UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Anette Habanen

Deepfakes for Paper Vote Privacy Defence

Master's Thesis (30 ECTS)

Supervisors:
Jan Willemson, PhD
Sven Laur, DSc. (Tech)

Tartu 2025

Deepfakes for Paper Vote Privacy Defence

Abstract:

The recent rise of artificial intelligence (AI) solutions has also had a significant impact on electoral processes. Most notably, deepfakes created by generative AI applications can (and have been) used to spread misinformation during the campaigns, but they can also be used for cyberattack automation, biased social media bots, etc. This thesis instead presents a positive use case for generative AI in manipulating video material required as proof of voting by potential coercers. For this, I have created a pipeline that takes a video of a voting ballot and replaces its critical content (in our case, the digits on the ballot). To achieve this, a YOLO model is used to find the digits, a WavePaint image inpainting model is used to cover up the old digits, and a separate image of the new digits is used to place it into the video. Additionally, I have implemented the prototype application in the form of a webpage.

Keywords: Artificial intelligence, machine learning, deepfakes, elections, voting ballot

CERCS: P176 Artificial intelligence; T111 Imaging, image processing; P170 Computer science, numerical analysis, systems, control

Süvavõltsingud paberhääletuse privaatsuse kaitseks

Lühikokkuvõte:

Viimaste aastate jooksul on tehisintellekti levik hakanud mõjutama valimisprotsessi. Peamiselt on generatiivse tehisintellekti poolt loodud süvavõltsingud põhjustanud valeinformatsiooni levitamist valimisperioodi jooksul. Samuti saab tehisintellekti kasutada küberrünnakute automatiseerimiseks ja kallutatud arvamusega sotsiaalmeedia postituste loomiseks. Käesolev magistritöö toob esile positiivse generatiivse tehisintellekti kasutusvaldkonna ning seda olukorra jaoks, kus potensiaalne ründaja nõuab tõestusmaterjali hääletamisprotsessist hääletuskabiinis. Selle jaoks koostasin programmi, kus võetakse hääletaja poolt filmitud video ning asendatakse videos hääletaja hääl ründaja sooviga. Selle saavutamiseks kasutasin YOLO mudelid numbrite leidmiseks, WavePaint mudelit numbrite kinni katmiseks ja eraldi pilti ründaja soovist, et asetada see sedelile. Programmi jaoks implementeerisin veebilehe prototüübi.

Võtmesõnad: Tehisintellekt, masinõpe, süvavõltsing, hääletamine, hääletussedel

CERCS: P176 Tehisintellekt; T111 Pilditehnika; P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

1. Introduction

Elections are a core mechanism to implement representative democracy in modern societies. Over 50 percent of countries can be categorized as democratic[1], and around 30 percent of the world's population (about 2.3 billion people) live in these countries[2].

Over centuries, many techniques for vote casting and counting have been experimented with [1]. For example, in the US, voting was long considered a public event, and hiding one's preference was considered non-democratic. However, announcing the preferred candidate out loud led to significant problems with coercion (e.g. vote buying) [2]. Thus the current main method of filling in an anonymous paper ballot in the privacy of a polling booth was introduced in mid-19th century Australia, and from there it gradually spread across the world [3].

In 2014, a new voting-related social media phenomenon started to emerge, where people started to photograph themselves in the voting booth with their voting ballots and upload it to social media [4]. The phenomenon is known as a ballot selfie or *stemfie* (a Dutch portmanteau of 'stemmen' – 'to vote' and 'selfie'). Although sharing the excitement of voting and encouraging other people to vote can be a good thing, it can also be used against the voter. Similar to 19th-century US, the coercer can force the voter to reveal their preference, this time in the form of a stemfie.

Disputes over stemfie legality and the equilibrium between freedom of speech and coercion resistance are currently ongoing in the legal community [5]. However, from the technical perspective, it is clear that even if the stemfies get outlawed, their use is rather hard to detect in the polling stations because the voting act should be private from the eyes of the polling station staff.

Thus, recovery methods are needed. In case the coercer requires a still image, an image editor like GIMP or Photoshop can be used to fake the proof. However, the situation is more complex if the coercer demands a video. Going over the recording frame by frame is a tedious task, and the result can be so inconsistent that the fake is easily detected by the attacker.

---

[1] Source: https://ourworldindata.org/democracy

[2] Source: https://ourworldindata.org/democratic-rights

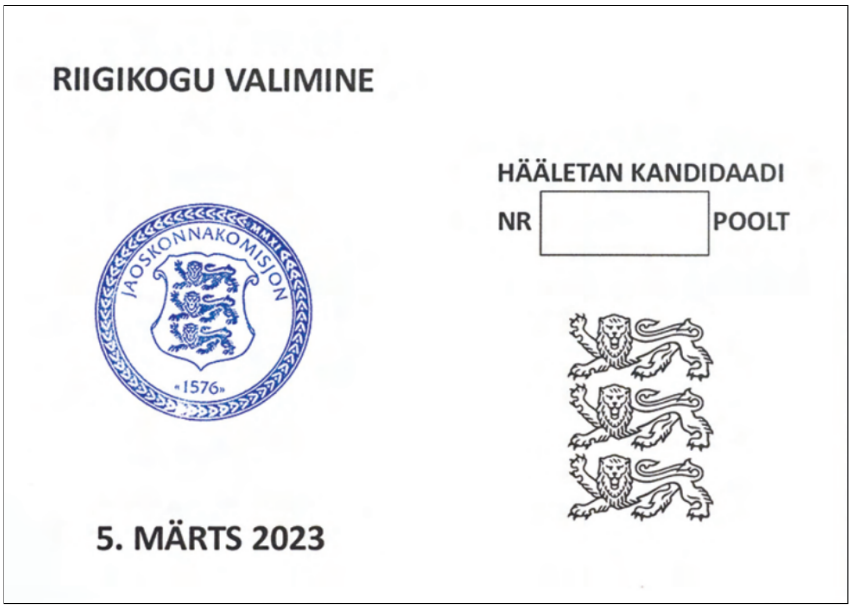


Figure 1. Example of a voting ballot in Estonia, where the candidate is selected by entering a 3-digit code of the candidate into the empty box[4].

In this thesis I am going to concentrate on ballots where the voter's preference is expressed by writing some numbers. In Estonia, for example, the voter writes the candidate's number, and in several other countries the ballot is filled by numbering the candidates in the order of preference[3]. For concreteness, we are using Estonian ballot sheets where 3-digit candidate numbers are written into the predetermined box. An example of these ballots is shown in Figure 1.

The aim of this thesis is to create a pipeline where we give it a video filmed of the voting ballot and an image of the new 3-digit code we want to replace in the ballot box. The pipeline should be able to create a convincing video with the new vote to deceive the potential attacker. The use case for this deepfakes pipeline is shown in Figure 2. In addition, the aim is to create a prototype of a webpage showing how this pipeline could be used in a real-life scenario.

The thesis is organized as follows. Chapter 2 covers the ways AI has been used in the election process and introduces different types of machine learning models. Chapter 3 describes the methods to process the video and ways to get a video with a different vote on the ballot. Chapter 4

[3]Estonian voting process: https://www.valimised.ee/et/valimiste-meelespea/valimispaeval-haaletamine/valimispaeval-valimisjaoskonnas-haaletamine

[4]Voting ballot image source: https://www.valimised.ee/sites/default/files/2023-02/17.02.2023_Lihtne%20keel%20rk%20valimised%202023%20%28002%29_0.pdf

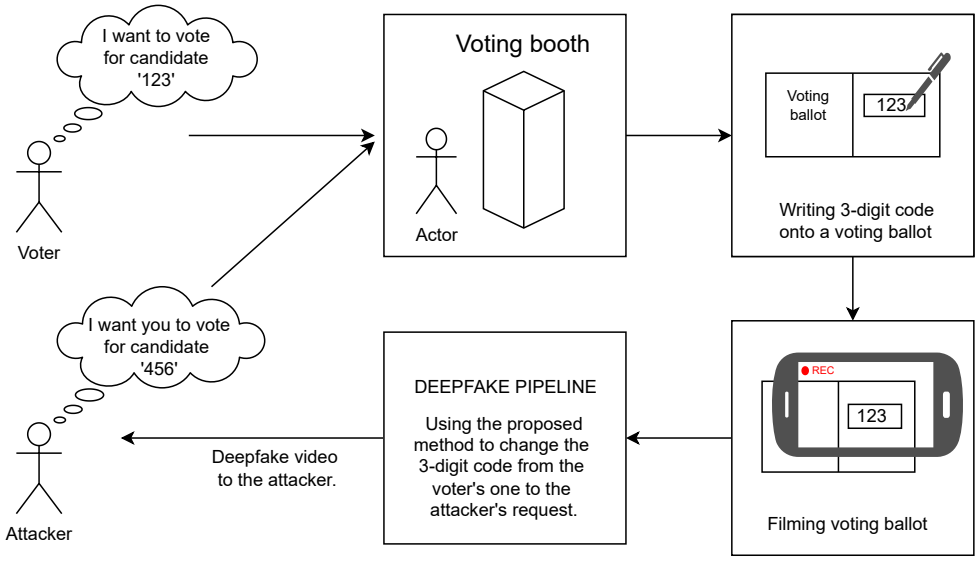


Figure 2. Use case for the created pipeline. The attacker says to the voter how they should vote. The voter goes to the voting booth, writes down their own preference, and films a video of the ballot. Then the voter uses the pipeline to create a video where the attackers preference is shown. The voter sends the video to the attacker as proof of how they voted.

shows the results and the evaluation of the methods, and Chapter 5 gives recommendations for future improvements.

2. Background

In this chapter, I am first going to introduce how AI is used and has impacted the electoral process. Then we take a look at multiple machine learning models. Lastly, I will present a couple of datasets used later in the thesis.

2.1 AI in elections

Recent research has identified several areas where the effect of AI on elections can already be observed. Of course, this effect can be both positive and negative.

On the positive side, AI can

- help maintain and cross-check voter rolls [6],
- quickly process huge numbers of voter registration forms [7],
- detect election fraud by looking for patterns in election data [7],
- tabulate hand-marked ballots and verify signatures on mail-in ballots [8],
- help under-resourced candidates to create campaign content [9],
- run voting advice applications, helping the voters to get a clear view about the elections and their options [10].

On the negative side, AI can be used to

- create deepfakes and other kinds of misinformation campaign materials [8, 9, 11],
- unduly influence the political debate using social media bots [12],
- automate cyberattacks on election infrastructure [8],
- undermine voter confidence in election integrity [13].

As already mentioned in Chapter 1, this thesis considers a novel approach to the usage of generative AI to fake a video of the ballot to protect the voter from undue pressure in potential coercive scenarios. As a lot of the media mostly focuses on the cases where AI is used negatively, it is important to highlight the ways it can have a positive effect.

2.2 Machine learning models

2.2.1 Object Detection

Object detection models are used to find objects in images or videos (frame by frame). The model outputs coordinates of a box where it thinks the object is located. There are many models available for object detection, and they can be separated into two categories: single-stage and multi-stage (two-stage) models [14]. The two-stage models use two models, one for extracting regions of objects and the other one for classifying the object and refining the location[5]. Most models in this category are CNNs (Convolutional Neural Networks), for example, R-CNN [15], Fast R-CNN [16] and Mask R-CNN [17]. The single-stage models only use one model and skip the first region proposal stage[6]. The most well-known and well-performing models in this category belong to the YOLO series. The main drawback of the two-stage models is that they are slow [14]. This means that single-stage models are more appropriate for real-time applications [14].

The first YOLO model was introduced in 2016 [18] and after that they have kept improving their model and adding new features with every new release. The earlier version mainly focused on object detection, but later versions have added segmentation, oriented object detection, and pose estimation[7]. With every new release, they also provide pretrained models with various parameter sizes (for example, model YOLO11m has 20.1 million parameters)[8]. One of the latest versions is YOLOv11 [19] which was optimized for efficiency and speed.

2.2.2 Image generation

There are multiple types of models that can generate new images. Text-to-image models generate new images based on the text input and keywords in it. Some of the examples are Dall-E [20], Imagen [21], Latent Diffusion Models [22], and Midjourney[9]. Some text-to-image models also have the opportunity to add an image to your text input to give additional information about what

---

[5]A guide to Two-stage Object Detection: https://medium.com/codex/a-guide-to-two-stage-object-detection-r-cnn-fpn-mask-r-cnn-and-more-54c2e168438c

[6]One-Stage Object Detection Models: https://paperswithcode.com/methods/category/one-stage-object-detection-models

[7]YOLO11: https://docs.ultralytics.com/models/yolo11/

[8]YOLO model (object detection): https://docs.ultralytics.com/tasks/detect/

[9]Midjourney: https://www.midjourney.com/home

the output should look like. One of the main examples in this category is Stable Diffusion[10], which is developed based on Latent Diffusion Models [22]. There are also image-to-image models, which take an image as an input and generate a new image based on that. Examples of these models are the GAN model [23], Pix2Pix [24], and WavePaint [25].

WavePaint [25] is a 'resource-efficient token-mixer for self-supervised inpainting'. Inpainting model means that we give the model an image with part of it covered up (masked). Then it is the model's task to fill in the places where the mask is with realistic results. When creating it, the WavePaint authors specifically worked on large mask inpainting where the model has less info about the surrounding pixels [25]. This model was created with the intention of being resource-efficient and parameter-efficient [25]. The main evaluation metric that the paper suggests is Learned Perceptual Image Patch Similarity (LPIPS) [26]. The other metric that is calculated throughout the training process is the Structural Similarity Index (SSIM) [27].

2.2.3 Digit recognition and generation

There is a significant body of research on the recognition of handwritten digits, and most of them are based on the MNIST dataset (see Chapter 2.4.1) [28–30]. Unfortunately, most of the work concentrates on classification and not object detection. There is a project called PaddleOCR[11] (OCR means Optical Character Recognition) that is multilingual and can recognize letters and digits. But in this thesis I need a model that can only recognize handwritten digits; hence, PaddleOCR would give too much unnecessary information. There are also works that explore creating new images of digits using machine learning [31, 32]. These digit generation approaches are applicable on the image level. However, generating smooth and consistent images (frames) to create a video is a lot harder.

2.3 Corner detection and tracking

Corner detection is a method that is mainly used for feature tracking [33]. Corners are very distinctive and invariant, so they are good features to track [33]. Because in this thesis I use videos, it is needed that the used method also be able to do object tracking. For this, usually optical flow is calculated. Optical flow analyses the motion in a video and computes the changes

[10] Stable Diffusion: https://github.com/Stability-AI/stablediffusion
[11] PaddleOCR: https://github.com/PaddlePaddle/PaddleOCR

of the pixel location[12]. This technique is based on pixel colour brightness and intensity[12]. One famous method that uses optical flow and is able to track objects in a video is called the Lucas-Kanade method [34]. Depending on the implementation, this method is either able to find random corners in the video and track them throughout. Or it is possible to give it predefined corners (pixel coordinates), and it is able to find their location in all subsequent frames.

2.4 Datasets

To train models, a good dataset containing handwritten digits is needed. However, the available datasets are usually black and white and do not have a lot of variety in them. In the real world, the environment does not have just two colours, and because of that, I had to create my own dataset featuring more diverse and colourful backgrounds. For digits I used the EMNIST Digits dataset, and for the background I used the Open Images dataset. Examples of these datasets are shown in Figure 3.

2.4.1 EMNIST digits

One of the well-known datasets in the machine learning industry is the MNIST digits dataset [35]. It consists of 28×28 pixel black and white images of handwritten digits (white digits on a black background, as seen in Figure 3(a)). MNIST is part of a bigger dataset called EMNIST [36] that consists of handwritten characters. The EMNIST dataset also features a split called EMNIST Digits. The main difference between MNIST and EMNIST Digits is the size: MNIST consists of 70,000 digits, and EMNIST Digits has 280,000 [36]. This makes it one of the biggest datasets for handwritten digits. In this thesis the EMNIST Digits dataset is used to train the object detection model YOLO to recognise and find handwritten digits from videos. For this the image colours are flipped, i.e., the digits are black and the background is white, as seen in Figure 3(b). The EMNIST Digits dataset can be acquired using the TensorFlow library[13].

2.4.2 Open Images dataset

The Open Images dataset [37] is a big collection of annotated images (classes, bounding boxes, segmentations), featuring everyday items and locations. The dataset has about 9,000,000 images, which can be filtered based on the annotations. Examples of the images in the dataset can be

[12]Optical Flow and Lucas-Kanade method: https://medium.com/@guptachinmay321/an-insight-into-optical-flow-and-its-application-using-lucas-kanade-algorithm-c12f9f6e3773

[13]Tensorflow library (EMNIST dataset): https://www.tensorflow.org/datasets/catalog/emnist

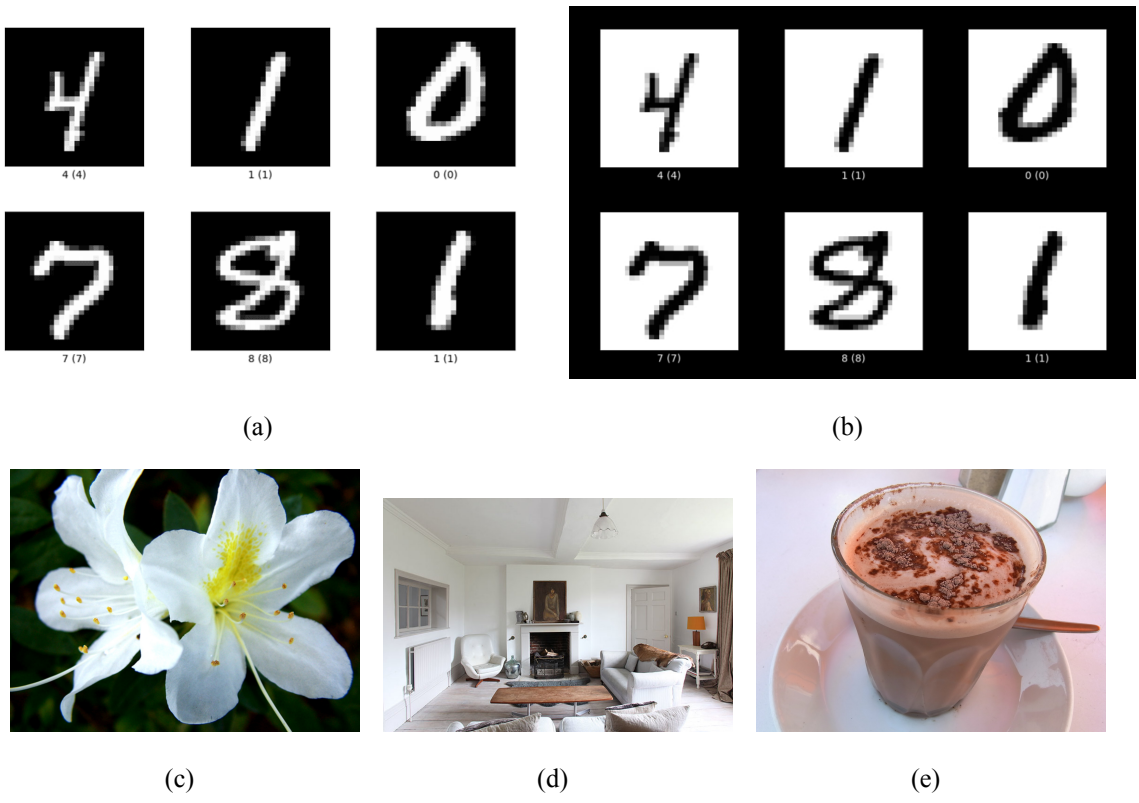


Figure 3. Example images from the datasets. (a) Original images from the EMNIST Digits dataset. (b) EMNIST Digit dataset examples with colours flipped for the YOLO training dataset. (c-e) Images from the Open Images dataset.

seen in Figures 3(c), 3(d) and 3(e). In this thesis, I used the dataset for two tasks: for object detection model YOLO training and image generation model Wavepaint training. The Open Images dataset can be easily acquired through the FiftyOne library[14].

[14]FiftyOne library integration with Open Images: https://docs.voxel51.com/integrations/open_images.html

3. Methodology

The task of changing the digits in the video frames is separated into three main tasks. First, I use an object detection model to find the digits from the video. Second, an image generation model is used to generate a new background and cover up the old digits. And lastly, I use image perspective transform and pixel tracking to place the new digits into the frames. Besides providing the video of the voting ballot, three additional things need to be given to the pipeline: an image of the new 3-digit code, the location of the code in the image and the corner coordinates for the vote box in the first frame of the video.

3.1 Corner detection

The first step is to use a corner detection and tracking algorithm. I use the Lucas-Kanade method [34] and its implementation[15] to track the vote box corners throughout the video. For this, OpenCV has a function called *calcOpticalFlowPyrLK* that is able to find the corners of the box based on the previous frames' data. To start finding the corners, we need to provide the first frame corner information to the algorithm. After running this algorithm, we know where the box is located throughout the video in every frame.

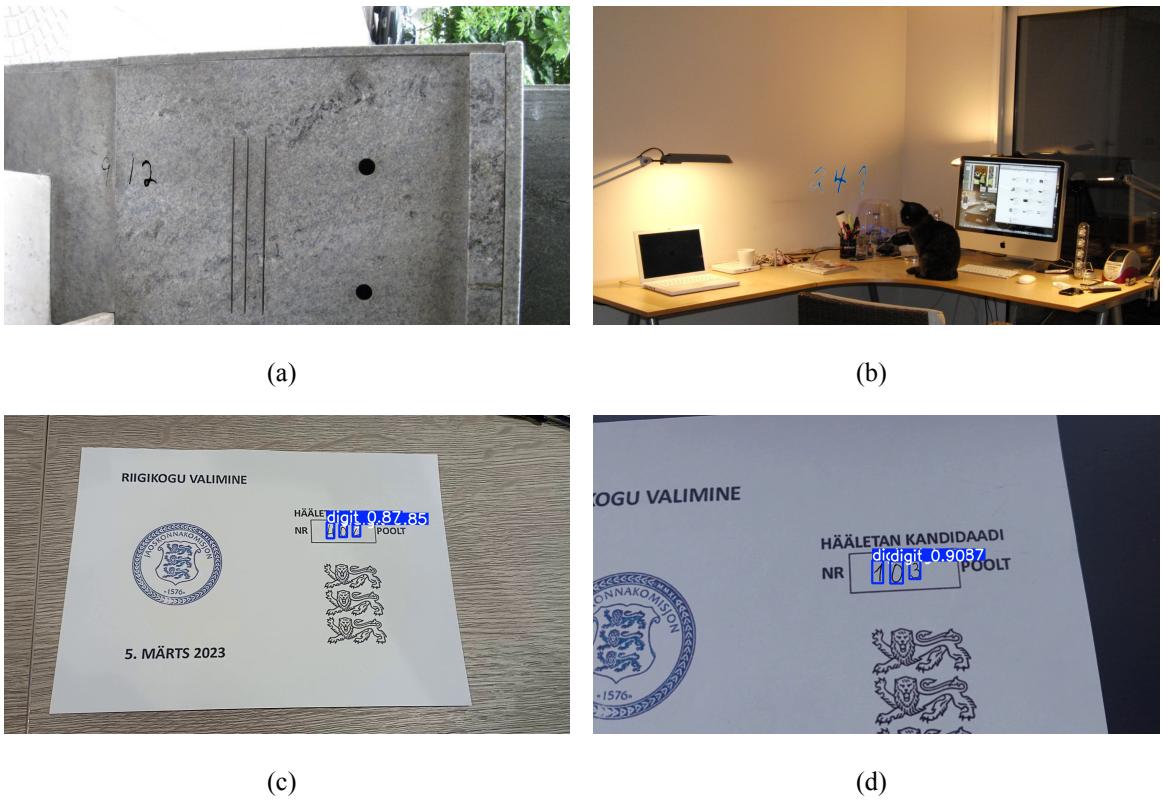
3.2 Object detection

I use an object detection model to find the 3-digit code from every frame of the video. I chose the YOLOv11 model to perform the object detection task.

3.2.1 Dataset for training YOLOv11 model

For training the YOLOv11 model to recognise the digits in videos, I created a dataset of Open Images dataset with EMNIST Digits characters. Three digits were randomly selected, placed next to each other and the colours of the digits were changed (either black or blue to represent common pen colours). This dataset was created using already existing code that added multiple MNIST digits together into one image [38]. I did some modifications with regard to changing the import of EMNIST Digits and added the color change for the digits. Additionally, I added to this code finding and saving bounding boxes of digits. Because the original EMNIST Digits

[15] Source code from the University of Tartu course called "Intelligent Transportation Systems": https://courses.cs.ut.ee/2024/ITS/fall/Main/Practicals?action=download&upname=Lab03_Fall24.zip



(a) (b)

(c) (d)

Figure 4. (a-b) Created images for YOLO model training. (c-d) Predictions from the trained YOLO model for video frames.

dataset is black and white, it is very easy to find the ground truth bounding boxes for every digit in the dataset using colour threshold.

To create the final training dataset for YOLO, image of 3 digits was selected, randomly resized and placed onto the Open Images dataset image. Two example images from the training dataset are shown in Figures 4(a) and 4(b). Based on the info from the YOLO developers[16], the dataset should have over 10 000 instances per class and over 1500 images per class. Because this dataset has 10 different digits, the final train dataset had 35 000 images, which means it had about 10 500 instances per class and 3500 images per class. The validation dataset had 3500 images. The image size was 1920×1080 pixels, which gives a 16:9 aspect ratio. I chose this aspect ratio because, by default, smartphones capture videos with this aspect ratio and specifically with the resolution of 1920×1080 pixels[17].

[16]Info from the developers of YOLO about dataset: https://github.com/ultralytics/yolov5/issues/5851
[17]Aspect ratio: https://www.digitalsamba.com/blog/video-aspect-ratio

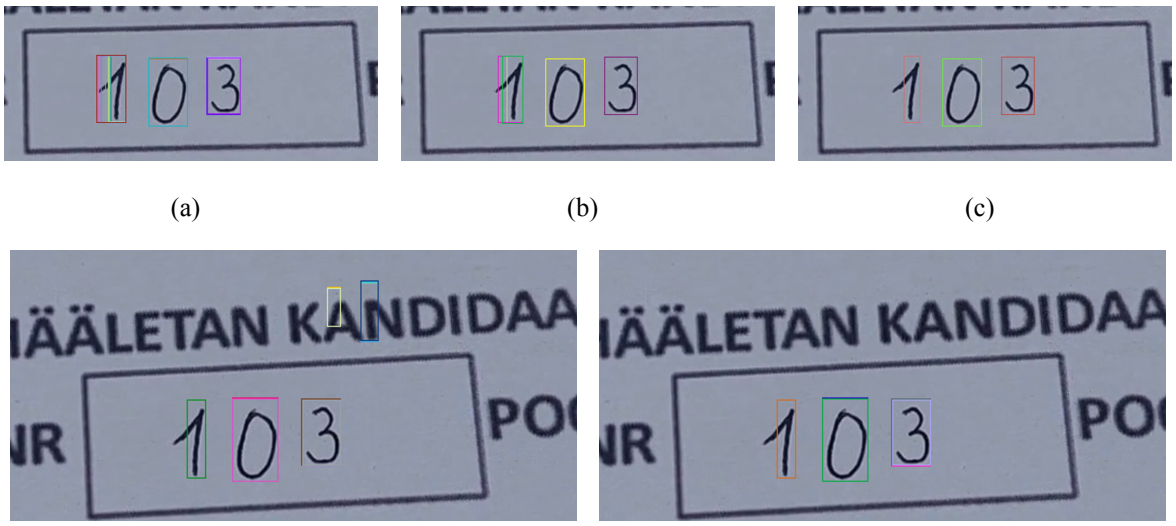


Figure 5. YOLO predictions for an image before and after postprocessing. (a) YOLO predictions when the IoU=1 and the confidence=0.2. (b) YOLO predictions when the IoU=0.8 and the confidence=0.2. As we can see, a lot of overlapping bounding boxes were removed. (c) YOLO predictions when the IoU=0.8 and the confidence=0.5. With this, the bounding boxes with low confidence are removed. (d) YOLO predictions before postprocessing. (e) YOLO predictions after using the Lucas-Kanade method and filtering out bounding boxes that were not in the voting box.

3.2.2 Training YOLOv11 model

For training the YOLO model, I used the Ultralytics package[18]. In Chapter 3.2.1 I mentioned the YOLO model developers' advice about the dataset size. When creating the dataset, these numbers were kept in mind. But when it comes to training, every single digit was grouped into one category called 'digit', so the model does not differentiate between numbers. I trained the model for 15 epochs with the image size set to 1920 pixels. It is important to note that the horizontal and vertical image augmentations were turned off (training parameters called *flipud* and *fliplr*). At the end of the training, the achieved precision was 0.99933 and recall 0.99941. Unfortunately, these numbers only show how good the model is at recognising EMNIST Digits from noisy backgrounds. These metrics really can't say how good the model is at detecting numbers that do not look like numbers in EMNIST dataset.

[18]Ultralytics package: https://docs.ultralytics.com/models/yolo11/

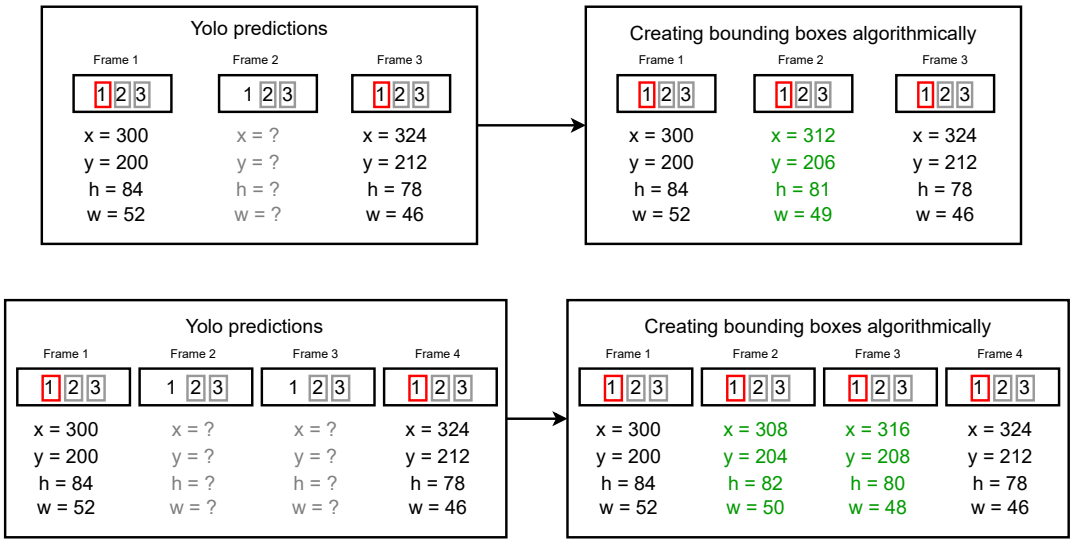


Figure 6. Algorithmically creating the missing bounding boxes after YOLO model inference. The *x* and *y* parameters are the centre coordinates of a bounding box for the digit '1' and *h* and *w* are the height and width of the bounding box. The upper flow shows how the missing box is calculated for one frame, and the lower shows it for two frames.

3.2.3 Postprocessing YOLO predictions

In Figures 4(c) and 4(c) we can see how the Ultralytics package shows the YOLO model predictions. Although in these examples the predictions look great, postprocessing of the bounding boxes is necessary. In Figure 5 we can see some examples of YOLO predictions and how they look like before and after postprocessing. First, when doing the inference, the confidence is set as 0.2 and the IoU score is set as 0.8 (Figure 5(a)) . When lowering the IoU score during inference from 1 to 0.8 (Figure 5(b)), a lot of bounding boxes are filtered out because of the big overlap. Next, I used the voting ballot box corner detection information. The bounding boxes that are not in the space where the box is are filtered out of the results (Figures 5(d) and 5(e)). After that I analyse the bounding boxes in order based on the confidence level. If there are bounding boxes that are very similar to previous predictions and are predicting the area of the same digit, the bounding boxes with lower confidence levels are filtered out. This can be seen in Figure 5(c), where there is only one bounding box per digit left after postprocessing.

3.3 Creating the missing bounding boxes

After getting bounding boxes from the YOLO model, there might be some frames missing the boxes. If there are only a couple of frames where the bounding box is missing for a digit, it is possible to generate the boxes with linear interpolation. To do this, I take two frames where the

bounding box info for a digit is known. Then it is necessary to find the difference of the two frames for the boxes centre coordinates, height, and width. And lastly, the difference should be divided between the frames.

To explain this further, let's look at Figure 6. In the upper part we have a situation where 'Frame 2' is missing the bounding box information for the digit '1'. To calculate the bounding box information, we look at the surrounding frames. 'Frame 1' and 'Frame 3' have this info. In the 'Frame 1' the digit '1' x centre coordinate is 300 and in the 'Frame 3' it is 324. With this we can calculate how much the x coordinate changes with every frame; in this case $(324 - 300)/2 = 12$. So the x centre coordinate for 'Frame 2' would be $300 + 12 = 312$. If we look at the bottom part of Figure 6, we have a situation where two frames are missing the bounding box info for the digit '1'. After finding the difference between 'Frame 1' and 'Frame 4', we can say that 'Frame 2' would get x centre coordinate of 308 and 'Frame 3' would have 316.

If we know bounding boxes (x, y, w, h) for frames indexed n and m, $n < m - 1$, we can use linear interpolation to find the bounding boxes for frames $n + 1, \ldots, m - 1$ as follows:

$$(x, y, w, h)_{n+i} = (x, y, w, h)_n + \frac{(x, y, w, h)_m - (x, y, w, h)_n}{m - n} \cdot i \quad (i = 1, \ldots, m - n - 1).$$

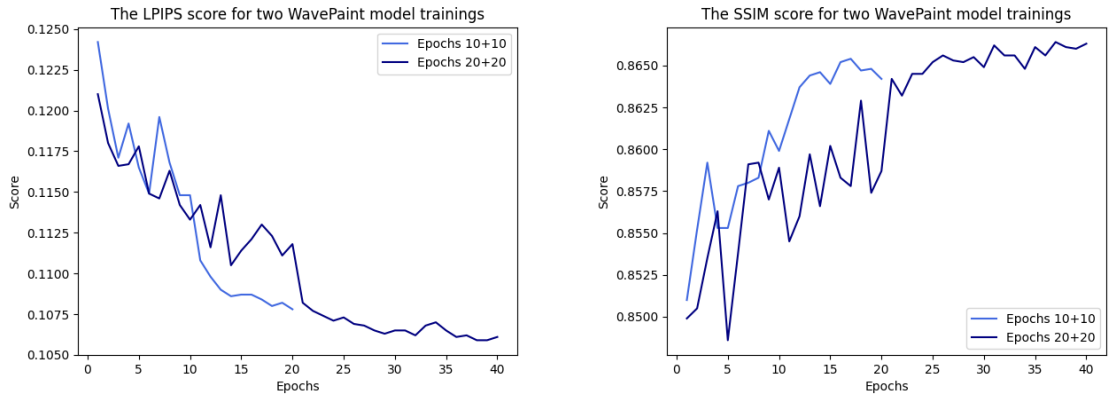
Creating these missing bounding boxes ensures that we have either all or most of the old digits covered at the end.

3.4 Image generation

I used an image generation model to cover up the wrong digits that were written on the voting ballot. I used the output of the YOLO model to find where the digits are and where I have to cover them. For this, image-to-image model is the best one to use because we can give the model an input image that we want the model to change and get back a new one. I tried out different models, for example, Pix2Pix[19] [24], cGAN[20] [39] and Palette [40]. In the end, I chose to use the WavePaint model.

[19]Pix2Pix code used: https://www.tensorflow.org/tutorials/generative/pix2pix

[20]cGAN code used: https://github.com/spmallick/learnopencv/blob/master/Conditional-GAN-PyTorch-TensorFlow/PyTorch/CGAN-PyTorch.ipynb



(a) (b)

Figure 7. Evaluation metrics during training for two WavePaint models. The model name 'Epochs 10+10' means that the model was trained for 10 epochs with the AdamW optimizer and 10 epochs with the SGD optimizer. (a) The LPIPS scores for two models: one trained for 20 epochs and the other trained for 40 epochs. (b) The SSIM scores for the same two trained models.

3.4.1 Dataset for WavePaint model

To train the WavePaint model, I created a dataset using the Open Images dataset. I took an image from the dataset and randomly cropped a 256×256 pixel image out of it. Then I created a mask of black pixels. This mask represents the place where the digit should be and the area that the model should try to inpaint. I tried to create the masks with the sort of ratios that the model would see when generating videos later on. Figures 8(a), 8(b) and 8(c) show example images from the training dataset.

3.4.2 Training a WavePaint model

I trained the WavePaint model using the code provided in the WavePaint model paper[21]. As a base model, I used a pretrained model provided by the authors that was trained on the CelebA-HQ dataset. This model has two stages. In the first stage they use the AdamW optimizer, and the second stage they use the SGD optimizer. I modified the code so the user can set the number of epochs for each stage.

[21]WavePaint model Github: https://github.com/pranavphoenix/WavePaint

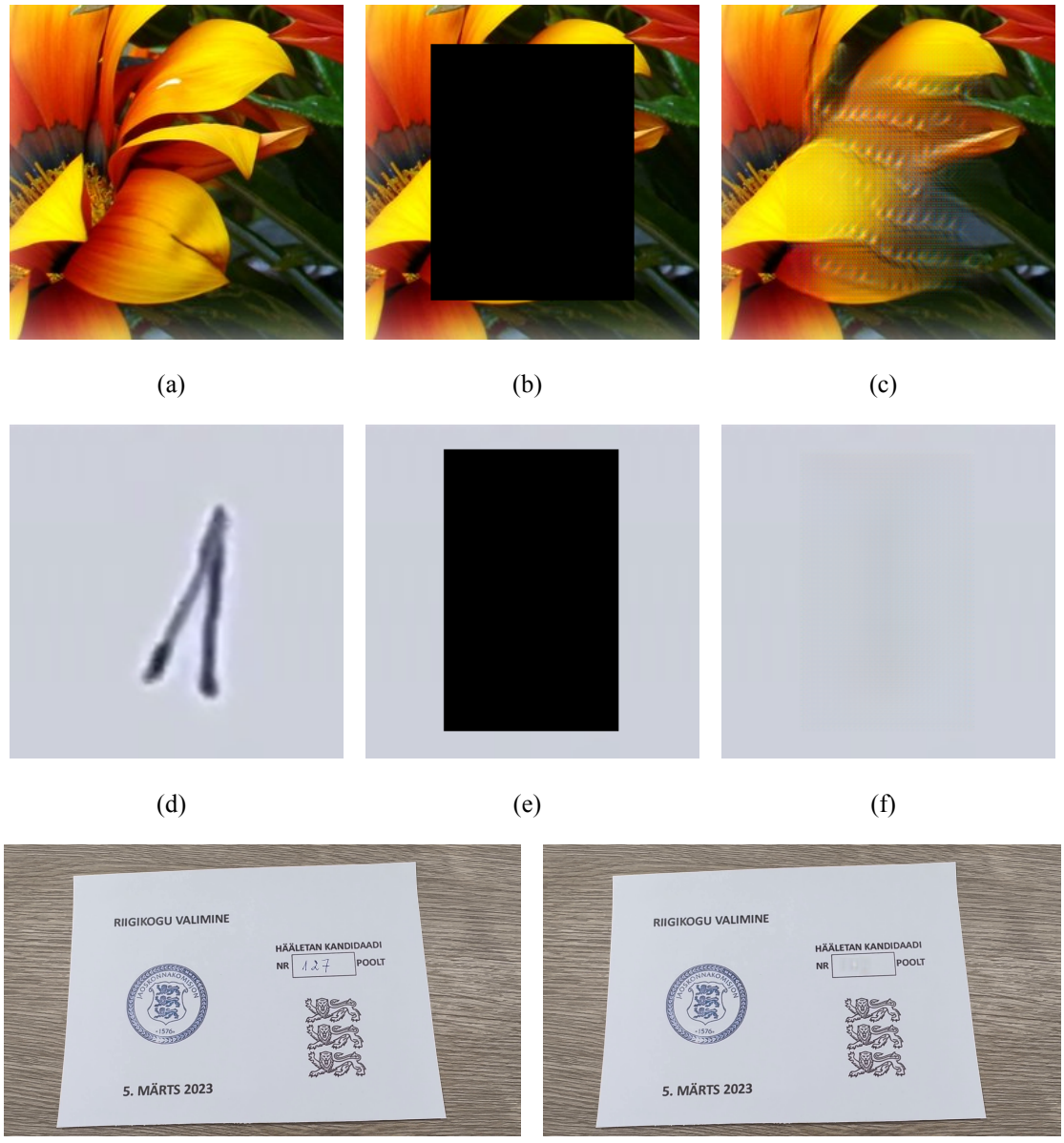


Figure 8. Example of WavePaint training and inference. (a) Image from WavePaint model training dataset. (b) Training image with a mask. (c) Training image after inference. (d) Image of a digit on a ballot. (e) Image of a digit where the digit is masked. (f) WavePaint model inpainting and returning image with the digit covered. (g) Entire frame before inpainting. (h) Entire frame after inpainting.

For comparison, I trained two models: one with 20 epochs and another one with 40 epochs altogether. As evaluation metrics, the authors calculated the LPIPS (Learned Perceptual Image Patch Similarity) [26] and SSIM (Structural Similarity Index Measure)[22]. In Figure 7(a) we can see how the LPIPS score decreases during training. For both models there is a big decrease after changing the optimizer. In Figure 7(a) we can see the increase of the SSIM score for both models. These figures show that there isn't a significant change in metrics when training the model longer. This means that the model that was trained for 20 epochs is sufficient. In the end I used the model that was trained with the batch size of 16 and where both optimizers ran for 10 epochs (20 epochs altogether).

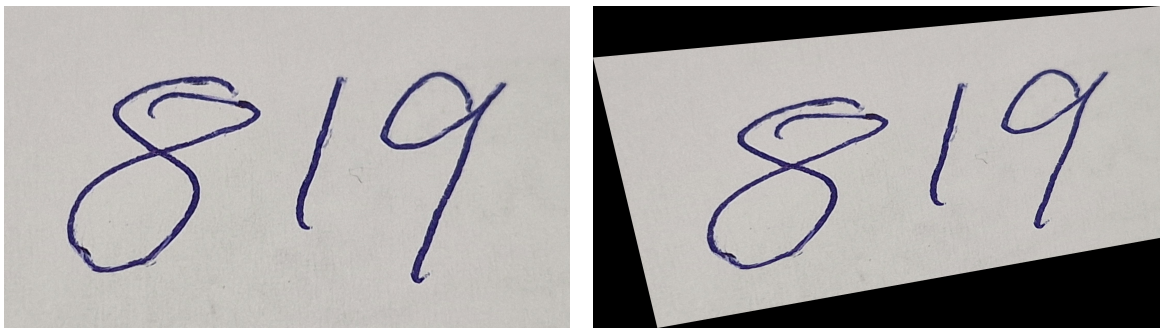
During inference the model's task is to generate a new image without the old digit. Figures 8(d), 8(e) and 8(f) show how WavePaint is able to cover up a real digit on a ballot. In Figures 8(g) and 8(h) we can see how the entire frame looks after covering up the old 3-digit code. The covering up is done separately for every digit instead of doing it together as one image because this way we have a smaller area to cover, and the digits are better covered when the voting ballot is not filmed straight on.

3.5 Placing the new digit

The last step after finding the wrong digits and covering them up is to get the new digits into the video. This can be done in a couple of ways.

One way would be to let an image generation model generate the new digits, but there are multiple disadvantages to this method. Firstly, this type of image generation task is much harder than just generating a background. It needs a lot more resources and training. Secondly, image generation for videos is still based on frame-by-frame generation. This means that the models do not "know" what they generated for the previous frames. This results in a flickering video in which the digit does not have a cohesive look throughout the video. Lastly, with image generation, it is difficult to make sure that the model output is presented using the same handwriting as the original video is.

[22] In the WavePaint paper [25] they mention using LPIPS and FID scores to evaluate the performance. But when actually looking at the code, there is no mention of the FID score. Their code calculated 5 different scores: L1, L2, PSNR, SSIM and LPIPS. I decided to use the SSIM score as the second evaluation metric to show the results.



(a) (b)

Figure 9. (a) An image that is created based on the input image and the cropping coordinates given to the pipeline. (b) Transformed new 3-digit code image based on the voting box corner coordinates.

To battle all of these challenges, I used a more algorithmic solution and OpenCV library[23]. For this I use the corner predictions and an image of the new 3-digit code (see example in Figure 9(a)). For each of the frames, I use the voting ballot box corners info and OpenCV libraries *getPerspectiveTransform* and *warpPerspective* to perform a perspective transform on the image (example in Figure 9(b)). I remove the background of the new 3-digit code image with colour thresholding and paste the transformed version onto the frame. In the end we have a video, where the original digits have been covered up and the new 3-digit code is placed on top of it.

Looking at the new video after this step, the result does not feel quite right. The new code in the voting ballot box shakes just enough that it does not convince the viewer of being a real video. For comparison, 100 continuous frames from one video were manually annotated with the box corner information. When a video was put together with these manual annotations, the result was still a little bit shaky. That shows that even the 'ground truth' solution isn't precise enough to create a believable solution. There is a way to reduce the shakiness. So far the videos have mostly been with the resolution of 1920×1080 pixels and with the frame rate of 30 fps. Lowering the resolution to 1280×720 pixels and the frame rate to 24 fps decreases the shakiness. Of course it is possible to lower them even more, but then at one point the video itself has too low of a quality.

[23] OpenCV library: https://opencv.org/

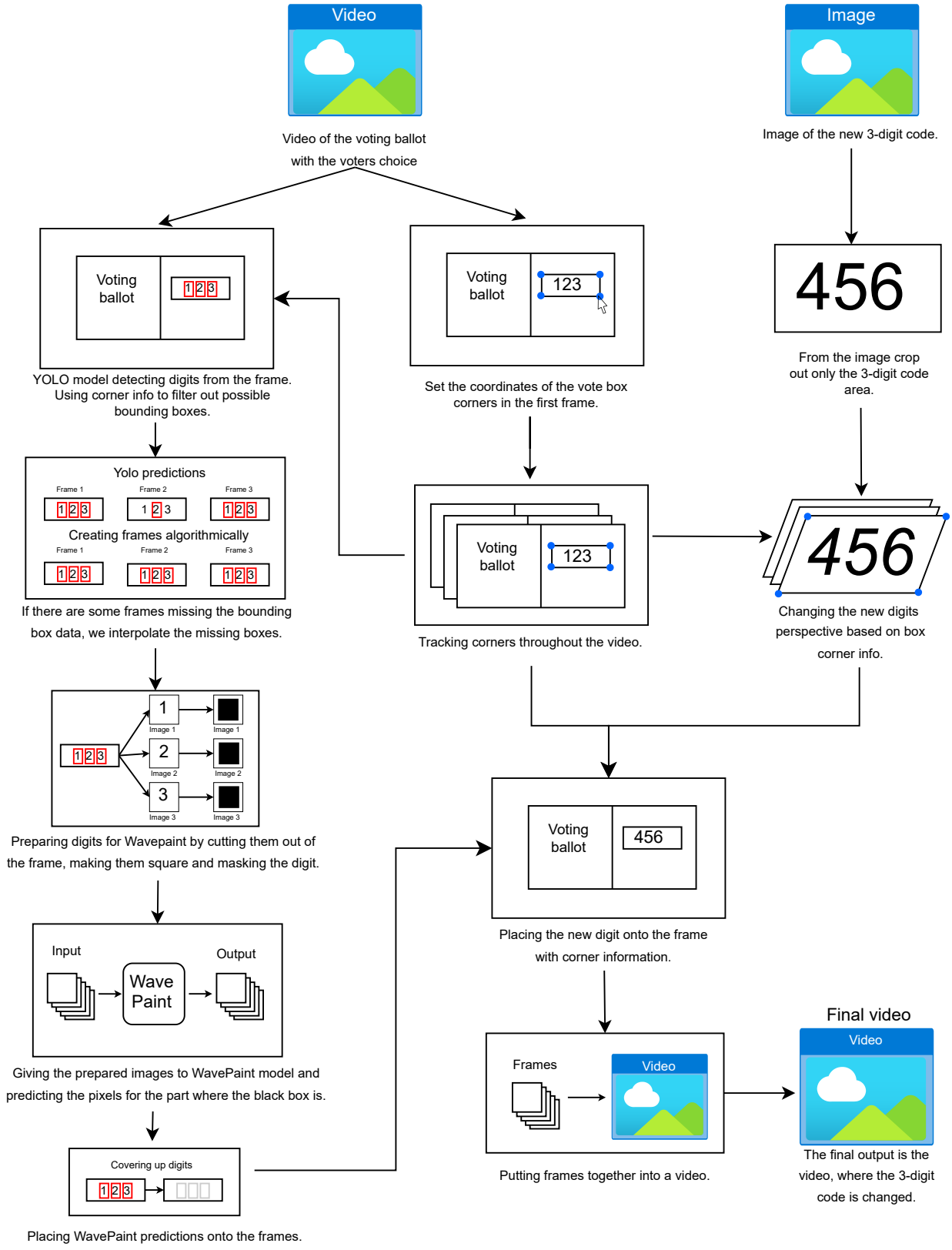


Figure 10. The pipeline of taking a video of the voting ballot and an image with the new code and turning them into a video with the changed 3-digit code.

1. Lowering the video frame rate and saving the video frames as images.
2. Manually finding where the corners of the voting ballot box are in the first frame.
3. Using the Lucas-Kanade method to track the voting ballot box corners throughout the video.
4. The YOLO model is used on the video to find where the digits are in each frame. The YOLO model gives back bounding boxes for the digits.
5. The YOLO model predictions are postprocessed using the box corner information.
6. If there are a couple of frames where the bounding box is missing for one digit, the bounding boxes are calculated for those frames.
7. The images are prepared for the WavePaint model.
    - YOLO predictions are used to cut out the digit and create a square image, generating extra pixels for the border.
    - The actual digit part of the square image is covered with a mask.
8. Giving the images to the WavePaint model to do inpainting.
9. Using generated images and placing them into frames to cover up the original digits.
10. Taking the image of the new 3-digit code and manually cropping out only the digit part.
11. For each frame, using the voting ballot box corner information and perspective transform to morph the new 3-digit code image.
12. Removing the background of the new digit and placing it into the frame in the voting ballot box.
13. Taking all the frames with the new digit, lowering the resolution of frames, putting them back together into a video and adding back the original audio.

3.6 Entire pipeline

After creating all of these parts separately, I can add them together to make one pipeline that changes the 3-digit code in the video into a new code.

After giving the pipeline a video of the voting ballot and an image of the new 3-digit code, we get out a video with the new code. Figure 10 visualises the same process of the pipeline, starting

with the input video and image and ending with the output of the new video. On average, if the video is 10 seconds and has 230 frames (with 24 fps), the entire pipeline takes 2.5 minutes. Out of that, YOLO predictions took 16 seconds, WavePaint predictions took 45 seconds and placing the WavePaint images and the new digit onto the frame took 46 seconds.

The entire repository is available at https://github.com/anettehabanen/Deepfakes_for_Paper_Vote_Privacy_Defence. It consists of the dataset creation files, model training files and final pipeline. To test out the training, small example datasets are also included. The Reflex app is located in a separate repository at https://github.com/anettehabanen/Change_My_Vote.

4. Results

4.1 Evaluation dataset

To evaluate the pipeline, I collected videos and images of voting ballots. I gave 19 people multiple voting ballots. They wrote random 3-digit numbers onto the ballots and took a 5-10 second horizontal videos. They also wrote the same 3-digit code on the other side of the page and took an image of it. In addition, they filmed empty voting ballots. From each person, there are on average 4 videos of different ballots. At the end there were 73 videos of filled out voting ballots, 68 images of 3-digit codes and 25 videos of empty voting ballots in total.

After collecting this dataset, I did some data cleaning and added annotations. For every video I took the first frame and found the four corner coordinates manually and for every image I found the cropping coordinates. These steps were important because that meant I was able to generate new videos very quickly without having to find this information again for every generation.

4.2 YOLO model

I used the collected dataset of filled out videos to evaluate the YOLO object detection model. I ran the inference on the videos frame by frame. In every video all the frames showed the 3-digit code. Based on this I could calculate how many bounding boxes the YOLO model should find (in one video there is $3 \times$nr_of_frames bounding boxes). I used the same post-processing for the bounding boxes here as for the pipeline (IoU=0.8, confidence=0.2 and boxes were counted if they were within the box corners). For evaluation, I calculated the percentage of bounding boxes the model found during inference.

Table 1. Comparissons of different trained YOLO models.

Base model	**Nr of classes**	**Nr of epochs**	**Imgsz**	**Found bboxes (all videos)**	**Found bboxes (good quality videos)**
yolo11n	10	15	1280	30.3%	35.2%
yolo12n	10	18	1280	23.7%	28.3%
yolo11m	10	15	1280	38.8%	46.0%
yolo11m	10	30	1280	38.2%	43.5%
yolo11m	10	15	1920	57.2%	67.7%
yolo11m	1	15	1920	67.3%	77.0%

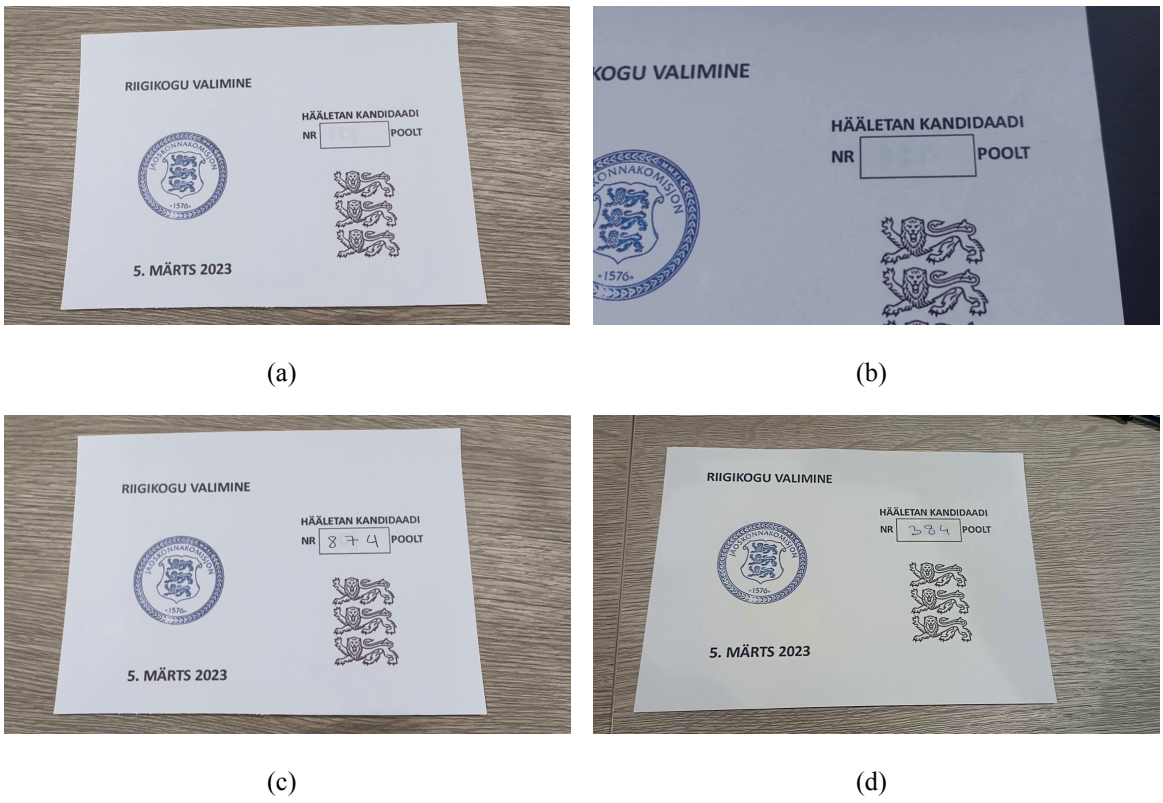


Figure 11. Resulting images from the pipeline. (a-b) Images of voting ballots, where the old digits have been covered up. (c-d) Images of voting ballots, where a new digit has been placed into to the voting box.

To see how different aspects and parameters affect the results, I trained multiple YOLO models with different parameters. For most of the models, I used the base model of yolo11m[24], but I also experimented with yolo11n[24] and yolo12n[25]. There are two main parameters that changed the results a lot. The first important parameter is *imgsz*, defining the maximum width of the image that the YOLO model processes. The second aspect is the number of classes to train on. I can either have 10 classes, which means that every digit is a separate class. Alternatively, I can have one class which means that every digit is in one class, and YOLO just has to learn to find handwritten digits.

Some of these models with their training parameters and results are shown in Table 1. As we can see, it is important to have a bigger *imgsz* and all the digits represented in one class. Surprisingly,

[24]Trained YOLOv11 model files: https://docs.ultralytics.com/models/yolo11/#supported-tasks-and-modes
[25]Trained YOLOv12 model files: https://docs.ultralytics.com/models/yolo12/#__tabbed_1_1

increasing the number of epochs from 15 to a bigger number really didn't achieve that much better results. Furthermore, having the YOLO model with the medium size (yolo11m) was necessary because the nano size versions (yolo11n and yolo12n) didn't perform as well.

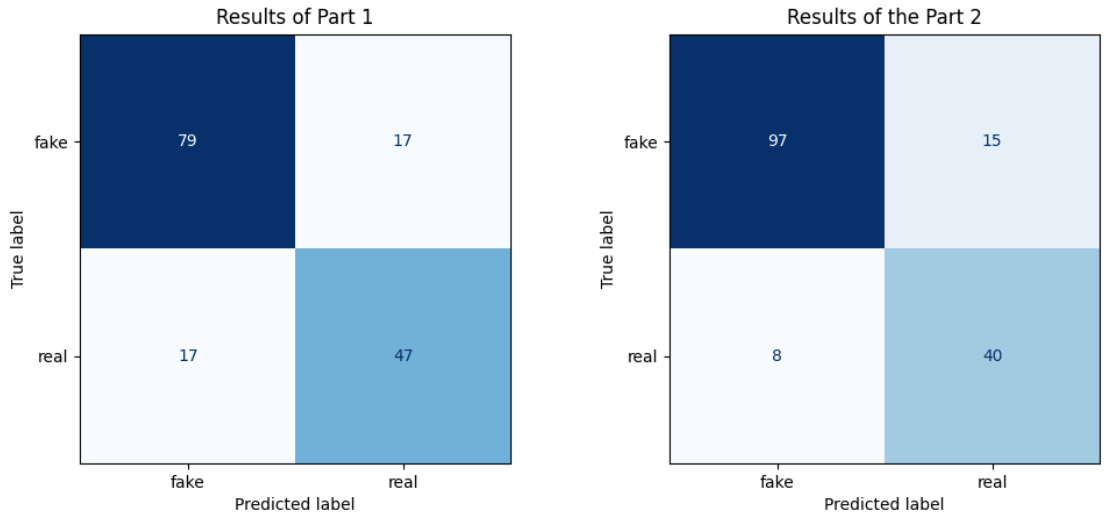
I evaluated the performance on two datasets. The first one (called 'all videos' in Table1) consists of every video of filled out voting ballots. This dataset had 73 videos, and these videos had 20 263 frames. The second one (called 'good quality videos' in Table1) consists of only videos, where the video resolution is 1920×1080 pixels or higher. In total this dataset has 60 videos and 16 967 frames. The best trained model (last row in Table1) was able to detect 67.3% of the numbers from the voting ballot videos. If I ran the inference only on the good-quality videos (82.2% of the evaluation dataset), the model detected 77.0% of the bounding boxes.

4.3 Evaluation of Results test

The 'Evaluation of Results' test was designed to measure the quality of generated images and videos. I wanted to see if the actual visual results were convincing and were able to fool the viewer. The test consisted of 4 parts: images of the covered up ballots, videos of covered up ballots, images of the new 3-digit codes and videos of the new 3-digit codes. Each of the parts had 10 examples, and at least 6 of them were generated.

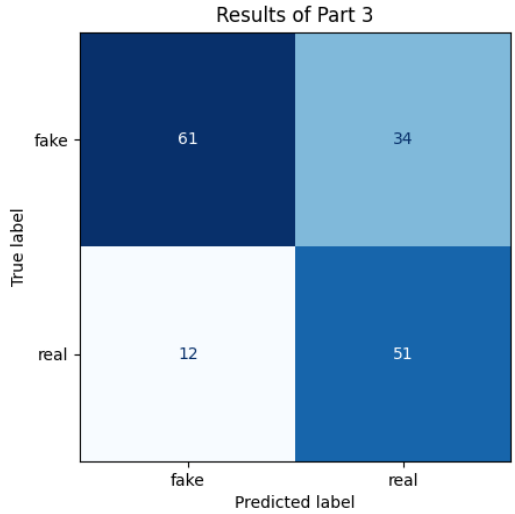
All these images/videos were put together into one video. This was done so that I could just play the video and I didn't have to go around my computer looking for files. Also, this way I could assure that every image and video could be seen the same amount. During the test, people were able to look at the images for 10 seconds and saw 5-second video clips twice. For each image/video, the test subject had to mark down if they thought that it was real or it had been tampered with. Additionally, for each guess they had to put down their confidence score. Confidence score measures how sure they are in their guess from 1 to 5. 1 indicates that they are not sure in their guess, and 5 indicates that they are very sure and they have no doubt. In Figure 11 we can see some of the generated images that people saw in the test.

Altogether I had 16 people do the 'Evaluation of Results' test. In Figure 12 we can see 4 confusion matrices that show how people's predictions compare to the reality. As we can see, most people were able to find the real and generated images/videos. In each part, there were some people who thought that the generated ones were real. The most difficult part seemed to be Part 3, where people were the most 'fooled'. In Table 2 we can see the confidence scores for the generated images/videos. The 'All' line shows the overall confidence in that part of

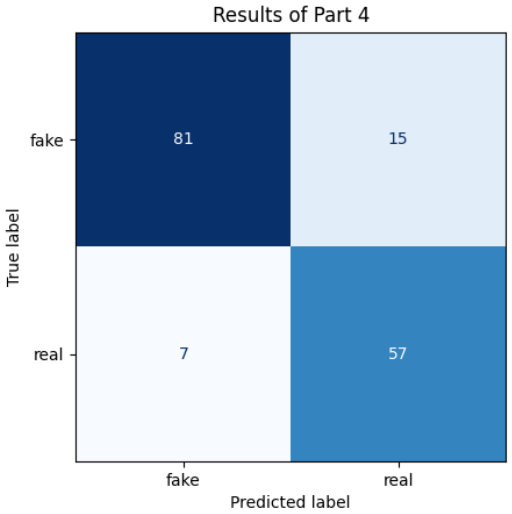


(a)

(b)



(c)



(d)

Figure 12. Comparison of the actual and predicted answers in the ’Evaluation of Results’ test per every part.

the test. The ’Real’ line shows average scores for cases where the person guessed ’Real’. And the ’Fake’ line shows people’s confidence score when they guessed ’Fake’. As we can see, the confidence score throughout the test increases. The only exception was Part 3, where the confidence score decreased compared to previous parts. This decrease can be explained by the increase of guessing ’Real’ for generated images in Part 3. In Part 3 people were the most unsure, and with that, their confidence score was lower.

4.4 Problems with end results

As seen from the confusion matrices, not a lot of people were fooled and they were able to tell apart the real and generated images/videos. When asking after, what were the main giveaways, similar answers were given.

First, for covering up the digit, the WavePaint model generated images that had a lighter centre. So the covering up wasn’t pixel perfect, and it didn’t create the illusion of being an empty box. If there was even a tiny bit of flickering or some areas were lighter/darker than the rest, it is very easy to tell that the image/video is fake. This problem is mostly due to the model not having a lot of information about the surrounding pixels and how to inpaint the masked area on the voting box.

Second, when placing the new digit into the box, the digits would shiver just a little bit and didn’t feel like they were stuck on the paper. This shivering is due to the box corner coordinates that the Lucas-Kanade method generated. As mentioned earlier, I tried to reduce it by lowering the frames per second and video resolution, but that wasn’t enough. Even manually annotated video had the same problem. For comparison I have added two videos into the main project repository. One of them has manually annotated corner coordinates, and the other one used the

Table 2. Average confidence scores in the ’Evaluation of Results’ test. Score 1 shows low confidence and score 5 indicates high confidence.

Prediction	**Part 1 (covered image)**	**Part 2 (covered video)**	**Part 3 (new digits image)**	**Part 4 (new digits video)**
All	2.96	3.78	3.21	4.06
Real	3.30	3.58	3.34	3.82
Fake	2.76	3.89	3.07	4.26

Lucas-Kanade method for that. It is hard to differentiate between them, and for me, the 3-digit code shakes equally in both videos.

Third, the missing bounding boxes are right now calculated using linear interpolation. If one frame is missing the bounding box, then using this method is quite good. But if we have a lot of frames with this missing info, then this way of generating is not that accurate. It doesn't take into account the acceleration and sporadic camera movements. This means more complex interpolation methods could be used to get improvements and more accurate bounding boxes.

Lastly, one interesting observation from some people was that they were able to learn what the aspects were that made the video fake. For example, when they realised that the fake ones had the lighter centre, they started to look for it in the next rounds. This means that as the test goes on, the confidence rises because the test subjects know what to look for. Unfortunately I think this effect is unavoidable. Even if I had shown the images and videos from different parts in random order, once a person finds the aspects of what to look for, they can't be unlearned. This effect can be observed from Table 2 where the average confidence score increases during the test.

4.5 End user application

For a better user experience and to make the process easily understandable, I also created a web application to run the pipeline. The framework I chose for the application was Reflex[26]. The reason for it is because it is written in Python, like the main pipeline, and it is a very easily understandable framework with nice predefined UI elements.

Using this application, the user can upload their video and the image of the new 3-digit code. Next, they can configure the exact place of the code and the voting ballot corners. Lastly, they can start the pipeline and wait for the finished result. This pipeline works exactly the same as the Python script version. The example figures of the application can be seen in the Appendix A.

I would like to emphasize that this application was only created to show a visual representation of the pipeline. In the real world this type of application has some security risks. For example, the uploaded video could be tied to the location of the phone or to the network that it is in. Because of that, the final solution should be something that can be run locally. Unfortunately,

[26]Reflex website: https://reflex.dev/

this would mean that this solution wouldn’t be easily accessible to a variety of people because of technical skills and restrictions.

5. Discussion

As we could see in Chapter 4.3, 'fooling' people and making them believe that the generated videos are real is very difficult. What is important to note is that when it comes to using this method in real-life scenarios, the conditions aren't the same as they were in this test. In the test people were looking for something wrong with the images/videos, but in real life the attacker wouldn't think about the provided video being fake or generated. The attacker would mostly just look if the number in the video is what they wanted, and would likely not analyse the video further.

One way to make the results better would be to have a higher-quality dataset for handwritten digits (remember that EMNIST dataset images are 28×28 pixels and get blurry when upscaled). Even though there are some better quality datasets available, they usually only have classification data and not bounding box or segmentation annotations. One example of such a dataset is the '3-digit occupation codes from the Norwegian 1950 census' dataset [27]. It has higher-quality images of handwritten digits but only class information is included in the filenames.

Furthermore, it would be useful to experiment with different resolution and fps values. I just chose to lower these parameters to certain numbers, but it would be interesting to see if there is a combination with which the shakiness disappears and the resulting video is very convincing. But it is important to not go to the extremes and create poor-quality videos. In that case, completely new methods should be tried. One method that could be tried regarding image resolution is to think about subpixel resolution. This means, for example, that if the Lucas-Kanade method predicts that the coordinate should be 100.2 and we round it down to 100, it is not precise. But if the image is resized to a very high resolution, the predictions can be more accurate. Then, after making very precise predictions, we can add all new frames together and lower the resolution for the final video. One big drawback with this method is that the high resolution makes the entire pipeline slower.

Additionally, it would be beneficial to find a model that would be good and consistent in generating background and handwritten digits. When talking about videos, consistency between the frames is a very important topic for creating convincing results. Because of that, the new

[27] '3-digit occupation codes from the Norwegian 1950 census' dataset: https://dataverse.no/dataset.xhtml?persistentId=doi:10.18710/OYIH83

model should somehow use the information about the surrounding frames to make the result smoother. If this kind of model could be used, that would remove the need to provide the input image and the 3-digit code location in the image, resulting in fewer steps for the user to perform before starting the conversion.

Another improvement that could be made (and that is on another level of difficulty) is being able to change the digits during the writing process because right now the pipeline works if the digits are already written on the voting ballot. And the last step after this would be to be able to perform this live. Right now it only works if you go to the voting booth, film a video and change it afterwards. But it doesn't protect people from scenarios where they are asked to be in a video call and show the proof live.

I think these results and observations show that deepfakes can't be used against this type of attack. Even though some images and videos were convincing, the majority of people were able to spot the generated videos. This means that the situations where the attacker asks for video proof are still dangerous, and people can't be protected from them. On a positive note, machine learning algorithms are rapidly evolving, and in the future it might be possible to create these videos with machine learning to protect people.

6. Conclusion

In this thesis I introduced a pipeline for changing a video of handwritten paper vote using machine learning. I focused on Estonian ballots, where casting a vote happens with 3-digit codes. The pipeline uses the YOLO model to find the handwritten digits from the video frame. After that a Wavepaint model is used to generate new images to cover up the old 3-digit code in the frame. Then the pipeline uses an image of the new 3-digit code that we want to add to the video. Corner detection method and image perspective transform are used to change the new image and add it to the frame. The last step is to take all the changed frames and put together a video with the new vote.

I evaluated the results achieved during the process. I collected a dataset of people filming their filled out voting ballots. On this dataset, the trained YOLO model was able to find 67.3% of the bounding boxes. I also conducted a test where people had to evaluate the results that the pipeline generated. Most people were able to differentiate between real and generated images and videos, but in some cases it was harder, and I was able to fool them. When asked for feedback after the test, people said that because the covering up of the old digit wasn't pixel perfect and the new placed 3-digit code had some shakiness, it was easy to spot the deepfakes.

In addition to being able to create convincing videos, the aim was to create a prototype for an application. For this I used the framework called Reflex. With this I was able to create a beautiful webpage that showed how this pipeline could be used in a real-life scenario and what the user experience would look like.

Moreover, I hope I have shown the positive side of using AI. Even though mostly people associate and the media shows the negative use of AI in the electoral processes, I showed that the opposite is also possible. With this thesis I showed a way to use AI models and deepfakes for good and help protect the voting system and people's privacy.

References

[1] Saltman R. G. The history and politics of voting technology: In quest of integrity and public confidence. Springer, 2006.

[2] Wasley P. Back When Everyone Knew How You Voted. *Humanities* 37.4 (2016).

[3] Brent P. The Australian ballot: Not the secret ballot. *Australian Journal of Political Science* 41.1 (2006), pp. 39–50.

[4] Hammelburg E. #stemfie: reconceptualising liveness in the era of social media. *TMG Journal for Media History* (Sept. 2015), pp. 85–100. DOI: 10.18146/tmg.108.

[5] Koutsoulias I. Ballot Selfies: Balancing The Right To Speak Out On Political Issues And The Right To Vote Free From Improper Influence And Coercion. *Journal of Law and Policy* 26 (2018), p. 349.

[6] Bender S. M. Algorithmic Elections. *Mich. L. Rev.* 121 (2022), p. 489.

[7] Stevenson K. Artificial Intelligence: A Double-Edged Sword In Elections. *Educational Administration: Theory and Practice* 30.4 (Apr. 2024), pp. 1660–1667. DOI: 10.53555/kuey.v30i4.1724. https://www.kuey.net/index.php/kuey/article/view/1724.

[8] McIsaac C. Impact of Artificial Intelligence on Elections. Tech. rep. 304. https://www.rstreet.org/wp-content/uploads/2024/06/FINAL-r-street-policy-study-no-304.pdf. R Street, June 2024.

[9] Bueno De Mesquita E., Canes-Wrone B., Hall A. B., Lum K., Martin G. J., and Ricardo Velez Y. Preparing for Generative AI in the 2024 Election: Recommendations and Best Practices Based on Academic Research. Tech. rep. https://harris.uchicago.edu/files/ai_and_elections_best_practices_no_embargo.pdf. University of Chicago Harris School of Public Policy and Stanford Graduate School of Business, 2023.

[10] Gemenis K. Artificial intelligence and voting advice applications. *Frontiers in Political Science* 6 (2024).

[11] Silbey J. and Hartzog W. The upside of deep fakes. *Md. L. Rev.* 78 (2018), pp. 960–966.

[12] Wei Z., Xu X., and Hui P. Digital Democracy at Crossroads: A Meta-Analysis of Web and AI Influence on Global Elections. *Companion Proceedings of the ACM Web Conference 2024*. WWW '24. Singapore, Singapore: Association for Computing Machinery, 2024, pp. 1126–1129. DOI: 10.1145/3589335.3652003. https://doi.org/10.1145/3589335.3652003.

[13] Lee-Geiller S. Using AI in Election Campaigns Erodes Voter Perceptions of Electoral Integrity with In-Party Use Facing Heightened Scrutiny. *SSRN* (2024). DOI: 10.2139/ssrn.4937187. https://ssrn.com/abstract=4937187.

[14] Boesch G. Object Detection: The Definitive 2025 Guide. 2024. https://viso.ai/deep-learning/object-detection/.

[15] Girshick R., Donahue J., Darrell T., and Malik J. Rich feature hierarchies for accurate object detection and semantic segmentation. 2014. arXiv: 1311.2524 [cs.CV]. https://arxiv.org/abs/1311.2524.

[16] Girshick R. Fast R-CNN. 2015. arXiv: 1504.08083 [cs.CV]. https://arxiv.org/abs/1504.08083.

[17] He K., Gkioxari G., Dollár P., and Girshick R. Mask R-CNN. 2018. arXiv: 1703.06870 [cs.CV]. https://arxiv.org/abs/1703.06870.

[18] Redmon J., Divvala S., Girshick R., and Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[19] Khanam R. and Hussain M. YOLOv11: An Overview of the Key Architectural Enhancements. *ArXiv* abs/2410.17725 (2024). https://api.semanticscholar.org/CorpusID:273532028.

[20] Ramesh A., Pavlov M., Goh G., Gray S., Voss C., Radford A., Chen M., and Sutskever I. Zero-Shot Text-to-Image Generation. 2021. arXiv: 2102.12092 [cs.CV]. https://arxiv.org/abs/2102.12092.

[21] Saharia C., Chan W., Saxena S., Li L., Whang J., Denton E., Ghasemipour S. K. S., Ayan B. K., Mahdavi S. S., Lopes R. G., Salimans T., Ho J., Fleet D. J., and Norouzi M. Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. 2022. arXiv: 2205.11487 [cs.CV]. https://arxiv.org/abs/2205.11487.

[22] Rombach R., Blattmann A., Lorenz D., Esser P., and Ommer B. High-Resolution Image Synthesis with Latent Diffusion Models. 2022. arXiv: 2112.10752 [cs.CV]. https://arxiv.org/abs/2112.10752.

[23] Goodfellow I. J., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., and Bengio Y. Generative Adversarial Networks. 2014. arXiv: 1406.2661 [stat.ML]. https://arxiv.org/abs/1406.2661.

[24] Isola P., Zhu J.-Y., Zhou T., and Efros A. A. Image-to-Image Translation with Conditional Adversarial Networks. *2017 IEEE Conference on Computer Vision and*

Pattern Recognition (CVPR) (2016), pp. 5967–5976. https://api.semanticscholar.org/CorpusID:6200260.

[25] Jeevan P., Kumar D. S., and Sethi A. WavePaint: Resource-efficient Token-mixer for Self-supervised Inpainting. 2023. arXiv: 2307.00407 [cs.CV]. https://arxiv.org/abs/2307.00407.

[26] Zhang R., Isola P., Efros A. A., Shechtman E., and Wang O. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. 2018. arXiv: 1801.03924 [cs.CV]. https://arxiv.org/abs/1801.03924.

[27] Wang Z., Bovik A., Sheikh H., and Simoncelli E. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.

[28] Shah F. T. and Yousaf K. Handwritten Digit Recognition Using Image Processing and Neural Networks. *World Congress on Engineering*. 2007. https://api.semanticscholar.org/CorpusID:6411926.

[29] Jain P. H., Kumar V., Samuel J., Singh S., Mannepalli A., and Anderson R. Artificially Intelligent Readers: An Adaptive Framework for Original Handwritten Numerical Digits Recognition with OCR Methods. *Information* 14.6 (2023). DOI: 10.3390/info14060305. https://www.mdpi.com/2078-2489/14/6/305.

[30] Ciresan D. C., Meier U., Gambardella L. M., and Schmidhuber J. Deep, Big, Simple Neural Nets for Handwritten Digit Recognition. *Neural Computation* 22 (2010), pp. 3207–3220. https://api.semanticscholar.org/CorpusID:1918673.

[31] Kazakçi A., Cherti M., and Kégl B. Digits that are not: Generating new types through deep neural nets. *ArXiv* abs/1606.04345 (2016). https://api.semanticscholar.org/CorpusID:13478578.

[32] Jha G. and Cecotti H. Data augmentation for handwritten digit recognition using generative adversarial networks. *Multimedia Tools and Applications* 79 (2020), pp. 35055–35068. https://api.semanticscholar.org/CorpusID:216085472.

[33] Gonzalez R. and Woods R. Digital Image Processing Global Edition. Pearson Deutschland, 2017, p. 1024. https://elibrary.pearson.de/book/99.150005/9781292223070.

[34] Lucas B. and Kanade T. An Iterative Image Registration Technique with an Application to Stereo Vision (IJCAI). Vol. 81. Apr. 1981.

[35] Deng L. The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142. DOI: 10.1109/MSP.2012.2211477.

[36] Cohen G., Afshar S., Tapson J., and Schaik A. van. EMNIST: an extension of MNIST to handwritten letters. 2017. arXiv: 1702.05373 [cs.CV]. https://arxiv.org/abs/1702.05373.

[37] Krasin I., Duerig T., Alldrin N., Veit A., Abu-El-Haija S., Belongie S., Cai D., Feng Z., Ferrari V., and Gomes V. OpenImages: A public dataset for large-scale multi-label and multi-class image classification. Jan. 2016.

[38] Sun S.-H. Multi-digit MNIST for Few-shot Learning. 2019. https://github.com/shaohua0116/MultiDigitMNIST.

[39] Mirza M. and Osindero S. Conditional Generative Adversarial Nets. *ArXiv* abs/1411.1784 (2014). https://api.semanticscholar.org/CorpusID:12803511.

[40] Saharia C., Chan W., Chang H., Lee C. A., Ho J., Salimans T., Fleet D. J., and Norouzi M. Palette: Image-to-Image Diffusion Models. *ACM SIGGRAPH 2022 Conference Proceedings* (2021). https://api.semanticscholar.org/CorpusID:243938678.

Appendices

Appendix A: Reflex application

For this thesis I created an application using the framework called Reflex. Reflex is written in Python and has a lot of predefined UI elements, which made creating the webpage very easy. In Figure 13 is the homepage, were user can upload their video of the voting ballot and an image of the new 3-digit code. In Figure 14 we can see a page, where the user can crop out the code from the rest of the image. In Figure 15 there is a page where user can set the corner coordinates for the voting box. Lastly, in Figure 16 there is a final page where the user can start the pipeline and after that download the finished new video.

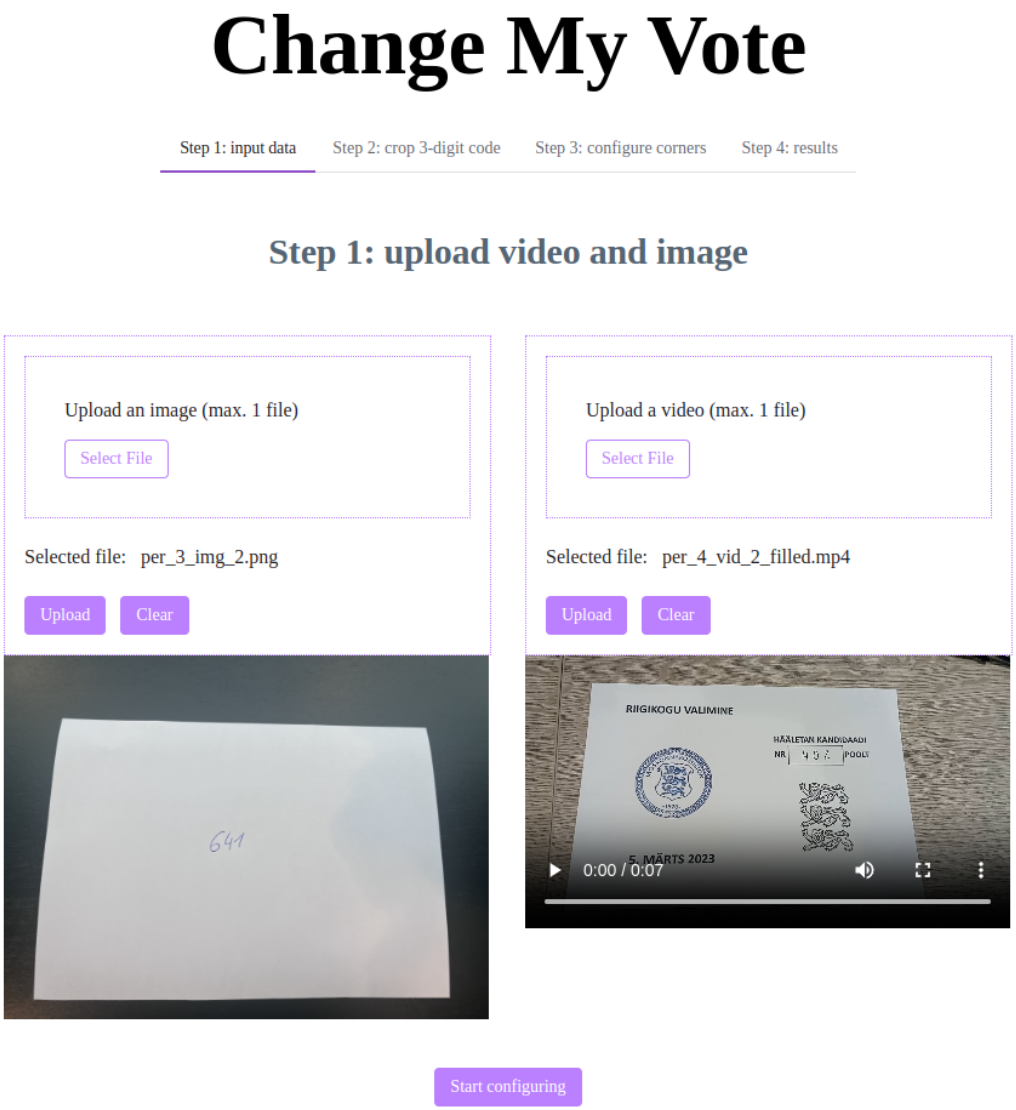


Figure 13. Uploading the video and image of the new code.

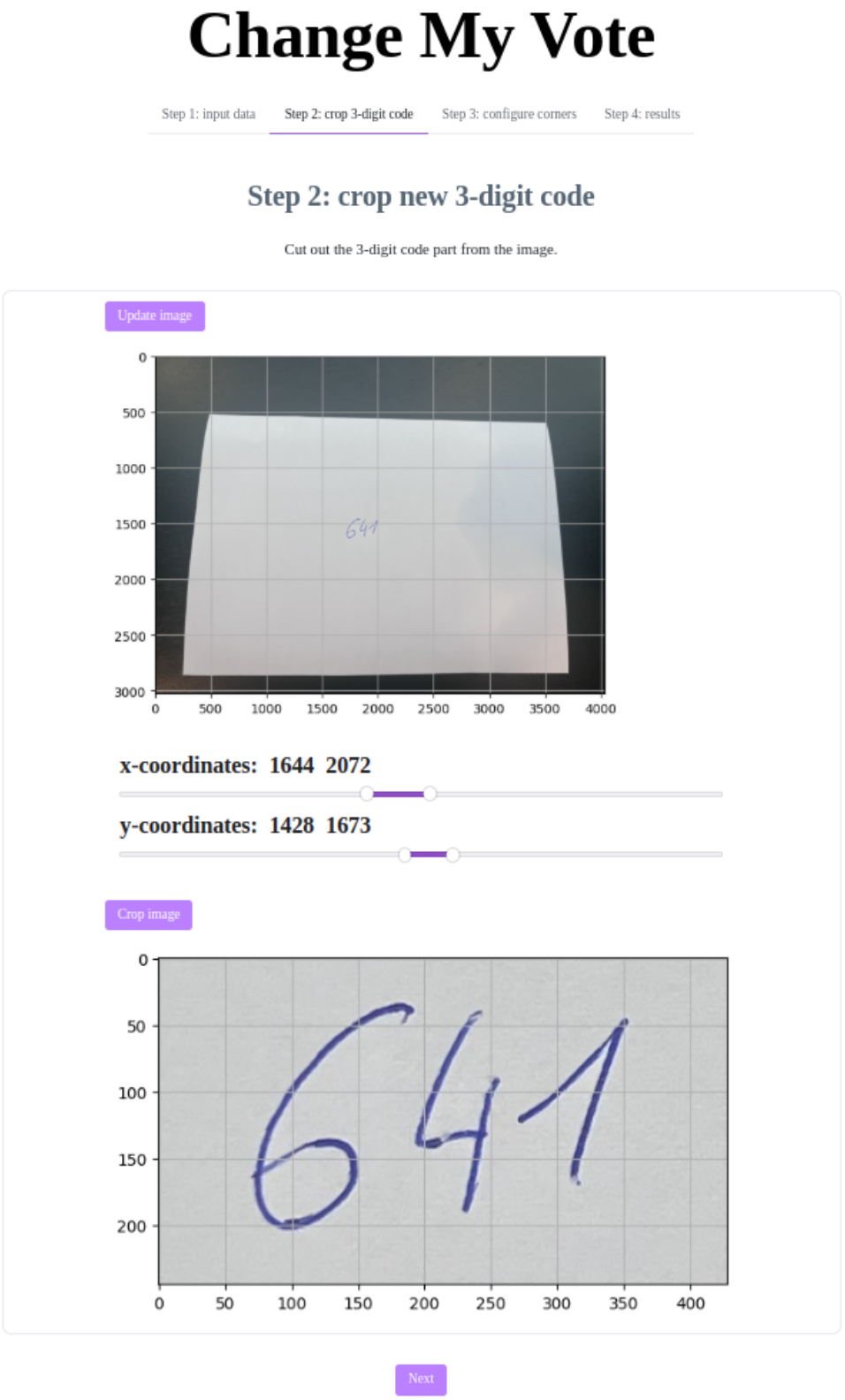


Figure 14. Cutting out the code from the image.

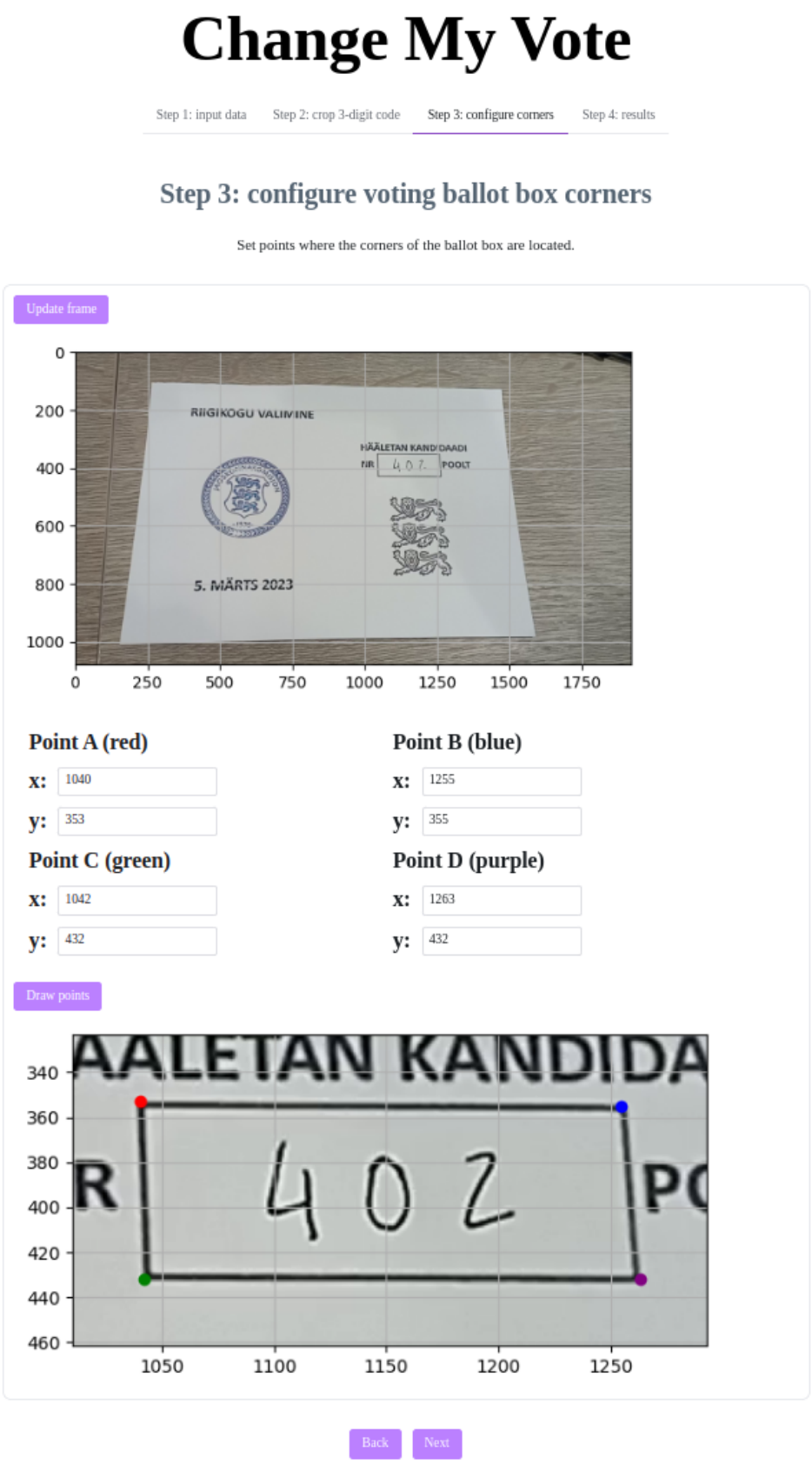


Figure 15. Configuring the voting ballot box corners.

Change My Vote

Step 1: input data | Step 2: crop 3-digit code | Step 3: configure corners | Step 4: results

Step 4: results!

Start converting

Part 3: getting YOLO predictions

Change My Vote

Step 1: input data | Step 2: crop 3-digit code | Step 3: configure corners | Step 4: results

Step 4: results!

Start converting

Here is the new video:

Figure 16. Starting the pipeline and getting back the result.

Appendix B: License

Non-exclusive licence to reproduce the thesis and make the thesis public

I, Anette Habanen ,
(author´s name)

1. grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright, my thesis

   Deepfakes for Paper Vote Privacy Defence ,
   (title of thesis)

   supervised by Jan Willemson and Sven Laur ;
   (supervisor´s name)

2. grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright;

3. am aware of the fact that the author retains the rights specified in points 1 and 2;

4. confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Anette Habanen
15/05/2025