Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

System for Automatic Checking of Solved Mathematical Equations in Raster Images

Daria Tunina

Supervisor: doc. RNDr. Daniel Průša, Ph.D.
January 2021

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Tunina Daria** Personal ID number: **474530**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Branch of study: **Computer and Information Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

System for Automatic Checking of Solved Mathematical Equations in Raster Images

Bachelor's thesis title in Czech:

Systém pro automatickou kontrolu vyřešených matematických rovnic v rastrových obrázcích

Guidelines:

Summary: The goal is to implement an application that checks correctness of simple mathematical equations (addition, subtraction, multiplication and division of two integers up to 99) solved by a user. The equations are printed on a sheet of paper, their solutions are filled in by hand. A camera is used to capture raster images of the assignment and its (partial) solution. The application interacts with the user and gives him hints if some of the equations are miscalculated.
Instructions:
1. Implement procedures that generate synthetic training data.
2. Use YOLO system [2] to train a detector of printed equations and handwritten digits.
3. Use suitable OCR libraries [3,4] for recognition of printed and handwritten symbols.
4. Propose and implement an application that interacts with a user.
5. Evaluate accuracy of the detection/recognition methods.
6. Test the application and analyse its usefulness for children practising to solve mathematical equations.
7. Document the procedures and results from the above points.

Bibliography / sources:

[1] R. Sedgewick, K. Wayne, R. Dondero: Introduction to Programming in Python: An Interdisciplinary Approach, Addison-Wesley Professional, 2015.
[2] YOLO: Real-Time Object Detection, https://pjreddie.com/darknet/yolo/
[3] N. Kumar, H. Beniwal: Survey on Handwritten Digit Recognition using Machine Learning, International Journal of Computer Sciences and Engineering, 2018.
[4] Tesseract Open Source OCR Engine, https://github.com/tesseract-ocr/

Name and workplace of bachelor's thesis supervisor:

doc. RNDr. Daniel Průša, Ph.D., Machine Learning, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **10.01.2020** Deadline for bachelor thesis submission: **05.01.2021**

Assignment valid until: **30.09.2021**

doc. RNDr. Daniel Průša, Ph.D. Supervisor's signature	doc. Ing. Tomáš Svoboda, Ph.D. Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student's signature

Acknowledgements

With this, I would like to express my gratitude to my supervisor, doc. RNDr. Daniel Průša, Ph.D., for his patience, valuable advice, and friendly and helpful supervision during this project. I am grateful for the given opportunity to work under his guidance.

I am also grateful for the encouragement and support that my friends and family gave throughout my studies.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, January 5, 2021

.....

Abstract

This project solves the problem of recognising mathematic equations and handwritten digits in an raster image. It is a popular problem. This project varies from other projects that are developed to recognise text. A system that automatically checks correctness of completed or partially completed simple mathematical tests is developed. The localisation of mathematical expressions is implemented by the object localisation system YOLO. The detection of printed digits is implemented by OCR system Tesseract. And lastly, the detection of handwritten digits is implemented by the CNN algorithm. A user interface for interacting with the application is also developed.

Keywords: recognition, YOLO, Tesseract, CNN, mathematical equation, handwritten digit

Supervisor: doc. RNDr. Daniel Průša, Ph.D.

Abstrakt

V tomto projektu je vyřešen problém rozpoznávání matematických rovnic a ručně psaných číslic na obrázku. Je to populární problém. Tento projekt se liší od ostatních projektů, které byly vyvinuty pro rozpoznávání textu. V tomto projektu je vyvinut systém, který automaticky kontroluje správnost dokončených a částečně dokončených matematických testů. Lokalizace matematických výrazů a číslic je implementována systémem lokalizace objektů YOLO. Detekci tištěných číslic implementuje OCR systém Tesseract. A konečně, detekce ručně psaných číslic je implementována algoritmem CNN.Je vyvinuto také uživatelské rozhraní pro interakci s aplikací.

Klíčová slova: rozpoznávání, YOLO, Tesseract, CNN, matematické rovnice, ručně napsáné číslice

Překlad názvu: Systém pro automatickou kontrolu vyřešených matematických rovnic v rastrových obrázcích

Contents

Chapter 1

Introduction

Computer vision has different tasks that many people are trying to solve. Object Localisation and Detection is one of the most significant ones. In this problem, *"given an image, the algorithm has to decide the locations of one or more target objects, outputting bounding boxes for each that appears in the image or video frame"* [1]. Recognition of text in a raster image can be seen as object localisation and detection. The localisation of text means that the algorithm will give the location of blocks of text in the image. The output of the said algorithm is a list of coordinates of bounding boxes for each block of text on the image. When a location of the text is know, the next stage, text recognition, follows. Optical Character Recognition (OCR) is used for converting *"images of typed, handwritten, or printed text into machine-encoded text"* [2].

1.1 The problem explanation

This project will concentrate on recognising mathematical equations and their handwritten answers on a raster image. The recognition of mathematical equations can be seen as the text recognition. The considered mathematical expressions will be addition and subtraction. They are simple and designed for younger pupils to solve (around 2nd grade of an elementary school).
The localisation of mathematical equations and handwritten answers will be done with a system for object detection YOLO. It will be trained on custom generated data. The detection of printed mathematical equations will be done by an OCR engine Tesseract. The CNN algorithm will be used for the detection of the handwritten answers.
The goal is to implement a system that will be used for checking correctness of completed and partially completed mathematical tests automatically. The system will provide a user interface for interaction. The user is supposed to be a young pupil who can solve the mathematical tests. The tests will be located on a piece of white paper. The application will work with a photograph of the paper made by a camera. After the system got the image it will run it through the recognition algorithm. After that, the system will calculate the correct result of the mathematical equation and then compare it with a given

handwritten answer.
The system can work on different platforms. To simplify the project, it will only be implemented as a desktop application. The application will guide the user to the correct answer by checking their progress and putting an encouraging message near the assigned mathematical equation. The message will differ depending on the user's results.
This implementation can be divided into three steps: recognise a printed mathematical equation, recognise handwritten number and implement a program that informs user if written answer to the equation is correct or not.

1.2 Similar applications

The algorithms for (mathematical) text localisation and detection are commonly used e.g., in translation programs and programs that help solve mathematical equations. Some of the examples are MyScript [1], which can convert handwritten text, figures, and math equations to a digital form. Another example is MathPix Snip and MathPix OCR [2]. They can convert typed and handwritten text from different languages, mathematical and chemical equations into popular document formats, such as LaTeX, Microsoft Word, and Microsoft Excel.
These systems can recognise mathematical expressions and handwritten digits. They are not able to automatically check the correctness of the answer. It is the main difference between the system of this project and these systems.

[1] https://webdemo.myscript.com/views/math/index.html
[2] https://mathpix.com

Chapter 2

Localisation of mathematical expressions

2.1 Problem explanation

The recognition of printed mathematical equations will be implemented with a library for object detection in images. Object detection means that the algorithm can localise an object (find a bounding box around it) and classify it (in other words, assign a class from the predefined list to the object). The library YOLO (You Only Look Once) will be used. The specifications of this library will be discussed later.
There are a couple of ways how to implement recognition of equations. The first solution is to identify individual numbers and arithmetical operators by YOLO and after that combine them to get an equation. But there can be a problem with correctly recognising symbols of addition, subtraction and the equality symbol, so it will be useful to find out where these symbols are located. The second solution is to identify the entire equation by YOLO. Because of the complications of the first method, the second solution will be implemented in this work.

2.2 Generating images

A large training dataset is needed for the training. This dataset will consist of images for recognition and their annotation. The dataset should be large (around 25000 of different images), because of that, creating dataset of real data would take a lot of time. So the first step is to create this dataset by implementing a program that automatically generates synthetic images that contain the equations.
The expected look for the paper is described below.
The paper has an A4 size. It can have a portrait or landscape orientation. The paper contains the mathematical equations that are located in columns. Text that explains to a student what they are supposed to do can be before or after the columns of mathematical expressions. There is a space after a mathematical equation that is enough to write an answer to the problem. Blue pens are commonly used by a lot of people. Therefore, the answer will be expected to be written with a blue pen. Most of the mathematical tests

for children found on the internet have the same number of mathematical expressions in columns. Therefore, for simpler implementation, it was decided to follow this layout. A focus is also on the mathematical equations for the youngest pupils. For more uniform implementation, the maximum possible answer will be 99 and the minimum possible answer will be 0. Therefore, each digit in the mathematical equation is between 0 and 99.
Also, before implementation, it is worth noting that photographs of a document do not look as good as that original document in real life. The conditions of taking a photograph such as lighting, camera quality and angle are essential to have a dissent photograph. These conditions should be noted when developing a program that generates images replicating photographs.
Implementation of the synthetic images generator requires background images, which are images that replicate different types of paper (see Figure 2.1).

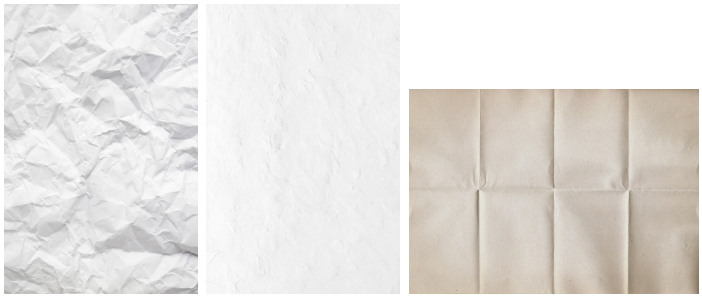


Figure 2.1: Examples of the initial images [3][4][5].

The program is developed in Python using additional libraries. First of all, library PIL (Python Image Library) is used for handling images. With the help of this library equations can be pasted on an initial image.
Secondly, a data augmenting application [1] is used for image rotation and shearing to make it look similar to photographs that are taken from different angles.
Thirdly, library scikit-image is used to make images blurrier with Gaussian noise, which simulates different qualities of cameras.
Correctly generated text is also important for a believable generated image. Library loremipsum is used for this task. The text can be pasted before or after equations. On the subject of equations, they are put in columns. The recognition will better work with larger fonts for the equation symbols. There are also needed to be spaces for the handwritten digits between the mathematical equation. Therefore, the number of columns will vary from 1 to 3 to keep the equations large enough for correct recognition. Reasons for keeping the number of equations in each column between 3 and 9 are the same. If the number of columns and the number of equations in each column is larger than the proposed limits, mathematical expressions will still be recognised. But the quality of the prediction will go down the smaller the equation will get. Text is pasted on the image in a colour on a grayscale from

[1] https://github.com/Paperspace/DataAugmentationForObjectDetection

black to dark-grey (from $(0, 0, 0)$ to $(50, 50, 50)$ respectively in RGB colour model).
And last but not least, images of handwritten digits are pasted to the right-hand sides of the equations. Handwritten digits are taken from the MNIST database (Modified National Institute of Standards and Technology database) [6], which is a large database of handwritten digits. Images of this database are white digits on a black background, therefore, images need to be modified before pasting them. Firstly, each image is inverted before pasting, so it becomes an image of a black digit on a white background. Secondly, the white background needs to be changed to the background of an image that the picture with the digit will be pasted on. Thirdly, the colour of handwritten digits should resemble the colour of a blue pen, therefore all of the black pixels on an image are changed to blue pixels. All of this is done with the help of the NumPy library.
With this implementation a large dataset can be created. Dataset is divided into training data and validation data. Training data is used for training YOLO on it, and validation data is used to validate the trained detector. Training data consists of 90% of all images, and test data is 10% of the initial dataset.

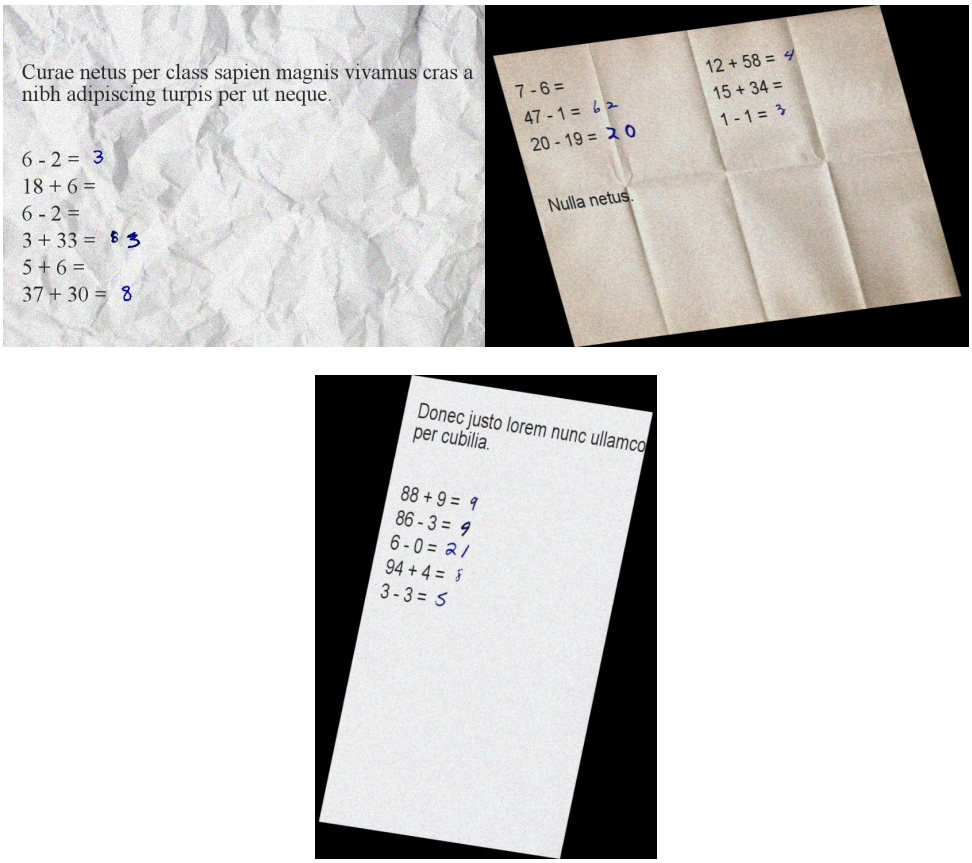


Figure 2.2: Examples of images generated using the method described in 2.2.

2.3 Labels and objects

The program that automatically produces training data images should also generate a text file, that contains information on the objects in the image. In this case, these objects are equations. Training of the object detector is done by using YOLO system. Therefore the text files should contain this information:
$< object >< center\text{-}x >< center\text{-}y >< width >< height >$, where

- $< object >$ is an integer in the range $0, .., (\text{classes} - 1)$, where *classes* is the number of different objects. In this work, the number of classes will be 2. YOLO will be trained to detect equations and handwritten digits separately
- $< center\text{-}x >$ is a float number between 0.0 and 1.0 that represents $x - coordinate$ of the object's border divided by the image width
- $< center\text{-}y >$ is a float number between 0.0 and 1.0 that represents a $y - coordinate$ of object's border divided by the image height
- $< width >$ is a float number between 0.0 and 1.0 that represents width of object's border divided by the image width
- $< height >$ is a float number between 0.0 and 1.0 that represents height of object's border divided by the image height

YOLO requires text files to be in the same folder and to have an identical name as the image that corresponds to the text file. For example, for an image "img_01.jpg" (see Figure 2.3) the information will be in the file with the name "img_01.txt", that contains a line:

```
0 0.14222873900293256 0.42130987292277616
 0.26099706744868034 0.06647116324535679
```

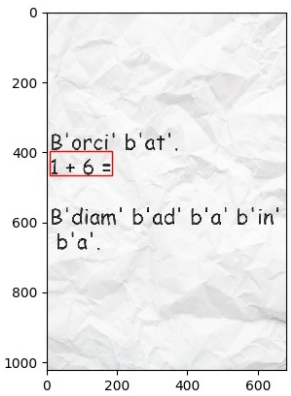


Figure 2.3: *img_01.jpg*

2.4 YOLO

The first version of this library (YOLOv1) was created by Joseph Redmon in 2016. YOLO uses a single neural network that predicts bounding boxes and class probabilities directly from full images in one evaluation [7]. Neural network divides an image into regions and predicts bounding boxes and probabilities for each region. YOLO is fast, and its main advantage is that it can accurately detect objects in real-time detection.
The second version YOLOv2 was released in 2017 [8]. YOLOv2 has new features that were not in YOLOv1 such as Batch Normalization, Higher Resolution Classifier, Anchor Boxes, Multi-Scale Training, Fine-Grained Features.
The third and last version of YOLO by Joseph Redmon was released in 2018 [9]. Here are some features that were improved in YOLOv3: Bounding Box Predictions, Class Predictions, Feature Pyramid Networks (FPN) [10]. After this realease, Joseph Redmond stopped working on new versions of YOLO.
Since YOLO is an open-source library [2], many other developers decided to work on improving YOLO. Alexey Bochkovsky has been working on updated variants of YOLOv2 and YOLOv3 since 2018 [3]. His updated variants provide useful functions that the original YOLO does not have. Alexey Bochkovskiy's version of YOLOv3 was used in this project since it was the most advanced YOLO version at the start of this project.
But since the project was started, newer, and in some ways better versions of YOLO were released. In April of 2020, Alexey Bochkovskiy released his version of YOLO, which he calls YOLOv4 [11]. In June of 2020, Glenn Jocher released a version of YOLO called YOLOv5. He is not associated with Joseph Redmon nor with Alexey Bochkovskiy [12]. In August of 2020, another version of YOLO was released called PP-YOLO [13].

2.4.1 Before training custom data

Some actions need to take place before training custom data.
Firstly, in *Makefile* set a graphical processing unit (GPU) to 1. The program will be built with Compute Unified Device Architecture (CUDA) to accelerate by using GPU. Without this, training will be immensely slow. After this, the program can be built by running *make* in the darknet folder.
Secondly, file *yolov3-obj.cfg* should be created. It is a file that has the same contents as the *cfg/yolov3.cfg* file. File *yolov3-obj.cfg* differs from *cfg/yolov3.cfg* file in some lines:

- It is needed to change the number of classes in [yolo]-layers and to change value filters, that is located before each [yolo]-layer. Value of filters is calculated by the formula: $filters = (classes + 5) \cdot 3$.

[2] https://github.com/pjreddie/darknet.git
[3] https://github.com/AlexeyAB/darknet.git

- Value of *max_batches* requires a change. It is usually $2000 \cdot classes$. For one class value, *max_batches* is better to be equal to 6000. After changing value of *max_batches*, value of steps is required to be equal to 80% of *max_batches*, 90% of *max_batches*.
- To solve the problem with the error: "CUDA Error: out of memory", value of *subdivisions* requires a change. It is a number equal to a power of 2.

Thirdly, files *obj.data, obj.names, train.txt, test.txt* need to be created.
File *obj.data* contains information about custom training data (see Figure 2.4).

1. The first line of that file is the number of classes that the training model has.
2. The second line is the path to the file *train.txt*.
3. The third line is the path to the file *test.txt*.
4. The fourth line is a path to file *obj.names*, which contains names of classes, each name on a separate line.
5. The fifth line is the path to folder *backup/*, where calculated weights will be saved to.

Files *train.txt* and *test.txt* contain paths to the files for training and testing respectively. Each path is on a separate line and refers to the image for training.

```
classes = 10
train  = data/train.txt
valid  = data/test.txt
names = data/obj.names
backup = backup/
```

Figure 2.4: Example of obj.data file.

2.4.2 Training in YOLOv3

After completing the initialisation part, training of custom objects can begin. Training is started by command:

```
./darknet detector train data/obj.data cfg/yolov3.cfg
 darknet53.conv.74 -dont_show -map
```

The example of one iteration is shown in Figure 2.5.

It starts with ($subdivisions \cdot number\_of\_[yolo]\_layers$) lines, that are followed by line that contains the number of the current iteration and average loss. So for $subdivisions = 8$ there will be $8 \cdot 3 = 24$ lines.
In the above example, the number 1001 on the last line is the iteration number,

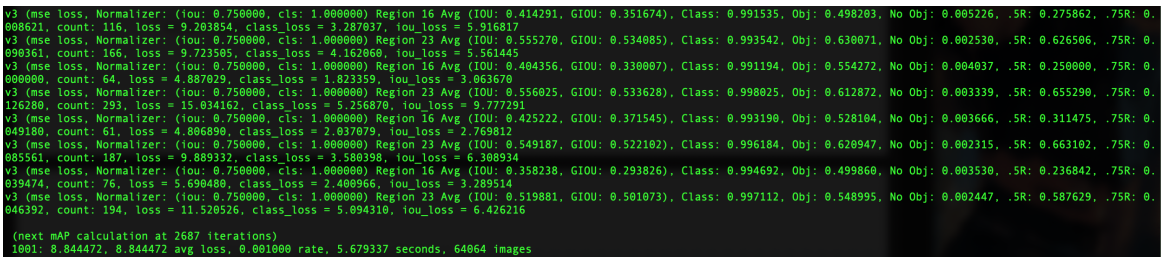


Figure 2.5: Example of one training iteration.

and the second number 8.844472 is the average loss number. The lower this number is the better the training goes.
If the average loss error is equal to *-nan*, it means that training is not going well. If all of the lines before iteration number have *-nan* on every place, besides *No Obj* value, then training is probably going wrong as well.

In addition to training command the second repository has these flags:

- The preciseness of written label files can be checked by using training command with a flag *-show_imgs*, that saves some images with borders around the objects to the *darknet/* directory.
- Flag *-map* calculates mean average precision for each class using files for testing. With the help of mean average precision, it is easier to know, when to stop training. If mean average precision stops decreasing, then training can be stopped. Map flag also saves best weights, so there is no need to check every iteration for the best weights.

Training is automatically stopped, when the number of iterations exceedes value of *max_batches*. But it can also be stopped manually. If training was started with flag *-map*, last weights will be saved to *backup/* folder.
Training should be stopped when the number of average loss stops decreasing.

For this project classes calculated mean average precision is **99.035%** for the class of a mathematical equation and **97.854%** for the class of handwritten digits. It is not the most precisely calculated recognition accuracy, but it gives an understanding of the accomplishments of the recognition algorithm.

2.4.3 Testing in YOLOv3

After training is done, and there are some weights in the *backup/* folder, testing can be started. It is activated be command:

```
./darknet detector test data/obj.data
        cfg/yolov3.cfg backup/yolov3_best.weights
```

Program asks for a path to an image. After receiving it, the program returns a prediction. The prediction is information about recognised objects. This information consists of coordinates of the objects and probability of

accuracy of the prediction. This prediction is written in a command line. The program also draws borders of recognised objects in the image and saves a new image with the name *predictions.jpg* (see Figure 2.6).

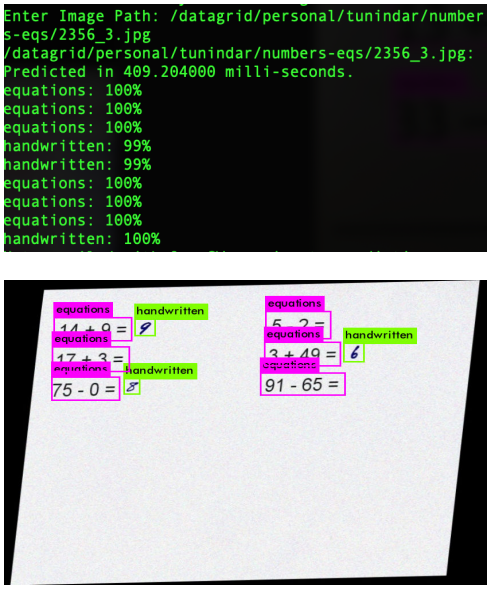


Figure 2.6: Examples of prediction in a command line and the predictions image.

Testing can also be done with command:

```
./darknet detector test data/obj.data
         cfg/yolov3.cfg backup/yolov3_best.weights
        -dont_show -ext_output < data/test.txt > result.txt
```

File *test.txt* contains paths to images. Predictions of that images as well as all of the information from the command line about that predictions are written to *result.txt* file.

Command

```
./darknet detector test data/obj.data
         cfg/yolov3.cfg backup/yolov3_best.weights
         -ext_output -out result.json < data/test.txt
```

writes information about recognised objects to *result.json* file. Example of information about one image in *result.json* file is shown in Figure 2.7.

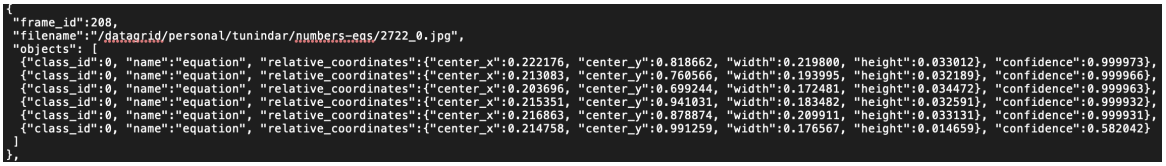


Figure 2.7: Examples of predictions of one image in *result.json* file.

Chapter 3

Detection of mathematical expressions

After getting the results of YOLO recognition, additional recognition of equations and handwritten digits is needed to be done. YOLO can show where equations and handwritten digits are, but not what are the equations and handwritten digits.

3.1 Detection of handwritten digits

There are methods that can recognise each handwritten digit. For example, methods from [14] can be used for this task. This repository offers an implementation of four methods for recognising handwritten digits: K-Nearest Neighbours, Supervised Vector Machine, Random Forest Classifier and Convolutional Neural Network (CNN).

Since the CNN implementation gives the best results out of these four methods (see Table 3.1 and Table 3.2), it will be used in this work. Firstly, the CNN model should be trained, and its weights should be saved for later use. It is done by the command:

```
python CNN_MNIST.py --save_model 1 --save_weights cnn_weights.hdf5}
```

The CNN model is trained on the MNIST dataset.
After that, the saved weights can be used for the recognition of handwritten digits. The algorithm that is used in the project was taken from [14]. Either one image or a list of images can be passed through the algorithm.

3.1.1 Pre-processing of an image

The image from the MNIST dataset differs from the custom image of a handwritten digit. Therefore, the image for recognition requires some pre-processing to make it resemble the image from the MNIST dataset. If the changes take place, the recognition results will be better.
Firstly, the sizes of the images from the MNIST dataset are 28 by 28 pixels. This means that the image for recognition is also required to have this size.
Secondly, the MNIST images contain a white digit on a black background. The custom images for recognition have a dark number written on a light background. This means that the image for recognition needs to be turned

	RFC	KNN	SVM	CNN
Trained Classifier Accuracy	99.71%	97.88%	99.91%	99.98%
Accuracy on Test Images	96.89%	96.67%	97.91%	98.72%

Table 3.1: Percent Accuracy of Each Classification Technique [15].

Model	Test Error Rate
Random Forest Classifier	3.11%
K-Nearest Neighbors	3.33%
Supervised Vector Machine	2.09%
Convolutional Neural Network	1.28%

Table 3.2: Classifier Error Rate Comparison [15].

to greyscale. Its colours need to be inverted as well. Now the image has a light number written on a dark background. Unfortunately, this still differs from the MNIST image. With the help of functions cv2.GaussianBlur, cv2.adaptiveThreshold and cv2.morphologyEx from OpenCV the image will be converted to an image that has an almost all-white number written on an almost all-black background.
Thirdly, the digits from the MNIST image are always in the center of the image. The same does not apply to the images for recognition. To make images similar, the following steps took place. All-black rows and columns at each end of the image were deleted. The new image was resized. Some of the all-black columns and rows were added back in a way, that makes the digit be in the center of the image.
Another obstacle for a better result of prediction was that some bounding box predictions are not perfect. Some of them cut part of the number off. Because of that, the number will be recognised incorrectly. The solution is to look around the bounding box for lightly coloured pixels. If the amount of pixels on one side is greater than the decided acceptable number, the bounding box will be extended on that side. The acceptable number is the same for each image. It was decided based on the results of the testing images. The previously described algorithm will continue until there are no more lightly coloured pixels around the bounding box. This workaround helps with recognition for most images, but there is a possibility that it will ruin good results of prediction for others. If the bounding box for the image is large, the algorithm can count pixels of another digit that was close to the original number. Therefore, at the end of the algorithm, the bounding box will include both digits. Because of that, the image is not going to be recognised correctly.

3.2 Detection of mathematical equations

3.2.1 Pre-processing

Similar to the images of a handwritten digit, the images of mathematical equations require pre-processing for better results. The pre-processing is done with the help of a function from OpenCV. Firstly, the image is turned to greyscale. Then the image is put through a Gaussian blur.

3.2.2 Tesseract and Python-tesseract

Recognition of the equation is done by Tesseract [1]. Tesseract is an optical recognition engine that is released under Apache Licence. It was originally developed by Hewlett-Packard as proprietary software [16]. In 2005 it was released as an open source to the public. Since 2006 the development of Tesseract is sponsored by Google [17]. At the moment, the latest stable version is 4.1.1, released on December 26, 2019. Tesseract can be trained on custom data, but it is not needed for the purpose of this application.
For this application, Python-tesseract[2] will be used. Python-tesseract is a wrapper for Tesseract. With the help of it, Tesseract can be used in a Python script. The following line will start a Tesseract algorithm in Python:

```
pytesseract.image_to_string(image)},
```

where the image is an OpenCV, NumPy, or Pillow image with a text that is needed to be recognised. OpenCV, NumPy, and Pillow are Python libraries that are most commonly used to process images in Python. This command returns a string that has a prediction of the text in the image. Unfortunately, it will look through all of the known for Tesseract symbols to find the best solution. Additional arguments can be used to limit the number of symbols that Tesseract will go through. Argument "config" can take different configurations, which are useful for modifying Tesseract recognition without the need to train Tesseract. With argument

```
-c tessedit_char_whitelist=
```

types of characters that will be recognised are specified.

For example, with

```
-c tessedit_char_whitelist=0123456789+-=
```

Tesseract will only look for numbers, addition, division, and equation signs.

Another useful argument is *–psm*, which stands for page segmentation modes. There are 14 different page segmentation modes. The ones that apply to this projects are:

[1] https://github.com/tesseract-ocr/tesseract
[2] https://pypi.org/project/pytesseract/

- 6 - Assume a single uniform block of text.
- 7 - Treat the image as a single text line.
- 8 - Treat the image as a single word.
- 13 - Raw line. Treat the image as a single text line, bypassing hacks that are Tesseract-specific. [18]

While testing all of the different modes in recognition, different results were achieved. That happened because the image with the mathematical equation contains an equation on one line, so it can be perceived as a single text line or as a single word. In some cases, psm 13 worked better. In others - psm 7. The first solution was to use different types of page segmentation modes and combine their results. This method has accurate results, but calculation takes more time since the same image goes through the Tesseract algorithm multiple times.
To save time one mode will be used. Mode 7 is the most accurate for the images of the equations. This mode treats the image as a single text line.

3.2.3 Post-processing

Sometimes YOLO does not recognise the bounding box of the image correctly. It can divide digits in half or does not include some of the digits of the equation. In that case, Tesseract would not give the right answer. That means that some post-processing must take place. The bounding box will be enlarged to the left and the right by the size of the width of one symbol. This width can be calculated since it is known what Tesseract has recognised. To get an approximate width of one symbol, the width of the image is divided by the number of recognised symbols. If Tesseract is not able to recognise any of the symbols on the image, the approximate width of symbols on the image will not be calculated. In that case, the default symbol width will be used. This width was calculated using testing images.

Even with these modifications, Tesseract can still have problems recognising some of the symbols. For example, a sign of addition is sometimes recognised as number four. This problem can be solved by looking at the predicted string. For example, if the predicted string looks like $2345 =$, there is a large possibility that the actual equation is $23 + 5 =$. Another example is when Tesseract recognises string $36 + 37 =$ as $364 + 37 =$. Since it was previously established that numbers lay in the range from 0 to 99, an extra digit 4 in the first number can be deleted. With that in mind, post-processing will become more complicated, but overall results will be more accurate.

Chapter 4

Evaluation and results

The recognition accuracy of YOLO and recognition accuracy of the CNN algorithm is known. As was previously established the mean average precision of YOLO for a mathematical equation is 99.035%. The mean average precision for handwritten digits is 97.854%. The recognition accuracy of the CNN is 98.72%. These are unfortunately not a recognition accuracy of the whole application. After the YOLO and CNN algorithms, additional post-processing and Tesseract recognition for mathematical equations are done. Because of that, the final recognition accuracy of the application is unknown.

4.1 Evaluation dataset

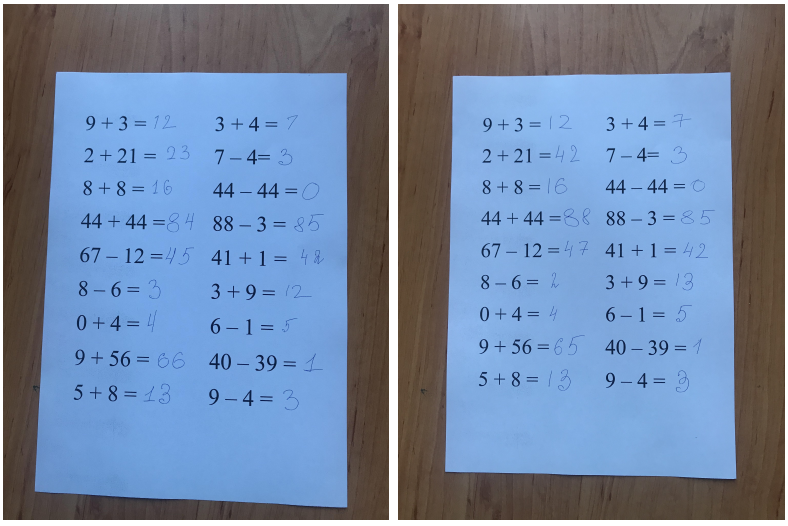


Figure 4.1: Example of two types of papers with different answers.

There are several steps in order to find the final recognition accuracy of the application. Firstly, a database of testing images is needed to be created. Each image is a photograph of a paper that lies on a desk. The photograph is taken by phone iPhone 7 Plus, which has Dual 12MP Wide and Telephoto

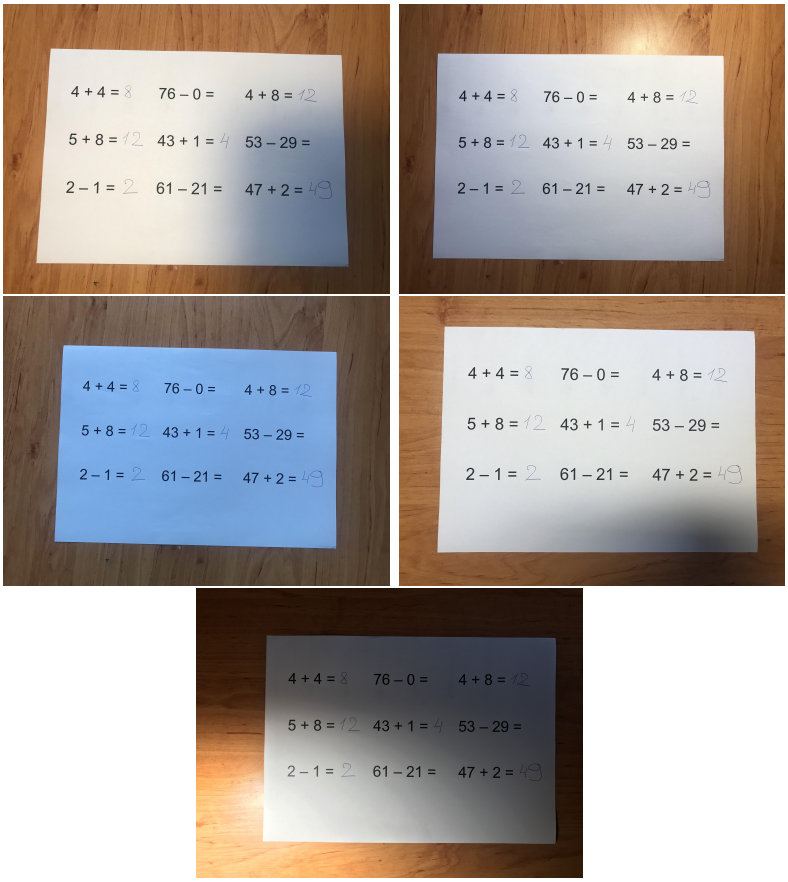


Figure 4.2: Example of one type of papers with different lighting.

cameras [19]. The papers are similar to the papers from the generated dataset. They have rows with mathematical equations and handwritten answers on them. Suppose that type of paper is different if the mathematical equations on it differ from other papers. In this dataset, there are 20 different types of papers. Each type is filled with answers (handwritten digits) 2 times (see Figure 4.1). Each paper was photographed 5 times with different lighting or camera position (see Figure 4.2). In the end, the database consists of 200 images.
Secondly, evaluation needs to compare recognised results with the ground truth results (what objects the photograph actually has). These results can be divided into the outcome of bounding box recognition and the recognition results of handwritten digits and mathematical equations.

4.1.1 Bounding box annotation

Image annotation is the process of labelling data in the image [20]. Annotation tools are widespread applications that vary, for example, in types of image annotation they produce (bounding box annotation, point annotation, polygon annotation, etc.). For this work, a bounding box annotation tool was needed,

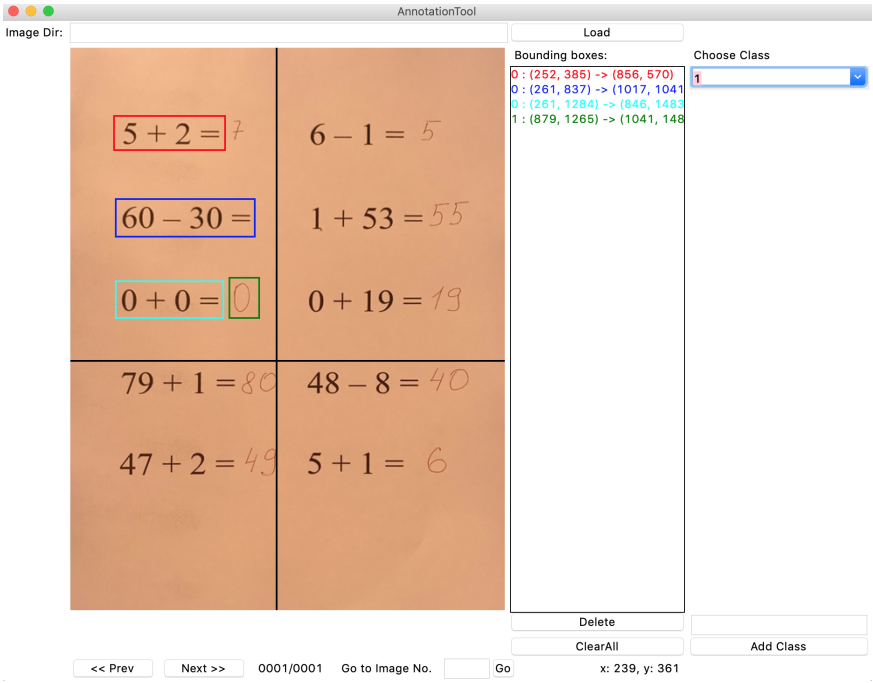


Figure 4.3: Annotation Tool interface.

essentially an application that will simplify the process of creating files with correct bounding box coordinates in them. For this work, Annotation Tool [1] was used. It is based on BBox-Label-Tool [2] which is another popular bounding box annotation tool. The first tool modifies the second one in many ways. One of the added functionalities is the YOLO annotation format converter, which was useful in this project.

While working with this annotation tool, some problems were encountered. Therefore, the code of the tool required modifications. The adjustments were done in three instances. First of all, this tool requires that the images to be labelled reside in specifically named folders [21]. Second of all, the application of the project modifies an image (cuts paper from an image). That means that annotation should be made out of a modified image rather than the original. Third of all, this tool was not written to work with large images. Because of that, only part of a large image was shown.

To work with this tool, setting of parameters in *config.py* is required. These parameters include the path to the folder with images to be labelled, the path to the folder where output files will be added to, extensions of the images, and names of the objects' classes. The program is launched by *python main.py*.

The interface of the annotation tool is simple and straightforward (see Figure 4.3). The application will show one image at a time from the selected folder. The user can choose the object's class and then draw a bounding box around the object by dragging the cursor. The annotation file is saved with the

[1] https://github.com/maverickjoy/bounding-box-annotation-tool
[2] https://github.com/puzzledqs/BBox-Label-Tool

same name as the image. If the user goes to the next image or closes the application, the output file is updated.
The annotation file has N lines, where N is the number of objects in the image. The first number in each row is a class of that object. Then there are four numbers that represent the bounding box coordinates. The first two numbers are XY-coordinates of the upper-left corner of the bounding box. The second two are the width and height of the box.

4.1.2 Text annotation

Text annotation does not have coordinates that it bounds to. Therefore, a system was created to indicate different objects and how they are located in relation to each other.
The mathematical equations have numeric answers that are non-negative integers less than 100. It is thus correct to say that there can be at most two handwritten digits together. The annotation file consists of several lines. The number of lines equals to the number of rows of text in the image. Each line contains the same amount of text blocks. This amount is equal to the number of the columns in the image. There are three elements in each block. The first one is always a mathematical equation. The last two are either a handwritten digit or a blank space (see Figure 4.4 for examples of the blocks). In the annotation file, every two consecutive objects are separated by the '|' symbol, and each row starts on a new line (see Figure 4.5).

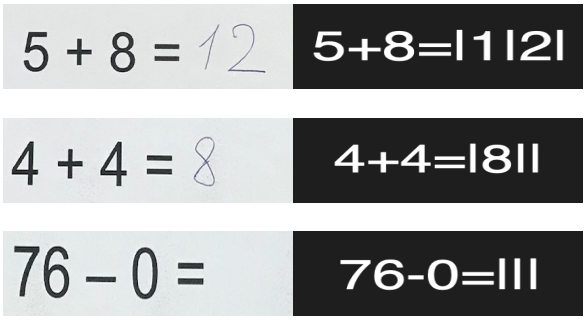


Figure 4.4: Examples of image blocks and their text block analogs: 1. block with two handwritten digits; 2. block with one handwritten digit; 3. block without handwritten digits.

The number of blocks per a line is the same within the whole annotation file. Because of that, it is possible to create a matrix of the size $M \times N$, where N equals the number of blocks on the line multiplied by three and M is the number of lines. This matrix can contain all of the objects of the image (see Table 4.1 for an example of a matrix).

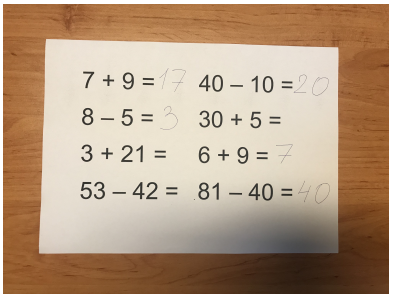


Figure 4.5: Image and its text annotation file.

7 + 9=	1	7	40 - 10=	2	0
8 - 5=	3		30 + 5=		
3 + 21=			6 + 9=	7	
53 - 42=			81 - 40=	4	0

Table 4.1: Matrix of annotated objects of the image in Figure 4.5 .

4.2 Processing recognised objects

After the image went through YOLO recognition, there is a list of recognised objects. The recognised objects in the list are not sorted in the order they appear in the processed image. It is a disadvantage, because the order is needed for displaying the objects by the application. The objects should be shown in the application as they are located in the image. That means that objects need to be sorted by rows and columns.
Firstly, the recognised objects are divided by rows using their y-coordinates coordination. If two objects' y-coordinates are close enough to each other, then they are on the same row. The closeness of the elements is decided by the constant variable, the value of which was determined while testing the application.
After that lines are sorted in ascending order. Lastly, elements in each line are sorted by their x-coordinates. Also, some bounding boxes can overlap (e.g., the bounding box of a mathematical equation can overlap with the bounding box of a handwritten digit).
There is a complication with making a matrix out of recognised objects. YOLO and Tesseract are not perfect algorithms. Because of that, some objects will not be recognised. That means that some of the created lines will be shorter or longer than the others. It was previously established that there are up to two handwritten digits after a mathematical expression. If a line has two mathematical expressions that follow each other, it will mean that two handwritten digits are missing. If there are three handwritten digits in a row, that will mean that a mathematical equation is missing. It also can just be the user writing three handwritten digits in a row. Because of that, the x-coordinates of these objects should be compared. The placement of that equation can be found by comparing the x-coordinates of found objects.

An empty object is placed in the matrix at the place where a mathematical equation or handwritten digit should be.
The resulted matrix resembles the matrix processed from text annotation. Because of that, an evaluation is possible.

4.3 Results of evaluation

4.3.1 Probability of recognition

The created dataset was used for evaluation. The resulted percentage is following. The probability that the bounding box will be recognised correctly is 88%, the probability that a mathematical equation will be recognised correctly is 78%, probability of recognising handwritten digits correctly is 59%. The overall probability of correct prediction of an object is 65%. Keep in mind that to recognise mathematical equations and handwritten digits correctly, at first, the bounding box should be recognised correctly. And only then, Tesseract or CNN algorithms are used.

How to make it better? Some results can be worse than others due to lighting, camera position, or quality of the photograph. The outcome of the recognition can be improved by combining results from multiple photographs of the same paper. Because of that, a missed recognition in one image will not affect overall results.
New calculated probabilities are following. The probability that a mathematical equation will be recognised correctly is 88%, the probability of recognising handwritten digits correctly is 68%. The overall probability of correct prediction of an object is 75%.
As it can be seen, the probability of correct prediction in all cases went up by 10% (see results in Figure 4.6). These results were achieved by combining results from 5 images.

4.3.2 Count of incorrectly recognised objects

The previously described evaluation looks at all of the objects from all of the test images at the same time. This type of evaluation does not give information about separate images. Therefore, another evaluation should be done.
Another way to evaluate results is to count all of the mistakes made while recognising objects. There are multiple mistakes that recognition can make: recognising bounding box wrongly (the box is too big/small or placed where there is no object), not recognising objects (there is no bounding box around the object), mistaken a symbol between numbers in an expression, mixing up one or two numbers in the mathematical equation, mistaken a handwritten digit. Counting all of these mistakes and their combinations in all testing files will help to evaluate results.
Figure 4.6 shows for each mistake its percentage as well as its total number

in the processed images. As can be seen on the figure, most of the text recognised object mistakes are with handwritten digits. Concerning the printed mathematical equation, the most problematic is to recognise one number or numbers of the equation. The latter is fixed by merging the results.

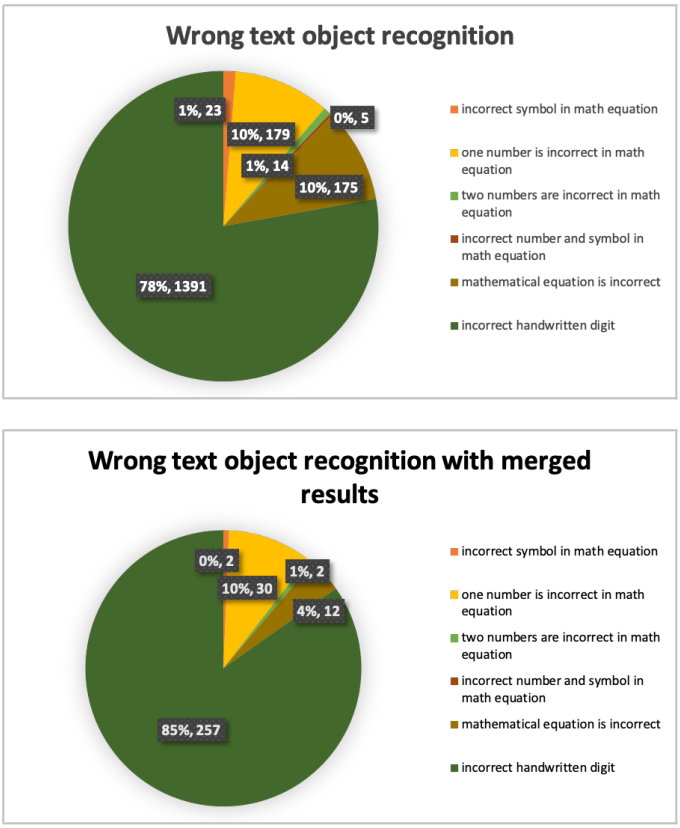


Figure 4.6: Counts of incorrectly recognised objects.

To evaluate the results, a decision was made to create a rating system that expresses how well/bad the application has recognised objects in a given image. Each mistake costs a specified number of points. The amount of points for mistakes is different due to their severity. For example, if a mistake was made in recognising the bounding box of the image, it would affect the latter recognition of handwritten digits or mathematical equations. Because of that, the mistake of wrongly recognising the bounding box will cost more than a mistake of wrongly recognising one number in a mathematical equation. A score for an image is obtained by summuning particular points. The points are determined by following rules. Not localising the bounding box correctly costs 5 points. Not detecting the handwritten digit correctly costs 3 points. If the application recognises a sign in the mathematical expression incorrectly, it costs 1 point. Not recognising one of the numbers costs 2 points. Not recognising one number and a symbol in the math equation costs 3 points. Not recognising the whole mathematical equation costs 4 points. Not recognising

two numbers in a mathematical equation costs 3 points. The points will be counted for each image.
Different images have a different number of objects in them. That means that just dividing images by points is inaccurate. The sum of points for each image will be divided by the number of objects in the image. Figure 4.7 does not provide comprehendible information about overall success or failure of image recognition. Because of that, the graph was divided into four parts. The first category includes images with the calculated number less than 1. The second category includes images with the calculated number greater than 1, but less than 2. The third category includes images with the calculated number greater than 2, but less than 3. And the last category includes the rest of the images. The number of objects in these parts was counted. Counts of images in the categories are depicted in Figure 4.8.

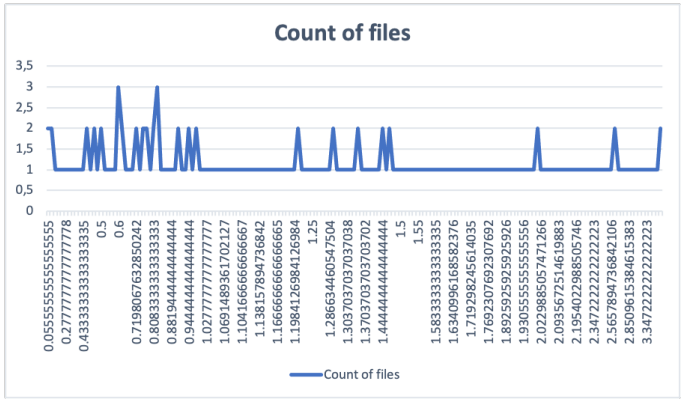


Figure 4.7: Count of incorrectly recognised objects.

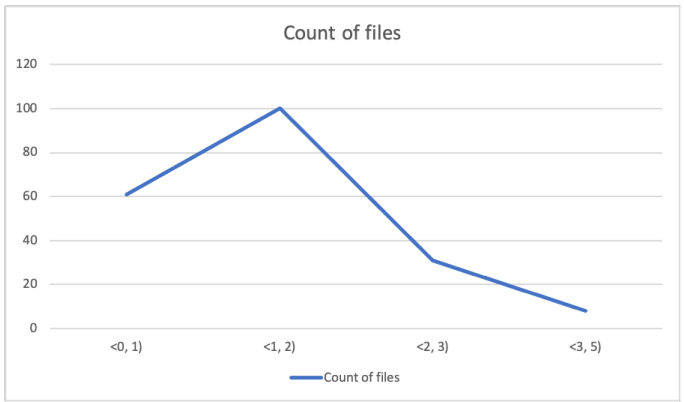


Figure 4.8: Count of incorrectly recognised objects.

The most numerous category is <1, 2) category. It means that each object of the image from this category costs around 1 point. The second largest category is <0, 1) category. This category means that each object of the image from this category costs less than 1 point. It would be best if most of the images are in the <0, 1) category.

Chapter 5

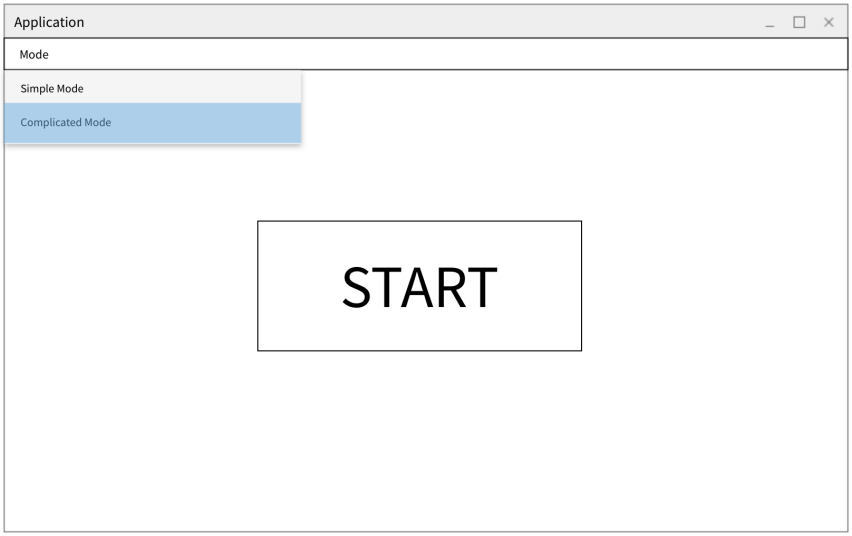
Application

This project requires an application that allows a user to use the functionality of written algorithms. The application will allow user to check the correctness of their answers from the image that the user will input. Since the Python language was used to implement the dataset generator, it was decided to write this application in Python as well.

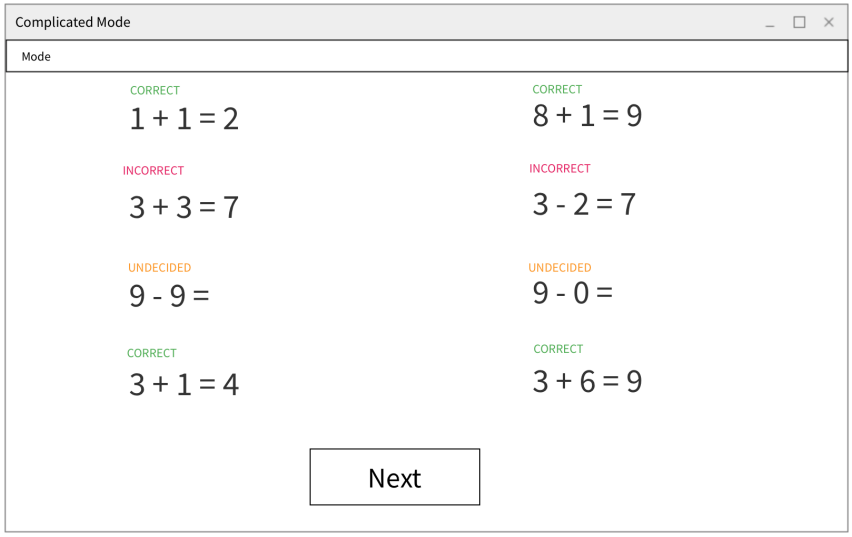
5.1 Application concept

The system can be implemented in an application on multiple platforms. The application can be mobile. In that case, the camera on the phone is used for capturing the current state of the paper. Another way is to implement a desktop application that will take an image from a web camera, or the image will be sent from a phone. Only one kind of application will be implemented to simplify work.
Before the implementation of the application, it is a good practice to create a concept of this application. The interface of the application as well as the desired use case scenario will be described here.
The description of the usage of the application is following. This application is a desktop application. The idea is that the user will sit in front of the computer. They will photograph the current state of the problems on paper. This photograph will be sent to the program, where it will be processed. The application will react based on the recognition results. The user can then send another picture for processing. Recognition results of all of the following photographs will be merged to gain better recognition results.
The application has two modes. The first one is a simple model that will only react to the accuracy of the user's answers by showing a face that will smile for good results and frown for bad ones. The second one is a more complicated mode that will show all of the recognised equations and their answers. It will react by writing a response next to the mathematical equation.
The application will have four different windows. The first one is the main window that has a button called "Start". If the user presses this button, the recognition of the first image will start. In this window, the user can change the modes of the application. Another one is the final window. It shows the results and a picture of a face that will smile widely if the results are

satisfactory. The face will have a smaller smile if the user makes a lot of mistakes.
Both of the other windows show a reaction to the results based on the mode the application is in. They have a button called "Next". If the user presses this button, the application will process the next image. If there are no more images to process, the application will write overall results (see Figure 5.1 for the windows concepts). The concept images were created by using MockFlow tool [1].



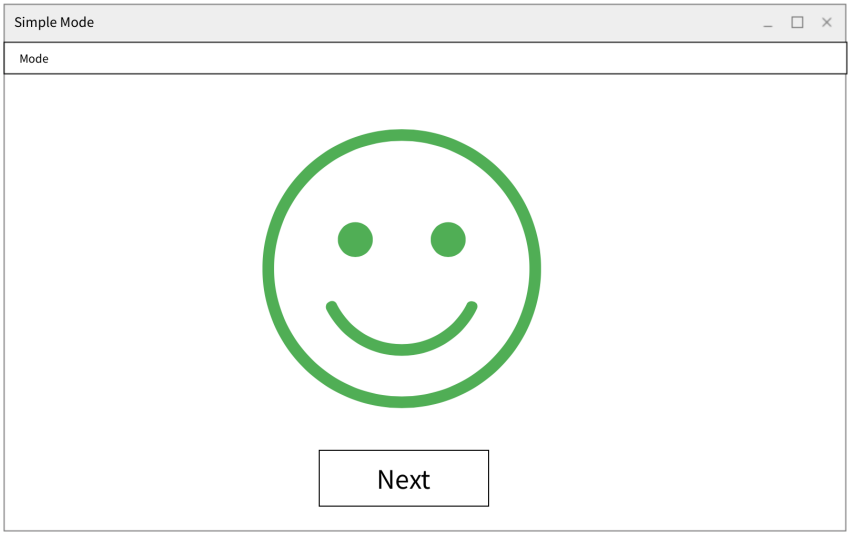
(a) : the main window



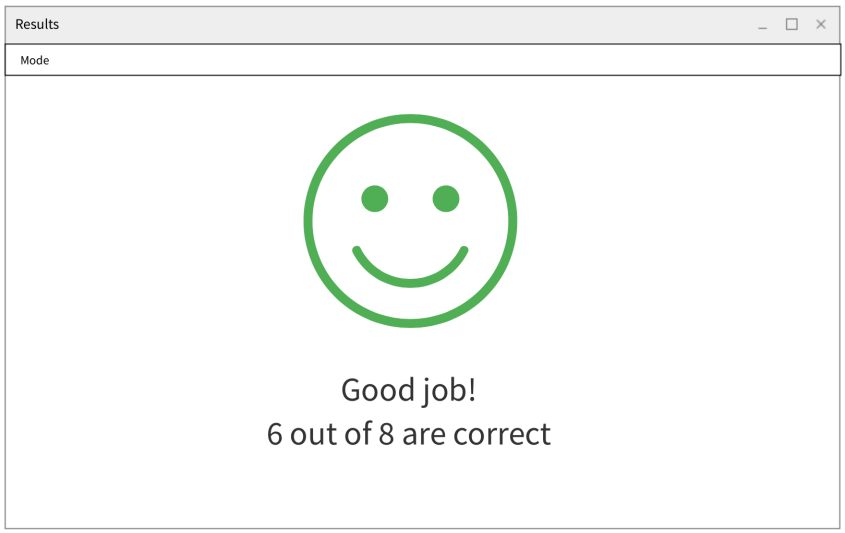
(b) : the widow with the detailed results

Figure 5.1: Concept of the application windows.

[1] https://www.mockflow.com



(c) : the window with the simple results



(d) : the final window

Figure 5.1: Concept of the application windows.

5.2 The application creation

There are a couple of ways to use YOLO algorithms in an application written in Python. The first one is to write a bash script that will activate the YOLO recognition algorithm through the terminal. That script will save the results of YOLO recognition to a file with a JSON extension. This file will be then processed in the application. This way is complicated and slow, especially while trying to run multiple photos through recognition. Because of that, another solution is used in this project. The second solution is to write an

application with the help of the functions from OpenCV. OpenCV can start a YOLO algorithm. Files such as *yolov3.weights*, *obj.data* and *yolov3.cfg* are needed for the OpenCV computation.

5.2.1 OpenCV

The solution for creating the application with OpenCV is based on [2] code. The description of the functions in OpenCV that are used in this application are following.
First, function

```
cv2.dnn.readNetFromDarknet(config_path, weights_path)
```

is used to load weights from the model. *config_path* is a variable that contains path to the file *yolov3.cfg*, *weights_path* is a variable that contains path to the *yolov3.weights* file. The result of this function is saved to the variable named *net*. Then function

```
cv2.dnn.blobFromImage(image, scalefactor, size, swapRB, crop)
```

"creates 4-dimensional blob from image" [22]. *image* is an input image that requires preprocessing for latter recognition. *scalefactor* is a factor by which the image is scaled by. In this project *"it scales the image pixel values to a target range of 0 to 1 using a scale factor of 1/255"* [23], *size* is a desired size for an output image. *swapRB* is a "flag which indicates that swap first and last channels in 3-channel image is necessary". *crop* is a "flag which indicates whether image will be cropped after resize or not".
The next part is to run the function

```
net.setInput(blob)
```

that sets the blob (that was created from the second function) to the network (that was created from the first function).
Then, with the help of the function

```
net.forward(layer_names)
```

the predicted objects are obtained. *layer_names* is the list with the names of the possible recognised classes. The output from this function contains bounding boxes, class ids, and confidences of a prediction. The latter will help to remove wrongly predicted objects. They are the ones with confidences of the prediction smaller than 10%.

5.2.2 Problems of the application

Some problems were encountered during the process of writing an application with OpenCV. The first problem is related to the bounding box recognition. The algorithm returns multiple bounding boxes around one object on the

[2] https://gilberttanner.com/blog/yolo-object-detection-with-opencv

image. This problem was solved with the help of the Non-maximum Suppression (NMS) algorithm. NMS uses prediction confidences of objects and coordinates of their bounding boxes. Intersection over Union (IoU) is needed to be calculated for the NMS algorithm. It is a value that is determined by dividing the intersection area of two bounding boxes by the union area of two bounding boxes (see Figure 5.2) [24].

Figure 5.2: Example of intersection and union of two bounding boxes.

IoU is compared to the overlap threshold. An overlap threshold is another value that is needed to use NMS. It tells NMS how big overlap of the bounding boxes is acceptable. NMS algorithm travels from the object with the highest prediction confidence to the object with the smallest prediction confidence. Each bounding box of an object is compared to bounding boxes of the objects that have already gone through the algorithm and were excepted. That way, only objects with the highest prediction confidence will be excepted.
NMS is implemented in OpenCV function:

```
cv2.dnn.NMSBoxes(boxes, confidences, confidence, threshold).
```

The second problem was that the application with OpenCV YOLOv3 recognised objects worse than the darknet YOLOv3 in the terminal. This problem was solved by changing the value of accepted confidence and overlap threshold to a smaller number.

5.2.3 Contour of an image

The photograph portrays a picture of a paper that lies on the table. Some additional processes needed to be performed to gain a better recognition result. It was decided to crop the paper from the image. It was done by implementing the method [3] of scanning a paper. The OpenCV library is used in this method. The goal is to find a contour of the images. That means the image has to have 4 straight lines that close an area of the required size (it is assumed that the paper will take more than one-fourth of the photograph area). The process of getting the contour is following. First, convert the image into the greyscale by using function

```
grey_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY).
```

Then functions

[3] https://bretahajek.com/2017/01/scanning-documents-photos-opencv/

```
cv2.bilateralFilter(image, d, sigmaColor, sigmaSpace)
cv2.adaptiveThreshold(image, maxValue, adaptiveMethod,
 thresholdType, blockSize, C)
cv2.medianBlur(image, ksize)
```

will blur the image and then distinguish the darkest pixels to make the edges of the paper more visible. Then

```
grey_image = cv2.copyMakeBorder(grey_image, 5, 5, 5, 5,
 cv2.BORDER_CONSTANT, value=[0, 0, 0])
```

is used in case the page is touching an image border. This function adds black border to the image. Then edges are find by

```
edges = cv2.Canny(img, 200, 250)
```

function. After that, the program will try to find the contour that will have 4 lines connecting to each other. All of the values in these functions were calculated on test data to gain better results.

5.2.4 Frontend of the application

The frontend of the application was also written in Python. Multiple libraries implement Graphical User Interface (GUI) in Python. The library choice for this project was made between Tkinter and PyQt5 libraries. They are the most popular GUI toolkits for Python. Library Tkinter is one of the oldest GUI toolkits for Python. It comes with Python, so it is not needed to be separately installed. Tkinter is easy to use and easy to understand library, so it is commonly used. Because of that, it is fairly simple to find any information about this GUI toolkit. Unfortunately, it has disadvantages such as old looking interface and lack of advanced widgets. On the other hand, PyQt5 has a modern-looking interface, many advanced widgets. But this library has a smaller number of tutorials that are written about it.
This application is supposed to appeal to younger users. Therefore, the main choice criteria between these two libraries was appearance. Because of that, PyQt5 was chosen.

5.2.5 Usage of the application

The application was created for this project. It will not be sold for profit or used elsewhere. Because of that, the application does not have an icon that can be pressed for the start. The application is started in the terminal by the command

```
python3 start_application.py.
```

The images for the recognition are taken from the specific folder. The path for the folder can be set, if the user starts the program with the argument -p. If the program started without the argument, the application will use the default folder path. The application also requires a copied repository Handwritten-Digit-Recognition-using-Deep-Learning, that is used for recognising handwritten digits.

5.3 Real test

This application is supposed to be used by children. Up to this point, there was only contemplation of what would children like and how they would react. Therefore, it was decided to test the application with a child. Due to the current situation in the world (this part was written in December of 2020), the application could not be tested with many children. The application was tested with only one child aged 10 years.
The process of testing was following. The child was given a paper with nine simple mathematical expressions on it. She wrote her answers with a blue pen. She was instructed on when to take a photograph of her work. There were times when the picture needed to be taken multiple times due to inadequate lighting in the room. Some numbers were recognised incorrectly, so the child was asked to white out these numbers and write them again. After the second try, the numbers were recognised.
She was also asked to make a mistake in the answer so she could see what the program would say to this. The wrong number was whited out and the new number was written in its place (see Figure 5.3 for this).

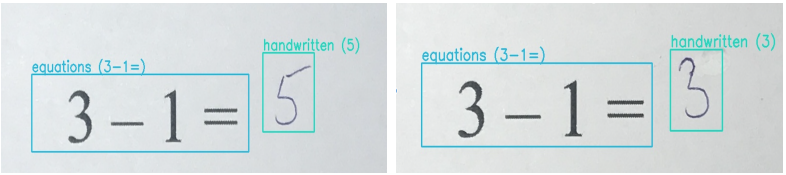


Figure 5.3: Examples of whited out number

The child enjoyed the experience. She said that "it was fun watching the numbers change on the screen". She also liked the design of the application. The motivational text that is printed along with the recognised mathematical equations was also praised.
There were some things that the child was not satisfied with. One of them is the amount of time it takes to take the picture, send it, and then wait for the results. It was also frustrating for her when her correct answer was marked as incorrect. The application was working that way due to the mistake in the recognition.
But overall, she was satisfied with the experience of using the application.

Chapter 6
Conclusion

The project was divided into three parts: implementing localisation on the image, implementing detection of the objects, and developing an application to use the created system. The results of localisation on the generated data are exceptional. Unfortunately, the results on real data are slightly worse. It could be because of the generated dataset. The dataset could be more diverse.

Thanks to the performed evaluation, the accuracy of the system can be judged. Accuracy percentage for the evaluation dataset is not ideal. Most of the mistakes are incorrectly recognised handwritten digits, which could be due to a wrongly recognised bounding box. CNN was trained upon the MNIST dataset. And although the image given to the CNN algorithm was converted to look similar to MNIST dataset images, it is still different. Therefore, it could be the reason why the greatest number of mistakes are the number of incorrectly recognised handwritten digits.

Overall, the goal of the project was met. The system that automatically checks the correctness of completed and partially completed mathematical tests on a raster image was written. It works good enough for a child to solve a test alone.

6.1 Suggestions on improvement

The localisation of objects can be improved by training the dataset on one of the newer versions of the YOLO system. They are superior to YOLOv3. Unfortunately, they were released after the localisation part of this project was done. It was decided to move on to improving the detection of the object rather than focusing on improving localisation.
Another thing to improve is to write an application for a mobile phone. That way, the user would instantly see the results without spending time sending a picture from the phone to the computer. Another way is to improve the system, so the user could use a web camera on the computer. The application also could be more interactive. It could give the user an answer after their multiple unsuccessful attempts at solving a mathematical equation.

Literature

[1] Maher. *6 Significant Computer Vision Problems Solved By ML*. Nov. 2020. URL: https://heartbeat.fritz.ai/6-significant-computer-vision-problems-solved-by-ml-623eb50544c5.

[2] Wikipedia Contributors. *Optical character recognition*. Dec. 2020. URL: https://en.wikipedia.org/wiki/Optical_character_recognition.

[3] *Folded Paper Textures*. Feb. 2020. URL: https://indieground.net/product/folded-paper-textures/.

[4] *Wrinkled paper*. URL: https://www.123rf.com/photo_60605247_wrinkled-paper.html.

[5] NordWood Themes. *White wall paint with black line photo*. Oct. 2020. URL: https://unsplash.com/photos/R53t-Tg6J4c.

[6] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. *THE MNIST DATABASE*. URL: http://yann.lecun.com/exdb/mnist/.

[7] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. May 2016. URL: https://arxiv.org/abs/1506.02640.

[8] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. Dec. 2016. URL: https://arxiv.org/abs/1612.08242.

[9] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. Apr. 2018. URL: https://arxiv.org/abs/1804.02767.

[10] Venkata Krishna Jonnalagadda. *Object Detection YOLO v1 , v2, v3*. Jan. 2019. URL: https://medium.com/@venkatakrishna.jonnalagadda/object-detection-yolo-v1-v2-v3-c3d5eca2312a.

[11] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. Apr. 2020. URL: https://arxiv.org/abs/2004.10934.

[12] Glenn Jocher et al. *ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements*. Version v3.1. Oct. 2020. DOI: 10.5281/zenodo.4154370. URL: https://doi.org/10.5281/zenodo.4154370.

[13] Xiang Long et al. *PP-YOLO: An Effective and Efficient Implementation of Object Detector*. Aug. 2020. URL: https://arxiv.org/abs/2007.12099.

[14] Anuj Dutt. *Handwritten Digit Recognition using Deep Learning*. 2017. URL: https://github.com/anujdutt9/Handwritten-Digit-Recognition-using-Deep-Learning.

[15] Anuj Dutt and Aashi Dutt. *Handwritten Digit Recognition Using Deep Learning*. July 2017. URL: http://ijarcet.org/?s=990-997.

[16] Google. *tesseract-ocr/tesseract*. 2008. URL: https://github.com/tesseract-ocr/tesseract/.

[17] Luc Vincent. Aug. 2006. URL: http://googlecode.blogspot.com/2006/08/announcing-tesseract-ocr.html.

[18] Filip Zelic. *[Tutorial] OCR in Python with Tesseract, OpenCV and Pytesseract*. Dec. 2020. URL: https://nanonets.com/blog/ocr-with-tesseract/.

[19] Apple. *iPhone - Compare Models*. URL: https://www.apple.com/iphone/compare/?device1=iphone7plus.

[20] Jiayin Low. *What is Image Annotation?* Jan. 2020. URL: https://medium.com/supahands-techblog/what-is-image-annotation-caf4107601b7.

[21] Shi Qiu. *puzzledqs/BBox-Label-Tool*. 2017. URL: https://github.com/puzzledqs/BBox-Label-Tool.

[22] *Deep Neural Network module*. URL: https://docs.opencv.org/master/d6/d0f/group__dnn.html.

[23] Sunita Nayak. *Deep Learning based Object Detection using YOLOv3 with OpenCV (Python / C++): Learn OpenCV*. June 2020. URL: https://www.learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python-c/.

[24] K Sambasivarao. *Non-maximum Suppression (NMS)*. Oct. 2019. URL: https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c.