Benchmarking of Different YOLO Models for CAPTCHAs Detection and Classification

Mikołaj Wysocki
ITTI Sp.z o.o.
Poznań, Poland
mikolaj.wysocki@itti.com.pl

Henryk Gierszal
Adam Mickiewicz University
Poznań, Poland
gierszal@amu.edu.pl

Piotr Tyczka
ITTI Sp.z o.o.
Poznań, Poland
piotr.tyczka@itti.com.pl

George Pantelis
UBITECH
Athens, Greece
gpantelis@ubitech.eu

Sophia Karagiorgou
UBITECH
Athens, Greece
skaragiorgou@ubitech.eu

***Abstract*—This paper provides an analysis and comparison of the YOLOv5, YOLOv8 and YOLOv10 models for webpage CAPTCHAs detection using the datasets collected from the web and darknet as well as synthetized data of webpages. The study examines the *nano (n)*, *small (s)*, and *medium (m)* variants of YOLO architectures and use metrics such as *Precision*, *Recall*, *F1* score, *mAP@50* and inference speed to determine the real-life utility. Additionally, the possibility of tuning the trained model to detect new CAPTCHA patterns efficiently was examined as it is a crucial part of real-life applications. The image slicing method was proposed as a way to improve the metrics of detection on oversized input images which can be a common scenario in webpages analysis. Models in version *nano* achieved the best results in terms of speed, while more complexed architectures scored better in terms of other metrics.**

***Index Terms*—Computer Vision, Object Detection, YOLO, CAPTCHA**

I. Introduction

As Completely Automated Public Turing Test to Tell Computer and Humans Apart (CAPTCHA) [1] is designed to protect data, copyrighted contents, servers, and services from bots. They can also be a blocking point for web-crawlers used to analyze illegal marketplaces and forums on the Dark Web, fake-new detectors, etc. CAPTCHAs are also used to prevent against spam and fraud, to check compliance, and to verify users. Classifying and pointing the position of a CAPTCHA code can be the first and crucial step of automatic CAPTCHA solving. Such a classifier can also serve as a tool for gathering the data about the most commonly used CAPTCHA types across the web. The development of the powerful machine learning algorithms made it possible to automatically solve different types of CAPTCHAs like text-based [2]–[4] or image-based also called ”reCAPTCHAs” [5], [6]. Increasing number of automated CAPTCHA solvers (such as Optical Character Recognition (OCR) algorithms and Deep Learning Models) resulted in the development of new CAPTCHA types and schemes varying from simple puzzles and math, through audios and videos, to interactive and mini-games [7], [8]. With the increasing variant of CAPTCHAs the challenge might not be only to solve the CAPTCHA but also to determine its type and find its position on the webpage. Recent technological advances in computer vision and real-time detectors have opened the opportunity for accurate and precise tool for CAPTCHA recognition.

There are many solutions used to the image recognition that are based on Machine Learning (ML) techniques like Neural Networks (NNs). One of the popular approaches involves the You Only Look Once (YOLO) framework. The YOLO architecture [9] as a Convolutional Neural Network (CNN) is well known for its accuracy and speed in computer vision tasks both while working with images and videos [10]. Starting from the analysis of accessible datasets, through the possible methods of enlarging the training data, tackling the issue of diverse size of input images, and ending on the training tuning and then evaluating the YOLO-based models, this paper aims to explore the possibility of composing Artificial Intelligence (AI) driven CAPTCHA detector that provides a classification model to identify and CAPTCHA types and determine where CAPTCHA pattern is located in the webpage.

In this paper we present a complex method for collecting a ML training dataset and also performance results of ML models trained for automatic classification of CAPTCHA codes embedded in webpages. We describe how a dataset for the training process was collected in a few steps to increase the quantity and diversity of labeled specimens. We focus on YOLO models as mature solutions used in image recognition. Moreover, a smart technique to process images dived into horizontal slides is implemented in order to minimize a problem with rescaled images. All YOLO models are compared using a set of performance metrics.

The structure of the paper is as follows. In section II there is an overview of the related works. Section III provides an insight into the methodology. Subsection III.A describes the process of obtaining and preparing the dataset used to develop the CAPTCHA detector. In the following subsection III.B we propose the image slicing approach to overcome the problem of oversized input images. Subsection III.C gives an overview of training and evaluation details. The section III ends with the subsection III.D which provides the detailed description of performance metrics. Results obtained and their discussion are presented in Section IV. The final Section V contains conclusions.

II. Related works

Research to solve CAPTCHA codes automatically using ML techniques has become a challenge for years. There are even commercial services available in Internet where everyone can buy a packet of Application Programming Interface (API) requests to solve CAPTCHAs in webpages which are visited. That is why new CAPTCHA types like Google's reCAPTCHAv2 or reCAPTCHAv3 systems are still being developed to counteract against bots that are feasible to provide CAPTCHA resolves in real time. To solve CAPTCHAs based on sequential information (like text of different lengths or with complicated features of characters) one can also concatenate a CNN with a Recurrent Neural Network (RNN) [11] that is able to correlate dependencies in CAPTCHA patterns. Another ML-based solution is Generative Adversarial Networks (GANs) [12] that allow providing synthetic CAPTCHA images as similar as possible to the real ones. A promising technique is Capsule Networks [13] that can detect the spatial relationships between different characters in the CAPTCHA. In [14] a customized CNN called Deep-CAPTCHA was developed to solve both numerical and alphanumerical CAPTCHAs, leading to cracking accuracy of 98.94% and 98.31%, respectively. The same approach based on a CNN named CapNet as well as VGG-19 and AlexNet deep CNN models were selected in [15]. The CapNet solver achieved accuracy of 96.08% (with slight differences for each of 5 digits in alphanumeric characters of CAPTCHA challenges) using advanced pre-processing techniques like noise reduction filtering, Grey-scaling, resizing, normalization, one-hot encoding, and image size reduction. Pre-trained YOLO v8 models for image segmentation and classification were used in [6] for three types of CAPTCHAs (and 13 classes) provided within the reCAPTCHAv2 system. According to the normalized confusion matrix top 5 accuracy was 99.5%. Reinforcement Learning techniques can be used to provide automatically upgrades for CAPTCHA-solving algorithms.

III. Methodology

A. Dataset preparation

Starting point of the experiment was to collect the images of webpages protected by CAPTCHAs. Although collections of CAPTCHA images can be found on popular portals like *Kaggle* or *Robflow* those are mostly plain images of CAPTCHAs cropped from the whole website. To make the training dataset more diverse and increase the neural network confidence to distinguish CAPTCHAs from typical webpage elements which may be falsely classified as CAPTCHAs (like heading, images, adverts, etc.), it was decided to collect and combine three datasets:

(1) Webpage images were obtained using the *Selenium WebDriver* [16] with Python scripts as screenshots of ca. 10,000 most popular webpages based on the existing collection [17]. In addition to that subset, additional images were obtained from *Kaggle* datasets [18], [19].

(2) CAPTCHA images were gathered using the Selenium WebDriver with Python scripts as screenshots of: Amazon [20], Weibo [21], Wikipedia [22], and Shopee [23] webpages. Additional images were also obtained from *Kaggel* [24]–[29] and *Robflow* datasets [30]–[32]. Moreover, the *PyCaptcha* [33] tool was used to generate text CAPTCHAs. Each CAPTCHA image was assigned to one of four classes:
   a) Text CAPTCHAs
   b) Puzzle CAPTCHAs
   c) Image CAPTCHAs
   d) Button CAPTCHAs

   that have allowed preparing (3)

(3) final training dataset.

Webpage images from the darkweb were collected using a data crawler alongside a Tor proxy [34] to enable access. Specifically, datasets were obtained from four different dark web sites to provide additional diversity to the training data. To overcome the unavailability of various training sub-datasets of webpages protected by CAPTCHAs, images from the CAPTCHA dataset collected (dataset (2)) were randomly resized and randomly pasted onto the images from the webpage dataset (1), in order to obtain additional synthetic specimens of webpages protected by CAPTCHA. Such a solution not only significantly enlarged the training dataset (denoted in the following as (3)) but also simplified the process of labeling the data specimens (CAPTCHA type) by calculating the label based on CAPTCHA image width, height, and coordinate of leftmost corner on a plane of the webpage image. Fig. 1 depicts the process of combining those two datasets in order to use the resulting dataset for neural network training.

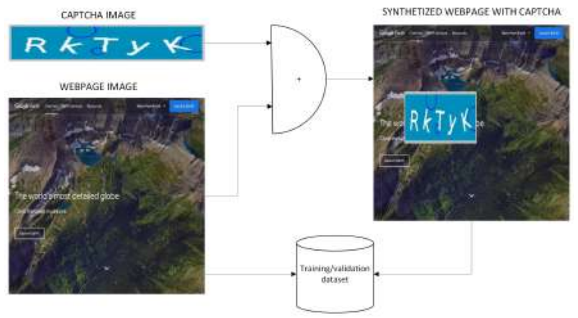


Fig. 1: The CAPTCHA image is combined with the webpage image to obtain a synthetized image of the webpage protected by CAPTCHA. Both original and synthetized webpage images are then used in training of the neural network.

The original webpage images without CAPTCHA (dataset (1)), used for synthetizing data specimens, were also added to the training dataset (3) with empty labels (no detection class - "unlabeled" in 4) to reduce the number of false positive detections. Fig. 2 shows a single sample of each CAPTCHA class.

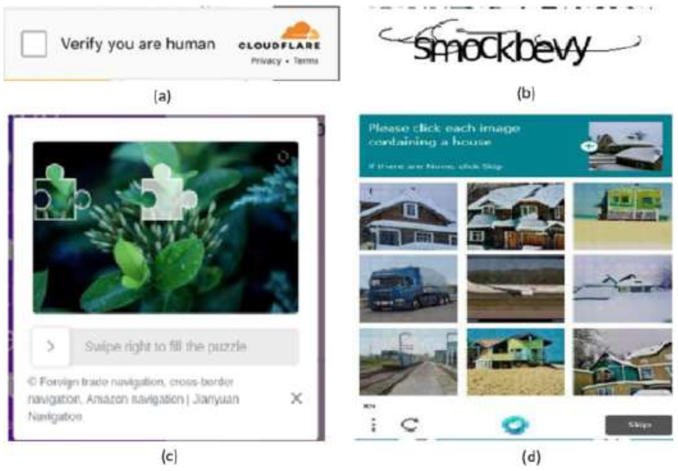

Verify you are human
CLOUDFLARE
Privacy • Terms
(a)
smockbevy
(b)
Swipe right to fill the puzzle
© Foreign trade navigation, cross-border navigation, Amazon navigation | Jianyuan Navigation
(c)
Please click each image containing a house
If there are None, click Skip
Skip
(d)

Fig. 2: Four CAPTCHA images classes (a) button [35], (b) text [22], (c) puzzle [20], and (d) image [36].

Fig. 3 presents a flow diagram containing all main steps of data preprocessing sequence. After collecting the datasets (plain webpage images (1) and CAPTCHA images (2)), the synthetized data are produced until the number of images in training dataset (3) meets the desired volume, which depends on available computational power (as the number of images in the training dataset increases, so does training time) and variety of images in both webpage (1) and CAPTCHA (2) datasets. Ideally the number of samples for each class (five classes in 4) should be equal. The number of samples used for the experiment, were 115,651. The last step of the preprocessing was to split the training dataset (3) between train, valid and test subsets as 70%, 20%, and 10% of cardinality of the whole dataset. The exact distribution of the classes in the dataset is presented in Fig. 4.

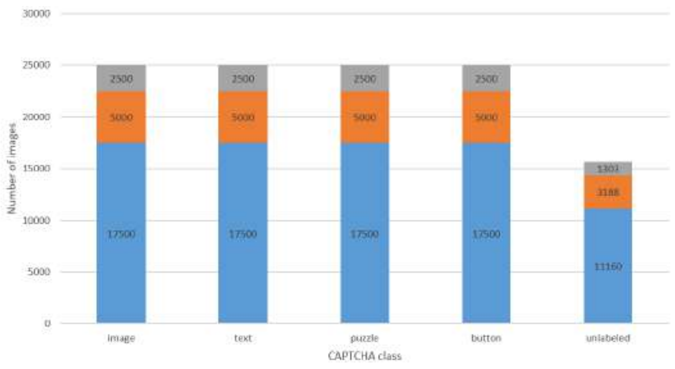


Fig. 4: Distribution of classes in the dataset.

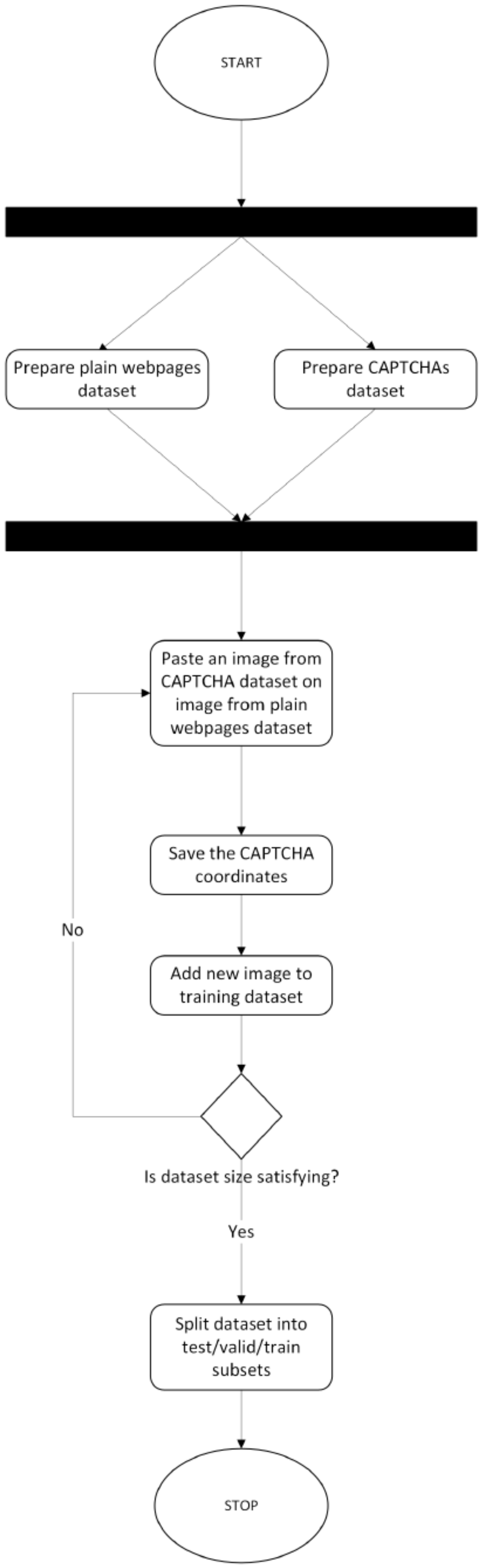

START
Prepare plain webpages dataset
Prepare CAPTCHAs dataset
Paste an image from CAPTCHA dataset on image from plain webpages dataset
Save the CAPTCHA coordinates
Add new image to training dataset
No
Is dataset size satisfying?
Yes
Split dataset into test/valid/train subsets
STOP

Fig. 3: Preprocessing process flow diagram.

B. Dividing the input image into slices

A trained network would have a fixed size of input, but depending on the layout of the webpage and the method how the screenshot is taken, the input images will vary in size and resolution. The trained neural network will always resize the image to fit to the model input size, which in extreme cases may notably distort the image causing the significant drop in the performance of the model.

To tackle this issue, an input image that exceeds a certain level of scale difference between current size and optimal one might be divided into smaller chunks of images (i.e. image slices). Each slice should contain the portion of the last slice to decrease the chance of having a desired classification object torn between neighbor slices. Fig. 5 is a simple demonstration of rescaling problem and solution. To calculate the starting and ending points of n slices along the image width and height, the equation (1) was used:

$$(a_n, b_n) = \begin{cases} (0, s) & : n = 1 \\ (L - s, L) & : b_{n-1} - s \times i + s > L \\ (b_{n-1} - s \times i, a_n + s) & : otherwise \end{cases} \tag{1}$$

Where a and b are the starting and ending points of the slice along the axis of either input image width or height. L is respectively input image width or height, s is the slice size and i is the percentage value of slices overlapping. Fig. 6 shows the equation parameters marked on an example of the image.

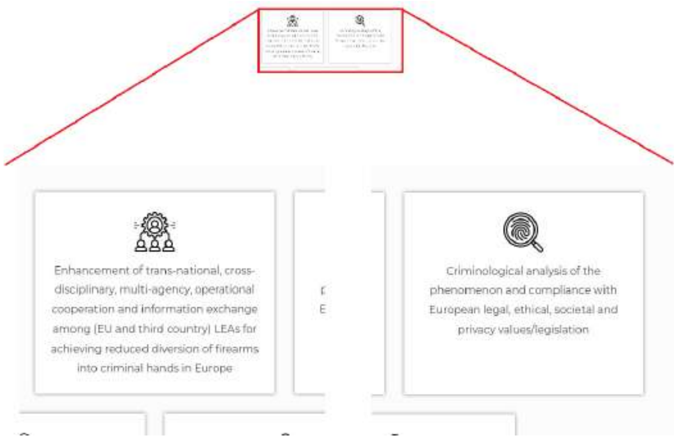
C. Training and evaluation details

In our study we decided to compare three YOLO models (versions) which are:

- **YOLOv8:**
  Built on the foundations of the YOLOv5 and launched by Ultralitics in January 10th, 2023 [38]. YOLOv8 introduced some key improvement; the most notable being anchor-free detection that increases the precision of the model comparing to its anchor-based ancestors [39], alongside with innovative feature named Spatial Pyramid Pooling Feature (SPPF) that improved model's performance in detecting objects of diverse scales [40].
- **YOLOv5u:**
  YOLOv5u integrates the features introduced in YOLOv8 (anchor-free, objectness-free split head) to the architecture of YOLOv5 model as well as offers the big ammout of pre-trained models, improving the accuracy-speed tradeoff, and makes the model more flexible in adapting to different scenarios [41].
- **YOLOv10:**
  Built on the Ultralitics Python package [38] and launched in May 23rd, 2024 by Tshingua University [42] aimed to find the balance between the performance and computational cost by eliminating Non-Maximum Suppression (NMS) to increase the postprocessing speed. Moreover YOLOv10 incorporates holistic efficient-accuracy driven model design strategy. Aditional optimization in the model architecture components aimed to reduce the computational overhead and improve overall model capabilities [42].

Versions YOLO5u and YOLOv8 were chosen because of their availability and the ease of use. Both of them can be trained through Ultralitics hub [38] which might make them the first choice for the users. YOLOv10 was selected because it was the newest version of release for the time of conducting the experiments (in September 2024).

We trained YOLO models based on the default parameters proposed by Ultralitics company. Each network was trained for 100 epochs. Moreover we conducted additional experiment in which we changed parameters of YOLOv10 model to see if we could obtain better results after enabling the cosine learning rate scheduler (cos_lr) and increasing the number of


Enhancement of trans-national, cross-disciplinary, multi-agency, operational cooperation and information exchange among (EU and third country) LEAs for achieving reduced diversion of firearms into criminal hands in Europe

Criminological analysis of the phenomenon and compliance with European legal, ethical, societal and privacy values/legislation

Fig. 5: Rescaling the webpage image [37] degrades pixel resolution and makes the text inside the red box unreadable. Slicing the original image prevents the information loss.

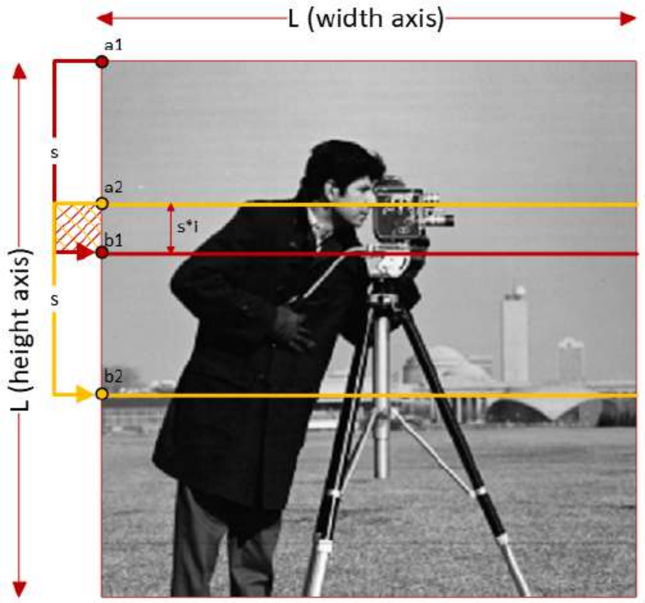


Fig. 6: The first two pairs of the coordinates (a, b) calculated along the height axis produces red and yellow slices that overlap over s*i height.

training epochs. Default parameters for YOLO and modified parameters are presented in Table I.

TABLE I: Training parameters

Setup	Learning rate	Optimizer	Update of model: cos_lr	Epochs
Default YOLO	0.01	SGD	Disabled	100
Modified YOLOv10	0.01	SGD	Enabled	234

Notes: SGD - Stochastic Gradient Descent

Following the training, the best model was evaluated by calculating the performance metrics for two test sets:

- **Set 1:**
  Part of the dataset depicted in Fig. 4. It contains the webpages with the same CAPTCHA pattern as in training and validation set but with a different content and background (webpage). A differences in the CAPTCHA pattern and content are presented in Fig. 7. Set 1 was used to validate the generalization capabilities of the model, while working with the known CAPTCHA patterns.
- **Set 2:**
  It is an additional dataset with no synthetized data and the completely new CAPTCHA patters gathered from the web. This small set of 225 specimens was used to evaluate the capability of the trained neural network to classify CAPTCHAs of unknown pattern. Set 2 was also used to explore the possibility of tuning the neural network for classifying the new CAPTCHA patters with small amount of the data without the huge impact on the network performance. Because of the limited number of the source data and imbalance in the CAPTCHA types to be embedded in the webpages, classes' instances in Set 2 were not equally numerous. The final class distribution in Set 2 was: text (79), image (74), puzzle (20), and button (52).

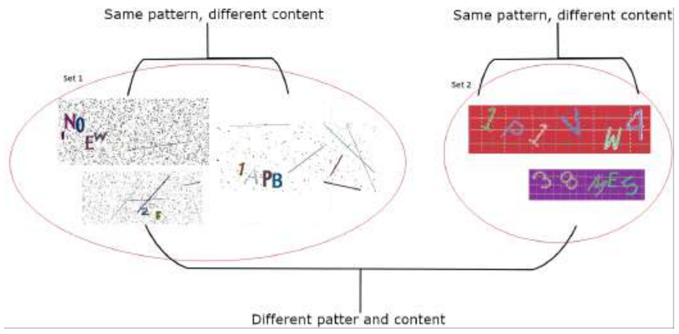

Same pattern, different content
Same pattern, different content
Set 1
Set 2
Different patter and content

Fig. 7: Differences in pattern and content between the text CAPTCHAs from Set 1 and Set 2

D. Performance metrics

a) *Precision*

Precision measures the accuracy of the model's positive predictions and is calculated via:

$$Precision = \frac{TP}{(TP + FP)} \tag{2}$$

Where True Positive (TP) is the count of correct predictions and False Positive (FP) is the number of incorrect predictions of certain class. For the experiment, high *Precision* means that the Neural Network (NN) model does not classify a fraction of webpage elements (text boxes, images, etc.) as CAPTCHAs and can correctly distinguish one CAPTCHA type from another.

b) *Recall*

Recall (or *sensitivity*) indicates the ability of the model to correctly detect and classify an object, and is defined by the formula below:

$$Recall = \frac{TP}{(TP + FN)} \tag{3}$$

Where False Negative (FN) stands for the number of the missed detections (class instances present on the image but classified as a different class or background). For the experiment, *Recall* tells how good is the model in finding and properly classifying the CAPTCHAs on the webpage.

c) *F1* score

F1 score is the combination of *Precision* and *Recall* to give the overall view on model performance. The equation that defines the *F1* score is the following:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4}$$

Achieving the high *F1* score can be interpreted as the model being able to correctly detect and classify the class instances, without missing or misclassifying the objects. For the CAPTCHA detector, *F1* score indicates that the CAPTCHAs are properly localized and classified. *F1* can also signal if the different CAPTCHA types are correctly distinguished from each other and if webpage elements are not misclassified as the CAPTCHAs.

d) *Mean Average Precision*

Mean Average Precision (*mAP*) is the metric that indicates the balance between *Precision* and *Recall*. It can be done by calculating the mean value of the area under the *Precision-Recall* curve at different levels of *Recall* for each detection class. *mAP* is defined by formula (5):

$$mAP = \frac{1}{n} \times \sum_{k=1}^{k=n} AP_k \tag{5}$$

Where n equals to number of classes and AP_k is average *Precision* for class k. For the CAPTCHA detector *mAP* defines how good the task of detecting and classifying the CAPTCHAs is done across all of the detection classes. The metric used in experiments is mAP@50 which means that mAP is calculated considering all the detections that exceeds an intersection over union (*IoU*) of 0.50.

e) *Inference speed*

Inference speed is a time that the trained model needed for a single detection.

IV. Experimental Results and discussion

A. Trained model performance

Table II provides the metrics for each neural network model tested with Set 1 (the known CAPTCHA patterns). Letters *'n'*, *'s'*, and *'m'* stand for YOLO model type (nano, small, and medium), respectively. Additionally 'YOLOv5un320' represents the parameters for YOLOv5 model with an input size 320x320 [px], instead of 640x640 [px] (input size of other models). Results are very similar and almost perfect. The differences in detection score between each pair of the models are marginal between 2 or 3 false detections. Such an outcome may be interpreted as a warning signal about overfitting. In such case, the results in Table III as well as comparison of the confusion matrices in Fig. 8 and Fig. 9 were crucial for the experiment as they pointed the potential vulnerabilities. While the detection of the new CAPTCHA patterns for image class is good, there is a visible drop in performance of other classes (button, text, and puzzle).

YOLOv5us and YOLOv10s models seems to be the best in terms of quality of detection, but they are slower than YOLOv8s and YOLOv5un. In terms of speed both YOLOv5 and YOLOv8 perform similar but YOLOv5un performed slightly better in tests on Set 2. Halving the input size of the YOLOv5un (to 320 px) doubled the detection speed without the performance drop.

B. Image slicing test

To check if proposed input slicing method can increase the performance of the model, the images from the Set 2 with a width or height higher or equal to 1920 px (three times neural network model input size) have been sliced using the formula (equation (1)) from Section III.B.

30 images that exceed 1920 px threshold have been obtained from Set 2. After slicing the images, we tested the response of the neural network based on model YOLOv8m for sliced and unsliced images, calculating TP, FP, and FN metrics. Results are presented in Table IV. It can be observed that the image slicing can be a response to the problem of oversized input images, but it is important to keep in mind that such a method can have a huge impact on the model speed. The detection has to be done for each image slice. There is also a small risk of creating false positive detections.

TABLE IV: Detection results on sliced images

Mode (sliced/unsliced)	TP	FN	FP
Unsliced	20	10	0
Sliced	25	5	1

C. Tuning the network

Analysis of the result tables (Table II and Table III) and confusion matrices (Fig. 8 and Fig. 9) with the succeeding reanalysis of the training dataset pinpointed to the problem of puzzle CAPTCHA instance representation being not enough diverse (this type is used rather seldom) and also, as expected, some of the new patters of the text CAPTCHAs were not recognized by the trained neural network (cf. Fig. 9).

As a possible solution, the additional training session was conducted to tune the network for analyzing new instances of text CAPTCHAs. The portion of 1,000 images of the new instances of text CAPTCHAs was mixed with the random images from the previous training dataset (cf. Section III.A), resulting in a new set of 34,304 images, divided into the training and validation subsets (26,154/8,150). The number of images from the previous training included in this retraining dataset was selected as a result of experiment. Training on such a dataset made it possible to classify the new text CAPTCHA patterns without the significant decay in recognizing the patterns learned before. Training was conducted for 50 epochs. Table V and Table VI show the network performance metrics for the retrained neural network (Table V) and the neural network without this additional tuning (Table VI), respectively.

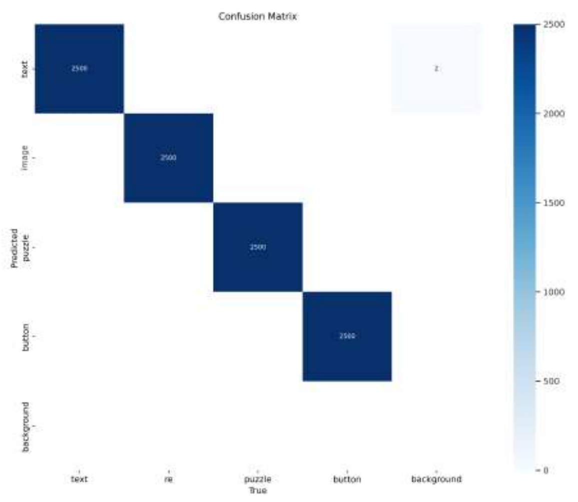


Fig. 8: Confusion matrix (referring to YOLOv8m) as a result of the test on the Set 1.

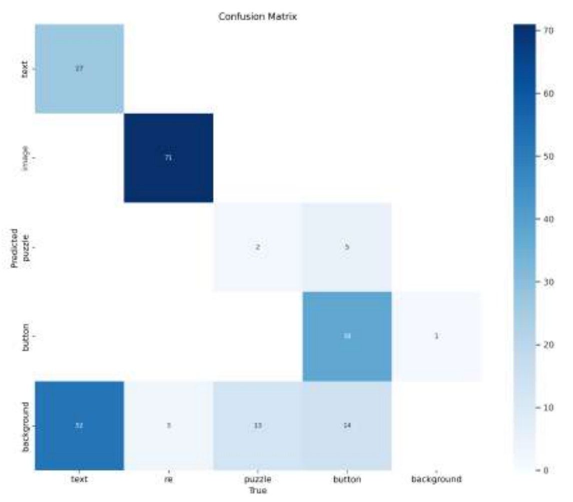


Fig. 9: Confusion matrix (referring to YOLOv8m) as a result of the test on the Set 2.

TABLE II: Test results for Set 1 (known CAPTCHA patterns)

Model	YOLOv5 un320	YOLOv5un	YOLOv5us	YOLOv8n	YOLOv8s	YOLOv8m	YOLOv10n	YOLOv10s	YOLOv10s cos	YOLOv10m
Precision	0.998	0.998	0.999	0.999	0.999	0.999	0.994	0.999	0.998	0.998
Recall	0.999	0.999	0.999	1	1	1	0.994	1	0.999	0.999
F1 score	0.998	0.998	0.999	0.999	0.999	0.999	0.994	0.999	0.998	0.998
mAP	0.995	0.995	0.995	0.995	0.995	0.995	0995	0.995	0.995	0.995
Inference speed	0.4 ms	0.8 ms	1.8 ms	0.8 ms	1 ms	4.9 ms	1 ms	2.4 ms	2.4 ms	5.1 ms

TABLE III: Test results for Set 2 (unknown CAPTCHA patterns)

Model	YOLOv5 un320	YOLOv5un	YOLOv5us	YOLOv8n	YOLOv8s	YOLOv8m	YOLOv10n	YOLOv10s	YOLOv10s cos	YOLOv10m
Precision	0.957	0.957	0.959	0.955	0.951	0.958	0.942	0.958	0.911	0.957
Recall	0.596	0.596	0.618	0.569	0.6	0.613	0.573	0.609	0.636	0.591
F1 score	0.735	0.735	0.752	0.713	0.736	0.748	0.713	0.745	0.749	0.731
mAP	0.653	0.653	0.672	0.644	0.632	0.665	0.664	0.670	0.665	0.665

TABLE V: Test results of the retrained models

Model/testing set	YOLOv5un320 Set 1	YOLOv8m Set 1	YOLOv5un320 Set 2	YOLOv8m Set2
Precision	0.999	0.999	0.953	0.963
Recall	0.998	1	0.653	0.741
Precision for text class	0.998	0.996	1	1
Recall for text class	0.993	1	0.434	0.583

TABLE VI: Test results without the retraining

Model/testing set	YOLOv5un320 Set 1	YOLOv8m Set 1	YOLOv5un320 Set 2	YOLOv8m Set2
Precision	0.998	0.999	0.957	0.958
Recall	0.999	1	0.596	0.613
Precision for text class	0.998	0.999	0.818	1
Recall for text class	0.993	1	0.217	0.341

Retraining of the network not only increased the number of true positive detections of the text class in Set 2 for both neural networks, but also eliminated the false positive detections of the text class for YOLOv5un320. A small drop of metrics can be observed for detection on Set 1 but it is a small cost of learning new text CAPTCHA patterns.

D. Performance comparison of the YOLO models

If the inference speed is omitted as a less important metric related to the hardware computing power, the ML models developed can be compared focusing on the other performance metrics listed in Section III.D and evaluated in Table III. For this purpose a weighted arithmetic mean has been calculated using the following weights assigned to the metrics: F1: 50%, mAP: 25%, Precision: 12.5%, and Recall: 12.5%. According to this value a ranking of the models, starting from the best one, is: YOLOv5us, YOLOv8m, YOLOv10s, YOLOv10scos, YOLOv10m, YOLOv5un320, YOLOv5un, YOLOv8s, YOLOv10n, and YOLOv8n. When two models were retrained (cf. Section IV.C), an average improvement for both Sets was 4% for Recall and 44% for Recall (cf. Table V and Table VI).

V. Conclusion and steps ahead

In this study we examined the capability and performance of different YOLO models in the task of detecting and classifying four distinct CAPTCHA types. This was motivated by the need to enhance the capabilities of the darknet web-crawler with the tool for detecting and solving text-CAPTCHAs to be able to explore the darknet more freely in search for the information of interest. Our analysis involved various metrics such as *Precision*, *Recall*, *F1-score*, *mAP*, and inference time.

To overcome the issue of not enough good quality data, we created the synthetized training data of webpages protected by CAPTCHA code. We tested the trained neural networks both on synthetized data and real-life scenario that included new CAPTCHA patterns. While the detection of the image CAPTCHA class was efficient for both synthetized and real-life specimens, the metrics for other classes were worse. The poor results of real-life detection resulted from the high variety of different puzzle and text CAPTCHA patterns. To overcome this issue an additional training with new text CAPTCHA patterns present in the real-life dataset has been done to increase the confidence of the model in recognizing CAPTCHA patterns.

Conducted experiment proved that it is possible to tune the network for recognizing new CAPTCHA patterns using the small amount of data (in the experiment 1,000 synthetized webpage images) but it is important to keep old CAPTCHA patterns in the training and validation sets to minimize the risk of learning new patterns while forgetting the old ones.

The proposed slicing solution for dealing with oversized images tends to be working efficiently and can increase the number of true positive detections in cost of slower response. As a way of enhancing this method, the real-life websites can be examined for the most common location of the CAPTCHAs on the webpage (top, middle, and bottom). With statistics like these above, sliced images could be analyzed starting from the slice where there is the highest probability of the CAPTCHA occurring.

Different YOLO models analyzed differ mostly in better performance metrics in cost of speed. The final model to be chosen as recommended one depends on the application. In the real-time environment YOLOv5un and YOLOv8n outscore other models because of their speed. On the other hand, in the application where *Precision* and *Recall* metrics are more valid than speed, other models such as YOLOv8m and YOLOv10s will work better.

It is possible to train the network for recognizing and localizing the CAPTCHA types on the webpage using YOLO models, but the diverse dataset is crucial for obtaining good results. While it is possible to train the model on the synthetized data as well as to work with real-life specimens, performance will mostly depend on diversity of CAPTCHA patterns used for training. That is why an important part of building the CAPTCHA classification and recognition model should be a constant collecting of the CAPTCHA data for future network tuning.

Acknowledgment

The research leading to these results has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No. 101073876 - CEASEFIRE project [37].

References

[1] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford, "Captcha: Using hard ai problems for security," in *Advances in Cryptology — EUROCRYPT 2003*, E. Biham, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 294–311.

[2] K. Chellapilla and P. Y. Simard, "Using machine learning to break visual human interaction proofs (hips)," *Advances in neural information processing systems*, 2004.

[3] E. Bursztein, J. Aigrain, A. Moscicki, and J. C. Mitchell, "The end is nigh: Generic solving of text-based captchas," 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:255684275

[4] J. Wang, J. Qin, X. Xiang, Y. Tan, and N. Pan, "Captcha recognition based on deep convolutional neural network," *Mathematical Biosciences and Engineering*, vol. 16, no. 5, pp. 5851–5861, 2019. [Online]. Available: https://www.aimspress.com/article/doi/10.3934/mbe.2019292

[5] D. Wang, M. Moh, and T.-S. Moh, "Using deep learning to solve google recaptcha v2's image challenges," 01 2020, pp. 1–5.

[6] A. Plesner, T. Vontobel, and R. Wattenhofer, "Breaking recaptchav2," in *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*, vol. 14. IEEE, Jul. 2024, p. 1047–1056. [Online]. Available: http://dx.doi.org/10.1109/COMPSAC61105.2024.00142

[7] V. P. Singh and P. Pal, "Survey of different types of captcha," 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:18390330

[8] A. M. Algwil, "A survey on captcha: Origin, applications and classification," *Journal of Basic Sciences*, vol. 36, no. 1, p. 1–37, Jun. 2023. [Online]. Available: https://journals.asmarya.edu.ly/jbs/index.php/jbs/article/view/208

[9] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. [Online]. Available: http://arxiv.org/abs/1506.02640

[10] P. Jiang, D. Ergu, F. Liu, C. Ying, and B. Ma, "A review of yolo algorithm developments," *Procedia Computer Science*, vol. 199, pp. 1066–1073, 02 2022.

[11] S. Y. and X. Y., "End-to-end captcha recognition using deep cnn-rnn network," in *2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*. IEEE, 2019.

[12] Z. N., E. M., L. W., and C. H., "A generative adversarial learning framework for breaking text-based captcha in the dark web," in *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 2020.

[13] M. I.G., Y. Z., and B. V., "Breaking captcha with capsule networks," *Neural Networks*, vol. 154, 2022.

[14] Z. Noury and M. Rezaei, "Deep-captcha: a deep learning based CAPTCHA solver for vulnerability assessment," *CoRR*, vol. abs/2006.08296, 2020. [Online]. Available: https://arxiv.org/abs/2006.08296

[15] J. S. Walia and A. Odugoudar, "Vulnerability analysis for captchas using deep learning," in *2023 IEEE International Conference on ICT in Business Industry & Government (ICTBIG)*, 2023, pp. 1–7.

[16] "Selenium web driver," acessed:2024-09-18. [Online]. Available: https://github.com/SeleniumHQ/selenium/

[17] "Top 10 million websites," https://www.domcop.com/top-10-million-websites, accessed: 2024-09-18.

[18] "Webpage element detection - coco json," acessed:2024-09-18. [Online]. Available: https://www.kaggle.com/datasets/desolationofsmaug/webpage-element-detection

[19] "Webpage elements annotated data," acessed:2024-09-18. [Online]. Available: https://www.kaggle.com/datasets/nilanjandebnath/webpage-elements-annotated-data

[20] "Amazon captcha," acessed:2024-09-18. [Online]. Available: https://docs.aws.amazon.com/waf/latest/developerguide/waf-captcha-puzzle.html

[21] "Weibo webpage," acessed:2024-09-18. [Online]. Available: https://m.weibo.cn/

[22] "Wikipedia captcha," acessed:2024-09-18. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Special:CreateAccount&returnto=Main+Page

[23] "Shopee philippines," acessed:2024-09-18. [Online]. Available: https://shopee.ph/

[24] "Puzzle captcha dataset," https://www.kaggle.com/datasets/riybot/puzzle-captcha-dataset, accessed: 2024-09-18.

[25] "Captcha collected from darknet websites," https://www.kaggle.com/datasets/njznjz/captcha-collected-from-darknet-websites, accessed: 2024-09-18.

[26] "Captcha images," accesed:2024-09-18. [Online]. Available: https://www.kaggle.com/datasets/aadhavvignesh/captcha-images

[27] "Captcha images dataset," accesed:2024-09-18. [Online]. Available: https://www.kaggle.com/datasets/amroelgar7ay/captcha-images-dataset

[28] R. Wilhelmy and H. Rosas, "captcha dataset," 07 2013.

[29] "Captcha dataset," accesed:2024-09-18. [Online]. Available: https://www.kaggle.com/datasets/parsasam/captcha-dataset

[30] "Captcha recognition 2.0," https://universe.roboflow.com/samip-timalsena/captcha-recognition-2.0, accessed: 2024-09-18.

[31] "Recaptcha seg 2," acessed:2024-09-18. [Online]. Available: https://universe.roboflow.com/joao-escribano/recaptcha-seg2

[32] "Slide captcha," acessed:2024-09-18. [Online]. Available: https://universe.roboflow.com/jarvis-umwee/slide_captcha-0kux4

[33] "Pycaptcha," acessed:2024-09-18. [Online]. Available: https://github.com/DrMint/PyCaptcha

[34] "dperson/torproxy," accesed:2024-09-18. [Online]. Available: https://hub.docker.com/r/dperson/torproxy/

[35] "Cloudflare captcha," acessd: 2024-09-18. [Online]. Available: https://www.cloudflare.com/products/turnstile/

[36] "hcaptcha," acessed:2024-09-18. [Online]. Available: https://accounts.hcaptcha.com/demo

[37] "Ceasefire project page," accesed:2024-09-18. [Online]. Available: https://ceasefire-project.eu/

[38] G. Jocher, A. Chauraisa, and Q. Jing, "Ultralitics yolo." [Online]. Available: 'https://ultralytics.com' and 'https://github.com/ultralytics/ultralytics'

[39] P. Zamboni, J. M. Junior, J. d. A. Silva, G. T. Miyoshi, E. T. Matsubara, K. Nogueira, and W. N. Gonçalves, "Benchmarking anchor-based and anchor-free state-of-the-art deep learning methods for individual tree detection in rgb high-resolution images," *Remote Sensing*, vol. 13, no. 13, 2021. [Online]. Available: https://www.mdpi.com/2072-4292/13/13/2482

[40] M. Sohan, T. Ram, and V. Ch, *A Review on YOLOv8 and Its Advancements*, 01 2024, pp. 529–545.

[41] G. Jocher, “Yolov5 by ultralytics.” [Online]. Available: ”https://github.com/ultralytics/yolov5 and https://docs.ultralytics.com/models/yolov5/

[42] A. Wang, H. Chen, L. Liu, K. Chen, Z. Lin, J. Han, and G. Ding, “Yolov10: Real-time end-to-end object detection,” 2024. [Online]. Available: https://arxiv.org/abs/2405.14458