



# Automatic School Handwriting Detection and Classification based on YOLO and Vision Transformers models

**Aico Schreurs**

Master Thesis

Date: July 2021

Supervisors:
Arno Siebes - Utrecht University
Tjeerd Hans Terpstra - Cito
Laura Kolbe - Cito

MSc Applied Data Science
Utrecht University

**Abstract**

The last decade has marked a rapid and significant growth of deep learning networks. The biggest challenge lies in the identification and handling of object recognition. The aim of this research is to investigate whether it is possible to identify handwritten objects and classify them in a machine readable language. There are two models used in this thesis.

You Only Look Once is built in Python and trained on a labeled dataset of 120 images from various handwritten objects. Training results show that the mAP value is close to one, and a maximum validation accuracy of 99% is achieved. To decrease overfitting and improve a more reliability accuracy, a larger training and validation dataset is needed. The second model is a Vision Transformers model and is trained separably on the MNIST dataset and custom labeled dataset. The highest testing results is provided by the MNIST dataset, which showed an accuracy of 99.5% on the testing set and an accuracy of 96.0% on the data that is provided by Cito. Although this accuracy is not even close to the human accuracy, it is up to Cito to decide how low the error rate should be and which confidence score and probability value is enough to implement this in practice.

# Contents

# 1 Introduction

Object detection or Pattern recognition for objects are dramatically changing over the last few years. Where Deep Neural Networks (DNNs) have recently shown outstanding performance on the task of whole image classification are these tools becoming more and more popular over the years [33]. While humans can in fact easily recognize objects in digital images or in the real world. The human visual system is quick and precise, allowing us to do complicated activities like driving with minimal conscious effort. Fast, accurate object detection algorithms would enable computers to drive cars without specialized sensors, assistive devices to communicate real-time scene information to human users, and unlock the potential for general-purpose [12].

In this research the main focus is on recognize handwritten objects in a so called "Rekenpeiltoets", which is a elementary school test for children. It is a traditional paper-based test which is created and manually grated by "Centraal Instituut voor Toetsontwikkeling" (Cito) [7]. This is a Dutch organization for developing and administering exams and tests. The aim of the measurement and tracking methods is an objective picture of knowledge, skills and competences. There are both multiple-choice and short-answer questions on the test, which need the result of a quick computation. A diagram, a schematic sketch, or a graph accompany the majority of the questions. According to Cito it takes a lot of time to manually grade these questions. Hence they are wondering which models can perform these task automatically, which certainly they can get of the position located on the answer sheet and lastly how can they interpret the handwriting objects.

In digital image processing and machine learning, handwriting recognition is a crucial challenge. The design and operation of systems that can recognize patterns in data are explored in this study field. A number of approaches have been offered in the literature, however the problem remains unsolved [3]. Another issue is that handwriting text is difficult for a machine to understand. Aside from the fact that each person's written characters are mainly distinct, some characters have a very similar form, unconnected or distorted characters, the written characters have a varied thickness, and they use different scanners.

Feature extraction and classification are commonly used to recognize patterns or objects. The feature extraction process often employs a number of approaches to obtain a representation of the data, which is subsequently classified using the classifier. The procedure is carried out manually and in a different manner. Recently, feature extraction and categorization have been combined into a single process or approach [27]. A approach for modeling high-level abstractions in data are Deep Learning techniques.

We aim to solve the problem of identifying handwritten text and categorizing it as computer readable text in this study. Handwriting recognition has been

the subject of several research. SVM and ANN were compared to the achieved accuracy rates in Dağdeviren [8] handwritten number recognition research with the Modified National Institute of Standards and Technology (MNIST) data set. On his MATLAB testing, he achieved a success rate of 99.97% in a data set of 10,000 data points for SVM. In a data set of 10,000 data points, he achieved an 80.39% success rate in the handwriting recognition system he created using ANN. Perwej and Chaturvedi [22] utilized the correlation approach to recognize handwritten characters and then used the KNN algorithm to improve his systems. Text samples from 172 different persons were used to build the data collection. Recognition rates of 93% in numerals, 90.4% in lowercase characters, and 91.2% in capital letters were attained in this investigation. In the case of words and numerals written together, these rates fell by 10%. In text with a combination of characters and numbers, 100% success was attained. The recognition method has also been enhanced by the use of a dictionary. Assegie [1] used a Decision Tree technique to create a digit recognition model on a data set of 42,000 rows and 720 columns. The success rate was 83.4%.

The purpose of this research is to presenting technological models with good performance of classifying handwriting objects in digital images. Firstly we started with detecting and extracting handwriting text in images, this is done with You Only Look Once (YOLO) [31] model which is very popular among classifying objects in a image or video. Secondly we used Vision Transformers (ViT) [29] combined with TrOCR to classify the handwriting digits into real numbers. The ViT model is trained on two different databases described in Chapter 3.

Based on the topic presented by Cito, the research question is as follows.

*Can we build a training model that can detect and recognize handwritten objects in a school test and is this approach reliable enough to implement in practice?*

This topic is researched in a team of three students, each with their own focus. However, there is overlap between the topics and the preprocessing steps which is seen in the next chapter.

# 2 Project outline

Mentioned in the introduction there is a overlap between the topics and the preprocessing steps by different student members. The project flow in Figure 1 gives a quick overview which steps are within the scope of this thesis.

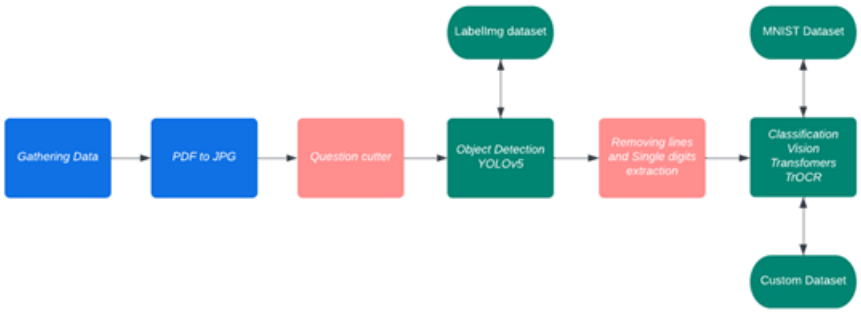


Figure 1: Processing steps

Step 1: Gather Data
Step 2: PDF to JPG extraction
Step 3: Question cutter [28]
Step 4: Object Detection YOLOv5 with sub step labeling images with LabelImg
Step 5: Removing horizontal lines and extracting single digits [14]
Step 6: Classification with ViT TrOCR and sub step creating custom dataset

The blue rectangles steps are the preprocessing steps that we did as a group. The green rectangles steps are the individual preprocessing and modelling steps, that are discussed in this thesis. The red rectangles steps are outside this project scope and will be referred to.

# 3 Data

## 3.1 Data Collection

The Cito 'Rekenpeiltoets' is a traditional paper-based arithmetic test and contains short-answers in the form of numbers and multiple-choice questions. The data exists in total of 73 tests, where each test contains a total of 14 pages with answers. The tests are scanned using a scanning device and uploaded by Cito. Each test is filled in by different children with a pen or pencil. The short-answers exists of a small number, mostly two or three digits long and are mostly written above a horizontal black line. The multiple-choice questions exists of four options namely, A, B, C or D that can be circled. There is no strict condition that the children must adhere, so answering the questions is quite dynamic.

This means that children can also write answers next to the horizontal line or circles the entire answer instead of the characters.

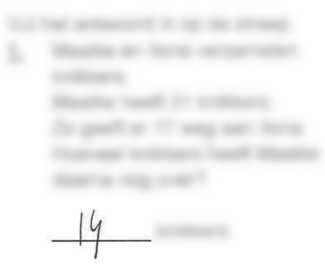

Figure 2: Short-answer question

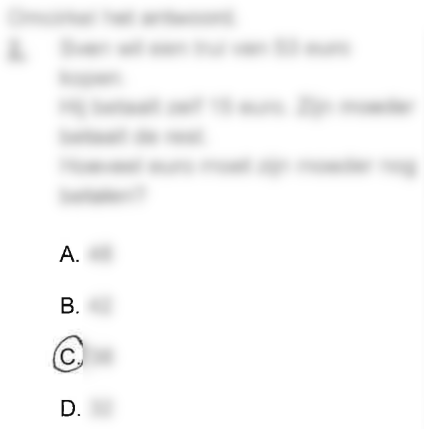


Figure 3: Multiple-choice question

The total number of short-answer questions are 20 and the total number of multiple-choice questions are 10. One example of a question is seen in Figure 2 and 3. Because of the data integrity the digital printed text is blurred out. The left part is an example of a short-answer question which is a handwritten number, the right part is an example of a multiple-choice answer which is circled.

## 3.2 Data Processing

Since it is an image recognition problem, the data already looks very clean. Therefore not many preprocessing steps need to be performed, as seen in Figure 1. The first step is to convert the scanned tests that has a pdf file extension into a usable image file, a jpg extension. The second preprocess step exists of separating each question from the tests, so that each question is one image, this step is performed by Tissings [28] and is outside this thesis scope. The last preprocess step falls between the two models, which exists of removing the horizontal lines and extracting the single digits, this is performed by Klopper [**9**].

## 3.3 Data for ViT Classification Tuning Training

Two different datasets are used for training the Vision Transformers (ViT) TrORC model. The first dataset is Mixed National Institute of standards and Technology (MNIST) and the second dataset is a custom created labelled handwritten digits dataset.

### 3.3.1 MNIST dataset

The MNIST dataset has 70,000 images of handwritten digits, with one class per digit, for a total of ten classes [18]. The numbers were written by 250 different authors, ensuring that the test sets digits were written by different people. Images in this database have a size of 28x28 pixels and contain grayscale intensity values expressed by 8 bits per pixel, yielding 256 intensity levels. The MNIST dataset was chosen because of its accessibility and prominence as the most widely used benchmark dataset for handwritten digits.

```
MNIST:
Patterns    Shape                Range
inputs      (28, 28, 1)          (0.0, 1.0)
targets     (10,)                (0.0, 1.0)
Total patterns: 70000
   Training patterns: 70000
   Testing patterns: 0
```

Figure 4: Raw summary MNIST

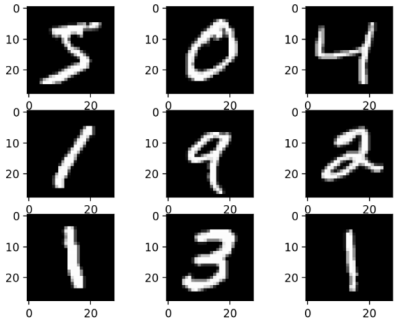

Figure 5: MNIST sample digits

A raw summary of the MNIST dataset is seen in Figure 4 and an example of a handwritten digit from zero to nine is seen in Figure 5

### 3.3.2 Custom label dataset

The second dataset is a created custom label dataset, which contains around 2,000 handwritten digits. Digits in these set were written by 10 different students from a secondary school with ages between 12-14 [26]. Every student filled in around 5-10 different digit numbers from 0-9 on a A4 paper. These filled in forms are scanned and the single digits were extracted with an open source tool in Python. This custom label dataset is created because the MNIST dataset lacks of children handwritten digits.

### 3.4 Preprocessing Custom dataset

The digit numbers are filled in a vertical ranked order from zero to nine, and on the horizontal line the same digit that belongs to the ranked digit, so that it is easily to extract the digits into one single digit. An example of one of the digit is seen below in Figure 6.

Figure 6: Extract single ranked digit

To extract every digit to a single image or array a third party Open Source Computer Vision Library (OpenCV) [20] software tool is used. OpenCV was created to offer a standard infrastructure for computer vision applications and to let commercial products integrate machine perception more quickly. The reason for choosing this over manually extracting the digits is that it is easy to use and it saves a lot of time.

To detect single digits the built in functionality of OpenCV for finding contours is used. Contours are just a curve that connects all of the continuous points (along the border) that have the same color or intensity. The contours are an effective instrument for item detection and recognition as well as shape analysis.

So the first part is to define the region of interests (ROI) of the image. This is done by the functionality "findContours" where OpenCV extract the bounding box coordinates points (x, y, w, h). One of the problems we encounter is that some of the bounding box coordinates are overlapping, especially when the handwritten digits are not fully connected within the digit itself. So for instance the first eight in Figure 6, the line right in top is not connected to the middle of the number. OpenCV could see this as a separate bounding boxes, which is seen in Figure 7 below.

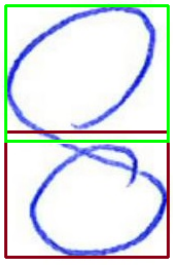

Figure 7: Overlapping bounding boxes

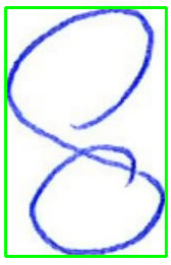

Figure 8: Union bounding boxes

So to minimalise this behaviour we first calculate the intersect, and if the value is $> 0$, which means there is an overlap, the union will be calculated to merge the two bounding boxes together. This is seen in Figure 8 above.

The intersect is calculate as followed:

$$dx = min(a.x + a.w, b.x + a.w) - max(a.x, b.x) \tag{1}$$

$$dy = min(a.y + a.h, b.y + b.h) - max(a.y, b.y) \tag{2}$$

if (dy and dx) $> 0$ then there is overlap between bounding box a and b.

The union is calculate as followed:

$$x = min(a.x, b.x) \tag{3}$$

$$y = min(a.y, b.y) \tag{4}$$

$$w = max(a.x + a.w, b.x + b.w) - x \tag{5}$$

$$h = max(a.y + a.h, b.y + b.h) - y \tag{6}$$

Which returns the new bounding box coordinates (x, y, w, h).

Then the union rectangle of the overlapping bounding boxes will return a list of bounding boxes. For every bounding box in the list the position will be extracted from the original image in Figure 6, resized to 28x28 pixels, reverted and then exported to an image file.

Figure 9: Digit export example

The total number of handwritten digits in the custom dataset is 2,475.

| Digit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Count | 319 | 300 | 268 | 265 | 256 | 254 | 237 | 230 | 190 | 156 | 2,475 |

# 4 Methods and Results

To answer the research question: "Can we build a training model that can detect and recognize handwritten objects in a school test and is this approach reliable enough to implement in practice?". We translate this question into a data science question.

*To what extent can the **detection model locate** and the **classification model interpret** the handwritten given answers on the answer forms?*

In this research two methods are used to answer the data science question. The locate subject will be answered with an object detection method YOLO, this algorithm makes it possible to detect single or multiple objects of handwritten objects. From here, this model extracts the cropped images into a useful image file format. The interpret subject is answered by using a Vision Transformer tool that is based of TrOCR and allows to interpret the handwritten digits into machine readable number.

All tests within the scope of this study are performed with Jupyter Notebook [13], which is an integrated development environment used in programming, specially for the Python language on the following machine: Intel CPU i5 9600K, Geforce NVIDIA GTX 1080 TI and 16GB RAM.

## 4.1 Method - YOLOv5

### 4.1.1 Motivation choice of YOLO model

Redmon et al. [24] invented as first the term "YOLO." Until then, Region-based Convolutional Neural Networks (R-CNN) models were the most used object detection models. In comparison to other models, YOLO was picked for its real-time accuracy and open source public availability. This technique is suitable for human detection, such as detecting handwritten digit objects [11].

YOLO, in contrast to other state-of-the-art object recognition technologies like Fast R-CNN or CNN, considers the whole image rather than just the region of interest. As a result of this there will be fewer background errors. Fast R-CNN is considered as one of the most effective object identification techniques [24]. Although these models are very accurate, it is slow since it required a multi-step technique to find the appropriate bounding box region, categorize these regions, and then refine the output using post-processing [19].

### 4.1.2 Architecture of YOLO

According to the previous mentioned studies, YOLO remains a top choice algorithm for object detection due to its rapid speed and average precision, as well as its open-source availability.

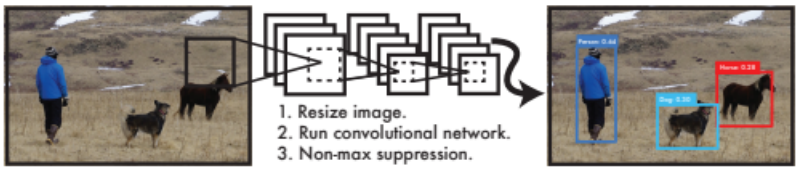


Figure 10: YOLO detection system [31]

Figure 10 shows how YOLO is delightfully easy. Multiple bounding boxes and class probabilities for those boxes are predicted simultaneously by a single neural network. In Figure 11 the following steps below are performed.

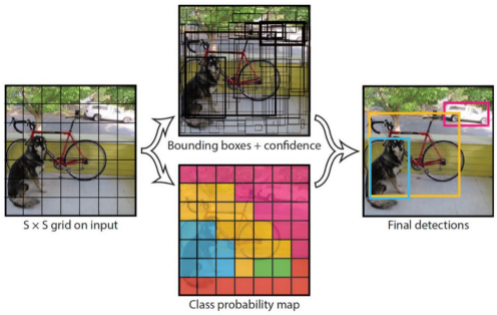


Figure 11: YOLO model [31]

The image is first split into a grid with a S X S dimension. Each cell predicts the number of bounding boxes, which is represented by the variable B, once the image has been divided into a grid. Grid cells with fully enclosed objects are in responsible of detecting the object within. For each box, confidence scores are predicted, showing the model's degree of confidence in detecting if the box truly boundaries an item, as well as in predicting what the object is. The following expression can be used to define the confidence score:

$$Pr(Class_i|Object) \times Pr(Object) \times IoU_{pred}^{truth} = Pr(Class_i) \times IoU_{pred}^{truth} \quad (7)$$

Where;
IoU = intersection over union
truth = the number of ground truth boxes
pred = the number of predicted boxes

Figure 12 and 13 are a visual representation of the intersection over union computation. The ground-truth bounding box entirely encircles the stop sign, but the algorithm's predicted bounding box is somewhat displaced from the true object. The IoU is the area where the two bounding boxes intersect or overlap.

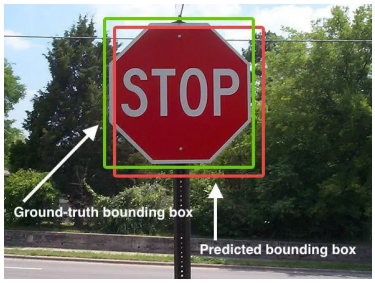


Figure 12: Difference between Ground-truth and Predicted [31]

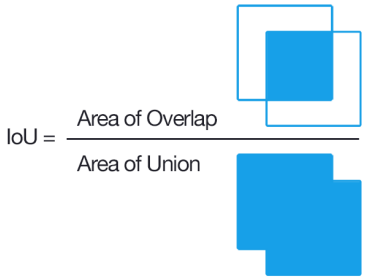


Figure 13: Intersect over Union visual calculation [31]

Grid cells with no objects should have a zero confidence score. The intersection over union between the algorithm predicted bounding box and the ground truth bounding box should be equal to the confidence score for cells with objects. There are five predictions in each bounding box: x, y, w, h, and confidence. The x and y variables reflect the bounding box center coordinates with relation to the cell boundary. The h and w variables represent the bounding box height and width in relation to the entire image, respectively, and the confidence is determined as the intersection of the algorithm predicted bounding box and the ground truth box.

For identifying numerous detections of the same item, Non-max suppression is utilized. This method only takes up the item once. Consider a scenario where the algorithm identified three bounding boxes with corresponding probability for the same object. First, we will choose the box to eliminate the duplicates with the greatest probability and provide that as a prediction. then exclude any enclosing box having an IoU greater than 0.25.

In this research the model version 5 is used. The model architecture is similar then the previous versions but includes some changes like it is written in PyTorch framework so it can easily be accessed in Python. Additional improvements in the YOLOv5 update include augmented data training through a data loader, specifically scaling, color space adjustments and mosaic augmentation [32].

### 4.1.3 Training on a Custom Dataset

Training the YOLO algorithm to a custom dataset can result in higher accuracy and precision for detection [32]. Also there is no publicity available pretrained dataset for detecting handwriting objects, therefore a custom dataset is needed. The first step is to create a custom dataset from the gathering data in Chapter 2.1. The first step is to divide the classification scheme into two different classes, namely:

1. Class number 0 - handwritten
2. Class number 1 - printed

The reason for chosen not only one class is that the algorithm need to know the difference between handwritten and printed text. One example is, if a trained handwritten number is perfect written then the model could think that a printed number is a handwritten number, this could increase the error rate in the model.

The next step is to manually draw these labels in the dataset that is provide by Cito. The tool labelImg is used for several reasons, it is free, open-source, easy to use and the most important one, the data is not uploaded to a third-party company and can even run locally, without the use of internet, on a machine.

### 4.1.4 LabelImg

LabelImg [2] is a open-source graphical annotation tool. It is developed in Python and has a graphical user interface built using Qt. Annotations are stored in text files, which are compatible with YOLO.

The next steps need to be performed to create a custom labelled dataset in an useful format.

1. In "data/predefinedclasses.txt" define the list of classes
2. Build and launch LabelImg
3. Switch to YOLO format by pressing on "Pascal VOC".
4. Click on OpenDir for processing single/multiple images.
5. Create a rectangle box and save the file

An example of a custom label image in LabelImg is seen in Figure 14, where the green rectangles are the printed classes and the yellow rectangles the handwritten classes. When saving this image into a YOLO format (step 3) it will export the raw values of the classes, x, y, w and h values into a text file, see Figure 15.

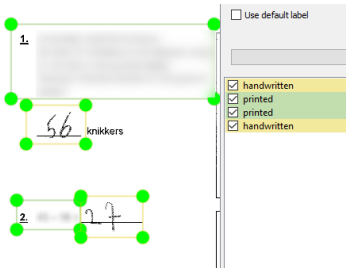


Figure 14: LabelImg Interface

```
1 0.208878 0.248846 0.113449 0.052335
0 0.316884 0.163845 0.387034 0.100564
0 0.194485 0.370703 0.120948 0.039507
1 0.315433 0.373268 0.118045 0.055584
0 0.307813 0.645374 0.365989 0.099367
0 0.303338 0.721225 0.093856 0.024628
1 0.212627 0.716436 0.069666 0.036258
```

Figure 15: LabelImg txt format

The text file format have the following specification:

- One row per object
- Each row is class x-center y-center width height format.
- Box coordinates must be in normalized x, y, w, h format from zero to one
- Class numbers are zero-indexed, which means they start at zero

#### 4.1.5 Preparing the dataset for training and validation

The dataset contains of:

1. Train – 120 question numbers of images with labels
2. Valid – 120 question numbers of images with labels
3. Test – 2,430 question numbers of images without labels

It is very time consuming to label all the data manually, for that reason only a small sample of the training and validation dataset is available.

Before training the model, YOLO need also a configuration file to know where the train/valid/test images are kept and which classes they belong to. It specifies an optional auto-download command, a path to a directory containing training pictures, the same for validation images, the number of classes, and finally a list of class names.

#### 4.1.6 Training and validation phase

When finishing all the previous steps the model can be trained with the following arguments:

- **img**: defines the input image size. The original image is 1024 by 1024, however it has been compressed to a smaller size to speed up training. After numerous tests, researchers [30] came to the conclusion that the size 416 by 416 is the best for input without losing too much information.
- **batch**: determines the batch size. The amount of weights that the model must learn in one period or epoch grows significantly when thousands of images are transmitted into it at once. As a result, the dataset is often split up into numerous batches of 32 images, with each batch being trained separately. After all batches have been trained, the findings from each batch are then stored to RAM and combined. The more batches there are, the more RAM will need to be used since the weights that are learned from the batches are kept there. For instance, if a training set has 3200 images and the batch size is 32, then there will be 100 batches in total.
- **epochs**: defines the number of training epochs. An epoch is responsible of training all input, or learning all input images. One epoch will be responsible of training all the batches because the dataset is divided into several batches. The quantity of epochs indicates how frequently the model learns all of the inputs and modifies the weights to get closer to the true labels. frequently determined by intuition and experience. It is common to have 300 or more epochs.
- **data**: the path to yaml configuration file containing the summary of the dataset. The model will also utilize the location in the yaml file to reach the validation directory and use its contents for evaluation as the model evaluation procedure is triggered immediately after each epoch.
- **cfg**: specifies the model configuration path. This command line enables the train.py file to compile and create this architecture for training input images based on the architecture specified in the model yaml file before.
- **weights**: specifies a path to the weights. The usage of a pretrained weight can reduce training time. The model will automatically establish random weights for training if it is left empty.
- **name**: name of result folder. The model will create a directory containing all the results performed during training.
- **cache**: True or False, cache images for training faster.

The following parameters are used for training the model:

1. Image size of 640x640, this seems to be enough according research.
2. Batch size of 32, a higher value was not possible due the memory errors.
3. Total of 300 epochs, which seems to be normal.
4. The configuration filepath to the custom created architecture (yaml) file.

### 4.1.7 Testing phase

At the testing phase the built in function of the model is used, which is "detect.py". The weights of the training model output is used for the input of the detect function. The results are exported to an image file, where only handwritten classes are specified, into a desired output folder given by the user. A Python function is used to loop over all the school test folders and export all the images into separate test folders. Each test folder starts with "toets" and ends with a number, as reason to separate all the student tests. Also the results are saved into a pandas DataFrame [23] that includes the bounding box prediction positions x, y, w, h, the class number, confidence score and path to the image file (this includes the test and question numbers). Only bounding boxes with a threshold greater then 0.25 are classified as True Positive (TP), otherwise it will be classified as False Positive (FP).

### 4.1.8 Metrics

Localisation loss, confidence loss, and classification loss are three different forms of loss functions. The localisation loss indicates how effectively the algorithm can detect an objects center and how well the projected bounding box fits it. Confidence is a probability measure for the existence of an item in a suggested region of interest. If the confidence is high, the image window is likely to have an object in it. The algorithm classification loss indicates how effectively it can predict the proper class of a given item. [2]

**Classification loss**: When an item is detected, the classification loss is defined as the squared error of the class probabilities for each class. The classification loss function is provided by the following equation:

$$Loss_{Class} = \sum_{j=0}^{K^2} \mathbf{1}_{obj,j} \sum_{c \in \{classes\}} (p_j(c) - \hat{p}_j(c))^2. \tag{8}$$

**Localisation loss**: The localisation loss function quantifies the errors in the projected border box locations and sizes and is connected to the coordinated that we described before, i.e. (x, y, w, h). The localization loss function is provided by if lambda coordinates is defined as a weight for the importance of the loss.

$$Loss_{loc} = \lambda_{coordinates} \sum_{j=0}^{S^2} \sum_{k=0}^{K} \mathbf{1}_{obj,jk}((x_j - \hat{x}_j)^2 + (y_j - \hat{y}_j)^2) + \\ \lambda_{coordinates} \sum_{j=0}^{S^2} \sum_{k=0}^{K} \mathbf{1}_{obj,jk}((\sqrt{w_j} - \sqrt{\hat{w}_j})^2 + (\sqrt{h_j} - \sqrt{\hat{h}_j})^2) \tag{9}$$

**Confidence loss**: The bounding box predictions, as indicated above, are made up of five values: the coordinates (x, y, w, h), and the confidence. The localisation loss is associated with the localization variables, thus it is only logical that we have a confidence measure. The two loss functions, as one might expect, are quite similar, with one minor change.

$$Loss_{Conf} = \sum_{j=0}^{S^2} \sum_{k=0}^{K} \mathbf{1}_{obj,jk}(C_j - \hat{C}_j)^2) + \lambda_{Nobj} \sum_{j=0}^{S^2} \sum_{k=0}^{K} \mathbf{1}_{Nobj,jk}(C_j - \hat{C}_j)^2). \tag{10}$$

The total loss function can then be defined as:

$$Loss = Loss_{Class} + Loss_{Loc} + Loss_{Conf}.. \tag{11}$$

Mean Average Precision can be defined as the area under precision-recall curve, where precision and recall are defined by the following equations:

$$Precision = \frac{TP}{TP + FP} \tag{12}$$

$$Recall = \frac{TP}{TP + FN} \tag{13}$$

$$mAP = \frac{1}{N} \sum_{n=1}^{N} APi \tag{14}$$

Where TP = True Positive, FP = False Positive and FN = False Negative.

Models that have a high level of precision as recall rises are considered to be high-performing. The mAP averages accuracy over a number of IoU thresholds. By limiting or extending what the model reports as a detection, an IoU threshold can affect the mAP value. [16]

## 4.2 Results - YOLOv5

In total there are 120 images trained, 120 images validate and 1,920 images tested. The YOLO model aims to detect which objects are handwritten in a specific image. The output is a image with a bounding box for a given confidence score based on the detection algorithm.

### 4.2.1 Train and validation results

Once the model was trained, it was validated on 120 questions for the validation set. The accuracy achieved for the validation set is 99%, this means that the model miss qualified only one question in the validation set. Figure 16 shows the confusion matrix of the validation set where the performance for each class (handwritten or printed) can be seen.

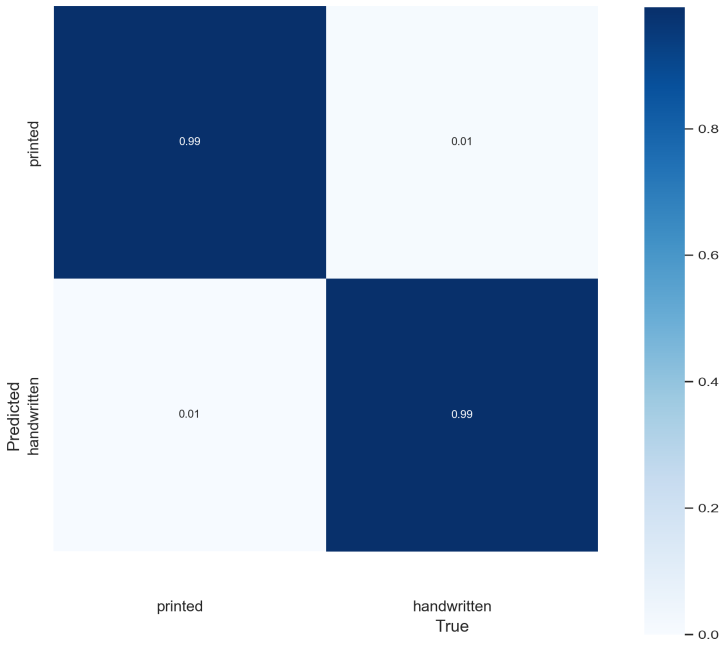


Figure 16: Confusion matrix of the validation set

In the graph below we can see the loss, precision, recall and the mAP functions both for training (train) and validation (val) set.

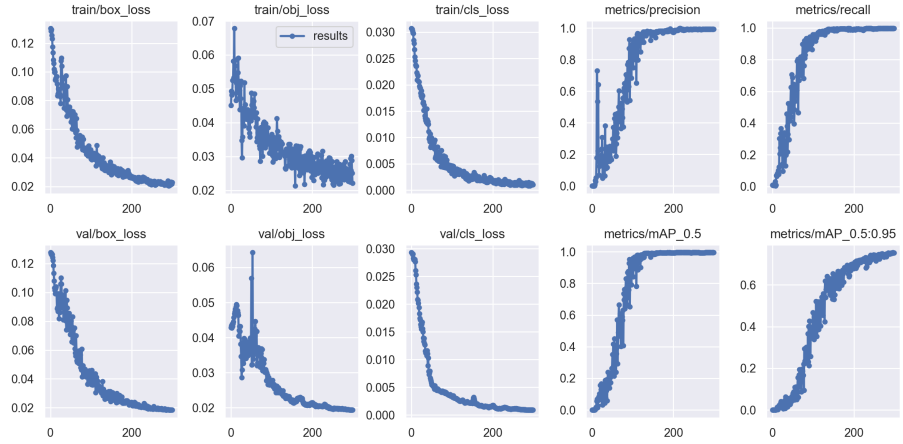


Figure 17: Metrics of the training and validation set

The first three upper left graphs are the train loss functions, which all three are decreasing over time. This is also true for the three bottom left graphs of the validation loss functions, only the object loss function had a slight peak after around 50 epochs, but after some epochs, it recovered to a decreasing trend. Also, the precision and recall functions are both increasing, which can be considered well-performing. The mAP_0.5 metrics value is increasing towards one already after around 100 epochs. This means that the more over-detection occurs will result in a higher value.

### 4.2.2 Testing results

After running the testing set in the object detection model we can display the properties in a Pandas DataFrame seen below.

| Pandas DataFrame output | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| n | x | y | w | h | conf | class | name | img_path |
| 0 | 551.39 | 116.70 | 2099.62 | 680.92 | 0.98 | 0 | printed | 1.jpg |
| 1 | 1102.85 | 872.04 | 1428.36 | 981.27 | 0.95 | 0 | printed | 1.jpg |
| 2 | 733.24 | 765.31 | 1024.08 | 1001.037 | 0.56 | 1 | handwritten | 1.jpg |
| 3 | 543.77 | 94.02 | 1240.04 | 272.39 | 0.95 | 0 | printed | 2.jpg |
| 4 | 1275.55 | 0.00 | 1697.84 | 282.57 | 0.92 | 1 | handwritten | 2.jpg |
| ... | ... | ... | ... | ... | ... | ... | ... | |

To get an insight of the total detected objects we group the DataFrame by name and count the total class values. In total we detected 5,596 printed objects and 2,152 handwritten objects. The distribution of the detected handwritten questions for each test is seen below in Figure 18.

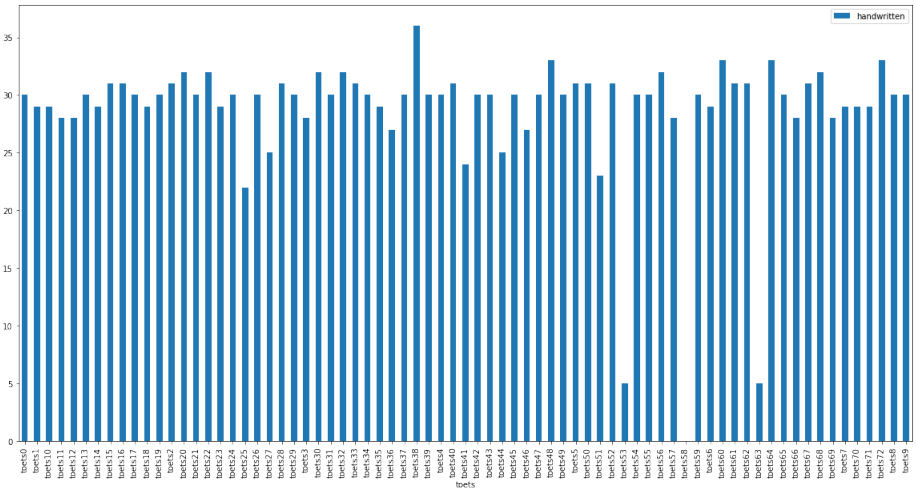


Figure 18: Distribution detection objects for each test

From the above plot we can see that we encounter a problem that needs to be taken into consideration. If the image detects two or more handwritten objects then the image needs to be manually analysed. The distribution plot is a little bit distorted, because the objects are summed together for each test. In Appendix A.1 there is a more detailed analyse for each test and question to detect if there are multiple detected objects within one question. Although for now this gives a global representation of the total detected objects within each test.

The objects that are more or less than 30 handwritten objects are called outliers, such as "toets 53" and "toets 63". Also there is a test "toets 58" that has zero detected handwritten objects. Looking manually into these first two tests, it is seen that only the first five questions are filled in and the rest are blanks. Test "toets 58" sees to be not filled in at all and after manually looked into this is also true. On the other hand "toets 38" has the highest amount of 36 detected objects, which seems to be due to crossing out multiple-choice answers. An example of a multiple-detection failure is shown in Figure 19 and a failure of the Question Cutter is show in Figure 20

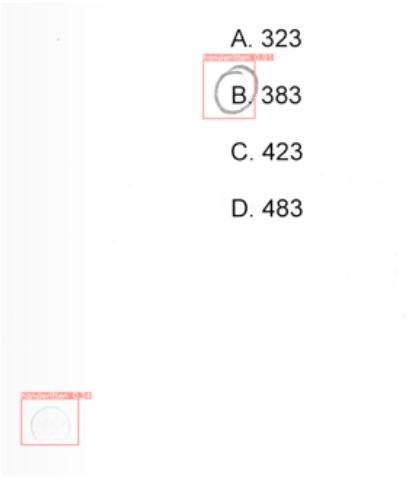


Figure 19: Failure multiple-detection YOLO

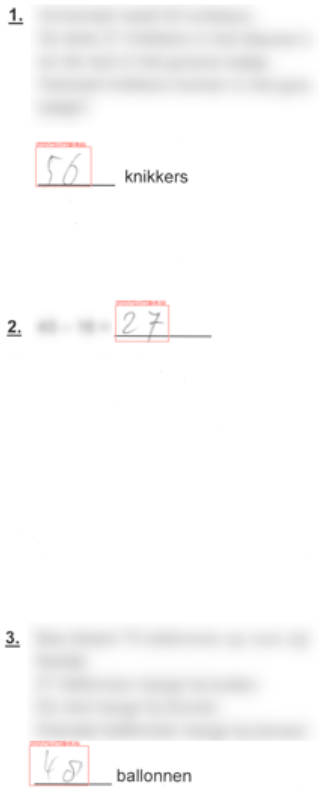


Figure 20: Failure Question Cutter

Another useful insight are the relationship between the different central coordinates of the bounding boxes for each test. Assuming that every test is scanned the same way, so not upside down or different angles, then there should be some relation between the coordinates of questions in different tests. In Figure 19 a scatter plot is seen of all the center coordinates.

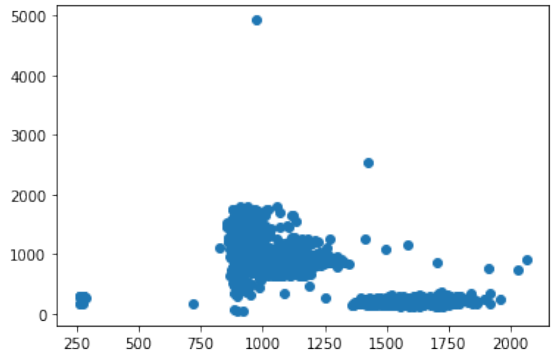


Figure 21: Center coordinates global tests

Filtering on individual questions gives a better representation of the relations of the questions between tests, which is seen in the two figures below.

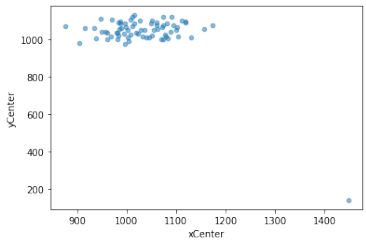


Figure 22: Coordinates Question 1

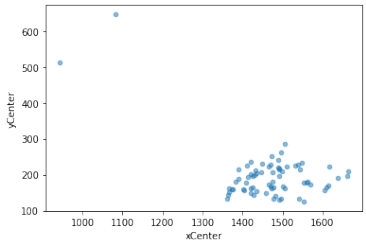


Figure 23: Coordinates Question 2

It looks like there is one outlier in the left figure and two outliers in right figure. This means that these handwritten objects are further away than the other center coordinates, this could be due a failure in detection (Figure 19) or any other failure in the preprocess stage (Figure 20).

The distributions of the confidence score of the handwritten objects is seen in a histogram and box plot below. The lowest confidence score is 0,25, because this is the default threshold value of the IoU.

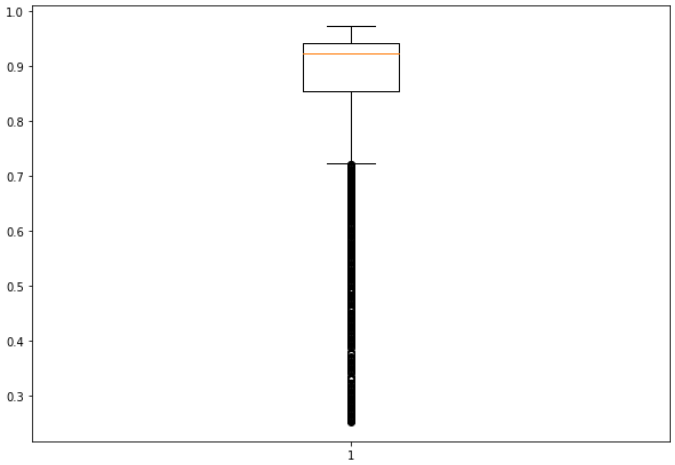


Figure 24: Boxplot confidence score

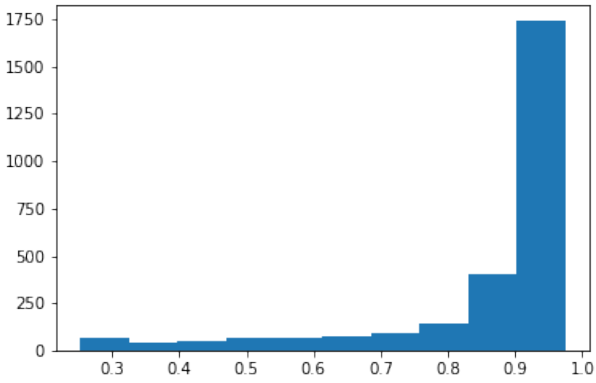


Figure 25: Histogram confidence score

Most of the confidence scores are within the 0.85-0.98 range, except for a few outliers. The histogram has a left-skewed distribution, where the majority of the objects are within the highest class range. This means that the model has on average a high confidence of predicting how sure the model is that the box contains an object and also how accurate it thinks the box is that predicts.

## 4.3 Method ViT TrOCR

The Transformer-based Optical Character Recognition with Pre-trained Models study by Minghao Li et al. [15] introduced the TrOCR model. To achieve optical character recognition, TrOCR uses an image Transformer encoder and an autoregressive text Transformer decoder.

### 4.3.1 Motivation choice of TrOCR model

There are three advantages that comes with the TrOCR model. First, without the need of an external language model, TrOCR leverages the pretrained image Transformer and text Transformer models, which benefit from large-scale unlabeled data for language modeling and image interpretation. Second, TrOCR does not require a complex convolutional network as the model's backbone, making it incredibly simple to set up and maintain. Finally, test results on benchmark datasets for OCR demonstrate that the TrOCR can produce cutting-edge outcomes on both printed and handwritten text recognition tasks without requiring any laborious pre- or post-processing processes.[15]

### 4.3.2 Architecture of TrOCR

TrOCR was developed using a Transformer architecture, which consists of a text Transformer for language modeling and an image Transformer for extracting visual information. TrOCR uses the standard Transformer encoder-decoder construction. The decoder is intended to produce the wordpiece sequence while paying attention to the encoder output and the previous iteration. The encoder is intended to get the representation of the image patches. The text decoder was initialized using the weights of RoBERTa [17] , whereas the picture encoder was initialized using the BEiT [6] weights. The full technical details [15] are outside the scope of this thesis and are not discussed.

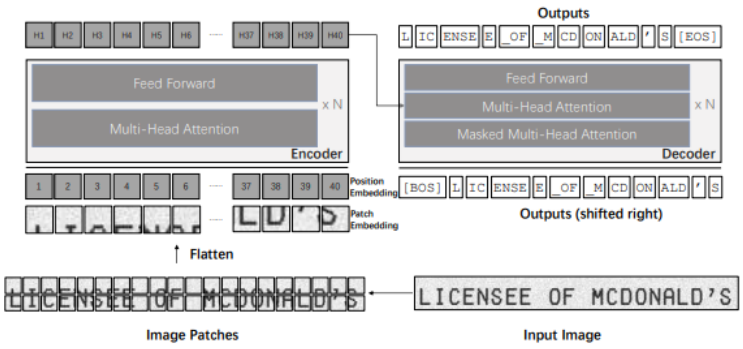


Figure 26: Model Architecture of TrOCR

### 4.3.3 Preprocess phase

A few steps need to be performed before training the model. First the preprocessing step of Chapter 2.4 is executed, which exist of 2,475 handwritten digits. This labeled dataset is loaded into Python that loops over the image folder and save this in a pandas DataFrame with two columns. The first column is linked to the image file path and the second column is linked to the folder of the digit number of that label.

Secondly the dataset is split into a train and test set with respectively 90% and 10%, this 10% is used for testing the performance of the model. Then the train (90%) is split into another train (80%) and validation (20%) set. This is simply done with the "sklearn" package built in function train_test_split. It is important to use a validation set or testing set, simply because this validation set is new data that is fed into the model, which then gives a better insight of the model performance [5].

### 4.3.4 Train, test and fine-tune phase

The transformer model consists of a TrOCRProcessor and a VisionEncoderDecoder model. The TrOCRProcessor uses a pretrained model that extract the features of the image so it resize and normalize the image for the input of the VisionEncoderDecoder model. When initializing this model we need to set two important attributes namely, the maximal length of the output language decoder which we set to 1, because we need only single digit output and a beam-search related parameter that are used to generate the output text.

Next we define some hyperparameters for the model. We need hyperparameter tuning [4] because it consists of finding a set of optimal hyperparameter values for a learning algorithm while applying this optimized algorithm to any data set. That combination of hyperparameters maximizes the model's performance, minimizing a predefined loss function to produce better results with fewer errors.

At last we trained the model on 3 epochs, with a batch size of 4 and validate steps of 200. The evaluation strategy is set to "steps", this means that after 200 steps the validate set will be "tested" or "validated". The metric to evaluate this step is Accuracy, Precision and Recall which is a built in function of the library load_metric [10]. The model is fine tuned both on the MNIST and custom dataset and saved to a PyTorch model file. The output of this file is the input of the VisionEncoderDecoder function in the testing phase.

## 4.4 Results - ViT TrOCR

### 4.4.1 Evaluating Model Performance

For evaluating the fine-tuned model the first testing set is used that is created in Chapter 4.3.3. For both the datasets the accuracy score is extremely high. The accuracy of the MNIST dataset is 99.5% which belongs to the top 50 highest accuracy of all models [21]. The accuracy of the custom dataset is 98.6% which is a bit lower then the MNIST dataset, but the test set is relative small 495 compared to 12,000 handwritten single digits. The two figures below shows the confusion matrices of the test sets where the actual and predicted value for each single digit is shown.

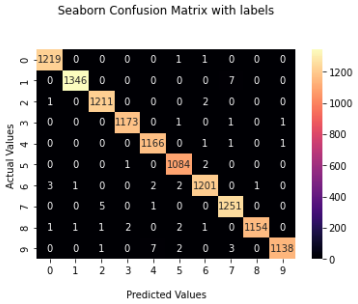


Figure 27: Confusion matrix MNIST

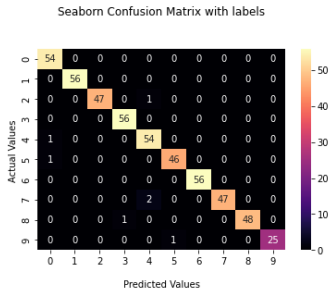


Figure 28: Confusion matrix Custom

### 4.4.2 Training Model Performance

For validating and training model performance the training and validate set is used in Chapter 3.3.4. The total number of training and validation examples of the custom dataset is 1,782 and 198. The MNIST dataset has 43,200 training and 4,800 validate images. The model is trained on three epochs and a batch size of 4. This means that the validation set will be tested on batches of 198/4 $\approx$ 50 images and for the MNIST set 4,800/4 = 1,200 batches of images. The training performance is determined by the loss functions and the accuracy of the validation set. In the figures below we can see the loss functions and accuracy for the training and validation set of the custom dataset.


Figure 29: Custom: Loss function of the training set

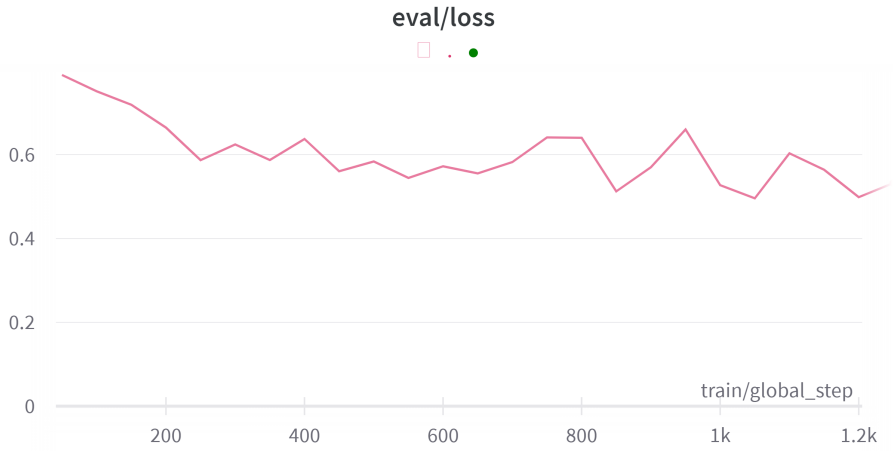


Figure 30: Custom: Loss function of the validation set

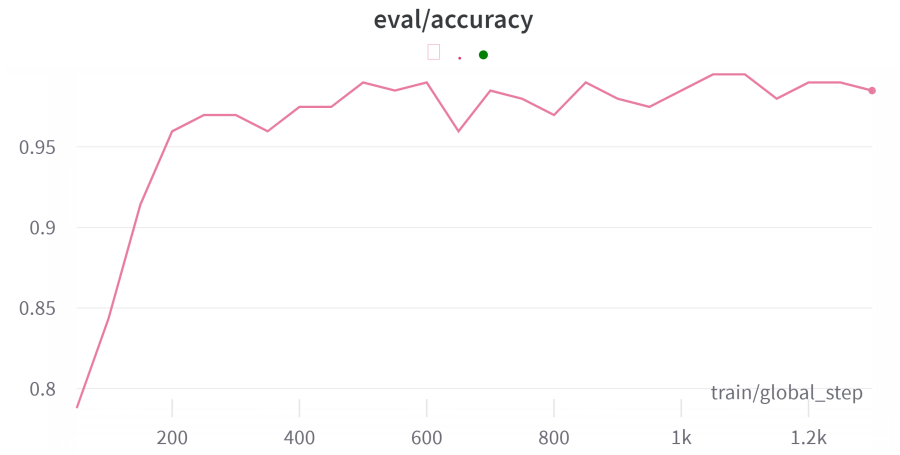


Figure 31: Custom: Accuracy of the validation set

Both the train and validation loss functions graphs decrease after every step. The validation loss decreases from 0.7901 to 0.5332. While the training loss decreased from 12.480 to 0.391. The validation accuracy increases after every step and reaches after the first cycle (200 steps) an accuracy more then 95%. At the end the validation accuracy is 98.5% which is considered high.

In the figures below we can see the see the loss functions and accuracy for the training and validation set of the MNIST dataset.

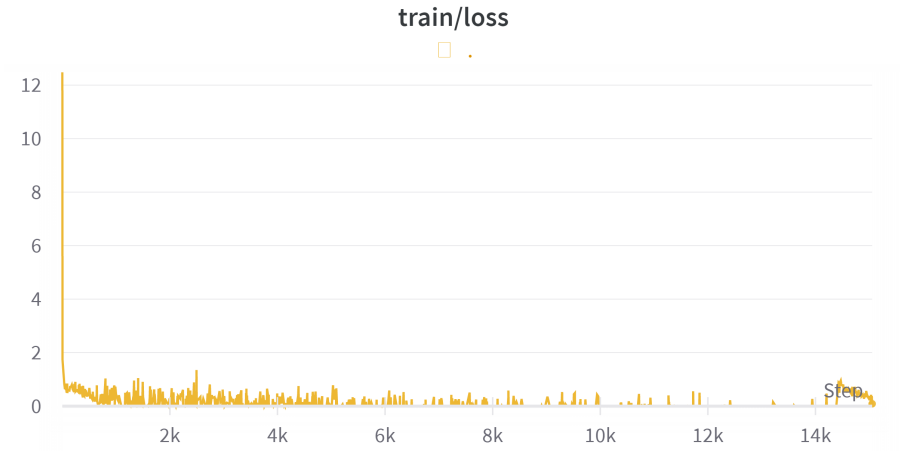


Figure 32: MNIST: Loss function of the training set

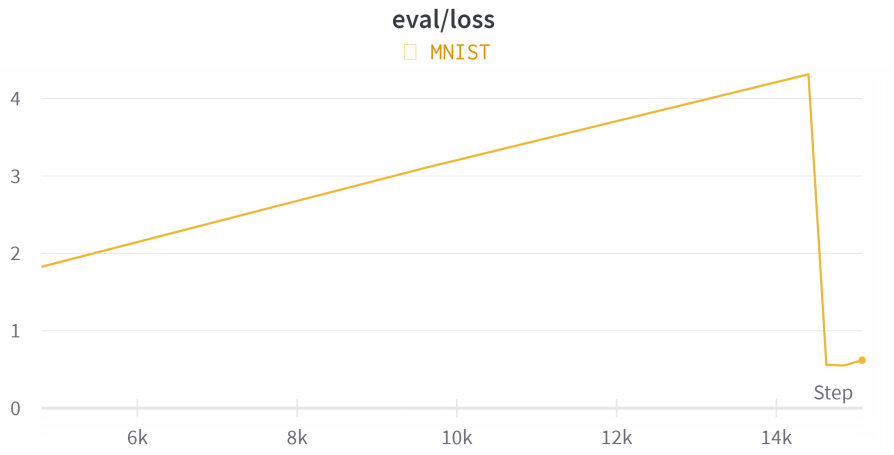


Figure 33: MNIST: Loss function of the validation set

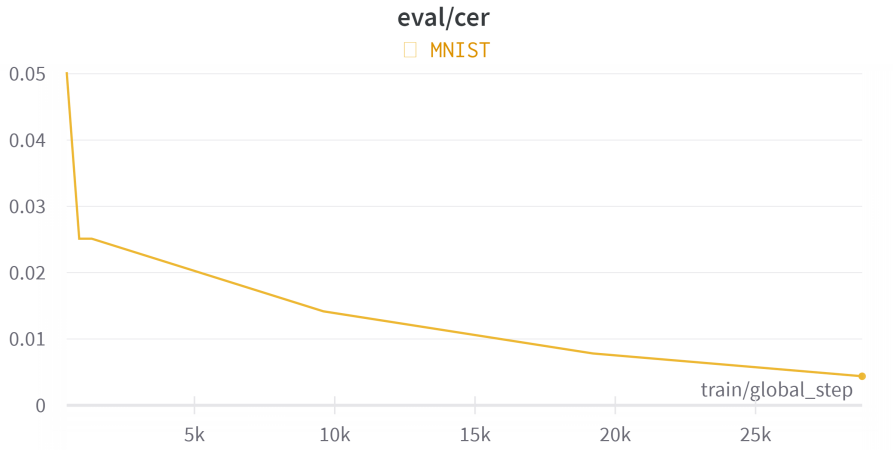


Figure 34: MNIST: Accuracy of the validation set

The validation loss function is increasing after every step, until it reaches step 4,315, then it drastically decreases to 0.5595 and at the end a total loss of 0.6194. The training loss decreases from 12.412 to 0.000, this value is extremely low for a loss function and could be due to overfitting the data. The validation accuracy is expressed in the Character error rate (CER) metric. This metric is usually used for the performance of an automatic speech recognition system, but it can also be used for single characters. A value close to zero does have a high accuracy, which is in this case 0.005375.

### 4.4.3 Testing Model Performance on Cito data

Once the model was trained, it was tested on the whole Cito dataset, which exists of 2,807 single handwritten objects. In the figure below the distribution of the digit numbers is seen:

| Class | Total |
|---|---|
| 0 | 321 |
| 1 | 301 |
| 2 | 278 |
| 3 | 375 |
| 4 | 329 |
| 5 | 218 |
| 6 | 290 |
| 7 | 261 |
| 8 | 304 |
| 9 | 130 |

When testing on the Cito dataset we see an accuracy of 96%. In the below figures the classification report and the confusion matrix where the performance for each class (0 to 9) can be seen.

| Classification Report MNIST | | | | |
|---|---|---|---|---|
| Class | Precision | Recall | F1-score | Support |
| 0 | 0.98 | 0.94 | 0.96 | 337 |
| 1 | 0.98 | 0.98 | 0.98 | 302 |
| 2 | 0.98 | 0.90 | 0.94 | 303 |
| 3 | 1.00 | 0.97 | 0.98 | 385 |
| 4 | 0.96 | 0.98 | 0.97 | 323 |
| 5 | 1.00 | 0.98 | 0.99 | 222 |
| 6 | 0.98 | 0.99 | 0.98 | 286 |
| 7 | 0.90 | 0.96 | 0.93 | 244 |
| 8 | 0.88 | 0.98 | 0.93 | 275 |
| 9 | 0.93 | 0.93 | 0.93 | 130 |
| accuracy | | | **0.96** | 2807 |
| macro avg | 0.96 | 0.96 | 0.96 | 2807 |
| weighted avg | 0.96 | 0.96 | 0.96 | 2807 |

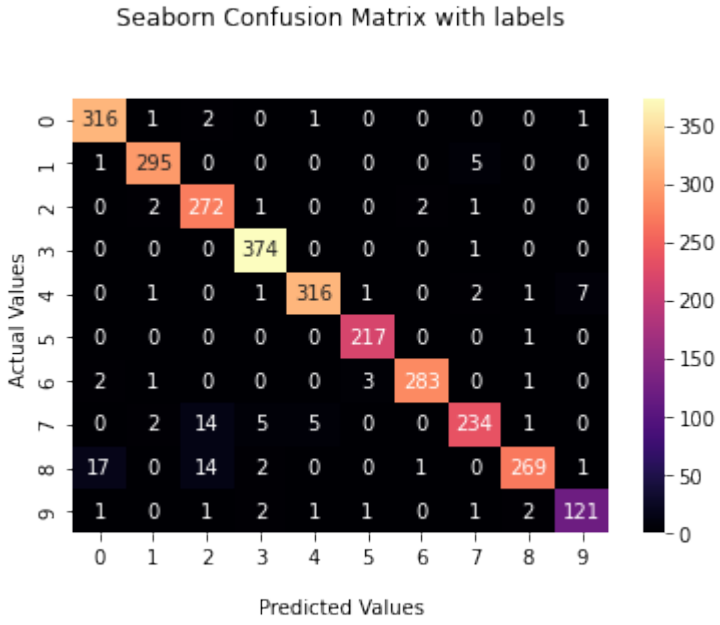


Figure 35: MNIST: Confusion matrix testing set

From the above confusion matrix we can see on the diagonal the correct predicted digits. There are some values that are miss classified, for example there are 17 predicted zero's that are actually an eight, but also some of them are a one, six or a nine. Another high miss classified value are the numbers seven and eight where the model predicted a two. An example of three miss classified numbers is seen below.


Figure 36: Three examples of miss classified digits

When testing on the custom dataset we see an accuracy of 93%. In the below figures the classification report and the confusion matrix where the performance for each class (0 to 9) can be seen.

| Classification Report Custom | | | | |
|---|---|---|---|---|
| Class | Precision | Recall | F1-score | Support |
| 0 | 0.95 | 0.97 | 0.96 | 315 |
| 1 | 0.99 | 0.77 | 0.87 | 387 |
| 2 | 0.88 | 0.99 | 0.94 | 248 |
| 3 | 0.95 | 0.99 | 0.97 | 362 |
| 4 | 0.94 | 0.91 | 0.93 | 341 |
| 5 | 0.91 | 0.99 | 0.95 | 201 |
| 6 | 0.98 | 0.93 | 0.95 | 308 |
| 7 | 0.91 | 0.91 | 0.91 | 261 |
| 8 | 0.88 | 0.99 | 0.93 | 272 |
| 9 | 0.81 | 0.94 | 0.87 | 112 |
| accuracy | | | **0.93** | 2807 |
| macro avg | 0.92 | 0.94 | 0.93 | 2807 |
| weighted avg | 0.93 | 0.94 | 0.93 | 2807 |

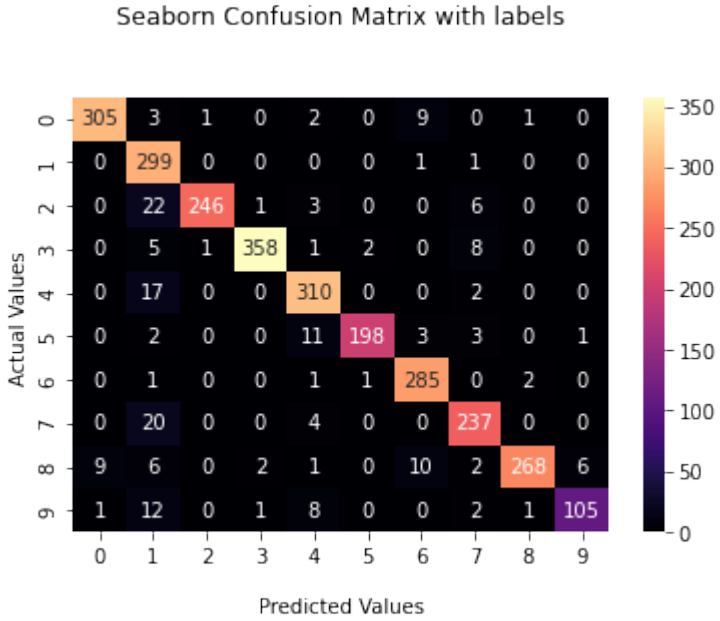


Figure 37: Custom: Confusion matrix testing set

Compared to the MNIST prediction results, we see that the custom data has some trouble of classifying the right digits. It seems that the model is over predicting the number one, which leads to miss classifying the actual values.

# 5 Conclusion and Discussion

## 5.1 Answering the data science question

Like mention before the research aims to find a answer to the question "To what extent can the detection model locate and the classification model interpret the handwritten given answers on the answer forms?". The answer to the first subject of locating a given answer is not really straightforward, because it heavily depends on the quality of the input of the data and the quality of the labeled data. In other words if the external source the "Question Cutter" is not working optimal, then it is possible that the algorithm can detect multiple objects. Also we saw multiple detected objects within one question, which could be due to crossing out or erasing answers. This can be improved firstly to use more training and validating data, secondly use more class labels, like "crossed out handwritten object" and "erased handwritten object", lastly use another format for filling in answer questions. One solution could be to restrict the answers to a specific part (boundary rectangles), instead of using a horizontal line, for an example format see Appendix A.3.

Although the model performed an accuracy of 99% on a validation set of 120 question numbers. This seems a high accuracy, but keep in mind this validation set is extremely low and could give a biased result. There was a total of 2,152 handwritten objects in 2,430 images, which is on average below 1 handwritten object for each question. Normally we would expect one handwritten object for each question. This lower average behaviour could be due that children are not filling in the answers on the form, which we saw in Figure 18. Although the loss functions are decreasing over time and the the confidence scores are within the upper high range we can conclude that the YOLO algorithm is performing really good.

The answer to the second subject depends on the output of step 5 in Chapter 2. This step is performed by Klopper [14], where single digits are extracted from the YOLO output. Not all written digits are single digits, it is also possible that numbers or digits are connected or overlapping each other. The model will then fail to extract single digits. If this is excluded then the model reaches a accuracy score of 96% with the fine-tuning model on the MNIST dataset. Compared to humans, that have an accuracy score of 99.77%, which is 23 errors on 10,000 images, that is a lot lower. Still the accuracy of the MNIST dataset is quiet high and could be improved to reach a much higher accuracy that eventually will be close to the human accuracy.

## 5.2 Answering the research question

This thesis proposes a solution for detecting handwritten objects and recognizing handwritten single digits. The research question was as followed: "Can we build a training model that can detect and recognize handwritten objects in a school test and is this approach reliable enough to implement in practice?" While both models for detecting objects and recognize are reliable enough according to the evaluating metrics and accuracy, it is still said that humans are better in recognizing digits than machine learning algorithms. Therefore the tests should still be inspected manually. However some of the answers could be excluded or graded with a certain confidence score (YOLO) and probability score (ViT), but that is up to Cito whether this approach is reliable enough to implement in practice. One could think of how low should the error rate be, which confidence score and probability value is high enough for a "human like" classification.

## 5.3 Limitations

There are a couple limitations that need to be considered:

1. YOLO model is restricted to two classes (handwritten or printed text) and the training and validation set is extremely small (bias prediction).
2. The ViT fine-tuning model is trained on 3 epochs, higher epochs could result in higher performance, but it takes a lot of computational resources (for 3 epochs it was 6 hours in total for training).
3. The MNIST dataset is a "toy" dataset and lacks of children handwriting digits.
4. The Custom dataset is a very small dataset that could lead to a lower accuracy score.
5. The two models heavily depend on other models that are out of this thesis scope.

## 5.4 Further research

Several improvements can be made to the two models. First the YOLO model could be expanded to three or more classes, adding for example crossed or erased answers as a label. Also the analyse of the coordinates bounding boxes could be improved to set a cut-off value for specific coordinates depending on the relations of the questions between tests. Secondly the ViT model has a lot of parameters that can be changed and optimized for a better result. Also a multiple digits classifier could be introduced to classify the whole YOLO output picture, which can lead to a lower error rate. Further a better bounded answer field could lead to a higher accuracy, because then it is easier to extract the digits. Lastly create a question where a student need to fill in the numbers from zero to nine, so this can be used as a training dataset for the ViT fine-tuning model.

## 5.5 Code and Model retrieval

All the code that is used in this thesis can be retrieved from Github [9]. The pretrained fine-tuning model of the MNIST dataset can be retrieved from the Hugging Face website [25].

# References


[1] Tsehay Admassu and Pramod Nair. "Handwritten digits recognition with decision tree classification: a machine learning approach". In: *International Journal of Electrical and Computer Engineering (IJECE)* 9 (Oct. 2019), p. 4446. DOI: 10.11591/ijece.v9i5.pp4446-4451.

[2] S. N. Agni. "Activity Recognition of Office Space Users using Thermopile Array Sensor". In: (Oct. 2020).

[3] Nafiz Arica and Fatos Yarman Vural. "An overview of character recognition focused on off-line handwriting". In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 31 (June 2001), pp. 216–233. DOI: 10.1109/5326.941845.

[4] Mohamadjavad Bahmani et al. "To tune or not to tune? An Approach for Recommending Important Hyperparameters". In: *CoRR* abs/2108.13066 (2021). arXiv: 2108.13066. URL: https://arxiv.org/abs/2108.13066.

[5] Yu Bai et al. "How Important is the Train-Validation Split in Meta-Learning?" In: *CoRR* abs/2010.05843 (2020). arXiv: 2010.05843. URL: https://arxiv.org/abs/2010.05843.

[6] Hangbo Bao, Li Dong, and Furu Wei. "BEiT: BERT Pre-Training of Image Transformers". In: *CoRR* abs/2106.08254 (2021). arXiv: 2106.08254. URL: https://arxiv.org/abs/2106.08254.

[7] *Centraal Instituut voor Toetsontwikkeling*. URL: https://www.cito.nl/ (visited on 06/14/2022).

[8] E. Dağdeviren. "Comparison of Support Vector Machines and Artificial Neural Networks for Handwriting Number Recognition". In: (2013).

[9] ADS Cito Groep. *Code thesis*. URL: https://github.com/ADS-thesis-CITO/CITO-thesis/tree/main/Classification%20Aico.

[10] *How to metrics*. URL: https://huggingface.co/docs/datasets/how_to_metrics.

[11] Md Nafee Al Islam and Siamul Karim Khan. "HishabNet: Detection, Localization and Calculation of Handwritten Bengali Mathematical Expressions". In: (Sept. 2019).

[12] Licheng Jiao et al. "A Survey of Deep Learning-Based Object Detection". In: *IEEE Access* 7 (2019), pp. 128837–128868. DOI: 10.1109/ACCESS.2019.2939201.

[13] *Jupyter Notebook*. URL: https://jupyter.org/.

[14] Tom Klopper. "Automatic grading of handwritten math tests: A convolutional neural network approach". In: (July 2022).

[15] Minghao Li et al. "TrOCR: Transformer-based Optical Character Recognition with Pre-trained Models". In: (Sept. 2021).

[16] Guoxu Liu et al. "YOLO-Tomato: A Robust Algorithm for Tomato Detection Based on YOLOv3". In: (2020). DOI: 10.3390/s20072145.

[17] Yinhan Liu et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *CoRR* abs/1907.11692 (2019). arXiv: 1907.11692. URL: http://arxiv.org/abs/1907.11692.

[18] *Modified National Institute of Standards and Technology*. URL: http://yann.lecun.com/exdb/mnist/ (visited on 05/30/2022).

[19] Naoki. URL: https://naokishibuya.medium.com/r-cnn-region-based-convolutional-neural-network-9a6ef37fd528.

[20] *OpenCV*. URL: https://opencv.org/.

[21] *Performance of MNIST models*. URL: https://paperswithcode.com/sota/image-classification-on-mnist.

[22] Yusuf Perwej and Ashish Chaturvedi. "Machine Recognition of Hand Written Characters using Neural Networks". In: *CoRR* abs/1205.3964 (2012). arXiv: 1205.3964. URL: http://arxiv.org/abs/1205.3964.

[23] Python. *Pandas DataFrame*. URL: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html.

[24] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[25] Aico Schreurs. *ViT MNIST model*. URL: https://huggingface.co/aico.

[26] *Sint Joriscollege*. URL: https://www.sintjoriscollege.nl/ (visited on 05/30/2022).

[27] Abhinav Somani. *Deep learning feature of OCR*. URL: https://www.forbes.com/sites/forbestechcouncil/2019/09/10/the-future-of-ocr-is-deep-learning/?sh=e16d52b6a049.

[28] Arend-Jan Tissing. "Automatic Grading of CITO Mathematics Tests: Multiple Choice Classification". In: (July 2022).

[29] *TrOCR*. URL: https://github.com/microsoft/unilm/tree/master/trocr (visited on 05/30/2022).

[30] Qiwei Wang et al. "Deep learning approach to peripheral leukocyte recognition". In: *PLOS ONE* 14 (June 2019), e0218808. DOI: 10.1371/journal.pone.0218808.

[31] *YOLOv5*. URL: https://github.com/ultralytics/yolov5 (visited on 05/30/2022).

[32] *YOLOv5 improvements and evaluation*. URL: https://blog.roboflow.com/yolov5-improvements-and-evaluation/.

[33] Zhong-Qiu Zhao et al. "Object Detection With Deep Learning: A Review". In: *IEEE Transactions on Neural Networks and Learning Systems* PP (Jan. 2019), pp. 1–21. DOI: 10.1109/TNNLS.2018.2876865.

# A Appendix

## A.1 Figures of detected objects distribution

The figures below are representing the distribution of the total detected handwritten objects for each test.

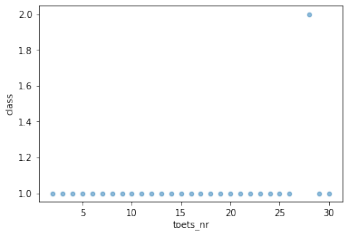


Figure 38: Distribution of Test 1

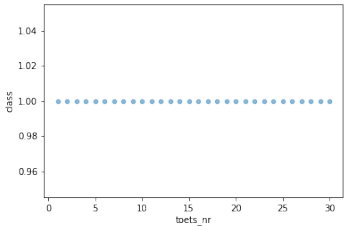


Figure 39: Distribution of Test 2

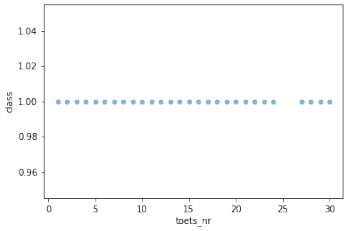


Figure 40: Distribution of Test 3

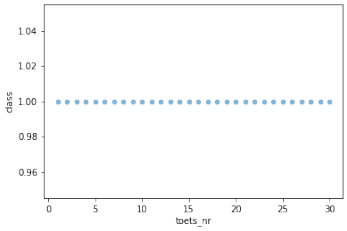


Figure 41: Distribution of Test 4

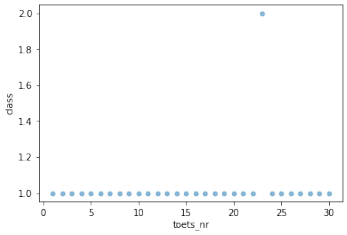


Figure 42: Distribution of Test 5

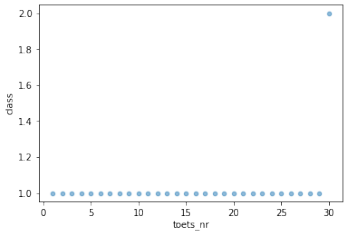


Figure 43: Distribution of Test 6

## A.2 False classification of ViT

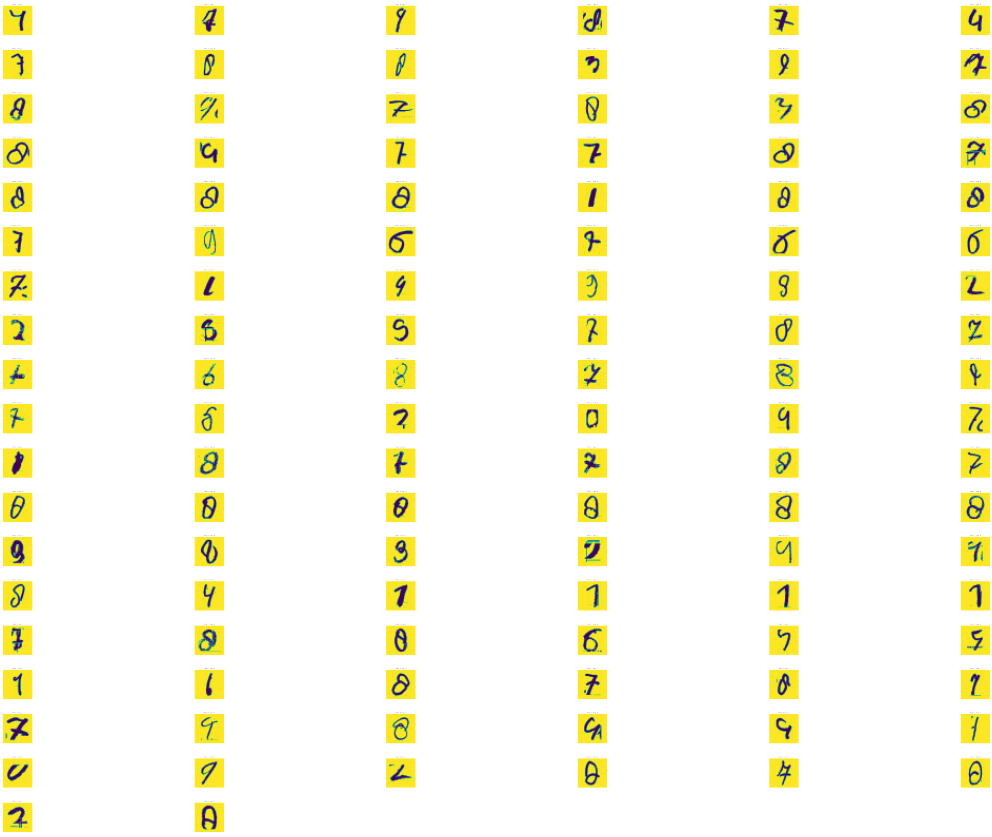

Figure 44: False classified numbers Cito

## A.3 Example of Cito format question

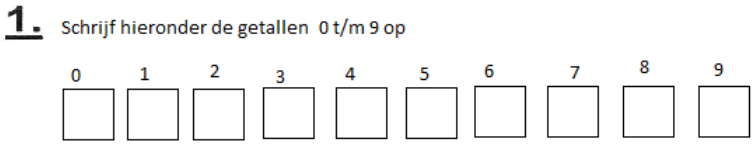


Figure 45: Question for custom label dataset

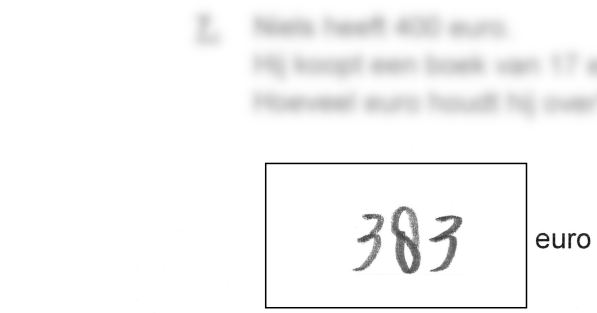


Figure 46: Format for bounding answers