Real-Time Pokémon Card Detection from Tournament Footage

Edwin Pua
Department of Computer Science
Stanford University
puaedwin@stanford.edu

Abstract

Match footage for Pokémon tournaments is blurry and hard to follow for average viewers unfamiliar with the cards. This paper outlines techniques dataset augmentation to improve an object detection model's ability to detect classes of Pokémon cards in real time, as well as their limitations. This includes established techniques such as random rotations or copy-paste augmentation, as well as novel techniques such as mosaic backgrounds or polygon clipping.

1 Introduction

The most popular use of Pokémon Trading Cards is for collecting, but they have another use - battling. Pokémon Trading Cards are used to play a tabletop game that simulates Pokémon battles.

However, there are two primary barriers that prevent people from getting into the sport. One is the set of twenty minute rules required to learn how to play, which is a surprisingly high activation cost. The other is how difficult it is to be excited as a spectator of the Pokémon Card Game.

Figure 1: Example of tournament footage of the Pokémon Trading Card Game (TCG).

For instance, consider Figure 1. In order to capture the whole playing field, the camera needs to be a certain distance above the match. However, because the Pokémon and text on the cards are illegible from this distance, the only way to follow what is happening is if you know what all the cards do in advance. Most normal spectators will not understand what is happening, which might turn away potential Pokémon trainers.

To alleviate this, the goal is to make a computer vision model that can detect which cards are on screen. When a viewer hovers over a particular card, the plan is to show a zoomed-in, high quality version of the card so an online spectator can easily read what a card does.

To be more specific, given a URL or MP4 recording of Pokémon TCG tournament footage, is it possible to generate real-time, frame-by-frame segmentation masks of each Pokémon card as output?

That way, when a user uses the mouse to hover over a card, and the cursor coordinates coincide with one of the coordinates of a segmentation mask, we can use the detected card class of that segmentation mask to retrieve the correct zoomed in image of the corresponding card.

2 Related Works

In the past, there have been similar works on card detection. For instance, Snyder developed a poker card detector without using a neural network, via corner detection and template matching. [1] However, these heuristics are not robust, since they assume close-up, clean images and no perspective distortion of the cards. Another poker card detector was made by Chen et al. using a "sandglass block", or encoder-bottleneck-decoder network architecture, which works even when each card takes up only 0.7% of the screen. However, these poker cards are not occluded by any objects like cards in the Pokémon card game are (e.g. dice, tokens, etc.), and there are only 52 classes of poker cards, compared to the thousands of classes of Pokémon cards. [2]

Generalizing to overall object detection, perhaps one of the most seminal papers is the YOLO paper by Redmon et al. [3] This model specializes in single-shot, real-time object detection, which is perfect for my use case. Given that I can easily finetune a YOLO model on my specific task, I feel confident in choosing YOLO as my weapon of choice.

Given that there exists only one of each Pokémon card, data augmentation will be very important for generating datasets. For this reason, I plan to implement many established augmentation techniques, such as random, noise, flip, rotation, etc. posited by Yang et al. [4] Another technique is copy-paste augmentation and scale jittering, as presented by Ghiasi et al. [5] I could also implement the techniques proposed by Roh and Chung, [6] which uses a diffusion model to counteract video degradation such as motion blur, camera defocus, and partial occlusion.

Some other related works that I will discuss in further detail later include a paper by Li et al. [7] which emphasized the importance of distribution focal loss, as well as various polygon clipping algorithms, such as Sutherman-Hodgman algorithm, [8] Weiler-Atherton algorithm, [9] and the Vatti clipping algorithm, [10] all of which can aid me in creating segmentation masks.

3 Dataset

First, I web scraped 460x640 resolution images of every tournament-legal Pokémon card from LimitlessTCG. Unlike most vision problems, Pokémon card detection has very little issues with deformation or intraclass variation - in other words, a given Pokémon card will always look the same.

Figure 2: Examples of web scraped cards, Bidoof (left) and Bibarel (right).

However, during tournament footage, there are a few reasons why two identical cards would have different pixel values. One is the lighting that a card is subjected to. Another is any potential

occlusions that are covering the card, like dice or tokens. Yet another is the presence of any holographic or foil effects on the cards.

Because the image and shape of a particular card class is relatively static, and the only changes to that image are the aforementioned lighting, occlusions, etc., this means that it is possible to artificially generate a dataset. My techniques for dataset generation are expanded upon in Section 4: Methods.

Although artificially generated data would serve as my training set, I created a validation set directly from actual tournament footage - this way, validation metrics can accurately showcase how well the model would perform in the real world. To do so, I took 37 screenshots from a tournament live stream, hand-annotated their segmentation masks using a labeling software called CVAT, and converted those segmentation masks into a YOLOv8-readable format using a labelling software called Roboflow to create the labels.

Figure 3: Rather than training the model on all several thousand tournament legal Pokémon cards, I decided to start with a small sample of ten cards. These cards are, from top to bottom, left to right: Radiant Greninja (ASR_046_R_EN_LG), Bibarel (BRS_121_R_EN_LG), Bidoof (CRZ_111_R_EN_LG), Comfey (LOR_079_R_EN_LG), Giratina (LOR_130_R_EN_LG), Frigibax (PAL_057_R_EN_LG), Baxcalibur (PAL_060_R_EN_LG), Chien-Pao (PAL_261_R_EN_LG), Iron Bundle (PAR_056_R_EN_LG), and Shuppet (SVI_087_R_EN_LG). Since there are many different Pokémon cards with the same name (e.g. there are multiple cards with the Bidoof Pokémon), I assigned a unique ID to each card as class names when training the model - these are in parentheses.

4 Methods

A large pretrained vision model can easily be fine tuned to detect Pokémon cards - this means that card detection itself is not the issue. The primary challenge with card detection is twofold: firstly, new Pokémon cards come out about every three months, meaning the model will have to be constantly updated with new classes. Secondly, annotating every current and future piece of tournament footage with segmentation masks is not feasible - many of these tournament streams last eight to ten hours. Additionally, cards that are less powerful may not show up in tournaments very often, so if only tournament footage is used as training data, this could lead to class imbalance and therefore high distribution focal loss. [7]

Thus, the principal goal is to experiment with artificial dataset augmentation - specifically, given a set of cards, is it possible to artificially generate a dataset using these cards, such that when the model is trained on this dataset, the model can extrapolate what it learns to real tournament footage? Not only would artificial dataset generation make it easy to create new training data as cards release, but this kind of dataset could also be automatically labeled with segmentation masks, since we could

memorize the locations that the cards were pasted and create masks that way, rather than annotate each image by hand. To illustrate, the following methods section explores the different datasets I created and why.

4.1 Architecture

To start, I chose YOLOv8's pretrained segmentation model to generate my segmentation masks, which was pretrained on the 2017 COCO dataset. I chose YOLO by Redmon et al. [3] because their pretrained model would be easy to use, immediately adaptable to my task, more effective than starting with a random initialization of weights, and already optimized for real-time segmentation.

I trained this model on each of my custom datasets using YOLOv8's default hyperparameters, which uses AdamW as the optimizer, learning rate α of 0.01, weight decay of 0.0005, momentum β_1 of 0.937, and dropout of 0. Although this baseline model is trained on images, YOLOv8 easily allows such models to be used on MP4s and video recordings.

Since this project is primarily about experimenting with data generation heuristics rather than model architectures, I chose to test the subsequent artificial datasets on only this model. In future work, I plan to perform grid or random hyperparameter search to determine the most optimal learning rate, weight decay, etc.

4.2 Dataset Generation

To start, I created a preliminary dataset of cards pasted directly on a white canvas to serve as a baseline.

Figure 4: Cards on a 1920x1080 blank canvas.

The next dataset I created uses the simple augmentation heuristics presented by Yang et al. on each card [4] These include random noise, random black occlusions, random blur using a Gaussian kernel, random alteration of hue, random alteration of saturation, random alteration of brightness, random chance to flip the image, and random rotation. I also incorporated copy-paste augmentation and scale jittering as presented by Ghiasi et al., where each image is resized and pasted onto a gray background.

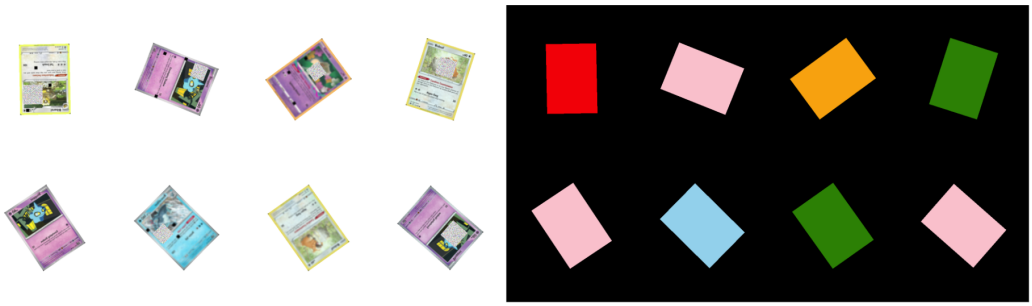

Figure 5: Preprocessed cards on a 1920x1080 blank canvas (left) and their corresponding, automatically generated segmentation masks (right).

After I trained on the previous dataset, I noticed that the model was struggling to segment the shapes of the cards. For example, if an image contained a card that was partially occluded by a hand, the segmentation mask would correctly cover the card, but also bleed into the hand, unable to identify the card's boundaries. Additionally, the model would also incorrectly detect card sleeves or the backs of cards to be a specific class of Pokémon, decreasing precision.

This was the motivation behind my next dataset. Instead of a white canvas as the background for the cards, I created collages of fake Pokémon cards to use as backgrounds. I used fake Pokémon cards because I wanted to teach the model that not every rectangular or card-shaped object is a detected Pokémon card. Fake Pokémon cards were produced by taking real Pokémon cards and replacing the art with some other Pokémon related artwork. The fake Pokémon cards were also preprocessed with the previous dataset's augmentation techniques.

These mosaic backgrounds would hopefully better teach the model what a card's edges look like, since it is now more difficult to determine where a card starts and stops, which gives the model more difficult training examples to learn from.

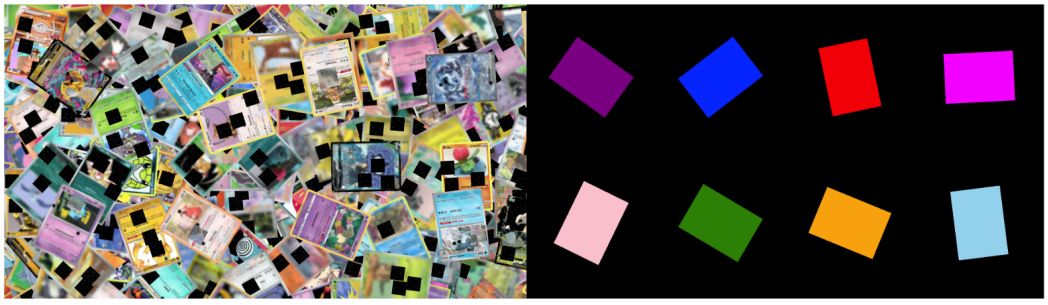

Figure 6: Preprocessed cards on a 1920x1080 mosaic canvas (left) and their corresponding, automatically generated segmentation masks (right). These mosaics was created by pasting randomly chosen, preprocessed, fake Pokémon cards at various locations on a white canvas.

After training on the collage dataset, I noticed that holographic cards were not being detected by the model. Holographic cards are rare Pokémon cards that emit shiny, prismatic patterns depending on the viewer's viewpoint angle. To account for this, I simulated the holographic effect as best as I could by increasing the card's contrast and increasing the brightness of slanted subsections of the card, and then added many simulated holographic cards to my dataset.

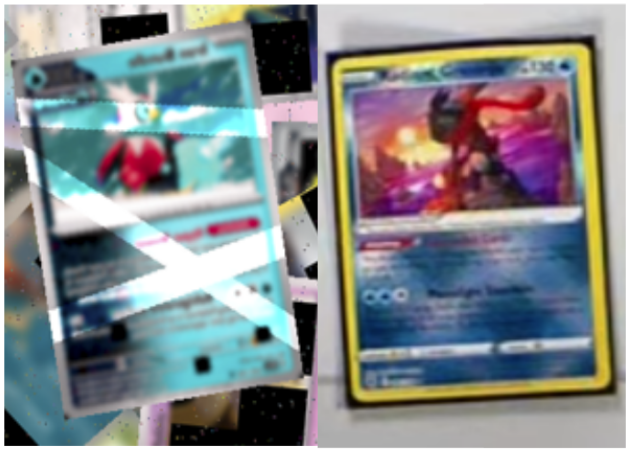

Figure 7: Simulated holographic card (left) versus real holographic card (right)

Finally, I noticed that some cards that were clearly visible, but partially occluded by a hand or another card, were sometimes not being detected by the model at all. To alleviate this, I constructed one final dataset that consisted of real cards partially occluded by fake cards. The challenge here is to obtain the segmentation mask of only the region of the real card that is visible. To do this, I can employ one of many polygon clipping algorithms, such as the Sutherman-Hodgman algorithm, [8] Weiler-Atherton algorithm, [9] or the Vatti clipping algorithm, [10] all of which can take in the coordinates of two overlapping segmentation masks as input, one top and one bottom, and output the coordinates of the visible portion of the bottom segmentation mask. I decided to use the Sutherman-Hodgman algorithm due to ease of implementation.

Figure 8: Real Pokémon card partially occluded by a fake Pokémon card (left) and its corresponding, automatically generated segmentation mask (right). The Sutherman-Hodgman algorithm takes in the coordinates of the bottom segmentation mask [(x_1, y_1), (x_2, y_2), (x_3, y_3),(x_4, y_4)] and the top segmentation mask [(x_5, y_5), (x_6, y_6), (x_7, y_7),(x_8, y_8)], and outputs the coordinates of the segmentation mask for the visible portion of the bottom segmentation mask. In this case, those coordinates are of the format [(x_1, y_1), (x_9, y_9), (x_5, y_5), (x_{10}, y_{10}), (x_2, y_2)], a pentagon.

5 Experiments, Results, and Discussion

For each generated dataset, I created 1000 training images and ran the aforementioned YOLOv8 network architecture for 10 epochs and observed which augmentations performed the best. To compare the effectiveness of each model, the primary metric I used was mean average precision (with IoU thresholds ranging between 0.5 and 0.95). I also looked at the precision and recall for both the bounding box and the segmentation mask.

	Box Precision	Box Recall	Box mAP50-95	Mask Precision	Mask Recall	Mask mAP50-95
No Preprocessing	0.00598	0.279	0.0437	0.00602	0.283	0.0423
Preprocessing	0.316	0.112	0.146	0.316	0.112	0.151
Preprocessing + Mosaic Background	0.709	**0.357**	0.379	0.704	0.351	0.371
Preprocessing + Mosaic Background + Holographic	0.604	0.253	0.276	0.604	0.253	0.256
Preprocessing + Mosaic Background + Mask Clipping	**.809**	0.352	**0.432**	**0.811**	**0.352**	**0.441**

5.1 No Preprocessing

We see that the initial mAP for segmentation masks with no preprocessing is abysmal at 0.0423. This is corroborated by the predicted segmentation masks. We can see in Figure 9 that many cards go undetected, and those that are detected are merged. Notice how the model mimics the bounding boxes of the training set - perfectly upright.

Figure 9: Detection using No Preprocessing model. The model classifies three cards as one. There are several false negatives.

5.2 Preprocessing

Preprocessing leads to a considerable increase in mAP from 0.0423 to 0.151. This is because preprocessing enabled the model to now detect rotated cards, cards that are blurry, or cards with considerable noise. However, we can see that the model still struggles with segmentation, failing to understand the boundaries between cards. It also falsely detects the backs of cards to be classes. This is the motivation behind including mosaic backgrounds - it trains the model on images that are difficult to segment, and also introduces fake cards that reduces the models false positive rate.

Figure 10: Detection using Preprocessing model. Notice how a card sleeve is erroneously detected a Pokémon, and two cards are detected as one.

5.3 Preprocessing + Mosaic Background

Figure 11: Detection using Preprocessing + Mosaic model. Features several accurate detections and classifications.

The mosaic background causes the mAP to jump from 0.151 to 0.371. Notice how the backs of the cards are no longer being picked up by the model. However, we can see that holographic cards are sometimes not being picked up, such as the Radiant Greninja on the right side. This is the motivation behind an additional preprocessing step that simulates the holographic effect on cards.

5.4 Preprocessing + Mosaic Background + Holographic

This holographic preprocessing leads to a decrease in mAP to 0.371 to 0.256. The reason seems clear - as seen by Figure 7 my simulation of holographic isn't quite realistic enough, which makes my training set less representative of the real world. To remedy this, I would need to create a more accurate holographic filter, or model the cards in a 3D software and use the software's lighting engine.

5.5 Preprocessing + Mosaic Background + Mask Clipping

Figure 12: Detection using Preprocessing + Mosaic + Mask Clipping model.

Mask clipping also causes the mAP to jump from 0.371 to 0.441. We can see how in Figure 12, the segmentation mask of Radiant Greninja on the left side is very accurate, and not running into the hand. However, we can see that the Holographic Giratina is not being detected on the right side, as well as the two more occluded Frigibax on the left side, indicating that all of these data augmentation techniques provide a partial, but not full, representation of real world Pokémon cards.

6 Conclusion

Although these techniques of preprocessing, mosaic backgrounds, and polygon clipping can be shown to increase YOLOv8's performance at detecting Pokémon cards, they still are relatively lackluster - mAP is still below 0.5, and many false positives and false negatives can be found in the outputs of all the models, regardless of training set. Thus, more powerful data augmentation techniques are necessary before a consumer-ready Pokémon card detector can be made.

For future work, the next step would be to upgrade to three dimensions, increasing complexity by modeling the cards, mosaics, and scenarios in Blender and create automatic segmentation masks by configuring different scenes, angles, and lighting. These would produce training sets that are more representative of real world Pokémon cards.

References

[1] Snyder, Daniel. 2019. Playing Card Detection and Identification. web.stanford.edu/class/ee368/Project_Winter_1819/Reports/snyder.pdf

[2] Q. Chen, E. Rigall, X. Wang, H. Fan and J. Dong, "Poker Watcher: Playing Card Detection Based on EfficientDet and Sandglass Block," 2020 11th International Conference on Awareness Science and Technology (iCAST), Qingdao, China, 2020, pp. 1-6, doi: 10.1109/iCAST51195.2020.9319468.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 779–788, 2016.

[4] S. Yang, W. Xiao, M. Zhang, S. Guo, J. Zhao, and F. Shen. Image data augmentation for deep learning: A survey. 2023.

[5] Ghiasi, G., Cui, Y., Srinivas, A., Qian, R., Lin, T.-Y., Cubuk, E. D., Le, Q. V., and Zoph, B. Simple copy-paste is a strong data augmentation method for instance segmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2918– 2928, 2021a.

[6] S.-D. Roh and K. Chung. Diffusionvid: Denoising object boxes with spatio–temporal conditioning for video object detection. IEEE Access, PP:1–1, 01 2023.

[7] Li, X., Wang, W., Wu, L., Chen, S., Hu, X., Li, J., Tang, J., & Yang, J. (2020). Generalized Focal Loss: Learning Qualified and Distributed Bounding Boxes for Dense Object Detection. arXiv preprint arXiv:2006.04388.

[8] Ivan Sutherland, Gary W. Hodgman: Reentrant Polygon Clipping. Communications of the ACM, vol. 17, pp. 32–42, 1974

[9] Weiler, Kevin and Atherton, Peter. "Hidden Surface Removal using Polygon Area Sorting", Computer Graphics, 11(2):214-222, 1977.

[10] Bala R. Vatti. "A generic solution to polygon clipping", Communications of the ACM, Vol 35, Issue 7 (July 1992) pp. 56–63.