BRNDTS - Technical Challenge

Development Report

David Carreira
June 2024

Contents

1 The pipeline

The whole pipeline leverages the principles of object detection and segmentation. The main goal is to produce realistic ad placement, hence the need to detect objects that we can replace the advertisement with. For that we, will work under the assumption that there is the presence of a television, windows and/or picture frames in the background. Because the surrounding environment is expected to be dynamic, a lot of obstructions and occlusions can occur, spanning the necessity to identify them and mask them.

With the aforementioned in mind I propose the following pipeline:

- **Step 1 -** Detect if there are televisions, windows and/or picture frames. If so, the code can proceed.
- **Step 2 -** Select the receptacle for the advertisement. There are multiple ways this can be done: at random, the one that creates a bigger area, if it is not occluded, etc.
- **Step 3 -** Detect occlusions and segment them out of the ad.
- **Step 4 -** Process the image to not look out of place. That can be achieved by applying a blur filter or finetuning brightness, contrast, exposure, etc.
- **Step 5 -** Perspective warp. Sometimes it is required to warp the advertisement in order to be in accordance with the perspective.
- **Step 6 (Optional) -** Burn-in prevention. When the final product is a video, it is essential to implement some burn-in prevention measure. This can be achieved by ever so slightly move the advertisement a few pixels in different directions through the span of time.

2 Test 1 - Pretrained YOLOv8

The logical first approach, is to employ a pretrained YOLOv8, and *ultralytics* distributes weights for both detection and segmentation. The detection task's weights were computed by training on Microsoft's COCO [6] or Google's OpenImagesV7 [1], whereas the segmentation task is only trained on MS COCO. An overview of both sets is resumed in Table 1.

2.1 Datasets' Overview

In order to solve our problem, the model must have been trained on a dataset that has, at least, the classes that we are looking for, i.e, televisions, windows or picture frames. This can be difficult because the object detection datasets are designed with a general purpose in mind, so images of less common objects might not be included. That being said, out of MS COCO'S 80 classes, it only has the television class. On the other hand OpenImagesV7 has classes for televisions, windows and picture frames, and other useful categories such as windows's blind, whiteboards, computer monitors, tablets etc. and a plethora of objects that might occlude the advertisement. Therefore, a model trained on OpenImagesV7 is the best choice for our problem.

Table 1 highlights how much OpenImagesV7 surpasses COCO both in terms of number of images and number of classes. It is expected for the OpenImagesV7's models to perform better independently of the context where it is deployed. However, there is a problem that must be taken

Table 1: A description of both datasets used by *ultralytics* to train their YOLOv8 implementation. Both instances of them have a set of labels that can be used to train the detection and segmentation task.

	Detection		Segmentation	
	COCO	OpenImagesV7	COCO	OpenImagesV7
# Images	1.2×10^5	1.7×10^6	1.2×10^5	9.4×10^5
# Classes	80	601	80	350

into consideration. On both datasets there is a great class imbalance. For instance, if we look for the class "person" and "television", COCO has, respectively, 66,608 and 4,768 occurrences, where in the case of OpenImagesV7, the distribution is 2,129,743 and 41,985. Henceforth, we can expect detections of our classes of interest to be difficult.

2.2 Models' Overview

We are looking for a good trade-off between performance and computational cost. For cases where a single detection is achieved, a more complex and demanding model can be selected, since it will be called only a few times and we need to make sure that the prediction is correct the first time. Assuming the following implementation is to be used in video, multiple frames will be fed to the model, hence, we need a faster and lightweight approach. Because there is a lot of data being processed, we can overcome worse performance (compared to heavier solutions) with bigger amounts of data. That is, there's a leeway to wrongfully classify a sample because we are performing more predictions over time.

Table 2: Models readily available to use. n, s, m, l and x are the size of the models in ascending order, "-oiv7" denotes models trained on OpenImagesV7 (the others are all trained on COCO) and "-seg" are the segmentation models.

	Detection										Segmentation				
	n	s	m	l	x	n-oiv7	s-oiv7	m-oiv7	l-oiv7	x-oiv7	n-seg	s-seg	m-seg	l-seg	x-seg
Params (M)	**3.2**	11.2	28.6	43.7	68.2	3.5	11.4	26.2	44.1	68.7	3.4	11.8	27.3	46.0	71.8
FLOPS (B)	**8.7**	28.6	78.9	165.2	257.8	10.5	29.7	80.6	167.4	260.6	12.6	42.6	110.2	220.5	344.1
Size (MB)	**6.2**	21.5	49.7	83.7	130.5	6.9	21.9	50.3	84.4	131.5	6.7	22.8	52.4	88.1	137.4

All in all, a good starting point would be to test a "middle ground" model. Table 2 presents our models' performance and computational cost metrics, and on a first analysis "s-oiv7" or "m-oiv7" seem to be the best choices. As previously mentioned in Section 2.1, we need a model trained on OpenImagesV7. Additionally, both networks appear to present a lower computational cost with much higher Floating Point Operations per Second (FLOPS) than the "n-oiv7" model without entering in the 44 millions of parameters and ∼80MB territory of the "l-oiv7".

2.3 Results

Running the image through our code, we get the results that can be seen in Figures 1, 2, 3, 4 and 5. We can observe that throughout the results, and as expected, there is a great difficulty in

detecting both the television and the picture frames on the wall. As previously mentioned on Section 2.1, this can be attributed to the highly imbalanced classes on the training dataset, therefore, the model requires further tuning.

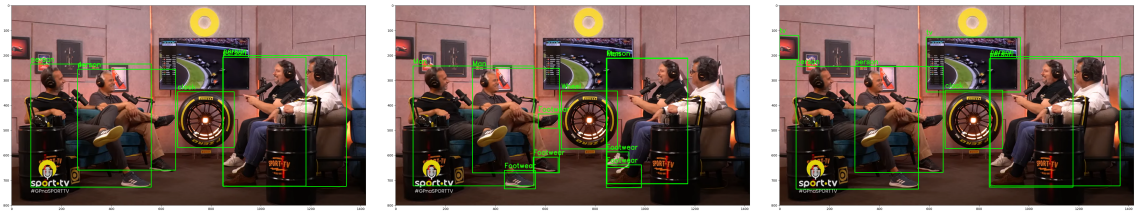

Figure 1: Results for the models: n, n-oiv7, n-seg (left to right order).

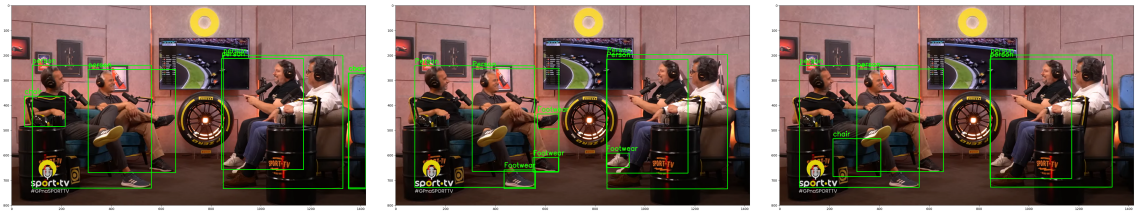

Figure 2: Results for the models: s, s-oiv7, s-seg (left to right order).

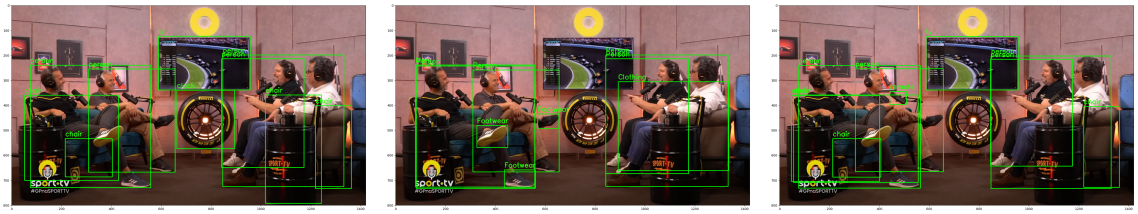

Figure 3: Results for the models: m, m-oiv7, m-seg (left to right order).

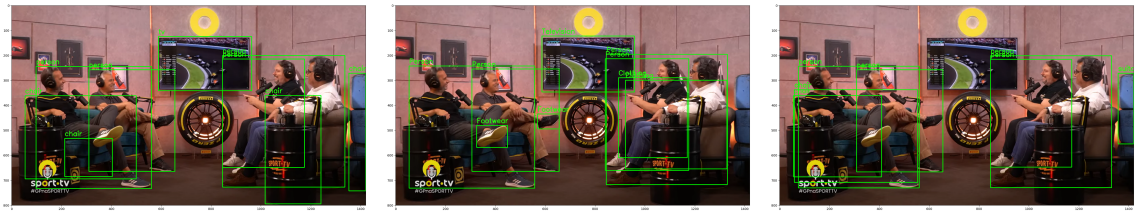

Figure 4: Results for the models: l, l-oiv7, l-seg (left to right order).

3 Test 2 - Meta's Detectron 2

For a sanity check, I ran the image through Facebook's Detectron2 [11]. However, because the models are also trained in the COCO dataset, the frames on the wall were also not detected.

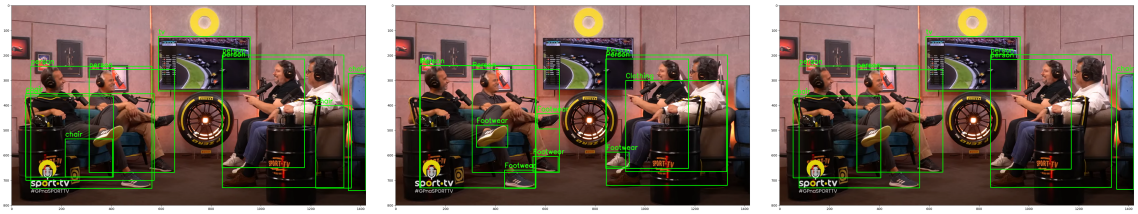

Figure 5: Results for the models: x, x-oiv7, x-seg (left to right order).

Nevertheless, the results are still interesting and it might be a framework worth of investigating since it offers state-of-the-art object detection and segmentation models, such as TensorMask [3] or Faster R-CNN [8]. The detections boxes can be seen in Figure 6.

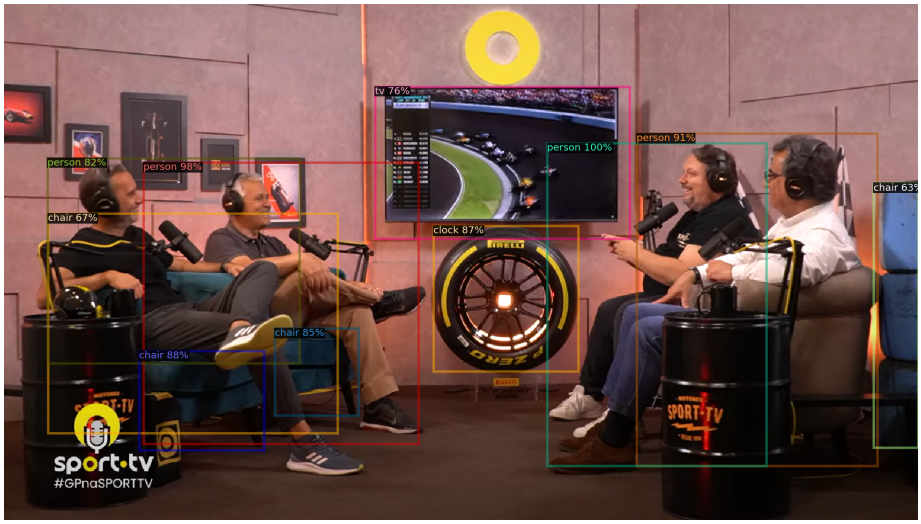

Figure 6: Detectron2 results. As expected the picture frames on the wall were also not detected, due to a COCO exclusive training procedure.

4 Test 3 - Confidence Threshold

A possible solution to our low detection rate is to reduce the confidence threshold of the model. Every detection has a confidence score and if it is less than a predefined threshold, the model rejects it as a possible detection of the class. The default value assigned by *ultralytics* is 0.25, but after experimenting, and as proven in Section 2.3, that value is too high. After testing, 0.01 appears to be the highest possible confidence threshold that produces all the intended detections. In the next section, the results from our methods of choice (s-oiv7 and m-oiv7) are highlighted.

4.1 Results

Allowing the model to be less strict on its detections helped on detecting the possible advertisement placement, and as a measure to reduce computational cost, only the classes of interest to the problem are being detected. The small model was able to detect all the frames and the television, wheres the medium one failed on detecting one of the frames. At first sight, that might not make sense, the expectation is for the more complex model to produce better results. But in this case, the more complex model accentuated the class imbalance previously mentioned, since its bigger depth and breadth biased [12] the model to better learn the abundant classes during training. Thus, moving forward, the model of choice is s-oiv7.

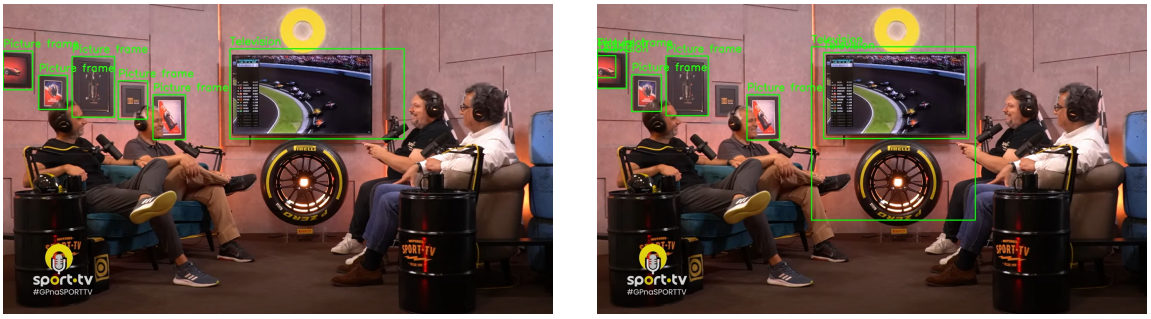

Figure 7: Detection results for s-oiv7 (left) and m-oiv7 (right).

5 Advertisement Placement Methodology

The detection stage produces the bounding boxes for our locations, however, they cannot be used directly due to their exaggerated permissiveness. Instead, they are going to be used as Regions of Interest (RoI) to intermediate processing stages, namely edge detection refinement, occlusions segmentation and realism improvement measures, as in perspective warp correction, constrained resizing and other aesthetic functions. The main intended purpose of these stages is to find more refined bounding boxes that better represent the edges of the potential ad locations and as a consequence enhance lifelikeness. Additionally, the use of RoI reduces the computational cost because only a small portion of the image is leveraged. For a single picture the improvement might not be noticeable, but for high demanding frame computation the overhead savings add up.

5.1 Refined Edge Detection

To improve computational cost and speed, for this step I decided to use classic Computer Vision techniques instead of Deep Learning-based ones. More specifically, Otsu thresholding [7] as a base and other image processing techniques, producing a more robust approach than the usual Canny Edge detector [2]. Otsu operates by iteratively thresholding the image in order to maximize the inter-class variance of pixel intensities, allowing it to separate all the elements.

First, the contrast and brightness are adjusted, the noise is reduced with OpenCV's Mean Shift Filter, and a small margin is added to the original RoI so more background is available around the edges. This helps the detection because it increases the gradient and variance between different

pixel clusters. Subsequently, after processing the grayscale version of the image, the Otsu method generates the binary RoI that is passed to a contour finding method that returns a warped bounding box to add perspective or a non warped version. Out of all the possible contours in the binary image, I am assuming that the best contour is the outermost one, that is, the one with the bigger area. Figure 8 exhibits the described process.

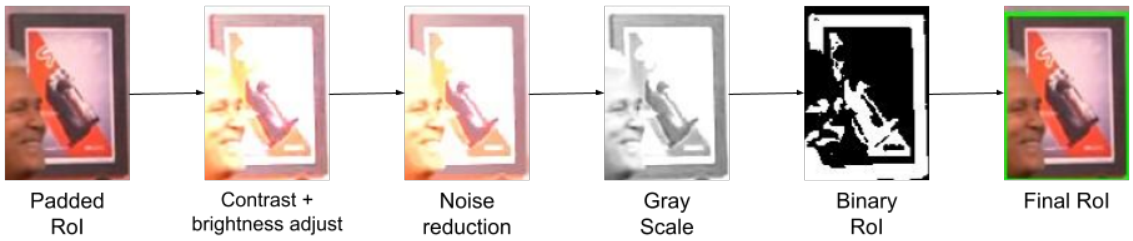

Figure 8: Edge detection process using classic Computer Vision Techniques. It involves padding the RoI, adjusting contrast and brightness, reducing noise, converting to grayscale, thresholding to a binary map, and applying an edge detection algorithm.

However, if the main interest is accuracy, this problem could have been easily solved with a small neural network - for instance the highly regarded MobileNet [5] - specifically trained for shape detection. Because the training images would be only shapes, the data could be generated synthetically using OpenCV and automatically annotated with the inherent coordinates from the generation (it would even be possible to add noise to the data for a better generalization of the model).

5.2 Segment Occlusions

To remove the occlusions from the front of the ads we could, once again, take advantage of the previous Otsu-based method, yet it would require too much finetuning of handcrafted features too achieve a inferior results compared to *ultralytics* segmentation solution. As noted before, this model was trained in the COCO dataset, so it incurs in the same disadvantages. Notwithstanding, it is still capable of handling most of the real-world possible occlusions.

Similarly to the previous method, the RoI are extracted and used to produce a binary mapping, in this case a binary mask of the occlusion. By converting the RoI to a RGBA (Red Green Blue Alpha) image, extracting its alpha channel (transparency) and performing a bitwise "and" between it and the binary mask, is as if the occlusion itself crops everything around it. This is done for every occlusion, a composite occlusion mask is generated and superimposed over the original image. Said procedure is reported in Figure 9.

5.3 Improve realism

How real the advertisement looks is very important, therefore, some considerations were taken into account. Namely, where to place the ad, the need to correct the perspective of the image and guarantee its aspect ratio, and optional aesthetic necessities, such as the possibility to add a border, color correction or resolution matching.

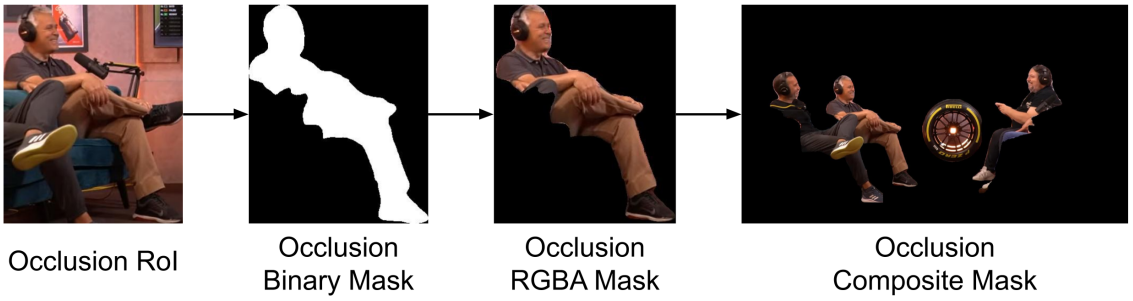

Figure 9: Process of segmenting the occlusions out of the original picture. First the RoI is determined, then a binary mask is produced, followed by a RGBA masks that create a composite of all the occlusions.

To choose where the ad's image is placed, there are 3 options: "total", "random" and "area. The method modifies the ad in accordance with the destination, and "total" runs through the list of possible locations applying the ad all of them, "random" selects an arbitrary position and "area" calculates the location with the bigger area, optimizing the ad exposure to the user. Regarding the perspective, it is similarly achieved to the fine edge detection, however, its main difference is that out of the 4 corners of the shape found, the resulting bounding box is a trapezoid instead of a rectangle. Finally, the correct dimensions are assured by computing the aspect ratio and evaluating if it is bigger than 1 (vertical image) or less (horizontal image). Accordingly, the advert is resized by maximizing either the width or height, and the resulting empty space is filled with a color.

The aesthetic parameters are a major help of warranting that the ad does not look out of place. If the option to include a border is set, then it can be easily customized in terms of thickness or color (by passing a tuple of RGB channels' values). Relating to the color correction it is achieved through 2 different modes: "histogram" or "color_transf". Both of the modes work on the LAB color space since changes to its channels produce changes more perceptible to the human eye than operating over the usual RGB space. While "histogram" matches the histogram of the advertisement with the source image, "color_transf" calculates the mean and standard deviation of the destination picture and adjusts the advertisement's statistics. Finally, the smoothing operation smoothes the ad and matches its apparent resolution to the destination's one. This is achieved by resizing the picture by a reduction coefficient defined by the user, then resizing it back to the original size.

5.4 Final Results

Now that all the detection, segmentation and other miscellaneous methods are implemented, the final results encompass all of the work done so far. In this stage is where it is possible to personalize the ad and the method has the following tunable parameters:

- **down_scale**: Reduces the size of the advertisement image to account for possible borders or just so the image fits better.
- **bordered**: Boolean that determines if the ad has a border or not.
- **border_t**: Border thickness.

- **border_color**: Tuple (R, G, B) that selects the color of the border.
- **warp**: Boolean that determines if the perspective correction is applied.
- **color_correction**: Two modes are available ("histogram" or "color_transf").
- **smoothing**: Matches the ad apparent resolution to the source's and smoothes.
- **mode**: Defines where the advertisement will be placed ("total", "random" or "area").
- **save**: If true, saves the picture to the current working directory.

Figure 10: Results without color correction and smoothing, and the different insertion modes available ("total", "area" and "random").

Figure 11: Results with color correction ("color_transf") and smoothing (0.65), and the different insertion modes available ("total", "area" and "random").

All in all, a solution to the initial problem is successfully proposed. Figures 10 (no color correction or smoothing) and 11 (color correction using the "color_transf" mode and a smoothing of 0.65) showcase some of the results that can be achieved with the available parameters. A bigger resolution of the depicted results is available in Appendix A.

6 Optimizations/Possible solutions

With more time available it would be interesting to explore other options. Although *ultralytics* is a powerful and intuitive framework, it quickly achieves its peak performance. One of its main drawbacks is the "black box" effect, meaning that it hides several stages of a deep learning domain. For instance, the whole training routine is hidden and that removes the finer debugging and optimization that a custom Pytorch/Tensorflow train can deliver. Different networks and/or datasets

require different training parameters, and as a default automatically optimizing it is not the best approach for advanced users that want a high performing state-of-the-art model. When dealing with a neural network it is of the utmost importance that we are in control of everything that is happening, and *ultralytics* removes that from us both during the training and inference time. Other problem, but that is more of a personal gripe of mine, is the fact that *ultralytics* still has not published a paper on the YOLOv8 to this day. Therefore, I have some suggestions that ideally would be tested on other object detection models, but for now I will focus only on the YOLOv8.

First of all, I would go about training the YOLOv8 by initializing its weights with the OpenImagesV7 ones and then retraining it on the classes of the LVIS dataset [4] that are relevant to where the model is going to be applied. This way, the training converges quicker (due to the domain aware weight initialization) and solves part of the class imbalance problem by leveraging a dataset with more images per class and excluding from the training classes that are not of interest. If the LVIS is not an option and there is a low abundance of data, there's always the possibility of taking what we have and use it as a prompt in a Generative Adversarial Network (GAN), such as Stable Diffusion V2 [9, 10], to produce different scenarios.

If time is a concern, I imagine that finetuning the YOLOv8 on the LVIS would also produce interesting results. This can be achieved by, once again, selecting only the classes that are relevant to the context. However, if the desired classes are not available in the original model, the problem starts to get more complicated, but not impossible to solve. For instance, the COCO trained YOLOv8 does not have the a class for "picture frames" and if we want to use LVIS to fix that we either need to train from scratch (wasting time) or modify the model's head. By creating a custom head designed for the new classes that are not included in the original model, we can train only that portion and then merge it with the pretrained model's head.

References

[1] Rodrigo Benenson and Vittorio Ferrari. "From colouring-in to pointillism: revisiting semantic segmentation supervision". In: *arXiv preprint arXiv:2210.14142* (2022).

[2] John Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[3] Xinlei Chen et al. "Tensormask: A foundation for dense object segmentation". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 2061–2069.

[4] Agrim Gupta, Piotr Dollar, and Ross Girshick. "Lvis: A dataset for large vocabulary instance segmentation". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 5356–5364.

[5] Andrew Howard et al. "Searching for mobilenetv3". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 1314–1324.

[6] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer. 2014, pp. 740–755.

[7] Nobuyuki Otsu. "A Threshold Selection Method from Gray-Level Histograms". In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (1979), pp. 62–66. DOI: 10.1109/TSMC.1979.4310076.

[8] Shaoqing Ren et al. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems* 28 (2015).

[9] Robin Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2021. arXiv: 2112.10752 [cs.CV].

[10] *Stable Diffusion Version 2*. https://github.com/Stability-AI/stablediffusion.

[11] Yuxin Wu et al. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[12] *YOLOv8*. https://github.com/ultralytics/ultralytics/issues/189.

A Appendix

Figure 12: Results without color correction and smoothing for the "total" mode.

Figure 13: Results without color correction and smoothing for the "area" mode.

Figure 14: Results without color correction and smoothing for the "random" mode.

Figure 15: Results with color correction ("color_transf") and smoothing (0.65) for the "total" mode.

Figure 16: Results with color correction ("color_transf") and smoothing (0.65) for the "area" mode.

Figure 17: Results with color correction ("color_transf") and smoothing (0.65) for the "random" mode.