Extracting Plate Numbers and Addresses

Arthi Aneel — Chandhu Bhumireddy — Hal Brynteson
Nygel Sotomayor — Venkata Rama Surya Pidaparthi

December 4, 2023

1 Summary of Problem

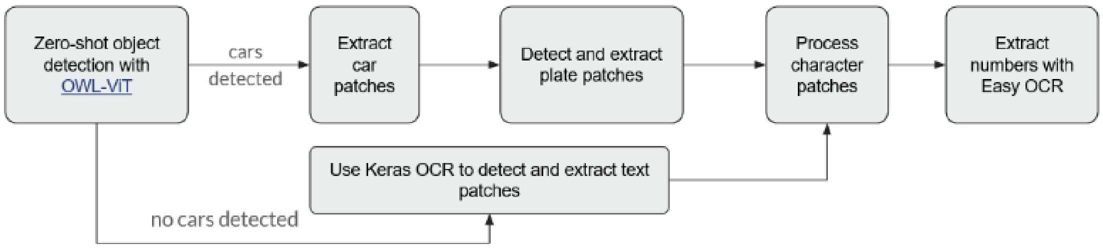
With vast amounts of photographic data collected in modern society, systems to perform object detection and optical character recognition (OCR) on these images are becoming increasingly useful. For our project, we seek to employ these two techniques to create a pipeline to extract text from an image and identify its source. Specifically, we will extract the identifying numbers of a house or a car, and identify which of those two objects the numbers are from.

Our task is inspired by Google Street View, where house numbers, or, addresses, are displayed as a part of Google Maps, and car numbers, or, license plates, are blurred. In this case, the dataset contains text and numbers on many different things, yet treats those numbers differently depending on what they are a part of. So, a combination of object detection and character recognition must be used to properly determine how text and numbers in an image should be handled.

To mimic this case, we created a dataset of 91 images containing either car numbers, house numbers, or both. These images were collected from either outside sources, or taken by our teammates. House numbers were pulled from the Google Street View House Numbers dataset, and cars from the PKU License Plate Recognition dataset.

(a) Figure 1: Examples of images in dataset gathered from Google Street View House Numbers (left), PKU License Plate Recognition (middle), and our own photography (right)

2 Approach

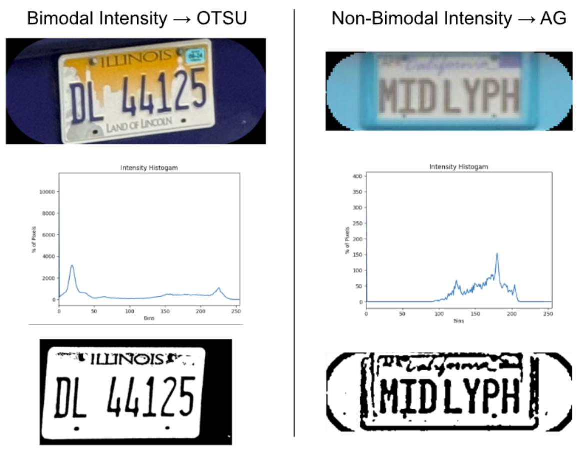

Zero-shot object detection with OWL-ViT
cars detected
Extract car patches
Detect and extract plate patches
Process character patches
Extract numbers with Easy OCR
Use Keras OCR to detect and extract text patches
no cars detected

(a) Figure 2: Diagram of complete pipeline

Generally, our approach relies on an object detection pass, a text-patch extraction pass, a preprocessing pass, and then a final OCR pass to extract the text. The initial object detection is done using OWL-ViT, and we detect all cars. To continue pushing images down the pipeline, we only continue to run detection on images where object detection was successful (which is measured by checking the object detection output against the image's label). Correctly detected cars are removed from the image, and then another OWL-ViT detection pass is performed to extract patches with license plates, which crops the image to only the necessary text.

Images where no car was detected are cropped to the text using Keras OCR. An initial Keras OCR recognition pass is performed, and only predictions that detected a number are saved. This is used to roughly estimate possible number locations. We then crop to the extents of the bounding box (or bounding boxes) to get the house numbers.

Once we have a list of all of our extracted text patches for both cars and houses, we run a few processing steps to make sure the text is as clear as possible. First, we upscale the image, and convert to grayscale. Next, we calculate and visualize a histogram of the grayscale image. For many images, especially license plates, there is enough of a difference between the character intensity and the background intensity, so an OTSU thresholding for binarization is sufficient. In cases where this fails, we switch to an Adaptive Gaussian (AG) thresholding method. In general, our pipeline assumes OTSU will be used. If text recognition fails for an image, we will isolate it, visually examine its histogram, and switch to the AG method. For AG thresholded images, we also perform a denoising step.



(a) Figure 3: Using histogram shape to determine threshold method

Once the image has been binarized, it is deskewed if needed. We take a naive approach to deskewing, where Hough lines are detected and, if found, the image is rotated to be horizontal. Finally, we convert the processed image back to RGB for OCR.

2.1 Machine Learning Approach used for Object Detection

We use OWL-ViT which is a zero shot text conditioned object detection model. This uses CLIP which stands for Contrastive Language–Image Pre-training as its multi model backbone. It has a ViT like transformer to get visual features and a casual language model to get text features. The architecture is shown here for pre training and open-vocabulary detection.

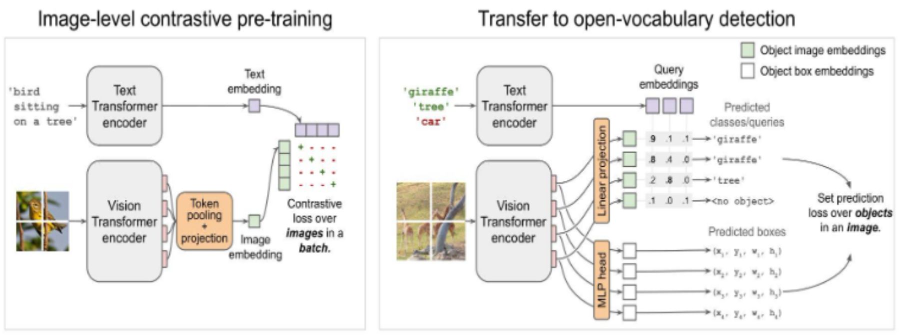


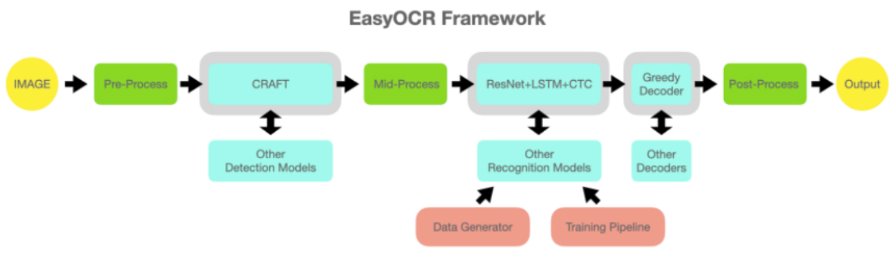
Image-level contrastive pre-training
Transfer to open-vocabulary detection
'bird sitting on a tree'
Text Transformer encoder
Text embedding
Vision Transformer encoder
Token pooling + projection
Image embedding
Contrastive loss over images in a batch.
'giraffe' 'tree' 'car'
Query embeddings
Object image embeddings
Object box embeddings
Predicted classes/queries
Linear projection
MLP head
Predicted boxes
Set prediction loss over objects in an image.
'giraffe'
'giraffe'
'tree'
<no object>

(a) Figure 4: OWL-ViT architecture

There are two encoders and inorder To use CLIP, the final token pooling layer is removed and instead a lightweight classification model and box head are attached to each token as shown. OwlViTImageProcessor can be used to resize (or rescale) and normalize images for the model and CLIPTokenizer is used to encode the text. OwlViTProcessor wraps OwlViTImageProcessor and CLIPTokenizer into a single instance to both encode the text and prepare the images.

2.2 OCR Approach used for Text Recognition

For our object recognition we used Tesseract and EasyOCR. EasyOCR is a lightweight OCR library designed to extract text from images. It utilizes deep learning techniques, specifically convolutional neural networks (CNNs), to recognize and interpret characters within images or scanned documents. It supports the use of either CPU based processing or GPU supported processing with support for only NVIDIA Cuda cores.



(a) Figure 8: EasyOCR Framework

EasyOCR initiates the text extraction process by taking an image in various formats, such as JPEG or PNG, as input. Prior to extracting text, the input image undergoes pre-processing, involving operations like resizing and noise reduction, to enhance quality and optimize OCR accuracy. Subsequently, EasyOCR employs deep learning models trained on extensive data-sets to perform text detection, identifying regions containing text within the image. Following text detection, another deep learning model is employed for text recognition, reading the content within each identified region and converting it into machine-readable text. The final output is the extracted text, provided as a string or list of strings, enabling us to further process, analyze, or store the information as needed.

3 Evaluation

For each image in our dataset, it was labeled for the objects in contains (car, house, or both), the subjective clarity (clear, unclear), and the text contained. Clear images have high resolution, high contrast, and/or little text distortion. Conversely, unclear images are low resolution, low contrast, and/or heavy text distortion.

(a) Figure 5: Images with labels

Object recognition techniques were evaluated based on the number of false positives and false negatives. A false positive would be a car detected in an image with a "house" label, and a false negative would be no car detected for an image labeled "car". We recorded the percentage of correct detections for cars, houses, and for each clarity group. Text recognition was evaluated based on the ability to recognize text in an image and, if text is detected, the accuracy. Accuracy was measured by the percentage of characters correct in the correct order.

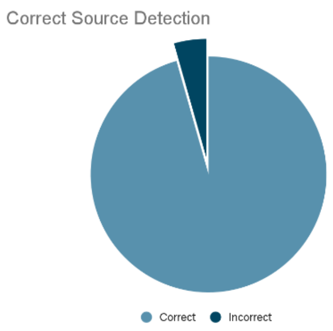
4 Results

The object detection pass of our pipeline had a 95% accuracy. 87 out of 91 images were detected correctly. Of the 4 incorrect, all were false negatives, where a car was not detected in a car labeled image. The majority of these false negatives were unclear images, with only one clear image failing the initial object detection process. In the patch extraction phase, 27 images failed to have an initial text patch found. 64 images made it to the preprocessing part of the pipeline. Of the 27 culled images, 63% were unclear houses, 25% were unclear cars, and less than 1% were clear houses.

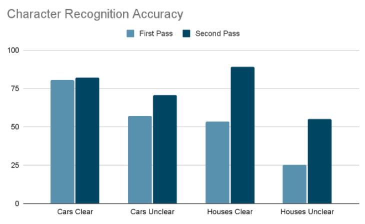
After a naive preprocessing approach where all images receive the same processing (OTSU thresholding, no deskew, uniform upscaling), 31 images were successful, where a "success" is defined as an accuracy of 70% or more. Of the successful images, the majority were of clear cars, and the total average accuracy was 54%.

A second preprocessing pass was performed where parameters for preprocessing were manually determined, such as setting the thresholding to AG, or enabling or disabling deskew. The second pass increased the OCR accuracy from 54% to 74%, with the (House, Clear) group seeing the greatest improvement of over 30%.

Overall, our method, with some user input and fine tuning, achieved 95% accurate object recognition and 74% accurate OCR. We finished with 59 images labeled with the correct source object, and a greater than 70% accurate text label.



(a) Figure 7: Results of Object Detection



(b) Figure 6: Changes in OCR accuracy between passes

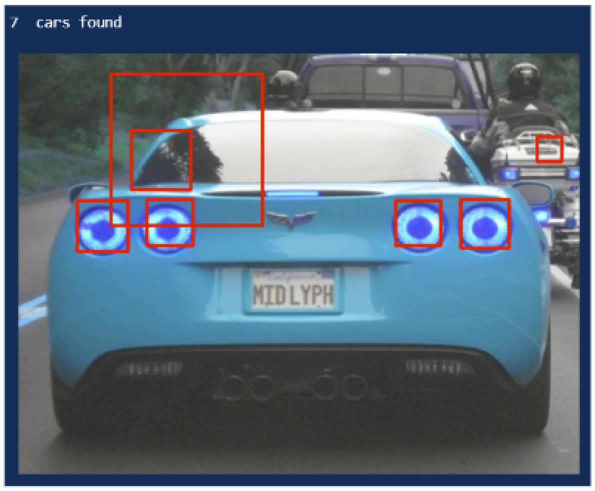
Overall, our method, with some user input and fine tuning, achieved 95% accurate object recognition and 74% accurate OCR. We finished with 59 images labeled with the correct source object, and a greater than 70% accurate text label.

5 Conclusion

We have made several observations by analyzing our results. First, the perceived clarity of the image did have an impact on accuracy, and this impact was considerably more pronounced for text recognition. 86% of unclear cars were correctly detected as cars, but many of them failed to have readable, extractable text. Also, house numbers benefitted the most from switching to Adaptive Gaussian thresholding over OTSU. This makes intuitive sense, as many license plates have intentionally contrasting text and background, while house numbers are not nearly as uniform in appearance. Similarly, house numbers rarely benefitted from the deskewing function, so it was disabled for most house number recognition preprocessing pipelines in the second pass.

From these observations, we can extrapolate many possible improvements to our pipeline. First, our dataset could be improved. Of the 5 images where no text could be read, many were entirely illegible to human eyes. These images could be very easily dropped from the dataset, as they don't really contain any numbers, even if they are sourced from the Google Street View dataset. In addition to this, our dataset is quite varied, which posed challenges in developing the approach. Initially, we attempted to train a Haar Cascade to handle object detection, but were unsuccessful, achieving 0 accurate detections. We believe this is due to the lack of uniformity in the appearance of the cars in our dataset. Even with our OWL-ViT approach, the varied nature of our dataset required fine tuning for the preprocessing phase, where user discretion was required when, at the very least, determining the thresholding method.

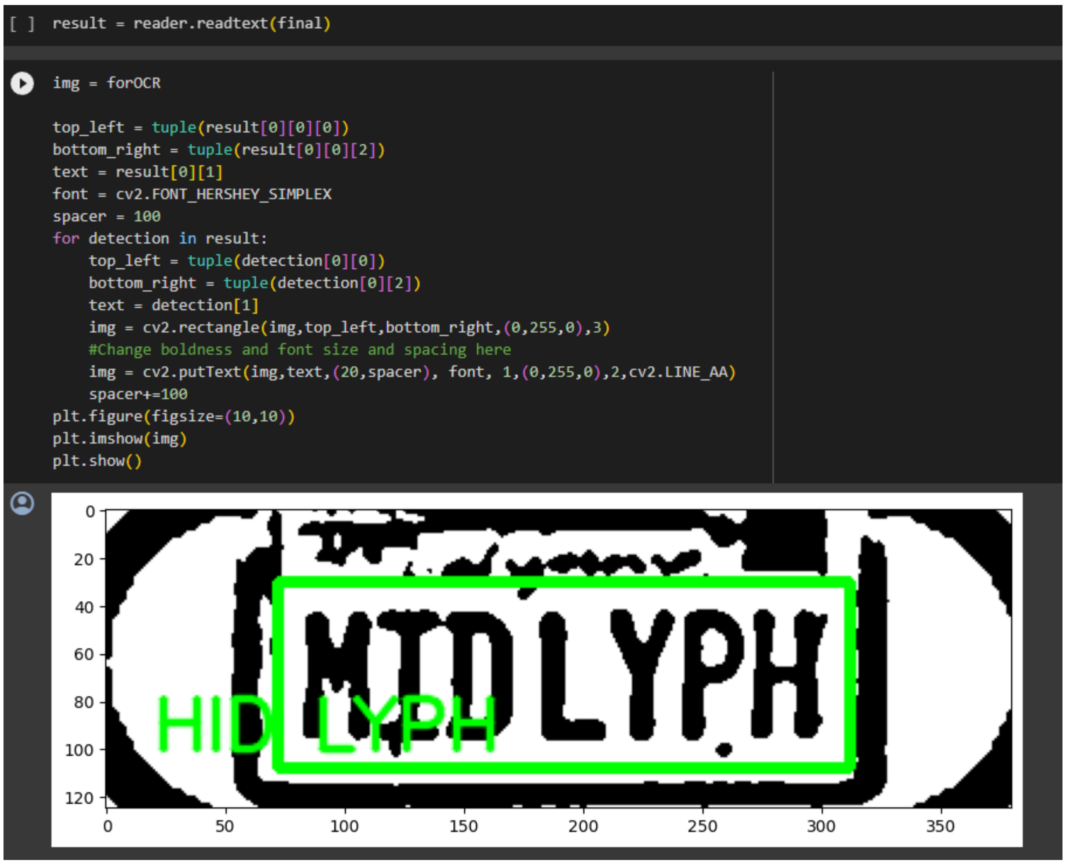
Finally, our approach relies on numerous assumptions, and routinely failed deviant cases. For example, our deskewing method worked on text on one line, such as license plates, but failed for cases such as vertically stacked numbers, common in house numbers. We assumed that most numbers would be on a horizontal line, so our approach is tailored to that case. Further, we assume that any numbers not on a car belong to a house. This assumption holds for our specific dataset, but prohibits our approach from being generalized.



(a) Figure 10: Unsuccessful object detection using Haar Cascade

6 Code Example

All of our code used is contained in a Juypter notebook CV_Team2_Final_PIpeline.ipynb that utilizes our dataset to perform our implemented object recognition and text recognition, example outputs are shown in the notebook aswell. An example of our code are as provided below.



```
[ ] result = reader.readtext(final)
```

```
img = forOCR

top_left = tuple(result[0][0][0])
bottom_right = tuple(result[0][0][2])
text = result[0][1]
font = cv2.FONT_HERSHEY_SIMPLEX
spacer = 100
for detection in result:
    top_left = tuple(detection[0][0])
    bottom_right = tuple(detection[0][2])
    text = detection[1]
    img = cv2.rectangle(img,top_left,bottom_right,(0,255,0),3)
    #Change boldness and font size and spacing here
    img = cv2.putText(img,text,(20,spacer), font, 1,(0,255,0),2,cv2.LINE_AA)
    spacer+=100
plt.figure(figsize=(10,10))
plt.imshow(img)
plt.show()
```

(a) Figure 9: EasyOCR implementation

References

Hugging Face Inc. (Year). Transformers. `https://github.com/huggingface/transformers`.

Inc., G. (2018). Tesseract ocr. `https://github.com/tesseract-ocr/tesseract`.

JaidedAI (2021). Easyocr. `https://github.com/JaidedAI/EasyOCR`.

Li, J. (2019). Street view house numbers. Retrieved 26 October, 2023 from https://www.kaggle.com/datasets/stanfordu/street-view-house-numbers/data.

Morales, F. (2019). Keras OCR. `https://github.com/faustomorales/keras-ocr`.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Yule, Y. (2018). Ofeeler/lpr: Pku lpr dataset. Retrieved 26 October, 2023 from https://github.com/ofeeler/LPR.