Senior AI Developer Homework Assignment

Documentation

Name

Balázs Attila Fekete

2024. február 2.

1. Analysis of the problem

I've been fortunate to examine the image examples and have undertaken specific challenges to determine the most suitable network. These challenges encompass the following:

- Small gaps between the words
- Various dimensions, hues, and designs of the letters
- The orientation of words is not deterministic.
- The background is not static

Indications that a sophisticated architecture may not be necessary for the solution include:

- Not required to identify handwriting.
- The outline of the text is linear, not curved.
- We possess screenshots; there's no need to consider issues related to focusing or lighting.

As a machine learning engineer, my responsibility involves crafting an architecture capable of addressing the challenges mentioned above, albeit within constraints such as cost and time. I strive to create a compact network design to optimize efficiency.

2. Quick research-Related work

Fortunately, optical character recognition (OCR) is an extensively studied domain in the neural network landscape. However, many networks are tailored for more intricate challenges, often boasting over 20 million parameters. In state-of-the-art solutions, even the smallest networks dedicated solely to text detection encompass approximately 7 million parameters. Exploring the comparison of state-of-the-art solutions on open-source datasets can be insightful.[1]

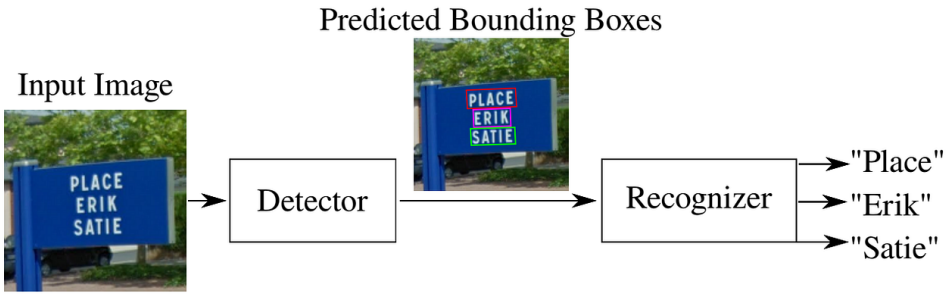
Using a large neural network for the task at hand seems like overkill. Hence, I've devised an architecture drawing from insights gleaned from experiments outlined across different articles, resulting in a swift solution. For quick solutions, leveraging the simple-to-use shell network, *easyocr*[2], is advisable. This is particularly handy for occasional checks. However, for frequent runs, opting for smaller, task-specific models would be more prudent.

I have crafted an architecture comprising two sequential networks. The initial network is dedicated to detecting and extracting words in the image, while the second is designed to recognize letters and words. While transformers have revolutionized the field of neural networks, certain studies highlight that their efficiency in image processing might not be as pronounced. Despite this, some architectures have surpassed convolutional solutions, albeit demanding substantial data.

[1] `https://paperswithcode.com/sota/scene-text-detection-on-total-text`
[2] `https://github.com/JaidedAI/EasyOCR`

3. Design

3.1. Model architecture



1. ábra. Concept overview

In Object Detection, YOLO stands out as a state-of-the-art solution, and I've chosen to apply it specifically for text detection. I've utilized the most miniature model, which comprises approximately 3.2 million parameters[3] and implemented a transfer learning approach on my end (further details will be provided later).

I opted to construct a Recognition system utilizing Residual blocks, drawing inspiration from ResNet, and incorporating a BiLSTM network. The network architecture is defined in the **TextDetectLSTM** class and is located in the Models/architectures/ocr_arc.py file.

3.1.1. Future development

It's advisable to incorporate a swift "Bounding Box Merging" technique to regulate the merging of adjacent bounding boxes. This will prove to be greatly beneficial in the obtained results.

3.2. Data generation

As previously stated, I created a dataset to train YOLO and own OCR to pursue our goal. During YOLO dataset generation, I endeavoured to replicate the environment by incorporating the following randomizations:

- Add text in different positions with different sizes, fonts and colours.)
- randomize the background colour and image size
- add icons to the background with random positions
- add random rectangles to the background.

While this dataset doesn't come labelled from the Graphisoft software, you can simulate it acceptably. If a user wishes to customize it according to their preferences, they only need to modify the config.json file and then execute the file, as demonstrated below:

[3] `https://docs.ultralytics.com/tasks/detect/`

```
python dataset_generator.py --config "path/to/config.json"
```

Examples of generated images are in the 8.1 section. Additionally, I have developed a YOLO visualization code to validate the generated dataset. For generating text recognition datasets, users can utilize the same Python file; they only need to reconfigure the config.json file.

3.2.1. Future development

I've utilized the *opencv* library to generate images and the *albumentation*[4] library for augmentation. However, as these libraries cannot produce rotated bounding boxes, it's worth noting that YOLO v8 can predict such boxes. Therefore, for future projects, developing a custom augmentation library would be advisable to address this limitation.

4. Train

The shell network solution operates without needing a training process as it comes with pre-trained weights, delivering excellent performance out of the box. Similarly, the YOLO model simplifies the training process by providing a predefined training function. Thus, adjusting the configuration file and executing the appropriate function were all needed.

However, building a custom model necessitates setting up a dedicated environment for execution. To streamline this, I opted for the *pytorch_lightning*[5] (pl) library, which facilitates extensive optimization and enables the creation of a model capable of seamless deployment across various environments such as the cloud or local setups.

I've developed the **PLBaseModel** class (Located at Models/base_model.py) to seamlessly integrate a PyTorch nn.Module into the PyTorch Lightning environment. For this adaptable model, selecting appropriate loss functions and, if necessary, metrics is crucial. I've chosen CTCloss and CERMetric to serve this purpose. To ensure the PLBaseModel class remains versatile for broader usage, these choices aren't hardcoded. Consequently, I had to override the default PyTorch functions, which are located at Data/utils_data.py. Both of these are considered state-of-the-art solutions to OCR problems.

I've additionally established a universal DataProvider to generate a DataLoader tailored for PyTorch Lightning models. This entailed defining Image and Label Preprocessor functions, which have been implemented in Data/utils_data.py. The ImagePreprocessor is responsible for resizing, normalizing, and converting images into tensors. Additionally, another function has been designed to manage label lengths and vocabulary. It transforms labels into number sequences based on the specified vocabulary. To accomplish this training, execute the following command:

```
python ocr.py --config path\to\config.json --max_epochs 100
```

5. Usage

The easyocr solution is coded in Models/test/easyocr_shell_network.py file, which can run with the following command:

[4] https://github.com/albumentations-team/albumentations
[5] https://github.com/Lightning-AI/pytorch-lightning

```
python easyocr_shell_network.py --images path/to/image/files
--save_path path/to/save --visualize True --save True
```

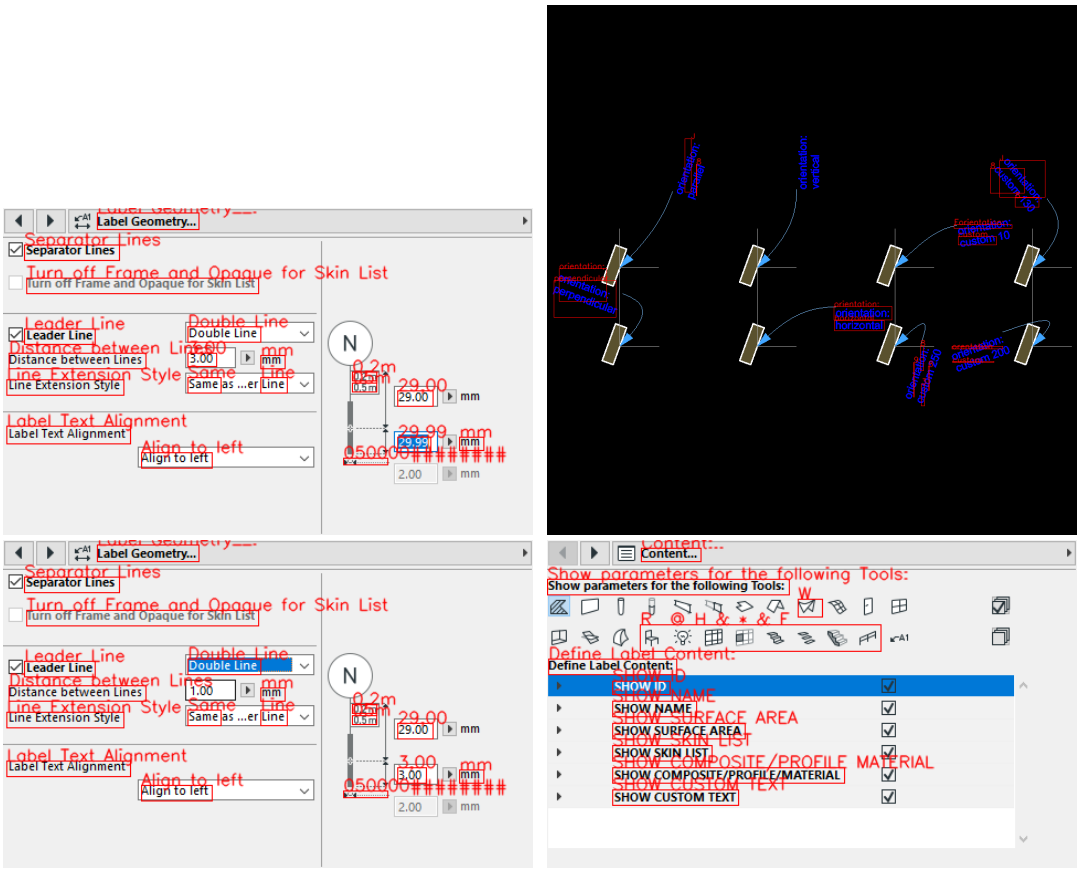
If the user chooses to save the results, a folder will be generated for each image. Each folder will contain a CSV file with the results and a "result.png" file visualizing the results on the original image.

Unfortunately my devices are not available to train the designed network from the ground, so I only can show the results of the yolo. The tester coded in Models/test/-test_yolo.py file, which can run with the following command:

```
python test_yolo.py --images path/to/image/files
--save_path path/to/save --visualize True --save True
```

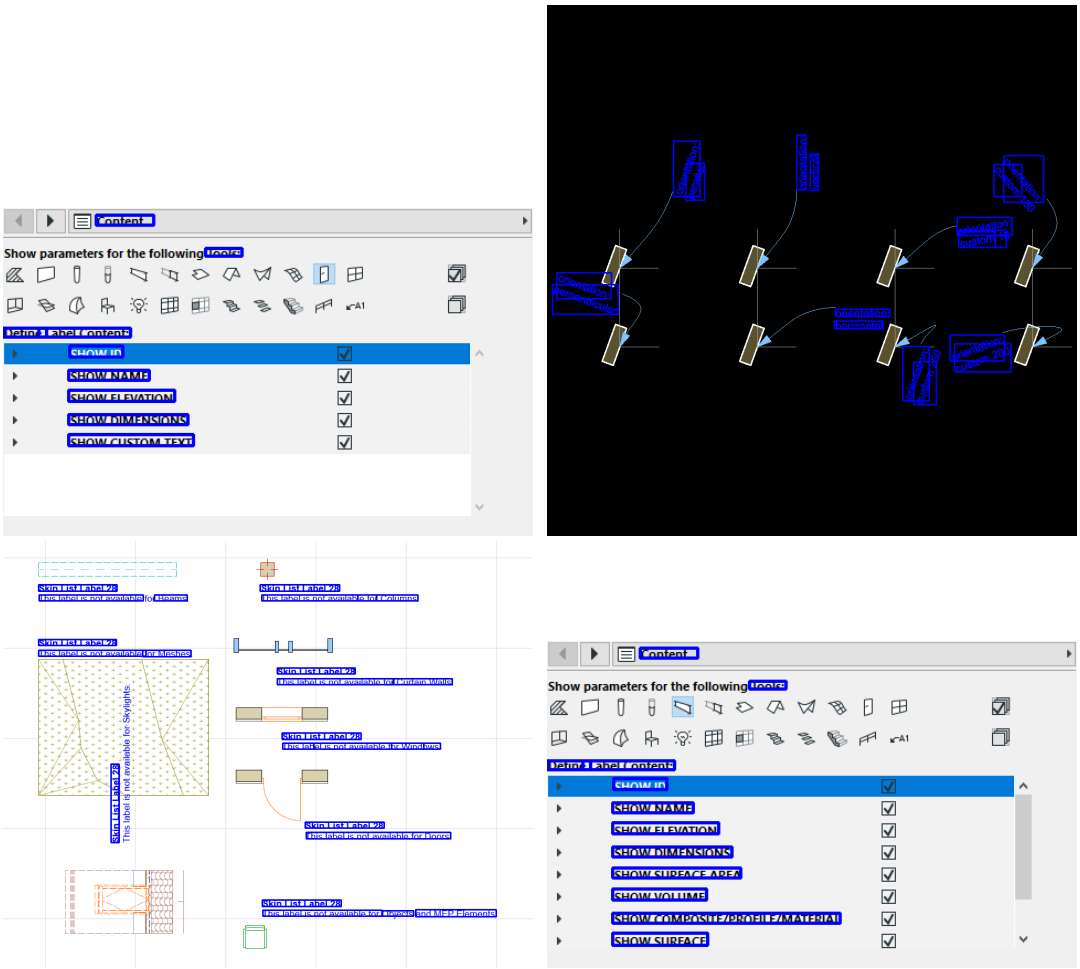
6. Results

While I won't assert full comprehensiveness, I aim to share the findings. The EasyOCR solution demonstrates effectiveness, yet it appears that implementing transfer learning in the future could be beneficial. Regrettably, EasyOCR currently struggles with handling rotated text.



2. ábra. Easyocr results

The YOLO results show considerable promise, and I anticipate that with the aforementioned future developments, we can attain nearly flawless results.



3. ábra. Yolo results

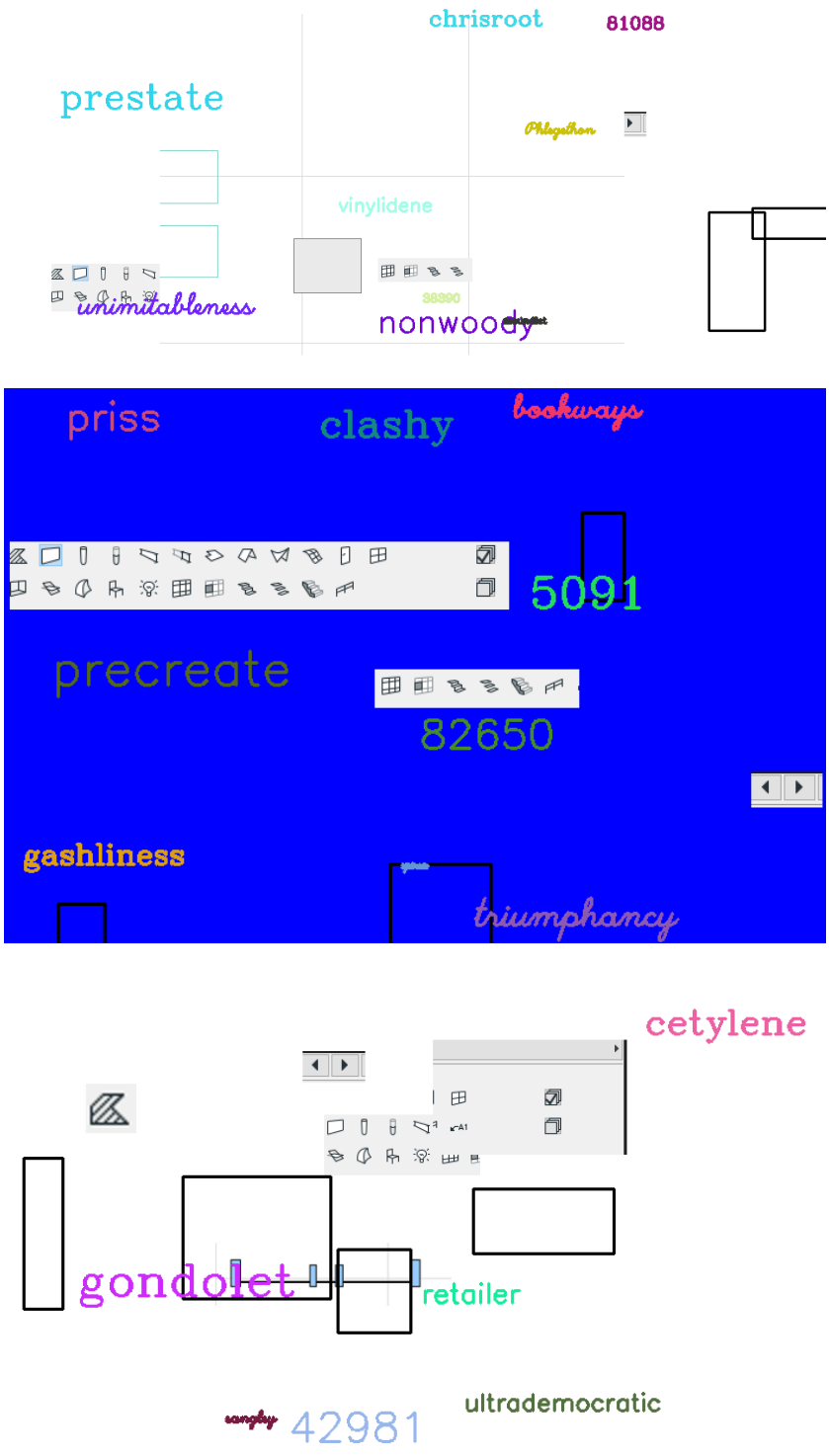
7. Conclusion

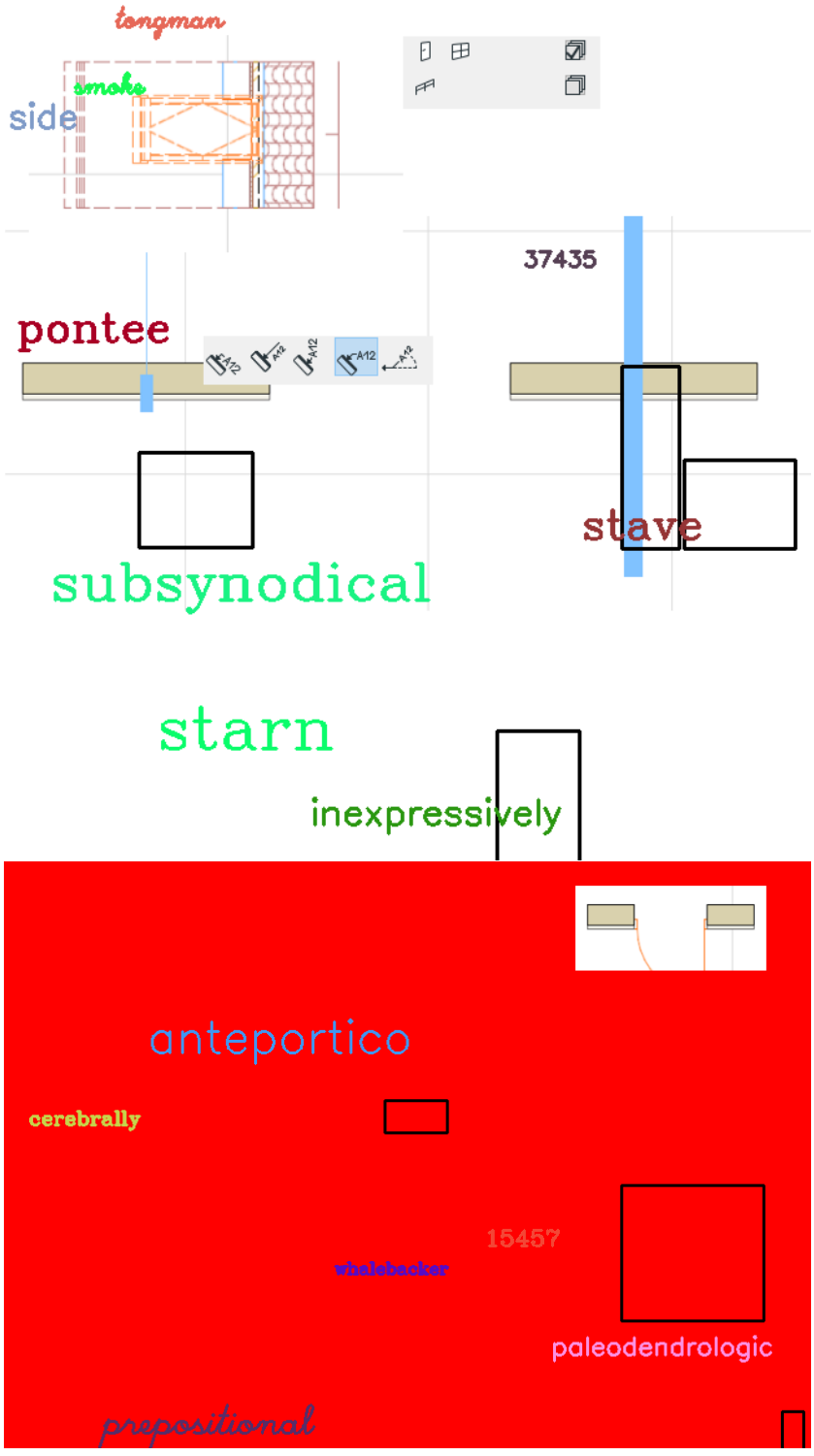
I've explored two solutions to address the issue. EasyOCR offers a straightforward approach and performs quite effectively. It can also be fine-tuned using custom data to enhance precision.

However, for frequent usage, setting up and training my model requires considerable effort. Regrettably, I lack the necessary environment for training a model from scratch without pre-trained weights. However, I defined an environment that will make this available in the future.

8. Appendix

8.1. YOLO dataset examples





8.2. Text recognition dataset example


indium
equison
thewness
sissiness
amphibiotic
cit