ChatGPT

Detecting Text-Based Watermarks in Images

Overlay text watermarks present unique challenges compared to ordinary scene text. These watermarks are often semi-transparent and placed over diverse image content, making them **low-contrast** and easily confused with background elements. A robust solution needs to accurately isolate the watermark text at the pixel level. Below, we survey candidate models and recommend the best option for this task, considering accuracy, support, and efficiency.

Challenges of Overlay Text vs. Scene Text

Watermarks differ from natural scene text in that they are *deliberately blended* into images. Key challenges include:

- **Transparency and Low Contrast:** Watermarks use semi-transparent text, causing low contrast with backgrounds [1] [2]. Models must detect faint text edges without many false positives.
- **Varied Placement and Orientation:** Like scene text, watermarks can appear anywhere, at various scales and orientations (e.g. diagonally across an image).
- **Confusing Background Content:** An algorithm must distinguish watermarks from actual scene text or background patterns [3] [4]. For example, writing on a sign or T-shirt in the image is *not* a watermark, but may look similar.

Because of these factors, **segmentation-based text detectors** are favored. They label text at the pixel level, capturing fine-grained shapes of letters (even if faint), rather than just coarse bounding boxes [5]. The model should output a segmentation mask for the watermark text.

Candidate Models for Text Overlay Detection

Below we examine prominent text detection models and their suitability for watermark text:

- **CRAFT (Character Region Awareness For Text Detection):** CRAFT is a popular deep learning model that predicts character-level heatmaps and affinity between characters [6]. It effectively detects difficult text areas by assembling characters into words. **Pros:** High recall for small or faint text; open-source PyTorch code available [7]; used in tools like EasyOCR. **Cons:** Tends to produce many false positives on textured backgrounds without special filtering [8]. In tests on overlay text, CRAFT "hallucinated" non-text regions as text until heuristics were applied [8]. It outputs character/affinity maps which can be thresholded to pixel masks, but extra steps are needed to merge characters into a watermark region. Maintenance is moderate (official code released in 2019). Good for research, but not the most lightweight or turnkey solution now.

- **PSENet (Progressive Scale Expansion Network):** An earlier segmentation-based detector (CVPR 2019) that generates **multi-scale text kernels** which expand to cover entire text instances [9] [10]. **Pros:** Handles arbitrary-shaped text (curved or oriented) well by progressively growing regions. **Cons:** The multi-step post-processing (expanding kernels) is somewhat complex and can be slower

[11]. The model is fairly heavy (ResNet-based). While official code exists [12], newer approaches have largely surpassed PSENet in both accuracy and simplicity. It's accurate, but not particularly lightweight or actively updated now.

- **PAN / PAN++ (Pixel Aggregation Network):** An efficient segmentation-based detector (ICCV 2019) known for speed [13]. It uses a pixel embedding/fusion approach to group text pixels (with a feature pyramid for multi-scale detection). **Pros:** Designed for **efficiency and accuracy**, achieving real-time speeds with a lightweight backbone [14] [15]. Official PyTorch implementations of PAN and its enhanced version PAN++ are available [12]. In one benchmark, PAN with a ResNet18 backbone reached ~18 FPS on standard text data [16]. **Cons:** Requires a C++ post-processing step to cluster pixels (the codebase includes custom CUDA kernels for this [17]). PAN++ extends PAN to end-to-end text spotting (detection + recognition) [18], which is powerful but beyond detection needs. Overall, PAN is a strong candidate if one prioritizes speed and has the capability to compile the custom layers. It outputs pixel masks for text regions and performs well on curved or rotated text. Active maintenance is moderate (often referenced in OpenMMLab's MMOCR toolkit).

- **DBNet (Differentiable Binarization Network): This model stands out as a top candidate for watermark text.** Proposed in 2020, DBNet is a segmentation-based text detector that introduced a **differentiable binarization (DB) module** to produce text masks [19]. Instead of manually thresholding the probability map, DBNet learns an optimal threshold map, enabling end-to-end training of the segmentation-to-mask pipeline [20]. **Pros:** It achieves **state-of-the-art accuracy and speed** on scene text benchmarks [21]. For instance, with a lightweight ResNet-18 backbone it reaches an F-measure of ~82.8% at **62 FPS** on a standard dataset [21] – indicating it's both accurate and extremely fast. Even with ResNet-50, it maintains high accuracy (Hmean ~85% on ICDAR2015) while staying real-time (20–30 FPS range) [22]. Multiple open-source implementations exist (the authors' code, plus reimplementations like **MMOCR** and **PaddleOCR**), so support is excellent. The academic backing is strong: it was published at AAAI 2020 and **is widely adopted as an OCR detection baseline** [23]. Importantly, DBNet outputs a fine-grained **pixel mask** for each text region by design – exactly what we need for watermark segmentation. It's also relatively lightweight in model size (tens of MB). Community feedback highlights its superior balance of accuracy and efficiency, calling it "faster, more accurate, and lightweight" than previous segmentation models [23]. **Cons:** DBNet's post-processing (shrinking and expanding text regions) can sometimes miss extremely long text lines [24]. In practice, this means a very long watermark string might be segmented into pieces or have gaps if default settings aren't tuned. However, this can be mitigated by adjusting the dilation (unclip) parameters [24]. Another consideration is that the model, as released, was trained on scene text datasets; it may not have seen highly transparent text. Fine-tuning on images with semi-transparent text (or augmenting training with lowered contrast) would further improve its watermark detection robustness. Despite these minor limitations, DBNet meets virtually all the requirements: pixel-level mask output, high accuracy on challenging text, efficient inference on a 12GB GPU, and strong open-source support.

- **DBNet++ (Adaptive Scale Fusion):** An improved version introduced in 2022 that adds a feature fusion module for multi-scale text [25]. **Pros:** Even higher accuracy (Hmean approaching 88–89% on benchmarks) [26] while retaining fast inference. It basically inherits all of DBNet's strengths and pushes accuracy further on small and large text by adapting to scale. This is likely the *single best* model for our needs, as it was designed to be a direct upgrade of DBNet. It is implemented in open-source libraries (MMOCR has DBNet++ models ready [27]). **Cons:** Slightly larger or slower than DBNet

due to the extra fusion module, but still within real-time range on GPU (in one study, ~10 FPS was reported for DBNet on a curved text dataset and similar or slightly lower for DBNet++ on standard images) [28]. Given that speed <10s per image is acceptable, DBNet++ easily clears this bar (in fact, ~0.1–0.2s per image on GPU).

- **Other Notable Mentions: FAST** (Faster Arbitrary Shape Text detector, 2021) is another segmentation model with a minimalist approach to text kernels [29]. It's implemented in DocTR and aims for speed, but it is less proven on a wide scale than DBNet or PAN. **FCENet** (Fourier-Contour Embedding, 2021) predicts text contours via Fourier descriptors [30] – it's academically interesting and accurate for complex shapes, but it's heavier and not needed unless watermarks have very irregular shapes. **ContourNet, TextSnake, TextBPN** and others exist, but they either have similar drawbacks (complex post-processing or less support) or focus on niche cases (e.g. dense curved texts). For watermark overlays, the simpler and well-supported models above are more than sufficient.

- **YOLOv8 (and Detection Transformers):** Some recent works have tried framing watermark text detection as a generic object detection problem. For example, a 2023 study introduced *YOLOv8-DCE* specifically for text watermarks [31], adding enhancements to Ultralytics' YOLOv8. **Pros:** Models like YOLOv8 are lightweight and fast, and the official implementation is actively maintained. They can detect text *bounding boxes* reliably. **Cons:** Off-the-shelf YOLOv8 treats text like any object, outputting boxes rather than segmentation masks. While one could use YOLOv8's instance-segmentation variant to get masks, it still may struggle to capture the exact translucent shapes of letters without specialized training. Moreover, scene text detectors have a built-in advantage: they are trained on text-specific features (e.g. strokes). A general detector might need a lot of watermark-specific training data to match the accuracy of DBNet or PAN. Thus, YOLO-based approaches are possible and fast, but they require custom training and do not naturally output the pixel-level masks that segmentation models do. They are a viable plan B if integration ease trumps mask accuracy – but in our case accuracy is the priority.

- **Watermark Removal Networks (SplitNet, etc.):** Interestingly, some **multi-task networks** designed for removing watermarks can also detect them. *SplitNet* (AAAI 2021) is one such model: it first predicts a watermark mask and a rough removed-image, then refines the result [32]. It was trained on synthetic watermarked images to perform *blind watermark removal*. **Pros:** It explicitly produces a watermark segmentation mask as part of its pipeline [32]. This mask is exactly the pixel-level output we want. The authors released code and even a pretrained model [33] [34]. **Cons:** This approach is significantly heavier – essentially a pair of U-Net style networks with attention, not optimized purely for detection speed. It's aimed at *removing* the watermark, so integration just for detection means running a lot of extra computation (and possibly requiring many GPU resources). Additionally, research has shown that a dedicated text detector can actually outperform these removal models on just detecting the watermark. In an overlay-text detection study, a fine-tuned CRAFT detector outperformed SplitNet at classifying/detecting watermarked regions (CRAFT-based method F1 0.95 vs. SplitNet-based F1 0.59) [35] [36]. This suggests that specialist text detectors are better at finding watermarks than a do-it-all removal network, unless you specifically need the removal functionality. Thus, while SplitNet is academically interesting, it's not the most practical or accurate choice solely for detection.

Recommendation: DBNet++ (Differentiable Binarization Network)

Taking all factors into account, **the best model for detecting text-based watermarks is DBNet, especially the improved DBNet++ version.** It offers an ideal combination of accuracy, granularity, and practicality:

- **Pixel-Level Mask Output:** DBNet++ was designed to produce segmentation maps of text regions, which are then binarized into precise text masks [19] [20]. This means it can directly output the *shape of the watermark text* in the image – satisfying the requirement for fine-grained masks rather than just boxes. Letters with holes, curved text, etc., are all captured in the mask. Figure 7 of the DBNet paper visually shows the model detecting various text shapes (curved, vertical, long lines) with corresponding probability and threshold maps [37]. This capability is crucial for transparent watermarks that might not form a contiguous high-contrast block (DBNet will still highlight the faint stroke regions).

- **High Accuracy on Challenging Text:** DBNet++ achieves **state-of-the-art detection accuracy** on multiple benchmarks [21] [26]. For example, on the Total-Text curved text dataset it reports Hmean in the 83–85% range, and on standard ICDAR15 scene text it can exceed 86–88% Hmean [38] [26]. This is on par with the best text detectors in research as of 2022. In practical terms, it means it will catch most text instances, even those that are low-contrast or oddly oriented. Watermark text (often large and translucent) should be well within its capabilities. If needed, one can fine-tune DBNet on a small watermark-specific dataset to further boost its sensitivity to transparency – but even out-of-the-box, it's likely to work well, since it was trained on a variety of backgrounds (SynthText, ICDAR, etc.) [21].

- **Speed and Efficiency:** Despite its high accuracy, DBNet++ remains lightweight enough for **local deployment**. It was explicitly designed as a *real-time* detector [39]. The use of a differentiable thresholding module eliminated expensive post-processing steps present in older models (no graph clustering or iterative expansion needed – it's a single forward pass and simple threshold) [11] [19]. The paper reports **62 FPS** on a 1080Ti GPU with a ResNet-18 backbone [39], and even with ResNet-50 it was around 32 FPS (which is ~30ms per image at standard resolution) [22]. On a modern consumer GPU (e.g. 12GB NVIDIA RTX series), you can expect inference times on the order of **50–100 milliseconds per image**, far below the 10 second requirement. Even processing high-res images or adding slight overhead for mask extraction will keep it well under 1 second. This means watermark detection can be done on-the-fly in a CLI tool or web service without noticeable lag.

- **Resource Requirements:** A 12GB VRAM GPU is more than sufficient. During inference, DBNet++ with ResNet-50 uses a few hundred MB of GPU memory for a single image (exact usage depends on image size and model size). The model weights are on the order of 50–100 MB. It also runs on CPU if needed (with optimization, ~1–2 s per image on a strong CPU), thanks to its streamlined nature. In summary, it **comfortably fits on consumer hardware** – no need for TPUs or 40GB GPUs.

- **Active Open-Source Support:** One major advantage of DBNet/DBNet++ is the robust support in open-source OCR ecosystems. For instance:

- **MMOCR (OpenMMLab)** includes DBNet and DBNet++ implementations with pretrained weights [40] [27]. MMOCR is a well-maintained toolkit; using it, you can load a DBNet++ model in a few lines of

Python and run detection on images. It even provides utilities to visualize the text masks or polygons.

- **PaddleOCR** (a popular PaddlePaddle-based OCR library by Baidu) uses DBNet as one of its default detection models [41]. PaddleOCR is geared towards deploying OCR in production and supports many languages. If Python and PyTorch are not requirements, PaddleOCR's CLI or Python API can perform detection with a pretrained DBNet (there's a "OCR Detection" pipeline which returns text region polygons). It's actively updated and optimized (with quantized and mobile versions of DBNet available for ultralight use cases).
- **Mindee's DocTR** library (PyTorch/TensorFlow) implements DBNet (and a variant of LinkNet) for document text detection [42]. DocTR is very user-friendly – you can install it via pip and do `model = doctr.models.detection.db_resnet50(pretrained=True)` to get a DBNet ready to run. This is a testament to how well-adopted DBNet is in the community.

- **Official and Community Repos:** The original authors released code on GitHub (MhLiao/DB) and there are reimplementations like `WenmuZhou/DBNet.pytorch` [43]. While some of these might not be updated to the latest frameworks, they serve as good references. The community forums (e.g. GitHub issues in MMOCR) show active discussion about DBNet, indicating it's widely used and understood (e.g. issues like handling long text in DBNet [24] have community-provided solutions).

- **Integration and Setup:** If using **PyTorch**, the simplest path is to use MMOCR or DocTR:

- *MMOCR approach:* Install MMOCR (which relies on MMDetection). Load the DBNet++ model and call the inference API. This gives you polygon coordinates of detected text or binary masks. The dependency footprint is somewhat large (MMOCR brings in a bunch of computer vision libs), but it's a proven solution.
- *DocTR approach:* Install `doctr` via pip. Use `ocr_predictor(det_arch='db_resnet50', reco_arch=None)` – this sets up a DBNet detector (ResNet50 backbone) without recognition. Then use `predictor.det_predictor(image)` to get detected regions. DocTR will output quadrangles for each text region, which in the case of a watermark would likely correspond to the text's bounding polygon. Converting that to a pixel mask is straightforward (fill the polygon on a blank mask).
- *Direct use:* If you prefer minimal dependencies, you can use the standalone PyTorch implementation of DBNet. It may require building some CUDA ops (for the "unclip" operation that expands the mask a bit). For example, WenmuZhou's repo provides a `predict.py` that reads an image and outputs a mask for text [44]. You'd need to install PyTorch and OpenCV. This gives fine control to modify the post-processing if needed (e.g. adjust the binarization threshold or unclip factor for faint text).
- *PaddleOCR approach:* Install `paddleocr` via pip, then simply call `PaddleOCR(det_model_dir='path/to/dbnet_model', rec=False)`. PaddleOCR's detection model returns boxes by default, but you can get the polygon output by enabling the unclip ratio and setting `det_db_box_thresh` appropriately. PaddleOCR is highly optimized (it can use MKL on CPU, etc.), making it a good choice for a CLI tool that might run on various platforms.

Regardless of approach, **Python integration is well-supported**. These models typically depend on standard libraries like PyTorch (or Paddle), OpenCV for image preprocessing, etc. No exotic dependencies are needed beyond the deep learning framework.

- **Accuracy on Watermarks & Limitations:** In practice, DBNet++ should accurately detect most overlay texts. It's especially good at **irregular and low-contrast text** due to the segmentation training [23]. That said, a few limitations to note:

- Extremely faint watermarks (opacity very low) might be near the threshold of visibility. If the model wasn't trained on such low contrasts, it might miss them or detect only partial text. Fine-tuning on synthetically faded text can address this. Another trick is to run the model at a higher image resolution than normal, which can help it pick up lighter strokes.
- If a watermark is very large text spanning the entire image width, the default mask dilation (unclip) might not connect all letters. This is what the community noted with long text lines [24]. The solution is to increase the `unclip_ratio` (which expands the detected regions a bit more to close gaps) – a simple parameter change. After such tuning, long watermark words should come out as a single mask.
- The model will detect **any text** in the image, not just watermarks. If an image contains both a watermark and some scene text (say a sign in the photo), DBNet++ will find both. Separating which text is the watermark might require an extra heuristic (e.g. perhaps the watermark will be the largest text, or have a known string like "© Company"). If you know the watermark text or pattern, you can filter detections by size or content. Otherwise, one might combine detection with a classifier that decides if an image has a watermark [4], but that's beyond this scope. Generally, if the use-case is "detect and remove the watermark," detecting all text is still fine as a first step (you might then remove the overlay text while leaving any genuine scene text intact).
- **Transparency handling:** DBNet++ itself doesn't explicitly model transparency – it just sees RGB pixels. However, because it learns to segment text, it often picks up the edges of transparent text (where the color changes). Empirically, segmentation detectors can detect watermarks that have as low as ~0.2–0.3 alpha (20-30% opacity), especially if the text is big. If the watermark is *very* transparent (nearly invisible), no vision model will reliably detect it without some form of image difference analysis or knowing the watermark pattern (which ventures into digital watermark detection techniques). For typical watermark opacities used by stock photo sites (often around 50% or a repeating pattern), DBNet++ should work. NVIDIA's OCR team notes that color and transparency can cause missed detections if not seen in training, reinforcing the idea to augment or fine-tune on such cases [2].

In summary, **DBNet++** is our top recommendation due to its excellent accuracy in finding text of all shapes (critical for overlay text), its pixel-wise mask output, and its efficiency and support in open-source tools. By integrating DBNet++ into a Python-based pipeline (via MMOCR, PaddleOCR, DocTR, or a direct model), one can achieve watermark text detection well under 1 second per image on a 12GB GPU, with highly accurate segmentation masks.

Setup and Usage Details

To concretize the recommendation, here's what using DBNet/DBNet++ might look like:

- **Installation:** If using PyTorch, install the OpenMMLab MMOCR package. For example: `pip install mmocr` (along with its dependencies). Alternatively, install Mindee DocTR via `pip install python-doctr`. For PaddleOCR: `pip install paddleocr`. These packages will handle model downloads internally.
- **Model Selection:** Use the DBNet++ model weights. In MMOCR, you could do: `from mmocr.apis import TextDetector; model = TextDetector(det='DBNetpp')`. In DocTR: `detector = doctr.models.detection.db_resnet50(pretrained=True)`. In PaddleOCR: initialize with `det_algorithm='DB'` (which loads a DBNet model).

- **Running Inference:** Provide the image to the model's inference function. The output will typically be a list of text regions. For instance, MMOCR returns a list of polygons (each polygon is the mask outline of one text instance). You can render these on the image to visualize, or create a mask image where those polygons are filled in.
- **Post-processing:** Because we specifically want the watermark mask, you might apply an extra step to isolate the likely watermark. If the watermark text is known or has a predictable size/location, use that. Otherwise, if the goal is simply to remove it, you can remove *all* detected text overlays (assuming the image normally shouldn't contain text except the watermark). If needed, a simple heuristic is to remove the largest semi-transparent text region detected.
- **Example:** Suppose we have a photo with a faint "© ExampleCorp" watermark across it. DBNet++ will output a mask covering the "ExampleCorp" text. We could take that mask and feed it to an inpainting algorithm to actually remove the text. Or if we just need to **detect** (perhaps to flag images containing watermarks), we could compute something like the mask area or confidence. The detection confidence from DBNet is typically high for the correct text region (since it was optimized end-to-end for that). In the overlay-text detection dataset by Amazon researchers, a text-detector-based approach achieved 0.95 F1 in distinguishing images with overlays [45], meaning these detectors are extremely reliable when appropriately configured.

Conclusion

DBNet++ emerges as the best choice for text-based watermark detection, combining the strengths of segmentation precision and efficient inference. It is academically validated and widely available in open-source. By using DBNet++ in a Python OCR pipeline, one can accurately segment watermark text on images in **under a second per image** on consumer GPU hardware, meeting the practical needs of a CLI tool or web service. The model's fine-grained masks will outline the exact pixels of the watermark, facilitating downstream tasks like removal or analysis.

While other models (CRAFT, PAN, etc.) are viable, they either require more tweaking to avoid false positives or don't offer the same balance of speed and support. YOLO-based detectors can be very fast, but need custom training and still fall short in mask accuracy. In contrast, DBNet++ offers a **proven, ready-to-deploy solution** with an excellent track record in scene text detection – which directly translates to the watermark overlay scenario. With proper setup and minor fine-tuning for transparency, it should reliably detect overlay text watermarks, enabling automated handling of protected images.

Sources:

- Liao *et al.*, *Real-Time Scene Text Detection with Differentiable Binarization*, AAAI 2020 – introduced DBNet [20] [21].
- Liao *et al.*, *Real-Time Scene Text Detection with Differentiable Binarization and Adaptive Scale Fusion*, 2022 – DBNet++ improvements [27] [26].
- Baek *et al.*, *CRAFT: Character Region Awareness for Text Detection*, CVPR 2019 – character-affinity detection (open-source) [8] [6].
- Wang *et al.*, *Efficient and Accurate Arbitrary-Shaped Text Detection with Pixel Aggregation Network*, ICCV 2019 – PAN detector (real-time text detection) [46] [28].
- Cun & Pun, *Split Then Refine: Stacked Attention-guided ResUNets for Blind Watermark Removal*, AAAI 2021 – watermark removal network (produces masks) [32] [35].

- **Open-source libraries:** MMOCR toolbox by OpenMMLab [40], PaddleOCR documentation [41], Mindee DocTR README [42] – all highlighting usage of DBNet for text detection.
- Kudale *et al.*, *Textron: Weakly Supervised Multilingual Text Detection*, WACV 2024 – notes on modern text detectors; specifically praises DBNet's speed/accuracy/lightweight nature [23].
- Amazon Science, *"Rooms with Text"* overlay text dataset paper, 2022 – discusses differences in overlay text vs scene text and compares CRAFT vs SplitNet for watermark detection [8] [35].
- NVIDIA Technical Blog, *Robust Scene Text Detection*, 2024 – notes on challenges of transparent text and importance of segmentation methods [2] [5].
- Ultralytics YOLOv8 paper – example of adapting general detectors for watermark text (YOLOv8-DCE) [31].

These sources and benchmarks collectively point to DBNet/DBNet++ as the ideal solution that is **accurate, fast, and well-supported** for the task of detecting text-based watermarks. [23] [21]

---

[1] [8] [35] [36] [45] assets.amazon.science
https://assets.amazon.science/12/17/2e4b442a4d068973add836ed599c/rooms-with-text-a-dataset-for-overlaying-text-detection.pdf

[2] Robust Scene Text Detection and Recognition: Introduction | NVIDIA Technical Blog
https://developer.nvidia.com/blog/robust-scene-text-detection-and-recognition-introduction/

[3] [4] Watermark Removal and Detection: Part I | by Ryan Murray | Medium
https://medium.com/@ryan_murray1/watermark-removal-and-detection-part-i-f17dbf93e886

[5] [25] [26] [27] [30] [38] Text Detection Models — MMOCR 1.0.1 documentation
https://mmocr.readthedocs.io/en/v1.0.1/textdet_models.html

[6] [7] GitHub - clovaai/CRAFT-pytorch: Official implementation of Character Region Awareness for Text Detection (CRAFT)
https://github.com/clovaai/CRAFT-pytorch

[9] [10] [12] [13] [15] [17] [46] GitHub - whai362/pan_pp.pytorch: Official implementations of PSENet, PAN and PAN++.
https://github.com/whai362/pan_pp.pytorch

[11] [19] [20] [21] [22] [37] [39] [1911.08947] Real-time Scene Text Detection with Differentiable Binarization
https://ar5iv.org/pdf/1911.08947.pdf

[14] [PDF] Efficient and Accurate Arbitrary-Shaped Text Detection With Pixel ...
https://openaccess.thecvf.com/content_ICCV_2019/papers/Wang_Efficient_and_Accurate_Arbitrary-Shaped_Text_Detection_With_Pixel_Aggregation_Network_ICCV_2019_paper.pdf

[16] [28] [PDF] Turning a CLIP Model Into a Scene Text Detector - CVF Open Access
https://openaccess.thecvf.com/content/CVPR2023/papers/Yu_Turning_a_CLIP_Model_Into_a_Scene_Text_Detector_CVPR_2023_paper.pdf

[18] [2105.00405] PAN++: Towards Efficient and Accurate End-to ... - arXiv
https://arxiv.org/abs/2105.00405

[23] Textron: Weakly Supervised Multilingual Text Detection Through Data Programming
https://openaccess.thecvf.com/content/WACV2024/papers/Kudale_Textron_Weakly_Supervised_Multilingual_Text_Detection_Through_Data_Programming_WACV_2024_paper.pdf

24 [DBNET]-Bad performance on long text detection · Issue #376 · open-mmlab/mmocr · GitHub
https://github.com/open-mmlab/mmocr/issues/376

29 42 GitHub - mindee/doctr: docTR (Document Text Recognition) - a seamless, high-performing & accessible library for OCR-related tasks powered by Deep Learning.
https://github.com/mindee/doctr

31 Text Image Watermark Detection Based on Improved YOLOv8 Model
https://dl.acm.org/doi/10.1145/3705391.3705396

32 33 34 GitHub - vinthony/deep-blind-watermark-removal: [AAAI 2021] Split then Refine: Stacked Attention-guided ResUNets for Blind Single Image Visible Watermark Removal
https://github.com/vinthony/deep-blind-watermark-removal

40 43 44 text-detection · GitHub Topics · GitHub
https://github.com/topics/text-detection

41 Unleash the Power of PaddleOCR: Your Guide to Best Open Source ...
https://generativeai.pub/unleash-the-power-of-paddleocr-your-guide-to-best-open-source-ocr-faabce26f5fa