size=128,
                callbacks=call_back,
                # shuffle = True
)
```

شکل ۴۳:

همانطور که می‌بینید از چندین کال بک استفاده شده که اولی برای این هست که اگر شش آپیاک متوالی در لاس دیتای

تست تغییری ایجاد نشد آموزش را متوقف کند که این به معنی اورفیت شدن هست

بعدی برای ذخیره مدل بر اساس بهترین لاس هست

بعدی برای این هست که بعد از آپیاک نوزدهم نرخ یادگیری را کمتر کند

Check model performance on validation set ۴.۲.۰

در اینجا دیتای را روی داده تست می‌سنجیم تا بر اساس معیار مختلف ارزیابی کنیم و عکس هایی که اشتباه حدس زده شده را نگاه می‌کنیم تا ایده بگیریم چه پردازشی لازم دارد تا نتیجه بهتری بدهد

ارزیابی ها از این قرار بود که در کل چند صفر داریم که به درستی صفر حدس زده شده اند یا به غلط و به غلط چه عددی حدس زده شده اند

در کل تعداد صفر بیشتر بود و همینطور بیشترین مشکل با صفر بود ولی به نتیجه ای نرسید چون تفاوت زیادی با بقیه اشتباهات بقیه اعداد نداشت و ارزش نداشت که مثلاً تعداد بقیه حروف را زیاد کنیم تا متعادل بشود یا کار هایی از این قبیل

ولی با مشاهده اشتباهاتی که مدل پیش بینی کرده بود به این نتیجه رسیدم که عکس ها نویز های متناوب دارند که ایده خوبی است و در کل هم که دیتاست را میتوان بیشتر کرد و هموار کردن هم میتواند کمک کند(filter smooth) و عملگر های مورفولوژی با داده ساختار کوچک و ساده کردن مدل به دلیل دیتاست کوچک

result of improvement ۵.۲.۰

هر کدام از ایده های بالا را همانطور که در کد دیده میشود اضافه کردم و در موارد زیر نتایج بهتر از بقیه ایده ها بود

حذف نویز متناوب با تبدیل فوریه

حذف نویز با هموار کردن

حذف نویز با عملگر مورفولوژی

در نتیجه بهترین عملکرد این شد

```
Average Levenshtein distance: 0.0482081911262798 66
Correct characters predicted : 98.67%
Correct words predicted      : 96.20%
false prediction:89
```

شکل ۴۴:

و با پیش پردازشی در آن ۸۹ داده هم توانستن حدود ۲۰ تای دیگر هم حدس بزنم(نتیجه پایین فقط بر روی ۸۹ داده اشتباه پیش بینی شده است)

```
print(f"Average Levenshtein distance: {total_distance / comparisons}")
# print(f"Average Levenshtein distance: {total_distance / total_char}") # this approach better than last line
print(f'Correct characters predicted : {corrected_char_percent}% = {corrected_char} from total chars:{total_char}')
print(f'Correct numbers predicted      : {corrected_numbers_percent}% = {corrected_numbers} from total numbers:{len(y
```

Python

```
total words:89
Average Levenshtein distance: 1.0337078651685394
Correct characters predicted : 73.65% = 341 from total chars:463
Correct numbers predicted      : 22.47% = 20 from total numbers:89
```

شکل ۴۵: