Validation results (made by contours masks)

Notes:

The glare (or blur) effect creates global noise, equalizing intensities and leveling the boundaries between the background and objects, therefore the task of restoration is *to correct the damaged relationships (gradients) between the background and objects, and not the distorted objects themselves*.

The surface defects creates local noise, distorting both the background and objects, while darkening the background and lightening the symbols, and, like the glare effect, enhancing the boundaries between the background and objects, therefore the task of restoration is the same as in the first case.

With this approach, 1) those local defects of objects that were present on them initially, i.e. potential artist mistakes, are naturally (and not artificially) preserved; 2) in the case of complex (fusion between glare and surface defects) noise pollution, the task does not change.

Images:	**24643 (good), 29508 (noised by surface defects), 29634 (noised by glare)**
Data on plots title:	**B - Background statistics [mean, var, std, range], S - Symbols statistics [mean, var, std, range]**

Image statistics before and after applying the gradient boosting algorithm.

agray	– original image
rs	– recovered image

24643 (good):

```
#-------------------------------------------------------------[mean, var, std, range] ********* [ mean, var,     std,  range]
stats for agray:            24643 [0.68, 0.13] *** B: [253  19   4 138] ,*** S: [  35 1555   39  255] 253-35=218
stats for rs:               24643 [0.68, 0.13] *** B: [254  24   4 106] ,*** S: [  26 1382   37  254] 254-26=228
```

29508 (noised by surface defects)*:*

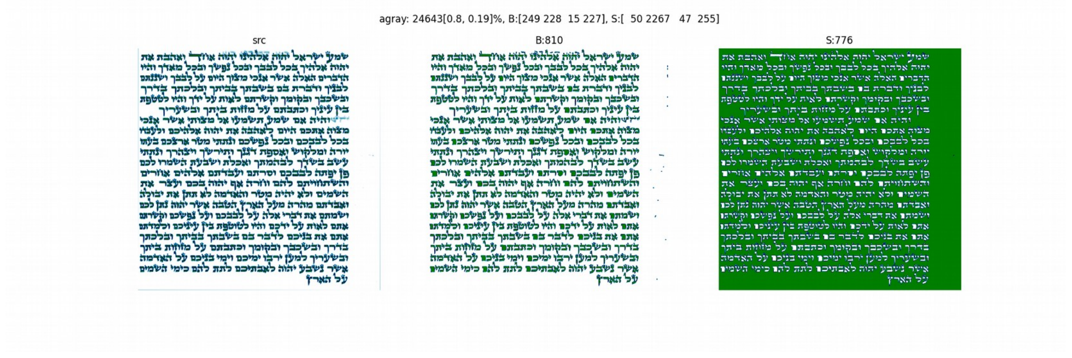
```
#-------------------------------------------------------------[mean, var, std, range] ********** [ mean, var,     std,  range]
stats for agray:            29508 [0.61, 0.14] *** B: [233 282  16 255] ,*** S: [  20 1577   39  255] 233-20=213
stats for rs:               29508 [0.61, 0.14] *** B: [252 102  10 202] ,*** S: [  18 1394   37  255] 252-18=234
```

29634 (noised by glare)*:*

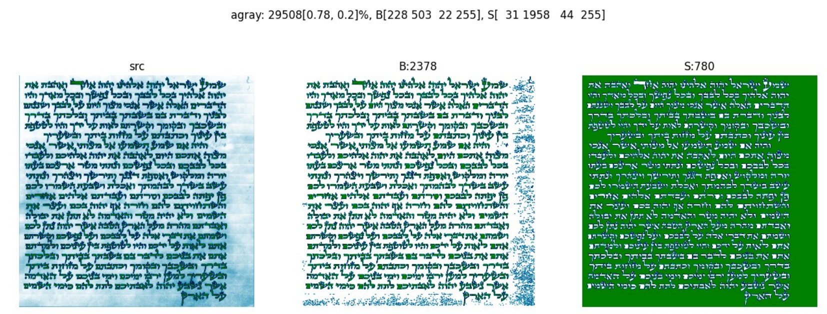
```
#-------------------------------------------------------------[mean, var, std, range] ********* [ mean, var,     std,  range]
stats for agray:            29634 [0.68, 0.13] *** B: [218  65   8 227] ,*** S: [  52  996    31  208]  218-52=166
stats for rs:               29634 [0.68, 0.13] *** B: [253  42   6 246] ,*** S: [  35  1567   39  244] 253-35=218

#----------------------------------------------------------------------
```

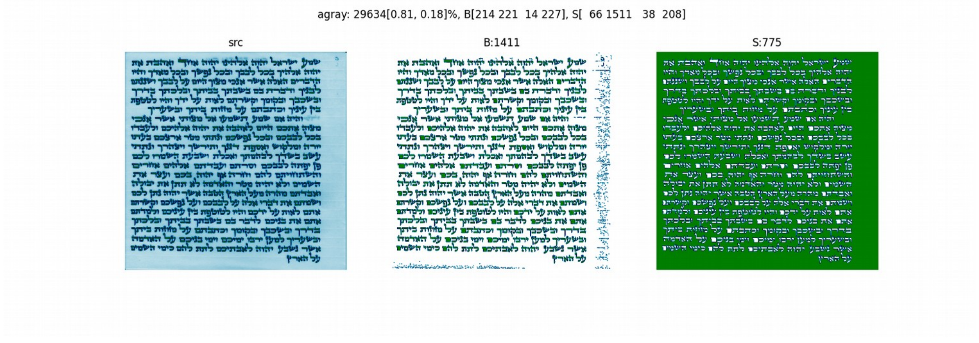
1. statistics BEFORE processing (original images without any transformations)


agray: 24643[0.8, 0.19]%, B:[249 228 15 227], S:[50 2267   47 255]
src
B:810
S:776

24643: Delta=Back – Symbols (intensity) = 249 – 50 = 199 (**normal**)
range (for symbols) = 255 (**good**)
var (for back) = 228 (**normal**)

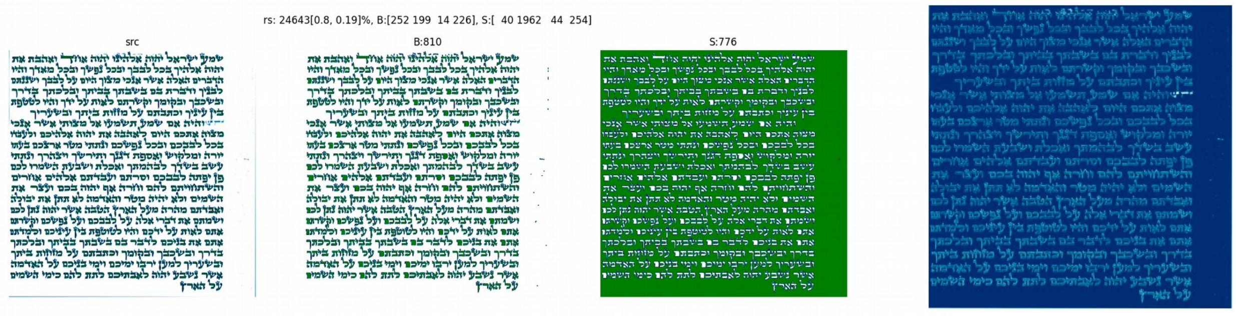

agray: 29508[0.78, 0.2]%, B[228 503 22 255], S[31 1958   44 255]
src
B:2378
S:780

29508: Delta=Back – Symbols (intensity) = 228 – 31 = 197 (**normal**)
range (for symbols) = 255 (**good**)
var (for back) = 503 (**too bad**) **!!! more surface defects**

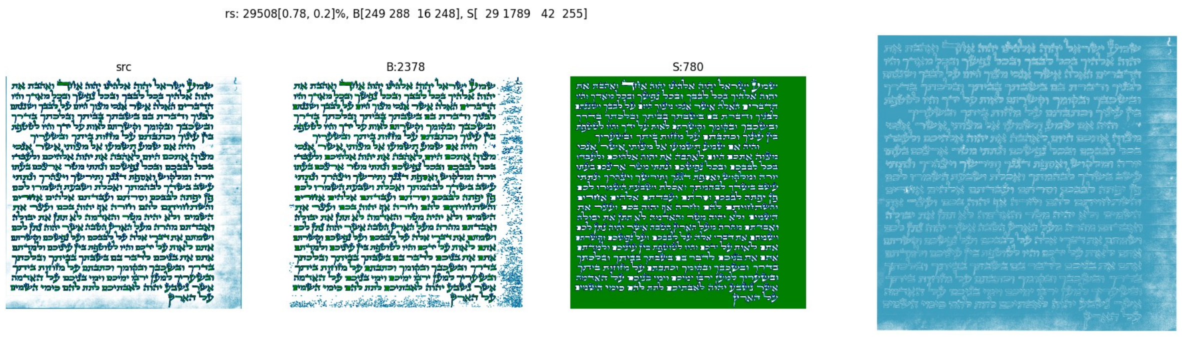

agray: 29634[0.81, 0.18]%, B[214 221 14 227], S[66 1511   38 208]
src
B:1411
S:775

29634: Delta=Back – Symbols (intensity) = 214 – 66 = 148 **!!! more glare defects**
range (for symbols) = 208 (**bad**) **!!! more glare defects**
var (for back) = 221 (**normal**)

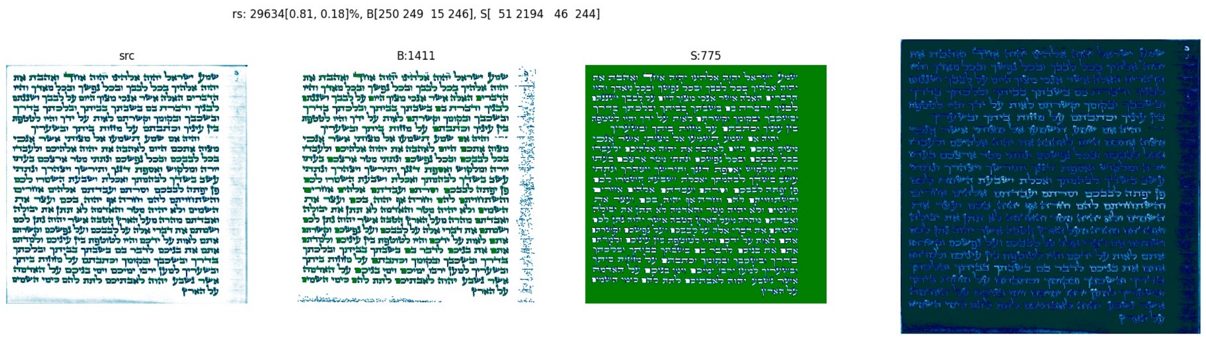
2. statistics AFTER processing (images after complex recovering)
*** the fourth image is the result of differences src and recovering images (results of processing)**


rs: 24643[0.8, 0.19]%, B:[252 199 14 226], S:[40 1962 44 254]
src
B:810
S:776

24643: Delta=Back – Symbols (intensity) = 252– 40 = 212 (**good**)
range (for symbols) = 254 (**good**)
var (for back) = 199 (**good**)


rs: 29508[0.78, 0.2]%, B[249 288 16 248], S[29 1789 42 255]
src
B:2378
S:780

29508: Delta=Back – Symbols (intensity) = 249 – 29 = 220 (**good**)
range (for symbols) = 255 (**good**)
var (for back) = 288 (**normal**)


rs: 29634[0.81, 0.18]%, B[250 249 15 246], S[51 2194 46 244]
src
B:1411
S:775

29634: Delta=Back – Symbols (intensity) = 250 – 51 = 199 (**good**)
range (for symbols) = 244 (**good**)
var (for back) = 249 (**normal**)

Conclusions:

1) the formulation of the problem "to remove the effects of glare (blur) or surface defects" while preserving real defects (artists' errors) is not only erroneous, but also harmful, since in this case the effects of the impact (changed pixels) will be affected incorrectly, which will distort the real mistakes;

2) in the case of using algorithms aimed at enhancing gradients, after the above-mentioned noise effects, under the given boundary conditions (almost a binary input image), the solution to the problem will be optimal and maximally effective both in terms of the result and in terms of processing speed;

3) additional algorithm aimed at improving the result of the gradient algorithm are standard statistical preprocessing of the original image, aimed at suppressing statistical noise (pixel flickering);

4) if the digitization technology is changed from single scanning to obtaining an array of images (or video with an array of frames) taken with a stationary camera for one document (the average number of shots is 7 images), then the pre-processing noted in point 3 can be reduced to an elementary sliding averaging of this array;

5) the code implementing both tasks: statistical pre-processing and the gradient sharpening algorithm itself, can be of varying degrees of complexity and adaptability, which in the most elementary and, as tests have shown, effective implementation in Python looks like this:

```
#----------------------------------------------------------------------------------------------- importing free libraries

import numpy as np                                  # numpy library
import cv2                                          # opencv standart library
import matplotlib.pyplot as plt                     # plot library

#----------------------------------------------------------------------------------------------- functions

def implotn(images,fig=7055, title=None, titles=None, color=None,tcolor='black', cmap='gray',label=None):
            fig = plt.figure( fig ); plt.clf(); plt.axis('off');
            if not color is None: fig.patch.set_facecolor(color);
            if not title is None: plt.title(title,color=tcolor)
            n = len(images)
            grid = plt.GridSpec(1,n)
            axs = []
            for i in range(n):
                        ax = fig.add_subplot(grid[0,i])
                        ax.axis('off')
                        ax.imshow(images[i],cmap= cmap[i] if (isinstance(cmap,(list, tuple, np.ndarray))) else cmap )
                        if not titles is None:
                                    ax.set_title(titles[i]);
                        axs.append( ax )
            if not label is None:
                        plt.legend(prop={'size': 10},loc='upper left',bbox_to_anchor=(1,1),handles=label)
            plt.show()
            return axs

def shine(img,sigmaX=99,sigmaY=99,K=5,vi=None):
            # statnoises removing
            b = cv2.GaussianBlur(img, (0,0), sigmaX=sigmaX, sigmaY=sigmaY)
            d = cv2.divide(img,b,scale=255)
            # normalization to float
            an = cv2.normalize(np.float32(d),None,0.,1.,cv2.NORM_MINMAX,dtype=cv2.CV_32F)
            # intensity spreading by koefficient
            s = 1 - cv2.pow(np.float64(1-an),K/10)
            # normalization to uint8
            r = cv2.normalize(s,None,0,255,cv2.NORM_MINMAX,dtype=cv2.CV_8U)
            if vi:
                        implotn((img,d,r),titles=('src','d','r'),cmap=None,fig=vi)
            return d,r

#----------------------------------------------------------------------------------------------- main begin

image = cv2.imread(path,0)                          # read image – the original grayscale image (unsigned 8 bit) with characters is darker than the background

d,r = shine(image,thr=(61,11),k=5)                  # recovered images: statnoises removed and fullstack recovering

# plot results
_= implotn((image,d,r), titles=('src','d','r'), title='Grayscale image recovering',fig='R')

#----------------------------------------------------------------------------------------------- end
```

6) It should also be noted that the efficiency of the algorithms is definitely better tested on the final result - after the formation of boxes or, even better, after the OCR, i.e. for this it is necessary to have the correct and effective code for the following post-processing procedures:

a) smoothing of harmful (inverted) pixel fluctuations (dots and small groups of inverted dots on the background and on the symbols);
b) binarization;
c) building an array of boxes and/or using OCR-system (which solving problem of building boxes automatically as the result of recognition)

7) in my own implementation of the post-processing procedures (which were not included in the problem statement), the final results for all 3 tested images give the following results:

The entire process of checking each image consists of the sequential operation of the following algorithms:

1) removal of statistical noise;
2) gradients increasing;

3) binarization;

4) construction of box contours on a binary image;

5) elementary filtering of boxes (without cutting and/or merging) exclusively by frequency and size;

6) detection and construction of reference lines (horizontal - lines and vertical - limiters on the right and left);

7) calculation of the binding of boxes to the line;

8) detection of subscript characters and their bindings to guide lines;

9) Detection of bad boxes;

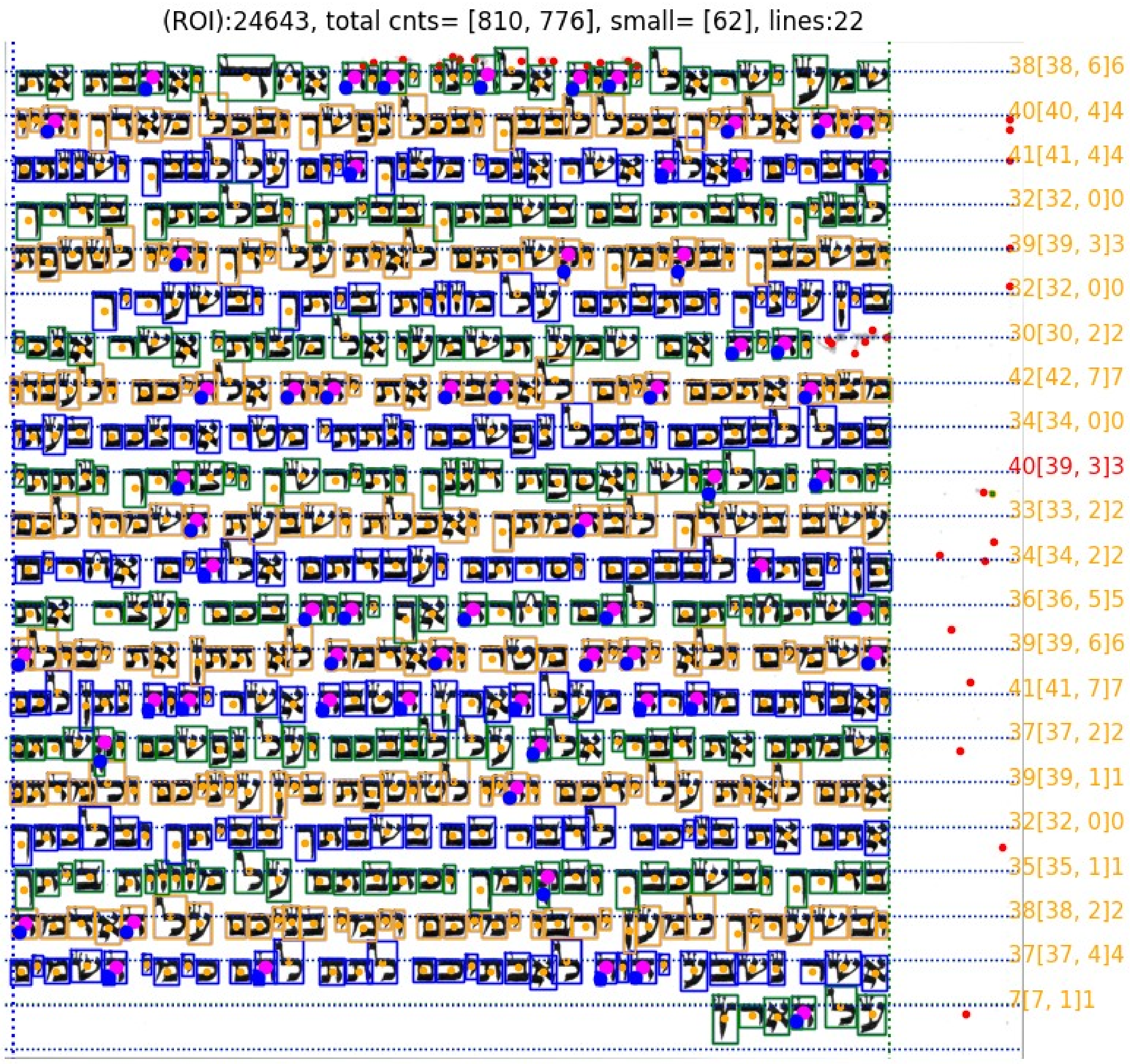
10) Visualization of results:

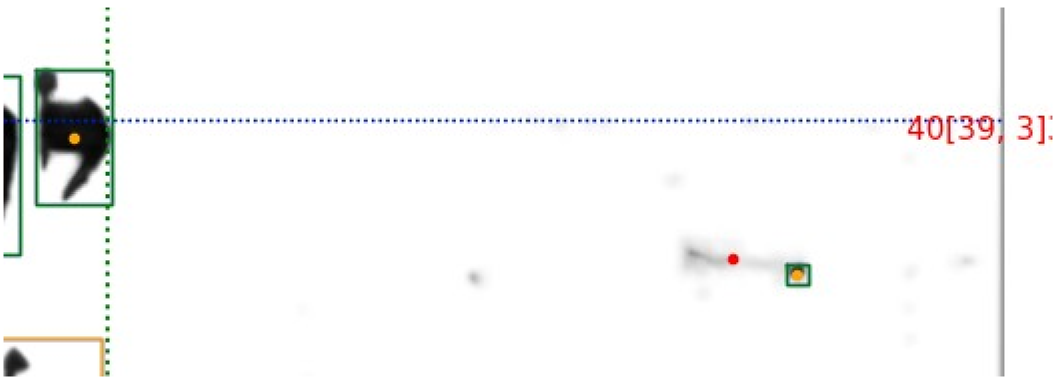
the column on the right of the plot shows the calculated and default values for each row in the following format:

Found number of large characters [default number of large characters, default number of lowercase characters] Found number of subscript characters

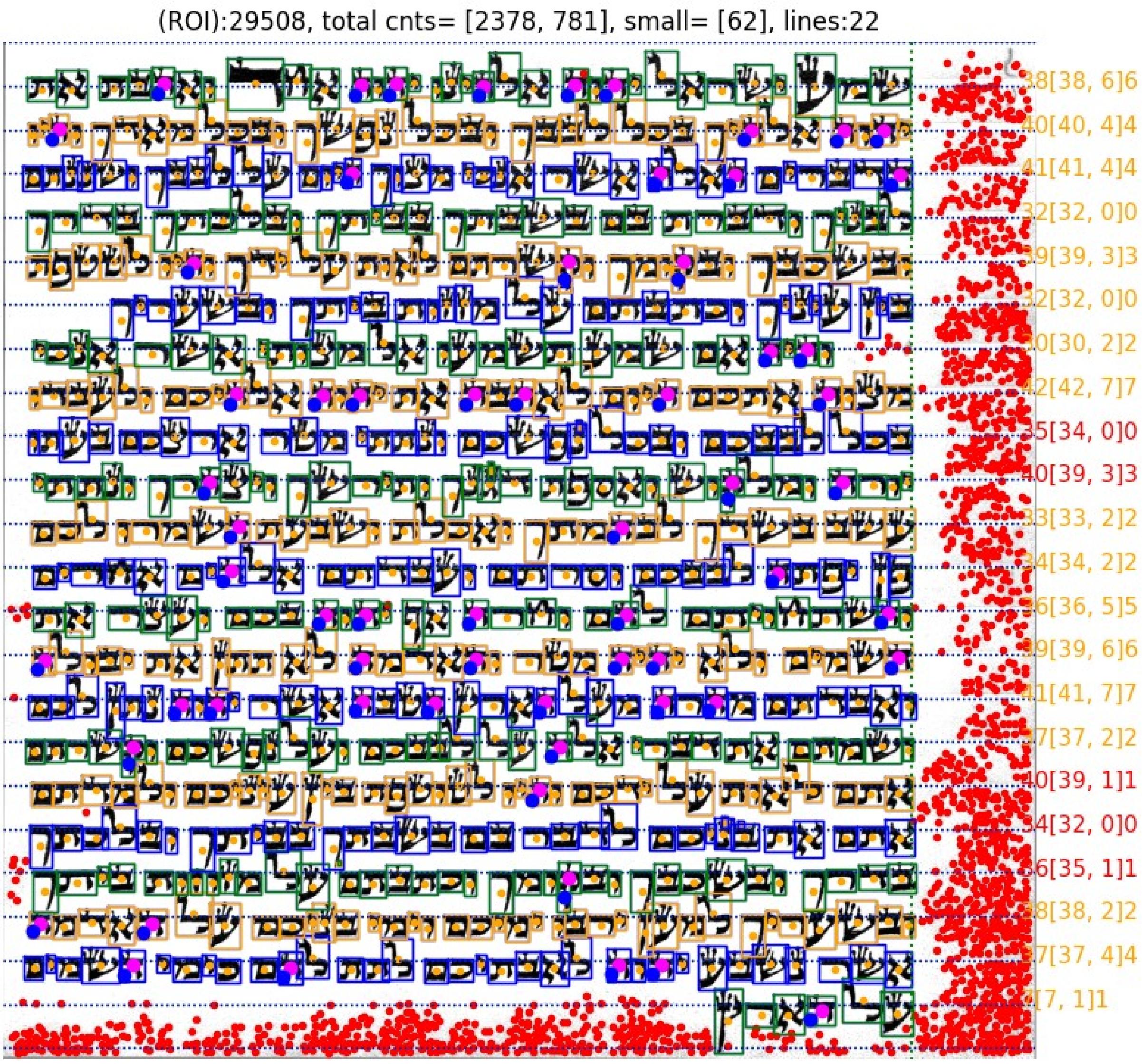
in this red color signals reliable fire safety in some boxes, detection of potentially erroneous boxes is not yet implemented

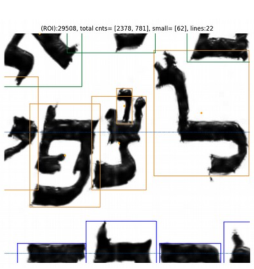
24643 (результат бинаризации и построения боксов на восстановленном изображении, вкл.ошибки), *красные точки отмечают отфильтрованные по интенсивности и размерам (и отброшенные) боксы*

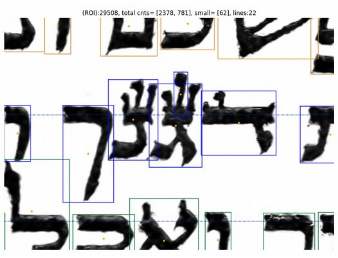


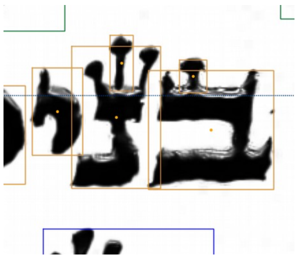


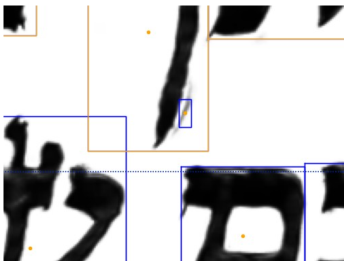
29508 (результат бинаризации и построения боксов на восстановленном изображении, вкл.ошибки), *красные точки отмечают отфильтрованные по интенсивности и размерам (и отброшенные) боксы*



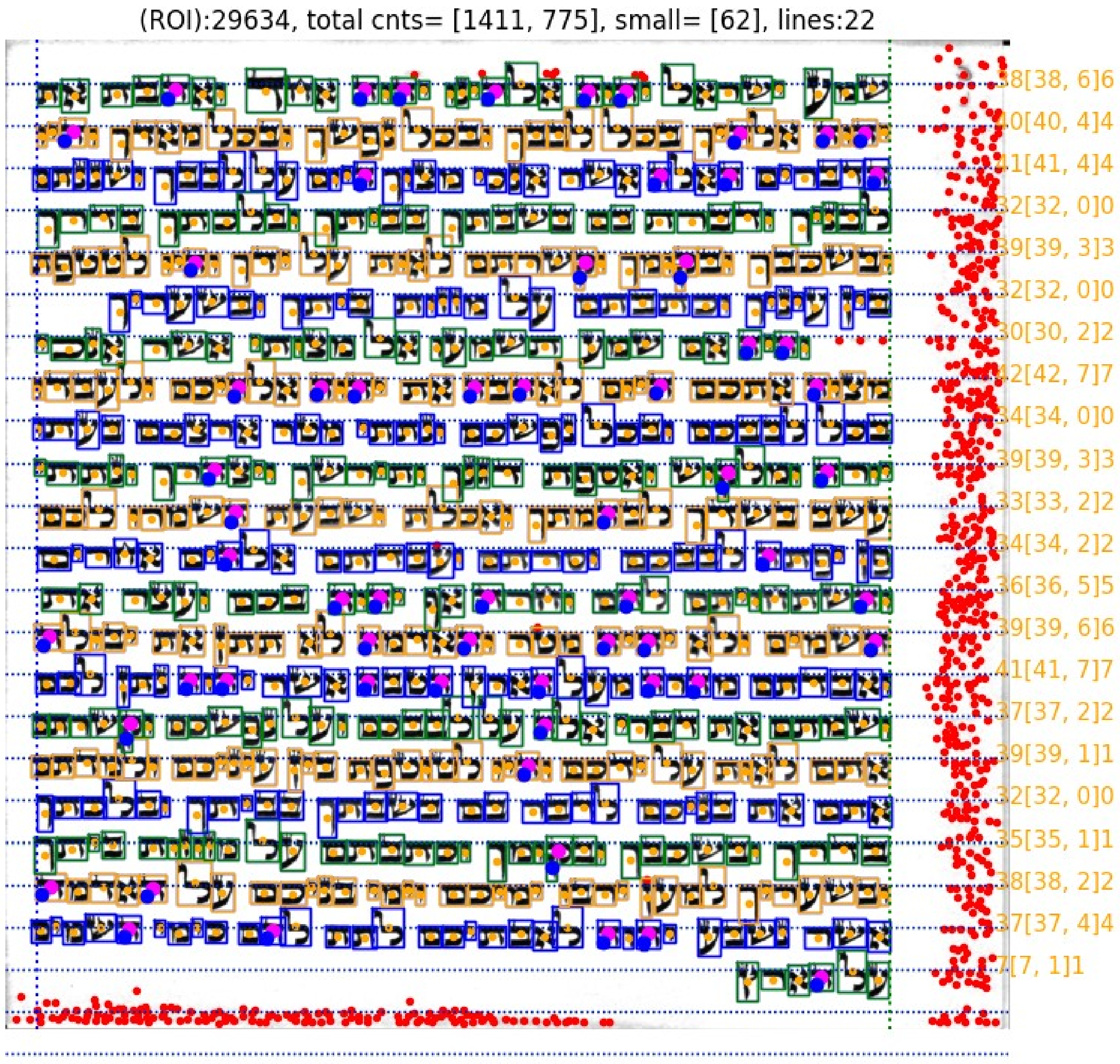






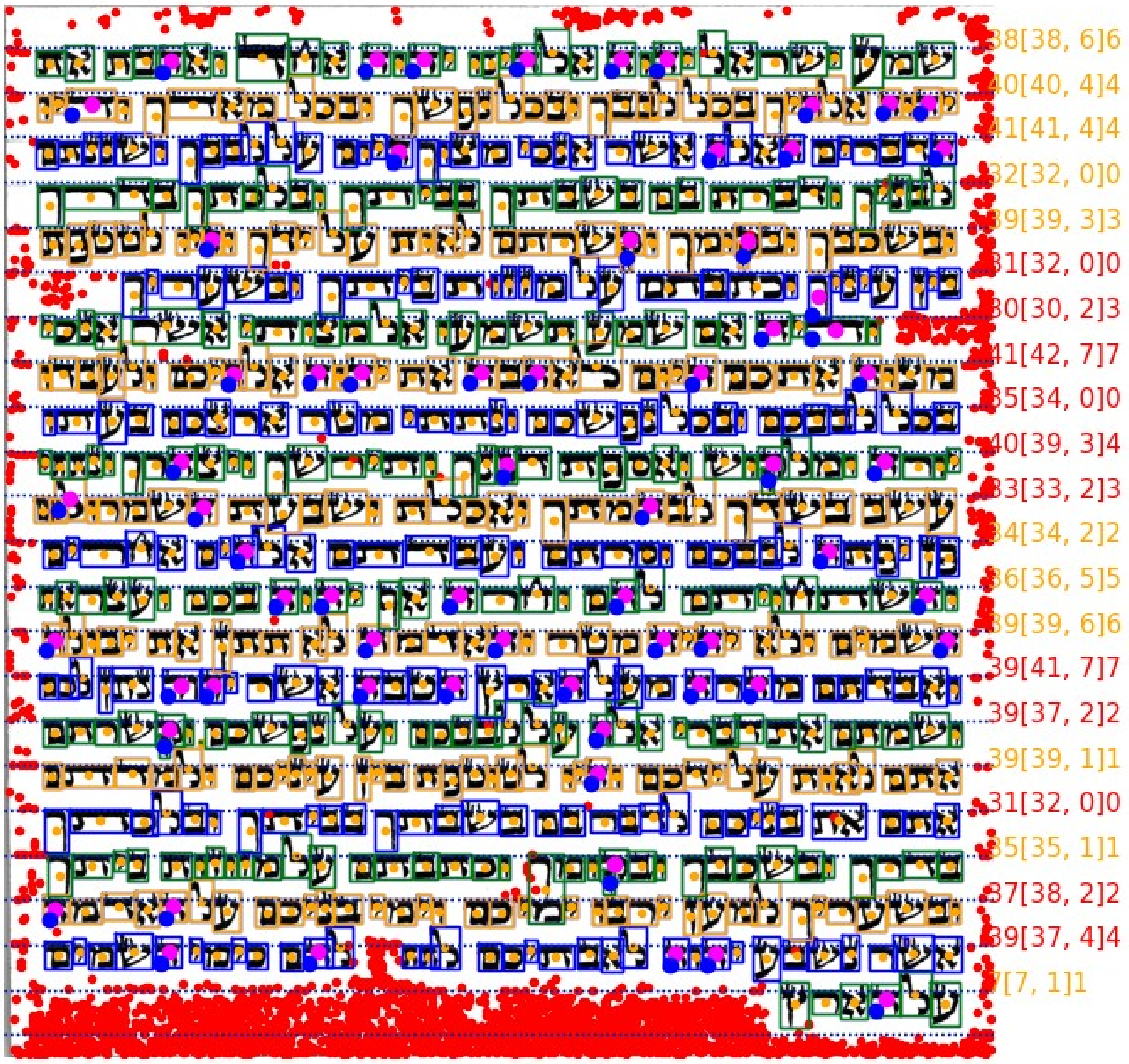


29634 (результат бинаризации и построения боксов на восстановленном изображении, вкл.ошибки), *красные точки отмечают отфильтрованные по интенсивности и размерам (и отброшенные) боксы*



1.png

(ROI):1.png, total cnts= [3458, 775], small= [65], lines:22


38[38, 6]6
40[40, 4]4
41[41, 4]4
32[32, 0]0
39[39, 3]3
31[32, 0]0
30[30, 2]3
41[42, 7]7
35[34, 0]0
40[39, 3]4
33[33, 2]3
34[34, 2]2
36[36, 5]5
39[39, 6]6
39[41, 7]7
39[37, 2]2
39[39, 1]1
31[32, 0]0
35[35, 1]1
37[38, 2]2
39[37, 4]4
7[7, 1]1