Optical recognition and solution of arithmetic expressions with fractions

Abstract. *Subject* Optical text recognition is one of the most important tasks of modern computer science. Contemporary recognition techniques mainly use a sophisticated mathematical apparatus. Recognition of arithmetic expressions with fractions differs from the classical text recognition because it needs additional pre- and post-processing to recognize the structure of fractions themselves. *Objectives* The objective of this work was to develop a simple recognition technique and to solve arithmetic problems for phones and tablets using it. It is relevant because people need to solve arithmetic expressions in printed form, and the calculation in the usual way is a time-consuming process. *Methodology* The standard for the Android set of IDE Eclipse with ADT has been selected as a development tool. The programming language was Java. Arithmetic expressions from real problem books were used for recognition and subsequent solving. *Results* An activity for the Android OS has been developed, which can be demonstrated on the tablet. The activity is fully operational and quite successful in recognizing targeted expressions. The obtained results provide a quick solution to arithmetic expressions with fractions by recognizing them from books or screens of various devices. *Conclusion* The targeted task consists of a sequence of subtasks (digital image acquisition, preprocessing, binarization, segmentation, character recognition, post-processing, result presentation, conversion of an expression into Reverse Polish Notation, and performing calculations). These subtasks can be solved on the Android OS. It is possible to achieve good results of recognition of even poor quality text images through the use of filters and complex algorithms of binarization and segmentation. For better results, it is necessary to improve the segmentation algorithm for the recognition of bad print in old textbooks. The way of further development includes, particularly building a working version and making it available on Google Play Market as well as expanding the program possibilities for recognition of decimal fractions and equations with fractions.

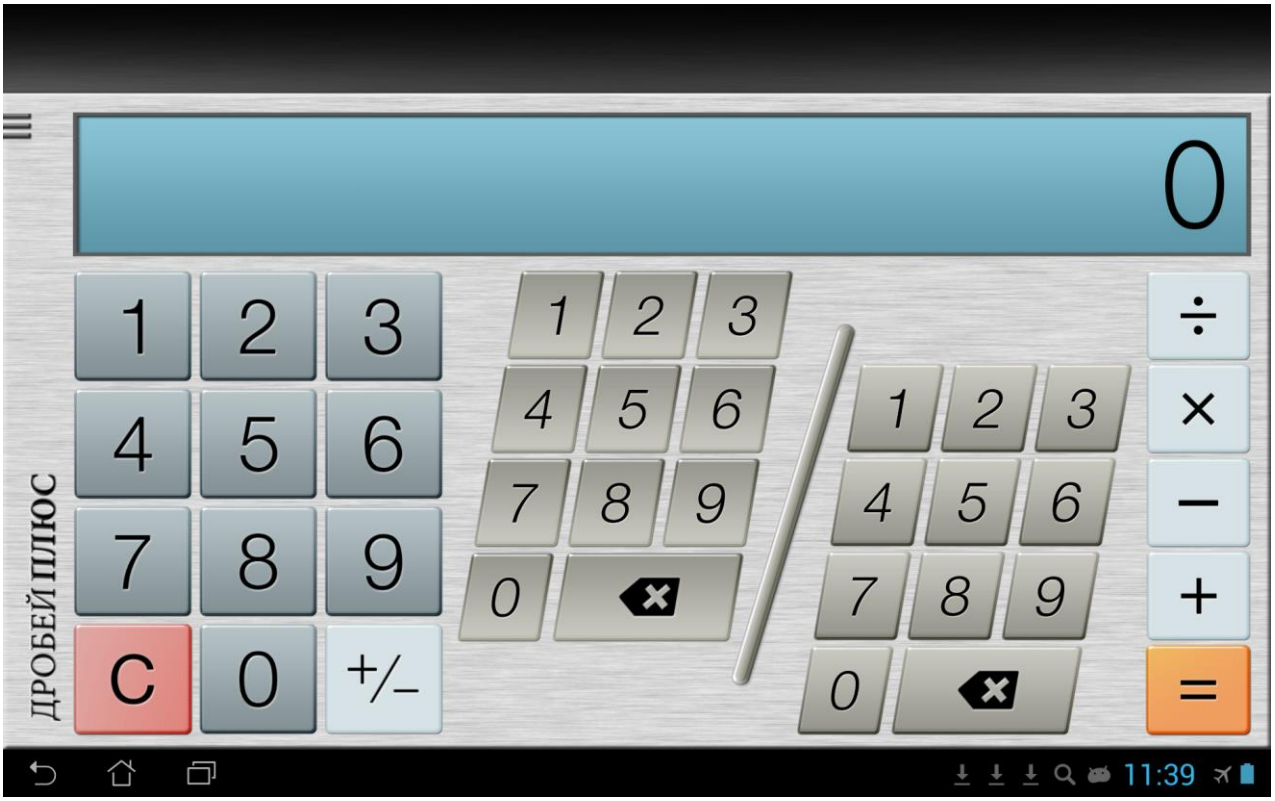
Key words: activity, Android OS, character, fractions, arithmetic expressions, OCR, print recognition, Reverse Polish Notation.

1. Introduction

In the fifth grade, we studied simple fractions and began to solve arithmetic problems. At first, it was really simple and even fascinating. However, the complexity of the examples quickly increased.
The next example [18]

$$\frac{(5\frac{1}{84}-(-\frac{31}{63})-(2\frac{31}{252}+3\frac{5}{21}))*(24:(1\frac{1}{2}:4\frac{3}{8}))}{(-\frac{7}{156}+\frac{1}{39}+1\frac{15}{26}):(20\frac{1}{4}:26)}$$

took 50 seconds to rewrite and 12 minutes to solve.
From the very beginning, it becomes clear that the problem of arithmetic operations with fractions can be easily solved with a computer. In order that such a program to be always with the user, it is necessary to have it for the phone and the tablet, that is, for the Android operating system. The search for programs to solve expressions with fractions of arbitrary complexity in Google Play Market did not give a result. However, there is a way for the input of fractional numbers into arithmetic expressions. Here is an example of a successful program:



This program appeared to be more convenient because it gave a unique input method of fractional numbers into arithmetic expressions (separate keys for an integer, numerator, and denominator). It was this program that gave the idea that the hardest part of the task at hand is not finding the result but the input of its conditions.

The research objective of this work was to build a simple recognition system as an activity for the Android OS and to solve a definite class of arithmetic problems for phones and tablets. It is a topical problem because many arithmetic expressions have to be solved based on their printed form (in books or on computer screens), and in the calculation in the usual way, it takes a long time.

2. Materials and methods

Input of fractional expressions

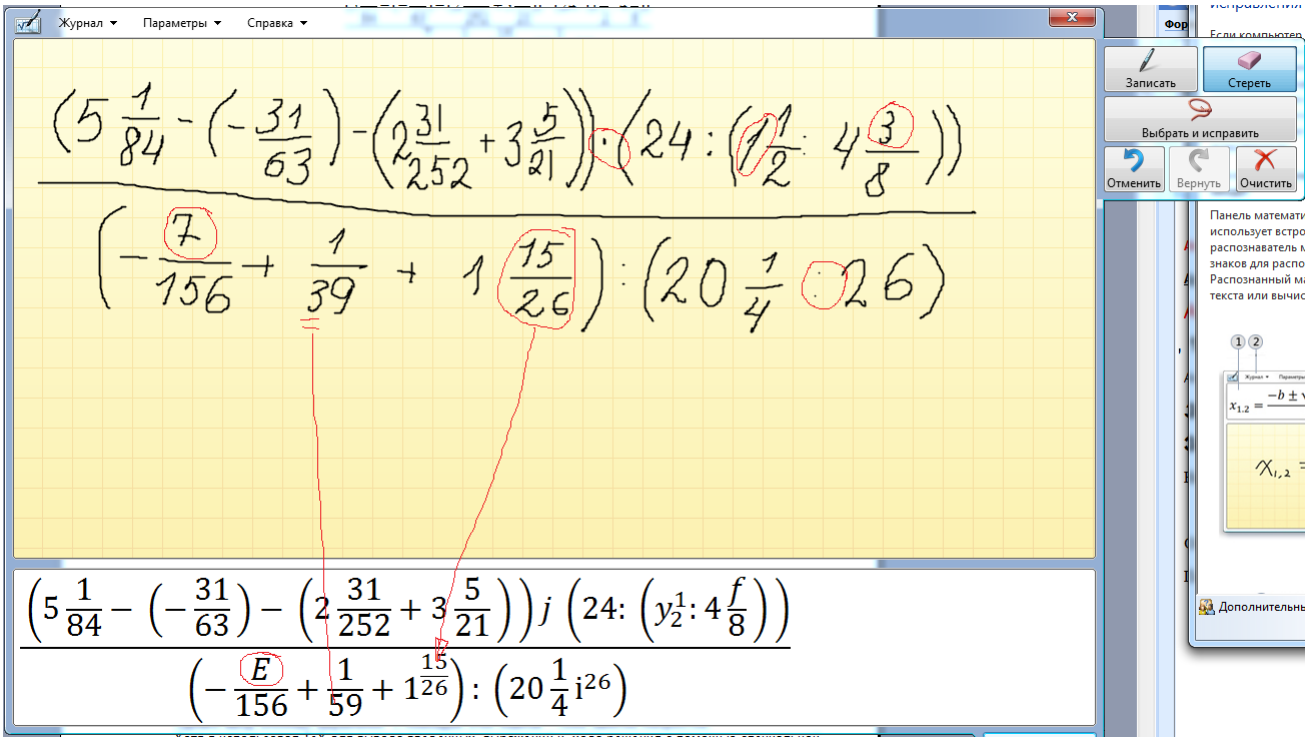
The standard for the Android set of IDE Eclipse + ADT has been chosen as a development tool [4]. The programming language of this choice is Java. The development in C# is also possible, but Monodroid appeared to be not well documented. All examples of mathematical expressions for the recognition have been taken from real problem books and other materials on mathematics [6-18].

Let us consider the same example. There are several ways of its input into the computer. There is a **special mode** for the input of formulas in **Microsoft Office** packages and the Google Docs application. But even if you have a full-fledged keyboard, the input of "two-story" fractions is not an easy task. During the work, the input of this expression in Microsoft Word 7 took **2 minutes and 45 seconds**. Windows 7 has a **Math Input Panel**, which is a convenient tool for handwritten input of formulas. It is designed for use with a stylus on a tablet PC. At the same time, it can also be used with any input device such as a touch screen, external digitizer, or even a mouse.

The input of the above fraction with the use of the **Math Input Panel** gave the following results:

– It took 5 minutes and 16 seconds (the initial input with the recognition of about 80% took 4 minutes and 30 seconds, and the correction before the full recognition took 46 seconds) with the tablet stylus.

– It took **6 minutes** with the **mouse**, but the desired result has not been achieved because of many mistakes. It is obvious that the mouse is not suitable for this kind of input.



Of course, these results are not representative because it is an issue of a lack of input practice and knowledge. However, it is obvious that both of these methods are not very suitable for the phone because there is no keyboard or stylus for handwritten input. Although, in the future, it would be preferable to provide the handwritten input for tablets with large screens, but there is an entirely different recognition technique (so-called online recognition). An alternative way to input expressions is to use specific markup languages for mathematical formulas. The most common are TeX and Mathematical Markup Language (MathML) [5].

The input of the above fraction with the use of **TeX** took 11 minutes and had the following form:
$$ \frac{(5\frac{1}{24}-(-\frac{31}{63})-(2\frac{31}{252}+3\frac{5}{21}))*(
24:(1\frac{1}{2}:4\frac{3}{8}))}{(-
\frac{7}{156}+\frac{1}{39}+1\frac{15}{26}):(20\frac{1}{4}:26)} $$

The input of the above fraction with the use of **ASCIIMathML** took 3 minutes and 32 seconds and had the following form:
(5 1/84-(-31/63)-(2 31/252+3 5/21))*(24:(1 1/2: 4 3/8))) /((-7/156+1/39+1 15/26):(20 1/4:26))

View in MathML (obtained by conversion) was as follows:
==========

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
 <mstyle displaystyle="true">
  <mfrac>
   <mrow>
    <mrow>
     <mo>(</mo>
     <mn>5</mn>
     <mfrac>
```

```
  <mn>1</mn>
  <mn>84</mn>
 </mfrac>
 <mo>-</mo>
 <mrow>
  <mo>(</mo>
  <mo>-</mo>
  <mfrac>
   <mn>31</mn>
   <mn>63</mn>
  </mfrac>
  <mo>)</mo>
 </mrow>
 <mo>-</mo>
 <mrow>
  <mo>(</mo>
  <mn>2</mn>
  <mfrac>
   <mn>31</mn>
   <mn>252</mn>
  </mfrac>
  <mo>+</mo>
  <mn>3</mn>
  <mfrac>
   <mn>5</mn>
   <mn>21</mn>
  </mfrac>
  <mo>)</mo>
 </mrow>
 <mo>)</mo>
</mrow>
<mo>&#x22C5;</mo>
<mrow>
 <mo>(</mo>
```

However, the input of complex expressions using these languages is also a difficult task. TeX was used to output the entered expressions and the progress in the solution using a specific program for the browser. The input time was measured when entering on the computer, and on the phone with its small on-screen keyboard, this input will be even longer. Based on the experience of these formats usage, an intermediate format was invented to represent the recognized expression. In this case, the expression will look like this:

```
[(5[1|84]-(-[31|63])-(2[31|252]+3[5|21]))*(24:(1[1|2]:4[3|8]))|(-
[7|156]+[1|39]+1[15|26]):(20[1|4]:26)]                          (*)
```

Optical character recognition (OCR)

It is obvious that the ideal solution would be optical expression recognition because modern phones and tablets, as a rule, have a digital camera that can be used to scan images. The task of optical text recognition is relevant to modern computer science. However, it is not an easy task. Most of the literature on this subject is based on a sophisticated enough mathematical apparatus. An additional

difficulty is that the best specialists work in large companies (for example, in the Russian ABBYY) and do not hurry to share brand-name algorithms. The task of recognition of mathematical formulas is somewhat different from the classical text recognition task because additional post-processing is required to recognize the structure of the expression itself.

Factors that make the task easier

The character set is limited to numbers and operational signs. Expressions in textbooks are printed in large and clear print. New problem books are printed on good paper.

Factors that make the task more complicated

Text without reference words complicates the task. It is not possible to use the frequency of letters and combinations of letters. Line division is not obvious because fractions often "hang" between lines.

The scanner also has limitations.

There is a difference between scanner and camera. The scanner provides uniform illumination of the treated area and presses the scanned page tightly. The scanner does not need to be focused because the distance to the scanned text is always the same.

3. Results and discussion

After analyzing the task and studying professional literature, the following sequence of program actions has been developed (with the first 7 points representing a classic sequence of the OCR task [1-3]):

1. Digital image acquisition.
2. Preprocessing (filters, vertical orientation, correction of geometrical errors).
3. Binarization and morphological transformations.
4. Segmentation.
5. Character recognition.
6. Post-processing including recognition of words, sentences, etc. In this case, it is highlighting the links between operational signs and numbers.
7. Presentation of the result as a text of any format. In this case, it is the format that has already been demonstrated before.
8. Conversion of an expression into Reverse Polish Notation.
9. Performing calculations.

Let us consider in detail the implementation of these paragraphs.

3.1. Digital image acquisition

At the first stage, the image is taken and cropped (Figure 1) using the built-in intents of Android (Capture and Crop).

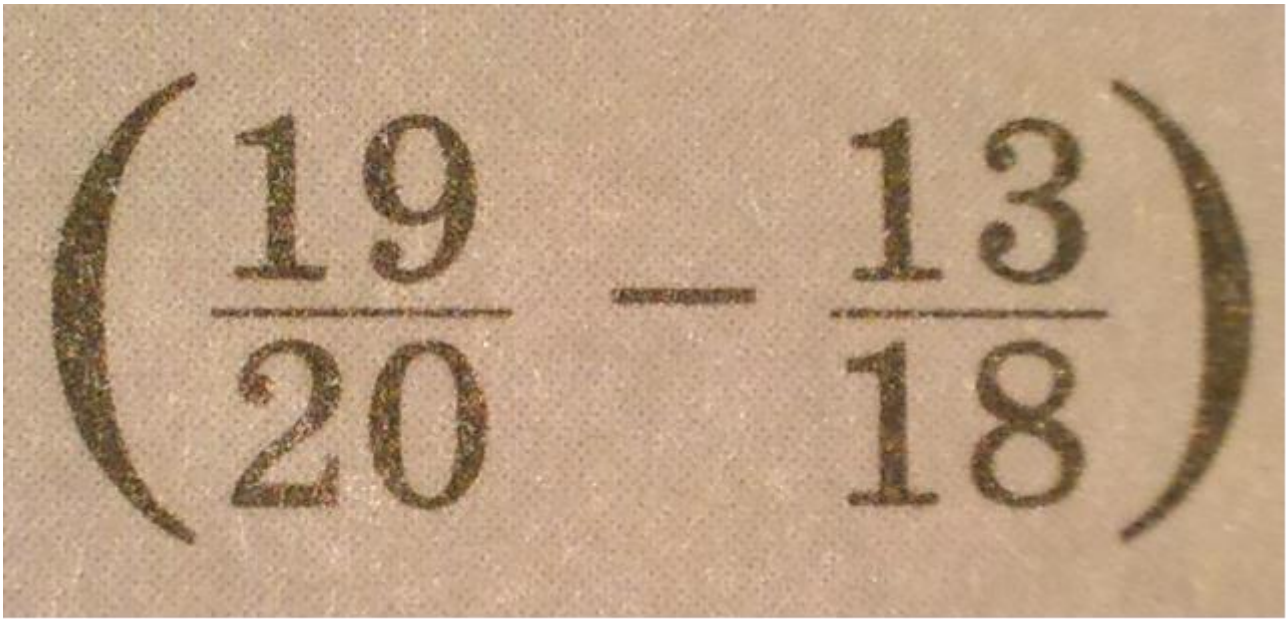


Figure 1. The initial image

Unfortunately, in the newest Samsung tablet (which was purchased specifically for the presentation because it had a Mirasast), the intent "Crop" is not as convenient as in the old Asus tablet, with which the development was implemented. The nearest work plans include making more convenient cropping. The result obtained in the form of BitMap is loaded into memory for further processing. There is an additional option to upload the image from the file.

3.2. Preprocessing

For all further actions, let us make the image black and white (more correctly, 256 shades of grey). The easiest way is to convert all colors to gray by simple summation and further normalization by the formula

$$Grey = ⅓ (R + G + B);$$

However, the human eye does not perceive all colors equally. Let us use the known formula for brightness

$$Grey = 0.299 \times R + 0.587 \times G + B \times 0.114$$

Figure 2 demonstrates how images look after being converted to shades of grey:

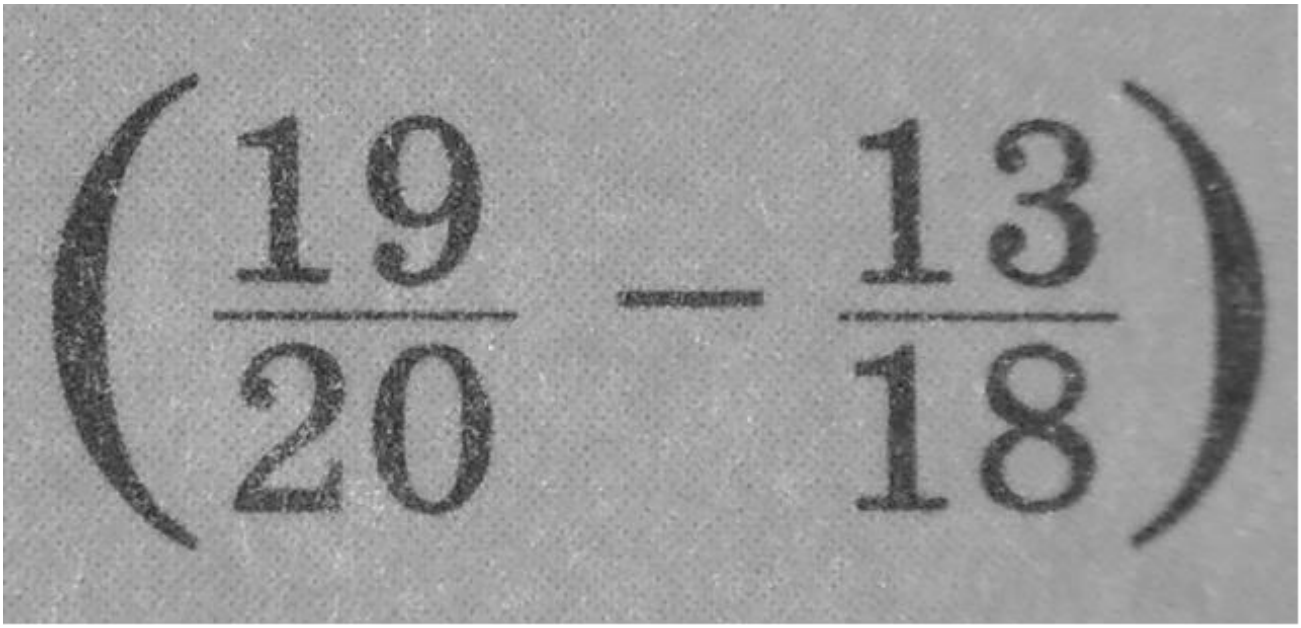


Figure 2. The image converted to shades of grey

The image contains a large number of defects or, in other words, noise. Different ways of image acquisition are characterized by different types of noise. In the figure below (Figure 3), you can clearly see the cavities, which, after binarization, will disintegrate the character.

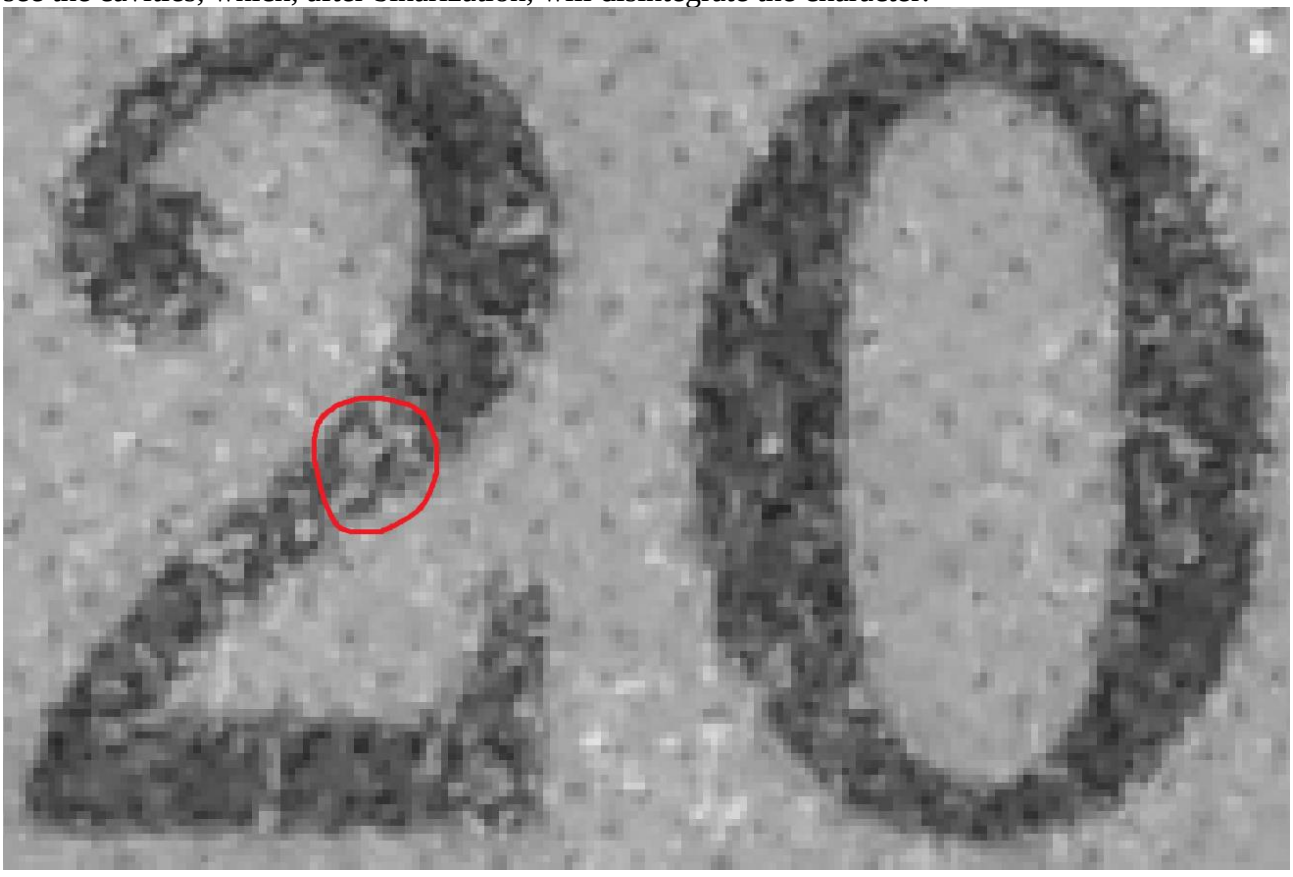


Figure 3. Cavities on the character

An additional source of problems may be non-vertical text orientation when shooting and geometrical errors (the book page hogging, etc.). Various filters are used to eliminate noise. A large class of filters

is linear filters. Commonly used non-linear filters include a median filter. The correct solution to the text recognition task depends on the qualitative solution of the preprocessing problem. Linear and median filters operate as follows. A two-dimensional square array with an odd side (called **window** or **aperture**) is taken. In this case, there is a central cell. The window moves all over the image surface. The color of the central point is redefined, depending on all other points in the window. In the median filter, it is the average of all values in the window.

Array M of length k2 contains elements of array D[i, j] ordered in ascending order.

142	147	149	149	155	165	169	177	178

$$D[x][y] = M[Int(k^2/2)+1]$$

where x, y *- are coordinates of the aperture center*

i, j *- are coordinates of the current aperture cell* $i=x-l, x+l;$ $j=y-l, y+l$

$l = Int(k/2)$ *, where k is the aperture dimension*

145	160	170	180	190	200	210	220	230	240	250
148	149	169	177	190	200	210	220	230	240	250
150	147	155 165	178	190	200	210	220	230	240	250
150	142	149	155	190	200	210	220	230	240	250
150	130	170	180	190	200	210	220	230	240	250
150	160	170	180	190	200	210	220	230	240	250

In the linear filter, each element of the window array is a coefficient by which the image element corresponding to that cell is multiplied. All these products are added together.

$$D[x][y] = \sum_{\substack{i=x-l,x+l \\ j=y-l,y+l}} (D[i][j] * A[i][j])$$

где x, y - *координаты центра апертуры*

i, j - *координаты текущей ячейки апертуры*

$l = Int(k/2)$, *где* k - *размерность апертуры*

145	160	170	180	190	200	210	220	230	240	250
148	1/9 149	1/9 169	1/9 177	190	200	210	220	230	240	250
150	1/9 147	1/9 159 165	1/9 178	190	200	210	220	230	240	250
150	1/9 142	1/9 149	1/9 155	190	200	210	220	230	240	250
150	130	170	180	190	200	210	220	230	240	250
150	160	170	180	190	200	210	220	230	240	250
150	160	170	180	190	200	210	220	230	240	250

*y[2][2]=(149+169+177+147+165+178+142+149+155)*1/9=159*

There is a vast diversity of linear filters. A broad range of effects can be achieved by using them. For example, the Sobel filter is designed to emphasize the borders. In this case, the simplest linear filter was used. It is a filter that averages the value across the window. Its effect is well visible in the Figure 4 because it smears over the image a bit.

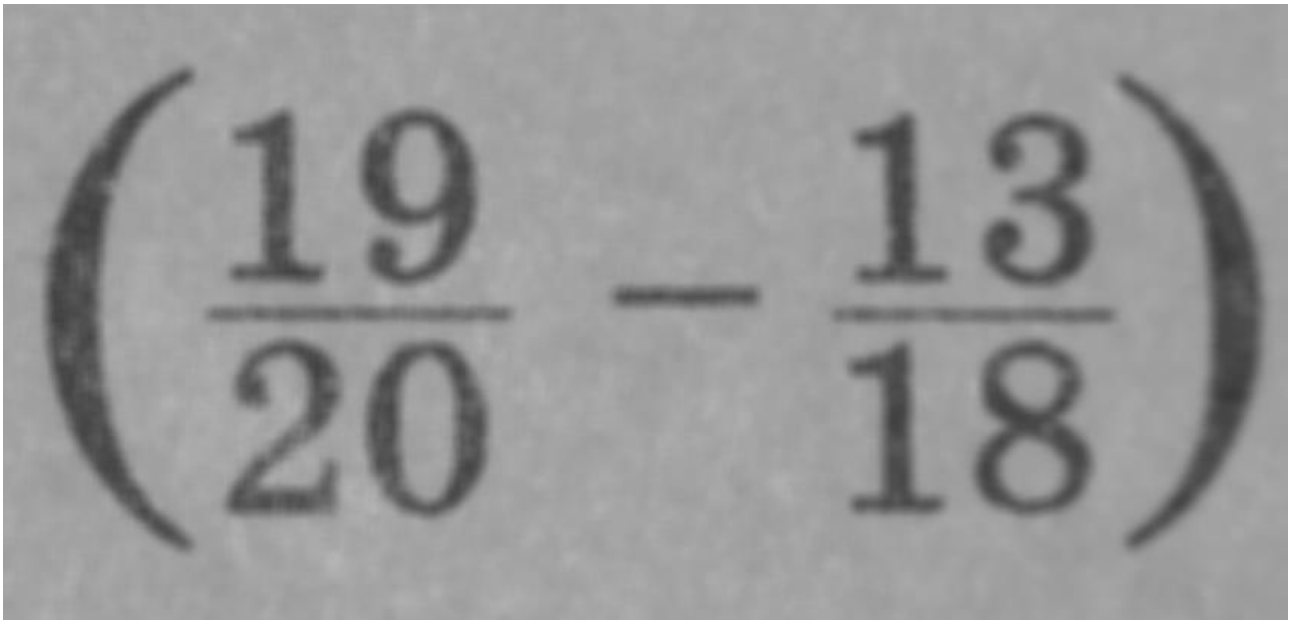


Figure 4. The linear filter effect

The linear filter operates much faster than the median one because it does not require sorting. In this case, only one multiplication by iteration was used. In the literature, the median filter is highly appreciated in terms of noise elimination. However, after the experiments, any advantages compared to the linear filter have not been found for the task. The chosen size of the window was 7×7 pixels. The results of the median filter operation are demonstrated in Figure 5:

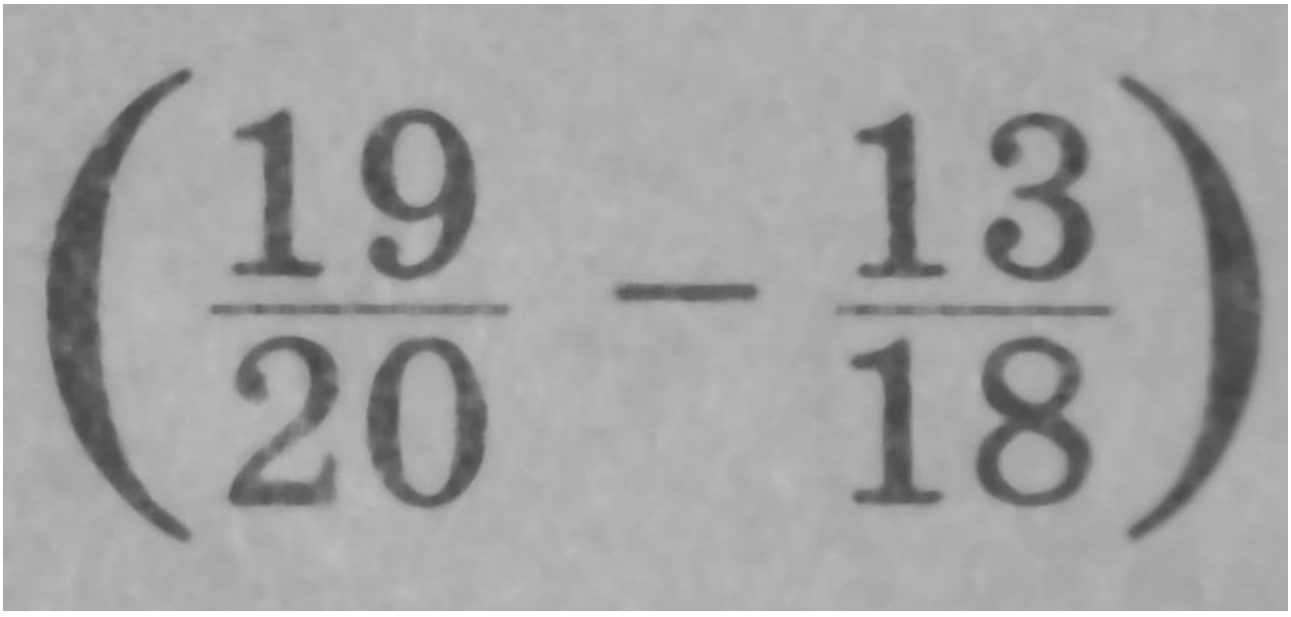


Figure 5. The results of the median filter operation

The task of the acquisition of the correct horizontal orientation is fully entrusted to the user. But looking ahead, it is necessary to note that the algorithm is quite good at dealing with small deviations from the horizontal. For instance, this example is correctly recognized and solved by the algorithm (Figure 6).

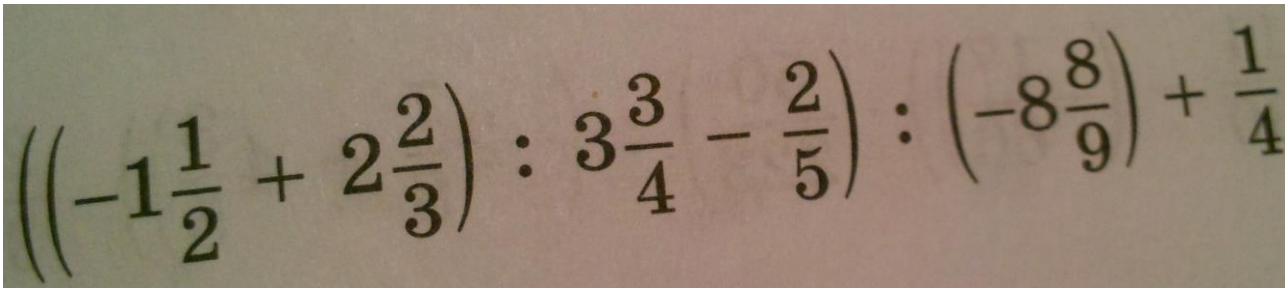


Figure 6. Correct results of the algorithm operation

3.3. Binarization

Binarization is one of the most important tasks of OCR. It is necessary to leave only two colors, white and black. The simplest but quite efficient way is threshold binarization. All points brighter than a certain threshold take the value of the **background**, and the rest points take the value of the **text**. It is important to choose the right threshold. The simplest way, which has been implemented at first, is expressed by the formula

$$\text{Threshold} = (\text{Max} - \text{Min})/2.$$

However, because of the noise, this method operates poorly.

Histogram analysis is indispensable for qualitative threshold selection. Color distribution histogram is an array containing the number of points of *i*-brightness in the *i*-element. In the case of 256 gradations, this array has a size of 256. A "good" text histogram with an example of arithmetic expression looks as follows (Figure 7).

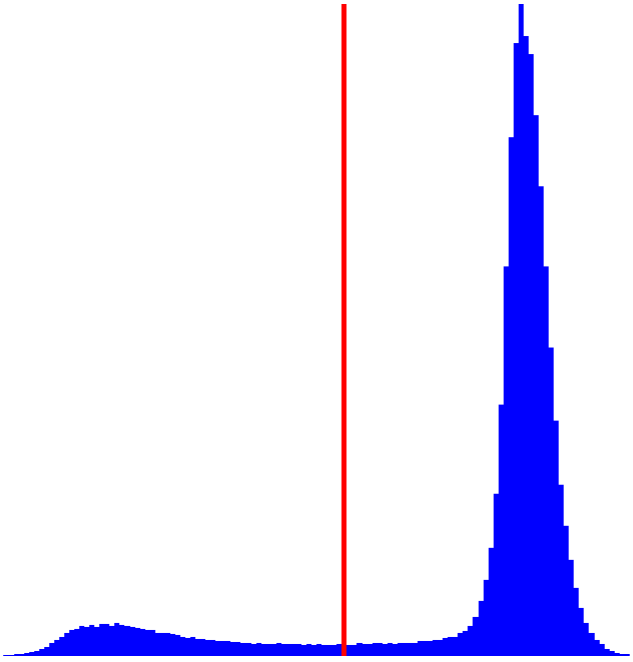


Figure 7. "Good" text histogram

There are two local maximums – the big maximum is the color of paper, and the small maximum is the color of ink. Some time was spent on writing a search program for these maximums. For the threshold value, the next one was taken

$$\text{Threshold} = \text{Max1} - (\text{Max1} - \text{Max2})/2.$$

For quality paper and uniform illumination, this algorithm worked well. However, the problems began when the ink maximum was very blurred. Here is a more typical histogram of the image (Figure 8). In this case, the proposed method, in general, finds a false threshold. It is a good result because there are many local maximums (yellow indicates the false threshold).

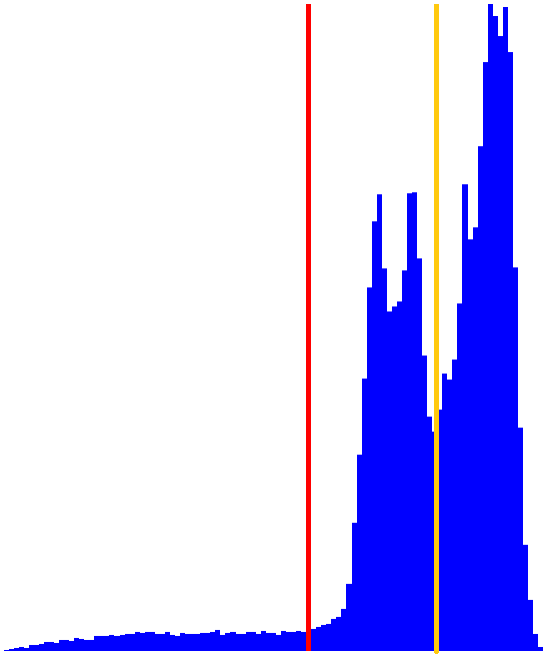


Figure 8. More typical histogram of the image

Then Otsu's method was used, which is one of the best binarization methods. The idea of the method is simple. You have to select a threshold so that the dispersion of each of the two colors is minimal. However, to implement this algorithm explicitly is very time-consuming. Otsu's method has proved that this task is equivalent to the task of intercolor dispersion maximization, according to the formula

$$W_1 * W_2 * (M_1 - M_2)^2$$

where W_i is the ratio of points included in the *i*-class to the total number of points; M_i is the arithmetic mean of color (brightness) of the *i*-class; $i = 1, 2$.

This algorithm is implemented in two passes over the histogram and works very well under uniform illumination. The results of its implementation are presented in Table 1 and Figures 9-11.

Table 1. Algorithm implementation according to Otsu's method

I	T	W1	W2	I*T	I*T all	M1	M2	FRESHOLD
0	1	1	99	0	0	0	5.4	2886.84
1	4	5	95	4	4	0.8	5.3	9618.75
2	7	12	88	14	18	1.5	5.8	19525.44
3	6	18	82	18	36	2	6	23616
4	4	22	78	16	52	2.4	6.1	23492.04
5	4	26	74	20	72	2.8	6.2	22241.44
6	66	92	8	396	468	5.1	7.9	5770.24
7	4	96	4	28	496	5.2	8.7	4704
8	2	98	2	16	512	5.2	8.5	2134.44
9	1	99	1	9	521	5.3	10	2186.91
10	1	100	0	10	531	5.3	0	28.09
Sum	100	569	531	531	2710	35	70	23616

6	6	2	2	3	2	4	6	6	6
6	6	8	7	4	3	5	6	6	6
6	6	8	7	5	3	7	6	6	6
6	6	2	1	3	2	0	6	6	6
6	6	3	5	4	5	10	6	6	6
6	6	1	1	7	9	6	6	6	6
6	4	2	1	3	2	6	6	6	6
6	6	6	6	6	6	6	6	6	6
6	6	6	6	6	6	6	6	6	6
6	6	6	6	6	6	6	6	6	6

Figure 9. Initial image in 11 shades of grey

6	6	2	2	3	2	4	6	6	6
6	6	8	7	4	3	5	6	6	6
6	6	8	7	5	3	7	6	6	6
6	6	2	1	3	2	0	6	6	6
6	6	3	5	4	5	10	6	6	6
6	6	1	1	7	9	6	6	6	6
6	4	2	1	3	2	6	6	6	6
6	6	6	6	6	6	6	6	6	6
6	6	6	6	6	6	6	6	6	6
6	6	6	6	6	6	6	6	6	6

Figure 10. The result of binarization

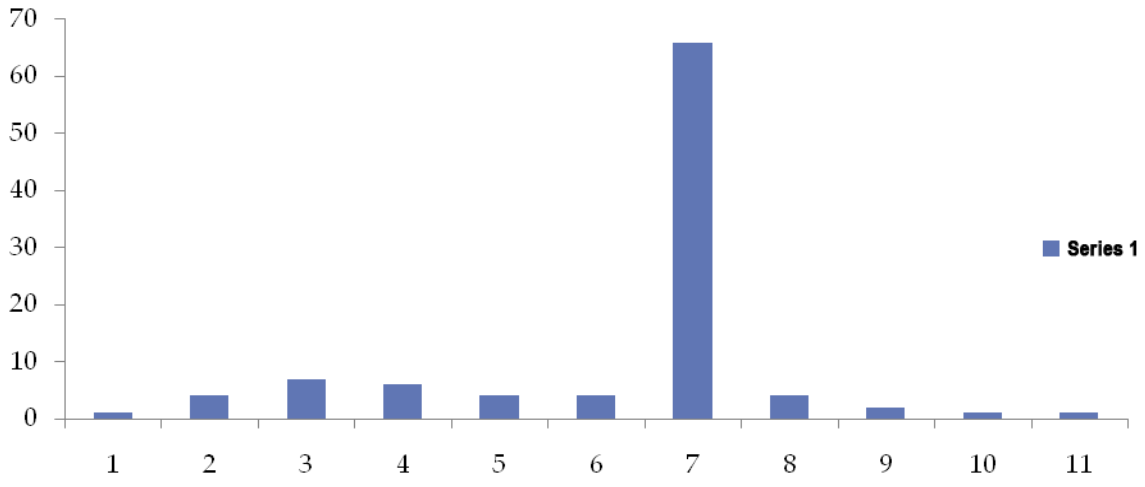


Figure 11. Series 1 histogram

Otsu's algorithm, unfortunately, is not always efficient enough. Problems occur as a result of non-uniform illumination. It has been found that the built-in camera is not necessarily the most appropriate scanner. When shooting without a flash, you often capture a picture that doesn't have enough contrast. At the same time, the flash does not give a uniform illumination. Figure 12 demonstrates an example of the image captured using the flash.

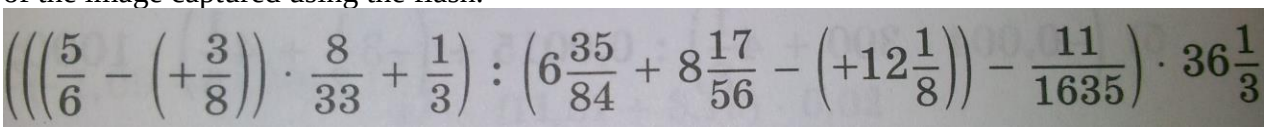
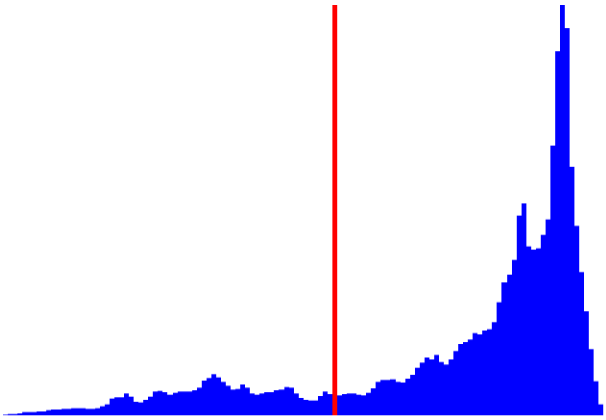


Figure 12. Example of the image captured using the flash

Figure 13 demonstrates the histogram and the result of binarization:



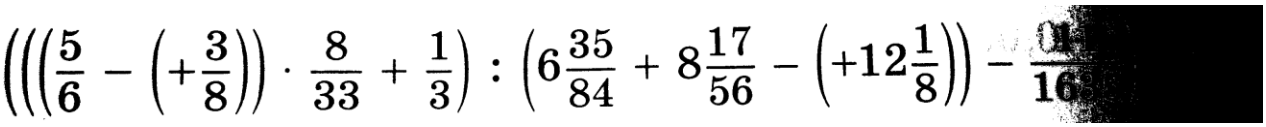


Figure 13. Histogram and result of binarization of the image captured using the flash

Besides, the flash gives glare on the new paper because it is not quite matte. It has been determined by experience that the flash should be used in extreme cases and take pictures from some distance. Furthermore, the proposed recognition method also works well with slurred images. For instance, the example below can be recognized well (Figure 14).

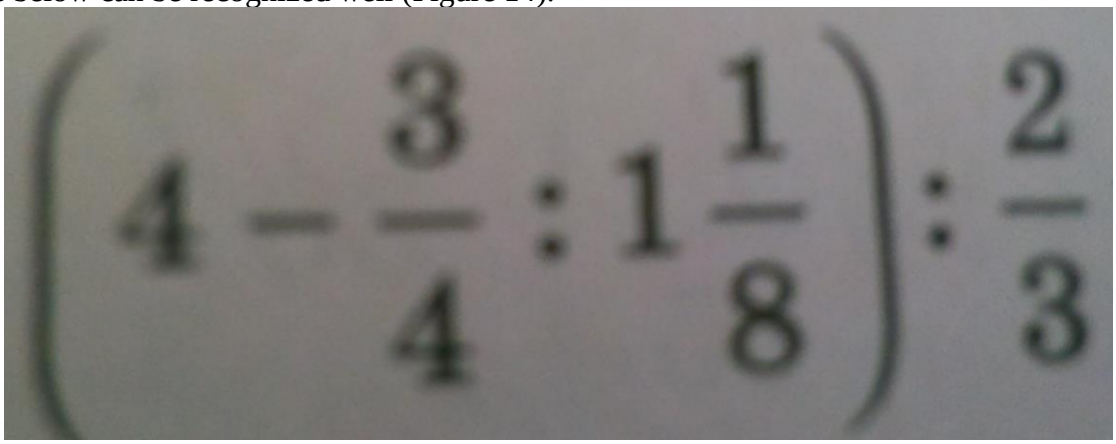


Figure 14. Example of the well recognizable slurred image

With the purpose of binarization improvement, it is necessary to split the image into parts to reduce the

nonuniformity of illumination. In this case, it is necessary to take parts of the size reliably larger than the character size. This problem has not been solved yet but postponed for the future. Binary images are often also subject to additional processing or transformations, which are called morphological. Morphological transformations have not been implemented yet. Figure 15 demonstrates binarization without preprocessing.

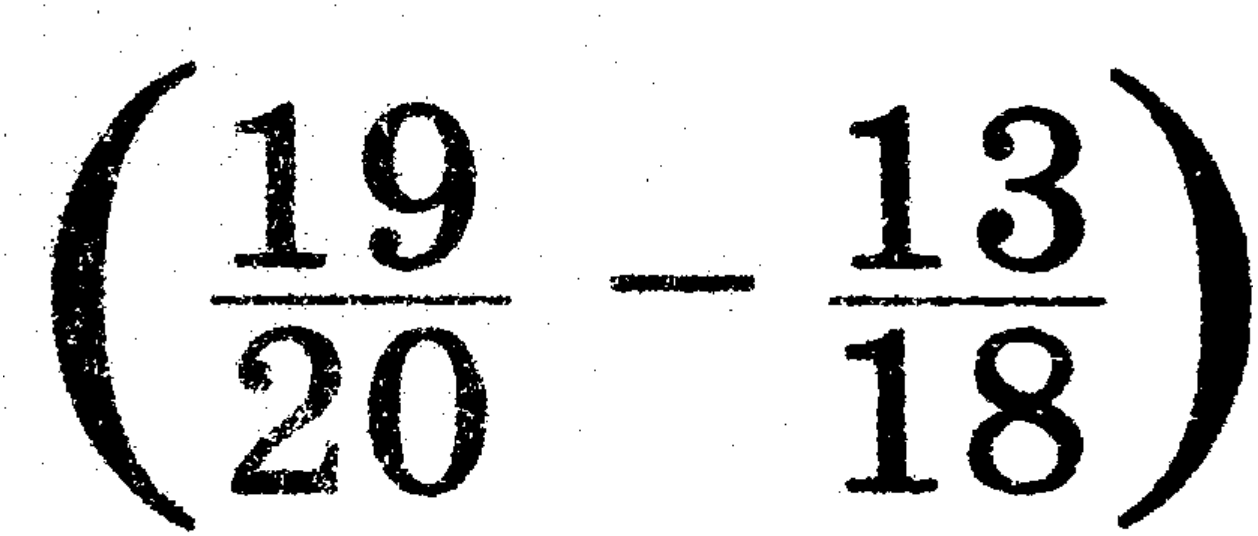


Figure 15. Binarization without preprocessing

Figure 16 demonstrates binarization after the linear filter.

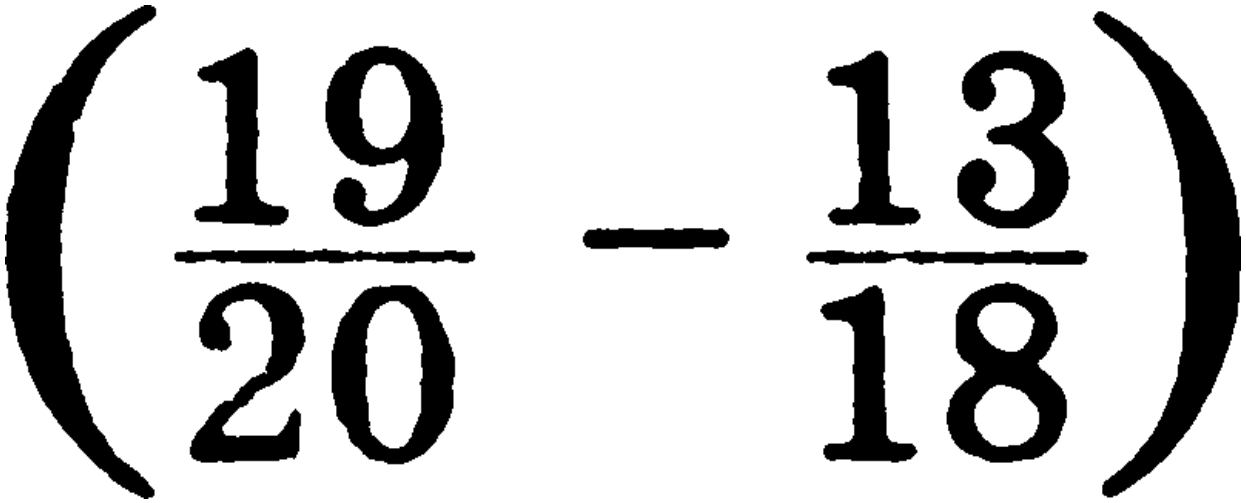


Figure 16. Binarization after the linear filter

Figure 17 demonstrates binarization after the median filter.

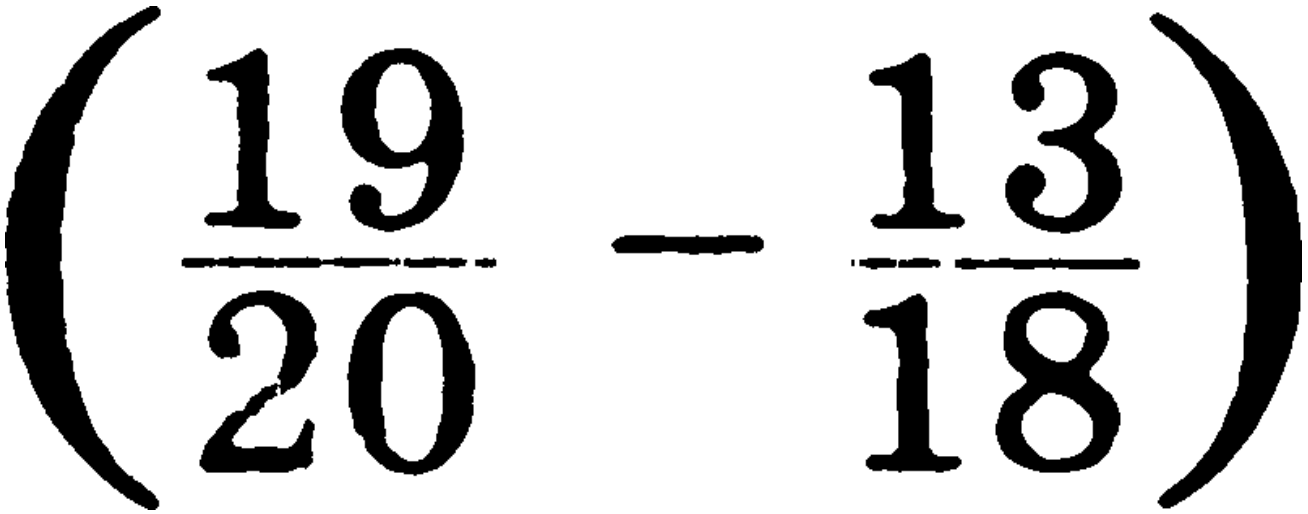


Figure 17. Binarization after the median filter

3.4. Segmentation

Segmentation represents the selection of individual characters in the text for further processing. The classic method of segmentation is three-stage segmentation on the basis of the average brightness of lines and columns. Thus, it is possible to select lines at first, then individual words, and then characters. This selection is followed by a fitting based on the geometric ratio of the height of lines, width and height of characters. At the final stage, the connectivity of characters is analyzed to get rid of false borders.

There were difficulties in the implementation of this algorithm for the following reasons:

- There is no guaranteed vertical text orientation because this pre-processing stage has been missed so far;
- Fractions break the simple line scheme of the text, acting outside the lines. It is especially noticeable in the case of different "number of stories" of individual parts of the expression;
- The height of lines and the width of the character vary greatly. For example, a long fractional line is a separate line with a minimal height.

As a result, it was decided to analyze only the connectivity of characters for the moment. Actually, a simple fill operation and the entire filled area selection are performed. Smaller areas are cut off as noise. Such simple segmentation imposes a significant restriction on the ability to recognize accurately. First of all, this is the case of old books with a bad print. In this case, the character often decomposes into several parts or, conversely, the characters merge. It is also the case of glare on the paper surface from a flash. It also can be the result of the significant nonuniformity of illumination. Thin signs, like fractional lines and minuses, decompose particularly hard. On the other hand, these are the signs that are “glued” together during the post-processing process.

3.5. Character recognition

There are many character recognition methods. The main "classic" methods of character recognition are

- Template matching,
- Feature,
- Structural.

In large systems they are combined.

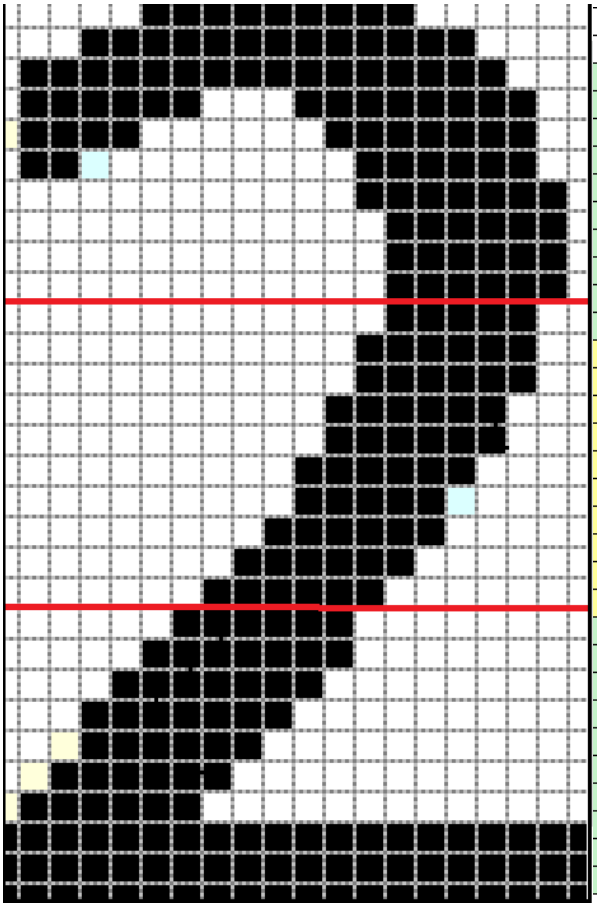
The template matching method in the simplest form is as follows. All characters are scaled to a certain size (for example, 16×8). The system has a set of templates of characters, typical for different types of character typefaces. The analyzed character is compared to a template, and the closest one is selected. The template matching method operates very fast but keeps away from high accuracy. Therefore, several candidate characters are usually selected, and these characters are analyzed using the template matching method.

The feature method determines a set of geometrical features of characters. The selection of these features is a difficult task. The literature often provides an example of a calculation of the number of line crossings with character boundaries.

The structural method implies the selection of geometric configurations and the links between them. This is the most complex method and it was not used.

Essential concepts of the project

The essential method, which is proposed, is a modification of the template matching method. Only not the character shape, but **the shape of the character boundary** is analyzed. The concept was as follows. The human eyes with the brain form a very complex organ, which knows how to distinguish the figures in their parts (interestingly, this is the basis of various optical illusions). However, when it is impossible to distinguish a figure (for example, in the dark), we often determine it by touch, of course, if the figure is three-dimensional. According to the proposed method, the selected rectangular segment is divided into N parts vertically and M parts horizontally. For each part, the average distance to the character boundary is found and normalized (Figure 18). Thus, an array of the size of 2N + 2M is obtained. By the way, the arrays for boundary measurements in the program have retained their names – Right hand, Left hand, etc.



Left hand			Right hand		
Distance	Distance norm	Difference	Distance	Distance norm	Difference
5	0,25	-0,10	6	0,3	-0,10
3	0,15	-0,10	4	0,2	-0,05
1	0,05	0,00	3	0,15	-0,05
1	0,05	0,00	2	0,1	0,00
1	0,05	0,00	2	0,1	0,00
1	0,05	0,55	2	0,1	-0,05
12	0.6	0,05	1	0,05	0,00
13	0,65	0,00	1	0,05	0,00
13	0,65	0,00	1	0,05	0,00
13	0,65	0,00	1	0,05	0,05
13	0,65	-0,05	2	0,1	0,00
12	0.6	0,00	2	0,1	0,00
12	0.6	-0,05	2	0,1	0,05
11	0.55	0,00	3	0.15	0,00
11	0.55	-0,05	3	0.15	0,05
10	0.5	0,00	4	0.2	0,05
10	0.5	-0,05	5	0.25	0,00
9	0.45	-0,05	5	0.25	0,05
8	0.4	-0,05	6	0.3	0,05
7	0.35	-0,05	7	0.35	0,05
6	0,3	-0,05	8	0,4	0,00
5	0.25	-0,05	8	0,4	0,05
4	0,2	-0,05	9	0,45	0,05
3	0,15	0,00	10	0,5	0,05
3	0,15	-0,05	11	0,55	0,05
2	0,1	-0,05	12	0,6	0,05
1	0,05	-0,05	13	0,65	-0,65
0	0	0,00	0	0	0,00
0	0	0,00	0	0	0,00
0	0	-	0	0	-

Figure 18. Division of selected rectangular segment into parts and determination and normalization of average distances to the character boundaries

However, it is easy to see that both red boundary lines have the same average distance to the character boundary. That is why another measurement – the differential measurement of the distance to the boundary has been introduced. Now, the boundary lines are easily distinguishable (Figure 19).

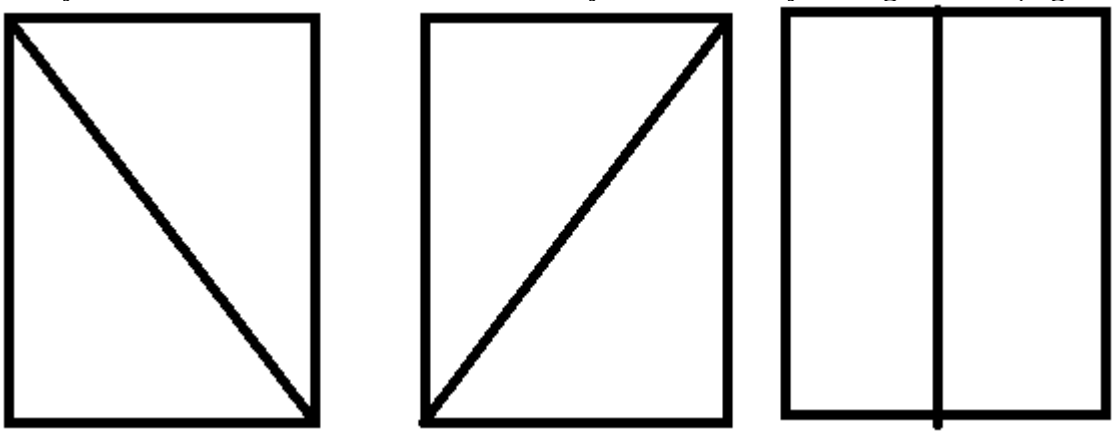


Figure 19. Difference between lines in the differential (left and central) and usual (right) measurements

For the template, characters of Arial font from Windows 7 have been chosen. It seemed that the character recognition does not require a large variety of fonts. But then, it turned out that the characters 1 and 2 have significantly different ways of writing. So, yet another option of these characters had to be added (Figure 20).

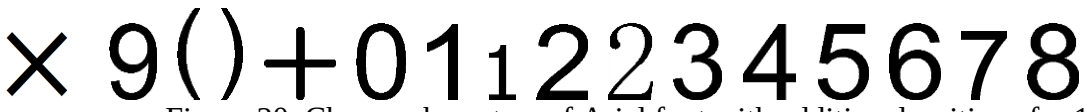


Figure 20. Chosen characters of Arial font with additional writings for 1 and 2

After that, the recognition became quite successful.
However, a specific "partial blindness" of the program has been found out. Sometimes the program recognized digits 6 and 9 as 0. It should be noted that this happened where the "tail" approached close to the circle. Also, sometimes the program recognized digit 3 for 8. While obtaining the recognition result of "0", it is necessary to look for the discontinuity points at the bottom left and at the top right. Discontinuity points are the points where the differential is larger than some threshold value. Similarly, when obtaining the result for "8", the discontinuity points to the left from the middle of the character are analyzed. As a result, a reasonably reliable way to recognize digits has been obtained. The characters of the "+" "×" operations are recognized in the same way. However, it is necessary to deal with the dot character in a different way. A dot is a character, which was recognized as "0". The dot has a very high ink density and is close to the "1" ratio of width to height. The "-" sign and the fractional line are recognized by the characteristic ratio of width to height.

3.6. Post-processing

All recognized characters are stored in **variables of a specific type**, where, among other things, there is a geometric location of the character. The first thing to do is to find the characters of a fraction. A fractional line is a line above which there are necessarily digits, and also, the nearest character from above is a digit. Otherwise, minuses and fractions can be confused. The next step is to find the characters ":". These are characters where the nearest character on top of the point is also the point. Additionally, it is necessary to "glue" two adjacent fractions or two adjacent minuses because such a record cannot be in the right expression.

3.7. Text presentation of any format

After that, a string is formed in the format described above (see Materials and methods. Input of fractional expressions, **(*)**). The string is generated recursively for the numerator and denominator.

3.8-3.9. Conversion into Reverse Polish Notation and performing calculations

Algorithms for arithmetic operations

Despite the experience in the development of the integer number calculator for complex expressions and the experience in stack implementation, it was decided not to reinvent the wheel but to use classical algorithms. It is easy to program the calculation of any complex expression written in Reverse Polish Notation. This record does not use brackets and has no operation priority. First, come operands, and then the operation. The calculation is easily organized on the stack. Operands from the input string are added to the top of the stack. If the operation appears in the input string, corresponding operands are removed from the top of the stack, and the necessary action is performed. The result is added to the top of the stack. The computation ends when the input string is empty. There is an answer at the top of the stack.
Dijkstra's shunting yard algorithm is used for the conversion of the infix notation. The algorithm "sorts" operands and operations using the stack by priorities, and the output string is the string written in Reverse Polish Notation. The analysis of the input expression for the unary minus should be provided separately. Two different operations of subtraction (binary) and reciprocal number (unary) are denoted in the record of expressions by one minus sign. The minus sign was replaced by the sign "~", where the minus is the first sign of the expression or the first sign after the opening parenthesis. During the program tests, a unary plus was suddenly detected. After that, the unary pluses are simply removed beforehand.

Syntactic parsing and token selection are performed, taking into account the peculiarities of fractions recording. After token selection in Reverse Polish Notation, it looks like this:

$$\frac{19}{20}\ \frac{13}{18}+$$

After parsing the expression, the calculation is only a technical matter. The course of calculation was memorized for further demonstration.

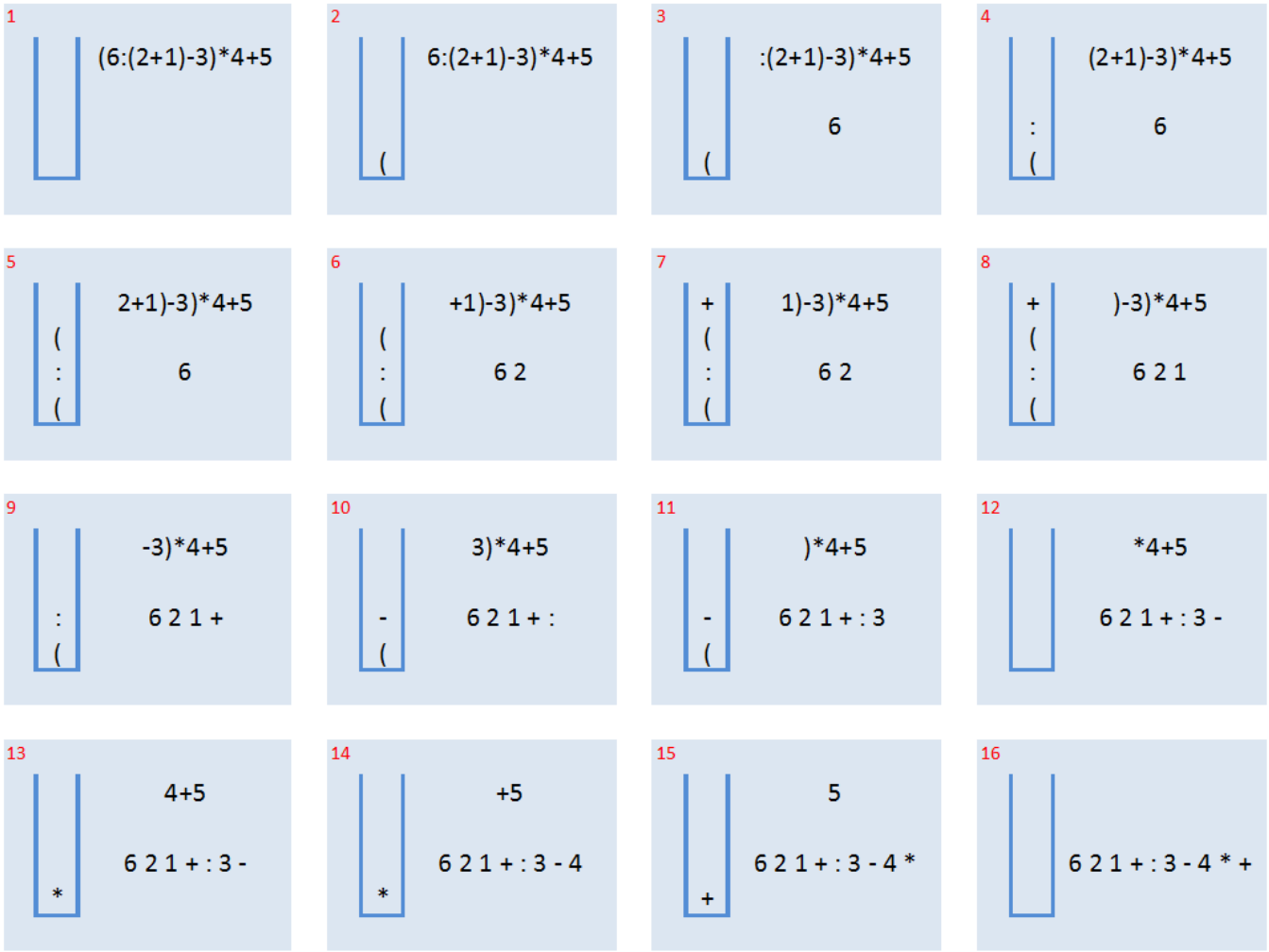


Figure 21. Practical demonstration of the shunting yard algorithm

4. Discussion

The results obtained after experimenting with a large number of problem books differ depending on the complexity of the task:

1. Small examples in new books are recognized with a very high probability.
2. If you have a successful image with uniform illumination, giant examples from new books are also recognized with a high probability.
3. Examples from new books scanned by the desktop scanner are recognized very well.
4. The print in old textbooks allows you to recognize only part of the examples. It is primarily due to the merged printing of some characters, the pale old color of the print, and blackened paper. To improve the recognition quality of these examples, it is necessary first of all to improve the

segmentation algorithm.

5. Expressions from computer screens at a definite resolution are recognized with a sufficient probability.

5. Conclusions

Throughout this project, an activity for the Android OS has been developed for optical recognition and solution of arithmetic expressions with fractions. The program has been compiled in the form aimed at demonstrating its principles and algorithms. The way of further development has been outlined. The program is fully operational and quite successful under certain conditions. The following conclusions have been drawn:

1. The OCR task consists of a sequence of subtasks. Except for the subtask of obtaining the text image, these subtasks can be solved at an adequate quality level on the Android OS. Each of these subtasks is of great importance for achieving the result.

2. Even with a degraded text image, it is possible to achieve good results of recognition through the use of filters and complex algorithms of binarization and segmentation.

3. Auto-testing is a critical element of a modern computer system.

The system needs further follow-up revision, more specifically, it is necessary to:

- Improve the binarization system by adding adaptive binarization mechanisms;
- Add morphological transformations to restore the connectivity of characters when the original image is of poor quality;
- Invent or adapt existing segmentation algorithms for fractional expressions;
- Add recognition of square brackets in an expression because they are found in some problem books;
- Supplement to post-processing the detection of an exponential operation with subsequent expansion of the class of tasks to be solved.

Plans for further development include:

- Building a working version and making it available on Play Market;
- Expanding the class of tasks by including decimal fractions and later – equations with fractions;
- Developing a system to recognize business cards with saving in the phone directory;
- Developing an activity to calculate the number of floors in a building.

References

1. Krasilnikov N. N. Digital processing of 2D and 3D images. Saint Petersburg. 2011 (in Russian).
2. Features of characters used for automatic recognition. Coursework. 2014. Available from: https://knowledge.allbest.ru/programming/2c0b65625a2ad79a5c53a88521316c27_0.html (in Russian).
3. The Internet tutorial of programming for Android http://www.developer.alexanderklimov.ru/ (in Russian).
4. Deitel P., Deitel H., Deitel A., & Morgano M. Android for Programmers: An App-Driven Approach. 2012.
5. Shiolashvili L. N. Presentation of mathematical texts on the Web [Electron resource]. Electronic Journals Library. 2005, Volume 8, Issue 6. Available from: http://www.elbib.ru/index.phtml?page=elbib/rus/journal/2005/part6/Sh (in Russian).
6. Ponomarev C. A. & Syrnev N. I. Collection of problems and exercises on arithmetic. 1964 (in Russian).
7. Zubareva I. I., Lepeshonkova I. P. & Milstein M. S. Self-directed learning. 6th grade. 2013 (in Russian).
8. Ershova A. P. & Goloborodko V. V. Self-directed and review works. 6th grade. 2014 (in Russian).
9. Erina T. M. Work-book on mathematics. 5th grade. In 2 parts. 2015 (in Russian).
10. Minaeva S. S. Calculations without errors. 5th and 6th grades. 2014 (in Russian).
11. Chesnokov A. S. & Neshkov K. I. Didactic materials on mathematics. 5th and 6th grades. 2013 (in Russian).
12. Popov M.A. Didactic materials on mathematics. 5th grade. 2015 (in Russian).
13. Zubareva I. I & Mordkovich A. G. Mathematics. 5th grade. 2013 (in Russian).
14. Vitanov T., Lozanov Ch., Dikulina L., Todorova P., Tsvetkova I., & Dzhonjorova I. Collection on mathematics. 6th grade. Bulgaria, 2014 (in Bulgarian).
15. Paskaleva Z. & Alashka M. Mathematics book with tasks and tests. 5th grade. Bulgaria, 2013 (in Bulgarian).
16. Erina T. M. Work-book on mathematics. 6th grade. 2014 (in Russian).
17. Evtushevsky V. A. Collection of arithmetic problems. 2014 (in Russian).
18. Gambarin V. G. & Zubareva I. I. Collection of problems and exercises on mathematics. 6th grade. Moscow, 2013 (in Russian).