Automatic Trading Card Identification and Grading

Lucas Lau, Andrew Ying, Kedar Krishnan, Shengzhe Wang

Background and Objective

Trading cards are collectibles that, in recent years, have seen a massive growth in popularity, both as a hobby and as investments for enthusiasts around the world. These cards not only hold value in entertainment and nostalgia, but also through very real and tangible monetary worth. Numerous cards have sold well into the millions of U.S dollars, and there are indeed countless cards selling daily for thousands.[1] These cards are a representation of a diverse popular culture, and provide a tangible representation of a wide variety of notable topics. Examples include cards of all types of sports, and popular media such as Pokémon, Yu-Gi-Oh, and Magic the Gathering. Thus, trading cards come in vast numbers, across many different categories and different sets. Learning and memorizing these defining characteristics may be time-consuming and a challenge for collectors and graders trying to determine a card's worth.

The exchange and transaction of trading cards occurs in several ways, including collecting, trading, purchasing, and selling—most commonly on secondary markets, as cards have only limited runs in retail stores.[2] Therefore, there is a need to build trust between buyer and seller through the characterization of a card's quality and veracity. This process is often delegated to third-party professional card grading companies, most notably PSA, Beckett, and CGC, and the results can drastically affect a card's market value.[2] Issues however stem from card grading companies relying on human graders, with the grading process suffering from inherent subjectivity stemming from differences in experience, expertise and personal opinion.[2]

Prior work in this domain includes Hernandez (2020), where the MTG Card Grader project utilized a pre-trained ResNet50 model for classification and an inception model from the Object Detection API for defect detection, achieving up to 94% accuracy on the training set.[3] The MTG project focused on grading cards based on their condition and provides valuable insights regarding the use of deep learning models for feature extraction and the application of transfer learning techniques, which can be adapted to our project. Limitations of this study

included poor accuracy for the Object Detection model, possibly resulting from limited training data and insufficient hardware to run numerous epochs.

In the work by Nahar (2023), researchers developed an automated corner grading system, DeepCornerNet, a deep learning model trained from a labeled data set of 593 sports card images and 4,744 corners.[2] For corner detection, rather than using the Harris algorithm, they used the Hough Transform algorithm to contour the image and identify the border line segments, the intersections of which are the corners. For the deep learning model, they used the CNN VGG-Net, obtaining 78% accuracy with VGG-16.

In the realm of feature extraction and classification for trading cards, template matching has emerged as a promising approach. Nazanin Sadat Hashemi (2016) has explored various algorithms to enhance this technique.[4] For instance, the Best-Buddies Similarity method, which utilizes mutual nearest neighbors for robust template matching, has shown potential in unconstrained environments. Additionally, fast template matching algorithms based on principal orientation histograms have been developed to expedite the matching process, for quick object recognition without high algorithm complexity. Their review showed robust results across a variety of applications from face detection to automatic bare PCB board testing to mammogram positioning. Template matching can be effectively applied to the identification of specific symbols on trading cards, aiding in their classification.

In text extraction, Tao Wang and David J. Wu (2012) leveraged large multi-layer convolutional neural networks (CNNs) to train robust text detection and recognition modules.[5] By combining these modules with simple post-processing techniques like non-maximal suppression and beam search, they construct a complete end-to-end text recognition system. Thus one can pre-train a model on the text extraction dataset to extract the text information on the trading card.

This project aims to establish an automated system which leverages deep learning and traditional computer vision algorithms to extract features from trading cards for the purpose of organization and objective grading. We will focus on Pokémon cards in this implementation, and aim to use these extracted features to identify specific cards. For identification, the system will need to detect a variety of characteristics, notably the set symbols, HP (hit points), abilities, and other certain features. For condition assessment, we aim to extract defects such as edge chipping, wear, creases, and dents. We aim for this project to be robust to different environments, with

different angles, lighting, and orientations. To implement this project, 5 models have been created spanning card extraction, text extraction, feature extraction and template matching, and defect object detection.

Methodology

Methodology 1: Card Extraction

The first step of the model pipeline was to create a system to automatically extract the borders of a trading card within a given image. This system needed to be robust and be able to extract the card regardless of factors such as rotation, position, etc. Traditional computer vision techniques were used to complete this task. First, the Canny Edge Detector from OpenCV was applied to preprocess the image and extract its edges. Different cards across different images required different parameters, so the thresholds of the edge detector had to be dynamically adjusted until the edges were detected.

Once the edges were extracted, the edge extracted image was examined and the contour with the largest area within the frame was found. This contour was approximated to a quadrilateral shape to find the card borders. The next step was to take a perspective transform. The approximated polygon was taken and reshaped into a (4,2) array, finding the four corner points of the quadrilateral. A perspective transform using OpenCV was taken, mapping the contour to a rectangular shape. To account for rotation, the orientation of the card was determined based on the edge lengths, and adjust the points as necessary. The performance of this model was not directly tested, as its purpose was to ensure the performance of the following text extraction model for cards surrounded by other objects.

Figure 1: Extracting Card from Surroundings using Contours

Left depicts the image following Canny Edge Detection. Middle shows the contour with the largest area. Right shows the extracted and perspective transformed card image.

Methodology 2: Text Extraction

The next step in the card identification pipeline was text extraction. Different text features on the card were utilized to distinguish the cards. Most notably, the name, HP (Hit Points), and moveset of the cards were extracted and used to identify cards. To apply text extraction to the card to extract these regions, the card images were normalized and bounding boxes were applied to the card around these regions, as they are typically consistent across different cards. These extracted images were binarized, and the following text extraction models were applied. Two different text extraction models, PyTesseract and EasyOCR were utilized and their performance was compared.

Figure 2: Extracting Regions Containing Text

Counter-clockwise from the left: image of card with bounding boxes around name, HP, and moveset; binarized name region; binarized HP region; binarized moveset region.

Text Extraction was tested on 2,000 images of Pokemon Cards taken from a dataset with 15.6k images of Pokémon cards in mint condition.[6] The extracted regions were searched for in a card attributes dataset containing the attributes of most existing Pokemon Cards.[7] Fuzzy Matching was implemented to find matches, as exact string matching would not be robust to minor mistakes made by the text extraction models. The number of correct matches was calculated.

Methodology 3: Feature Extraction - Template Matching and VGG-16

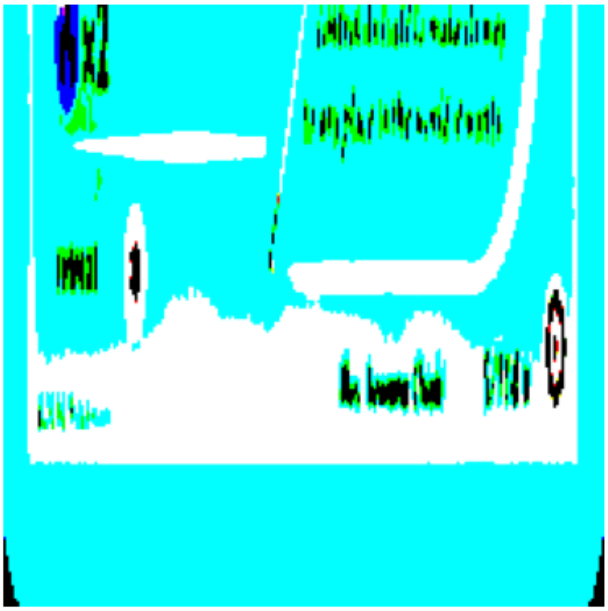
The step following text extraction in the pipeline was feature extraction. Pokémon cards have multiple distinguishing features on cards that can be used for identification purposes. For example, rarity symbols, elemental symbols, and set symbols can all be distinguishing features. Set classification was examined to start. The set symbol is usually in either the bottom left or bottom right of the card. Two methodologies were tested: Template Matching and the VGG-16 CNN model. The same dataset as above was used and contained 15.6k images of Pokémon cards in mint condition, with a corresponding table of attribute information for the card.[6,7] The set symbol information was of specific interest for testing accuracy of the model. Additionally, the dataset was split into training (60%), validation (20%), and testing (20%) sets.

To apply template matching, the image was converted to grayscale and cropped to the set symbol of the image. The template matching model assumed that the cards are already oriented correctly from the previous card extraction step. To apply the VGG-16 model, the image symbol was again cropped to the set symbol, and the assumption the card is oriented correctly was made as well. To account for different cards having different set symbol positions, template matching was applied to both sides of the card, while VGG-16 was simply applied to the entire bottom section of the card and trained over 50 epochs. RMSprop Optimizer was used for this approach along with a learning rate scheduler.

In addition, VGG-16 was also applied to the entire card as another approach. The images were rescaled to 224 x 224 with 3 color channels and converted from RGB to BGR with each color channel being zero-centered with respect to the ImageNet dataset. The base layers of the model were frozen, and layers for normalization, dropout, and fully connected layers were added on top. L2 regularization is applied to penalize large weights and prevent overfitting. Data augmentation (flips, rotations, and color variation) was applied and the RMSprop Optimizer was one again used for this approach.

Figure 3: VGG-16 Input Image after Preprocessing

Image of the bottom of the card following cropping, rescaling, RGB to BGR conversion, and zero-centering.



Methodology 4: Feature Extraction - ResNet50

A model utilizing the ResNet50 model was also trained for feature extraction, to be compared with the VGG-16 and Template Matching models' performance. A pretrained ResNet50 model was utilized, and this approach was begun by preprocessing. The images were rescaled to a standard 342 x 245 with 3 color channels. Random horizontal and vertical flips were utilized for data augmentation, along with color jitter to diversify the dataset variability. The Adam optimizer was used for model training. The same dataset as above was used contained 15.6k images of Pokémon cards in mint condition, with a corresponding table of attribute information for the card.[6,7] In this case, classification of four different attributes was performed with a different model trained for each task: set, weakness, type, and rarity.

No cropping was applied, as ResNet50 was applied to the entire image. Adoption of multi-task prediction was implemented to extract multiple features from the Pokemon Cards. Finally, the dataset was split into training (70%), validation (15%), and testing (15%) sets. The model with the highest validation accuracy over 9 epochs was selected to mitigate overfitting.

Figure 4: Class Activation Mapping for Classification Tasks

Left to right, the activation map for: weakness, type, and set classification.

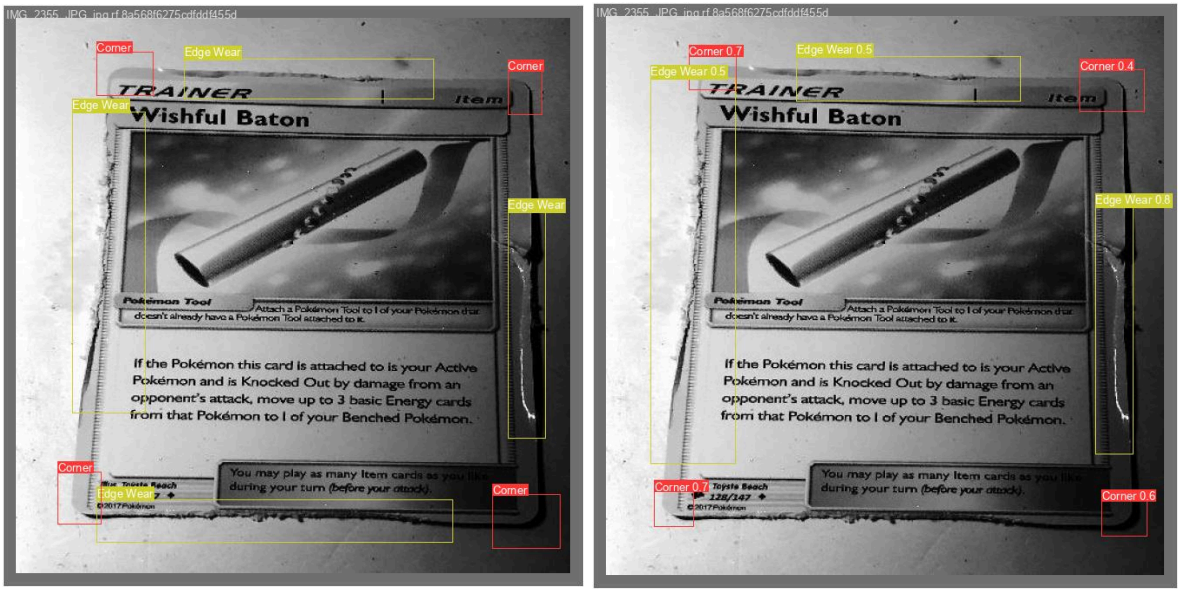
Methodology 5: Object Detection

The final model was an object detection model utilizing YOLOv5 to draw bounding boxes around defects. A dataset of 2,633 grayscale images of Pokémon Cards was split, with 70% for training, 20% for validation, and 10% for testing.[8] The dataset contained annotations and bounding boxes around defects. Different classes were implemented for defects were corners, creases, damage, damaged corners, edge wear, heavy wear, holes, scratches, and tears.

Preprocessing steps for this model were minimal. YOLO internally does imagespace and colorspace augmentations and it is not recommended to augment in preprocessing.[9] The images of the dataset were already grayscale, and thus the only preprocessing step is stretching the images to 640x640 pixels.

Figure 5: Object Detection Bounding Box Example

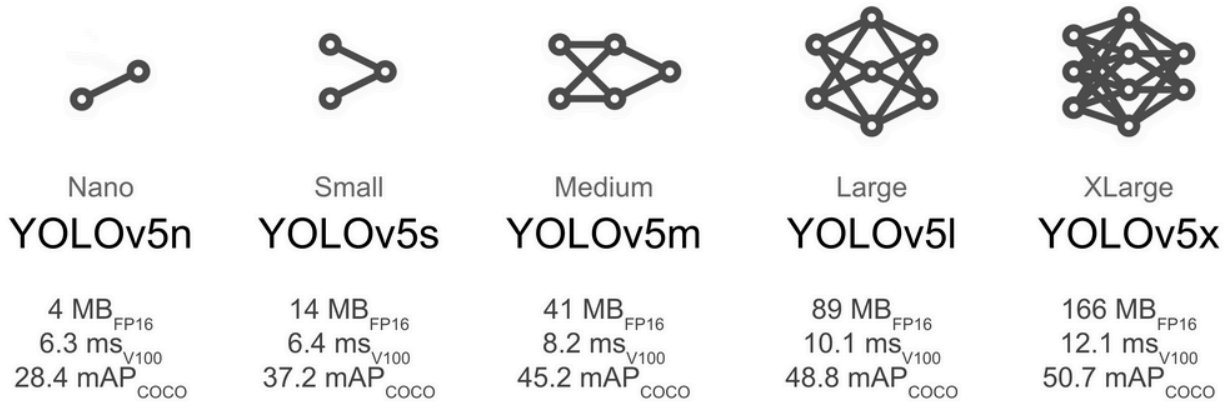
The left image shows the annotated target image and the right shows the predicted annotations and bounding boxes. Bounding box labels show mAP



The performance of the YOLOv5 model was compared across the different model sizes: nano, small, medium, large, and xlarge. This could be used to judge the cost-benefit of each size. Each model was trained for 300 epochs on the JHU Rockfish computing cluster, with the model with the best fitness (calculated from mAP, Precision, and Recall) being chosen for testing.[10]

Figure 6: YOLOv5 Models[11]

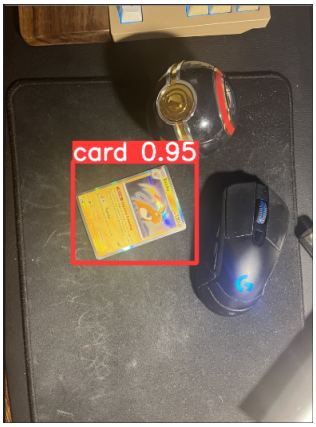
The size, inference time, and mAP (mean average precision) is given for each model



The performance of the YOLOv5s model was also compared with that of the YOLOv8s, to examine any improvements between generations of the detector. Finally, YOLOv5s was also used to perform object detection on a card detection dataset for comparison with the traditional CV card detection. The dataset used contained 972 color images of Pokemon cards in various backgrounds.[12] Note that YOLOv5 is only able to produce axis-aligned bounding boxes, however.[13]

Figure 7: YOLOv5 Card Detection Example

Bounding box label shows mAP



Results

Text Extraction:

Testing was done on 2,000 images of Pokémon Cards. Mentioned previously, two OCR models were implemented: PyTesseract and EasyOCR. For PyTesseract, the model identified 1649/2000 cards. 1620 were actually correct, with the other 29 being misclassifications. This was an accuracy of 81% with a misclassification rate of 19%. For EasyOCR, 1873 cards were classified, with 1860 of these being actually correct. This is an accuracy of 93%, with a misclassification rate of 7%. Overall, EasyOCR appeared to have superior accuracy over PyTesseract.

Table 1: OCR Model Accuracy and Misclassification Rate

Comparison of accuracy and misclassification rate over 2,000 test images

OCR model	Accuracy	Misclassification Rate
PyTesseract	81%	19%
EasyOCR	93%	7%

Feature Extraction: Template Matching & VGG-16

The following is divided into two sections: Before data color augmentation, and after data color augmentation. Before data color augmentation, template matching had an accuracy of 98%, with a misclassification rate of 2%. VGG-16 trained over 50 epochs had a classification accuracy of 95% with a misclassification rate of 5%. After data color augmentation was applied, template matching accuracy dropped to 36% with a misclassification rate of 64%. VGG-16 however stayed relatively high with an accuracy of 91% and misclassification rate of 9%. When the VGG-16 model is applied to the full card, the resulting accuracy was 74% with 26% misclassification. Template matching had slightly superior accuracy over VGG-16 before data color augmentation, but far worse accuracy afterwards. Both template matching and VGG-16 experienced a drop in accuracy following data color augmentation.

Table 2: VGG-16 and Template Matching Accuracy and Misclassification Rate

Comparison of accuracy and misclassification rate for set classification before and after data color augmentation

Model	Data Color Augmentation?	Accuracy	Misclassification Rate
Template Matching	No	98%	2%
VGG-16	No	95%	5%
Template Matching	Yes	36%	64%
VGG-16	Yes	91%	9%

Feature Extraction: ResNet50

The result metrics for the ResNet50 feature extraction model applied to the test set across each classification task are summarized in the following chart. Accuracy was similar between weakness and type classification, which were both greater than the set and rarity classification accuracy.

Table 3: ResNet60 Accuracy and Misclassification Rate

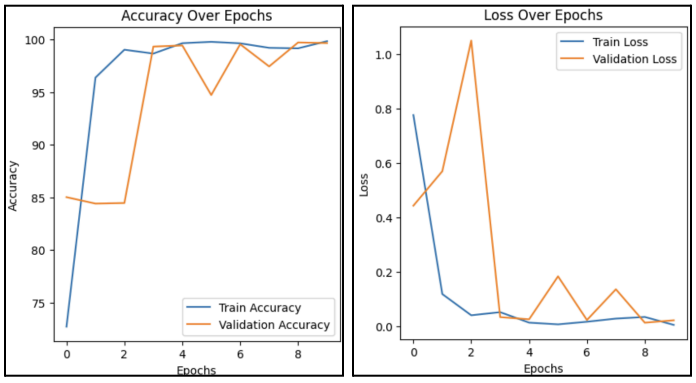
Comparison of number of correct classifications, accuracy, and misclassification rate for each classification task

Task	Correct	Accuracy	Misclassification
Weakness	1793/1804	99.4%	0.6%
Types	1847/1857	99.5%	0.5%
Set	2004/2158	92.9%	7.1%
Rarity	1944/2123	91.6%	8.4%

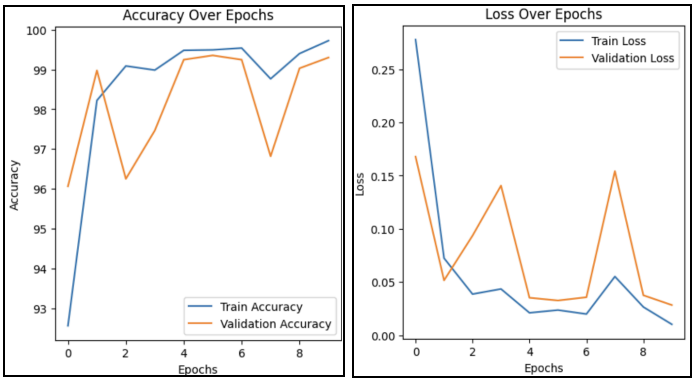
The loss and accuracy over 9 epochs can also be seen in the following graphs. As expected, accuracy generally improved over epochs while loss decreased. Training of the weakness and type models showed notable variation between epochs while set and rarity trainings were more stable.

Figure 8: Loss and Accuracy over Epochs

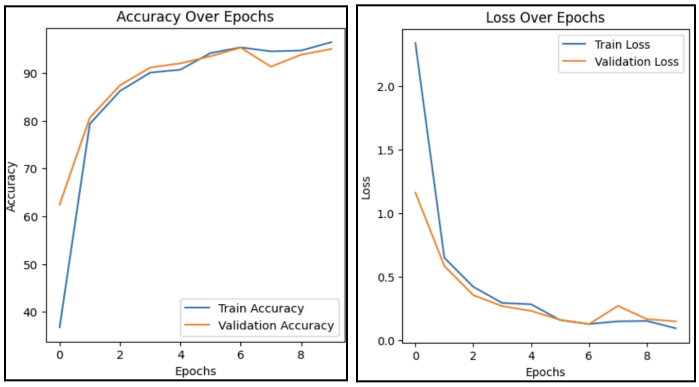
Loss and accuracy for each epoch for each classification task. Blue lines are for training and orange are for validation



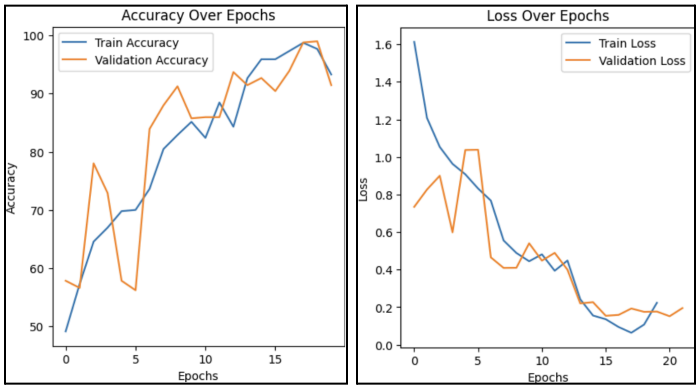
Weakness



Type



Set



Rarity

Object Detection

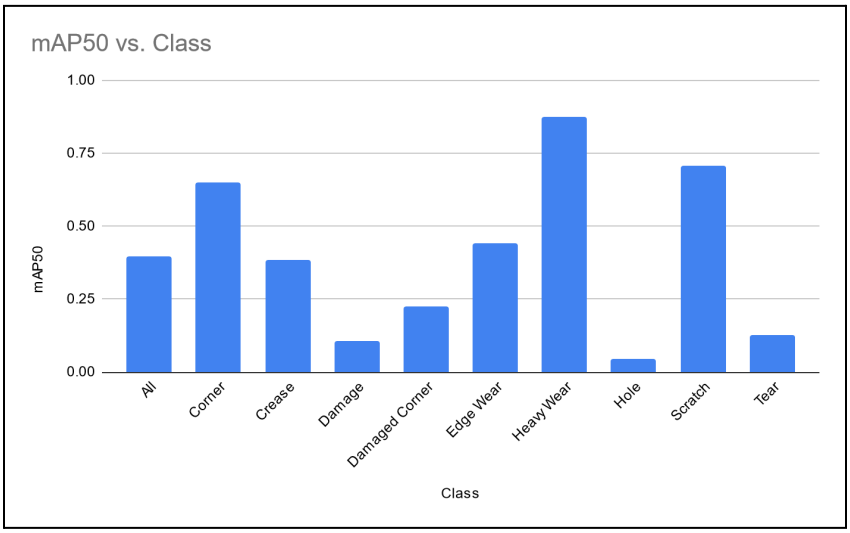
The result metrics for the YOLOv5s defect detection model applied to the test set across are summarized in the following chart. Note that certain classes were underrepresented, such as heavy wear and tear, and had low instance counts while others, such as corner, were relatively overrepresented with a high instance count. Significant variation across precision, recall, mAP50 (mAP for IOU > 50%), and mAP50-95 (mAP for 50% < IOU ≤ 95%) were exhibited between each class.

Table 4: YOLOv5s Defect Detection Accuracy for each Object Detection Class

Precision, recall, map50, and mAP50-95 given as estimations of classification and bounding box accuracy.

Classes	Instances	Precision	Recall	mAP50	mAP50-95
All	1519	0.512	0.444	0.395	0.171
Corner	728	0.717	0.650	0.649	0.276
Crease	100	0.466	0.430	0.385	0.213
Damage	37	0.239	0.108	0.105	0.0514
Damaged Corner	313	0.415	0.265	0.223	0.106
Edge Wear	30	0.588	0.533	0.441	0.126
Heavy Wear	8	1.000	0.870	0.876	0.324
Hole	5	0.121	0.200	0.0464	0.0255
Scratch	294	0.742	0.687	0.706	0.358
Tear	4	0.317	0.250	0.126	0.0624

Figure 9: YOLOv5s mAP50 for each Class

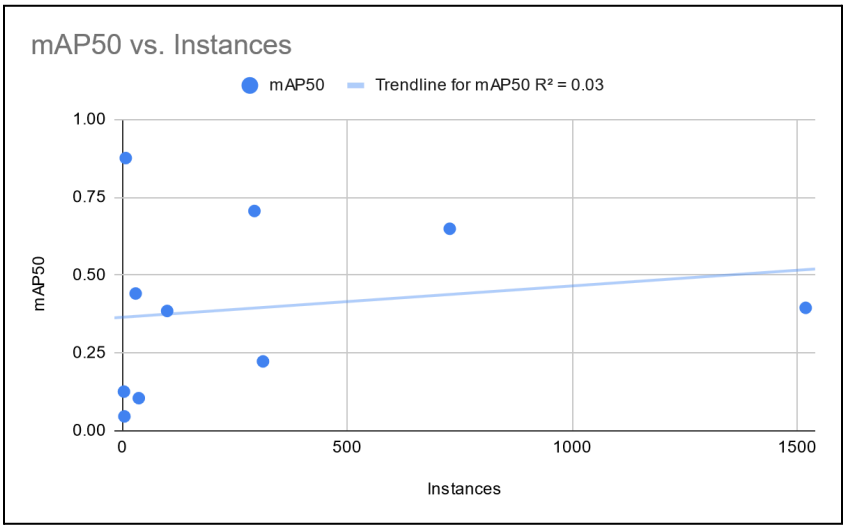


Classes are provided in alphabetical order

The relationship between the mAP50 and the number of instances for each class used in training is examined in the following graph. A linear trendline was calculated with the R^2 value provided.

Figure 10: YOLOv5s mAP50 vs. Number of Instances for each Class

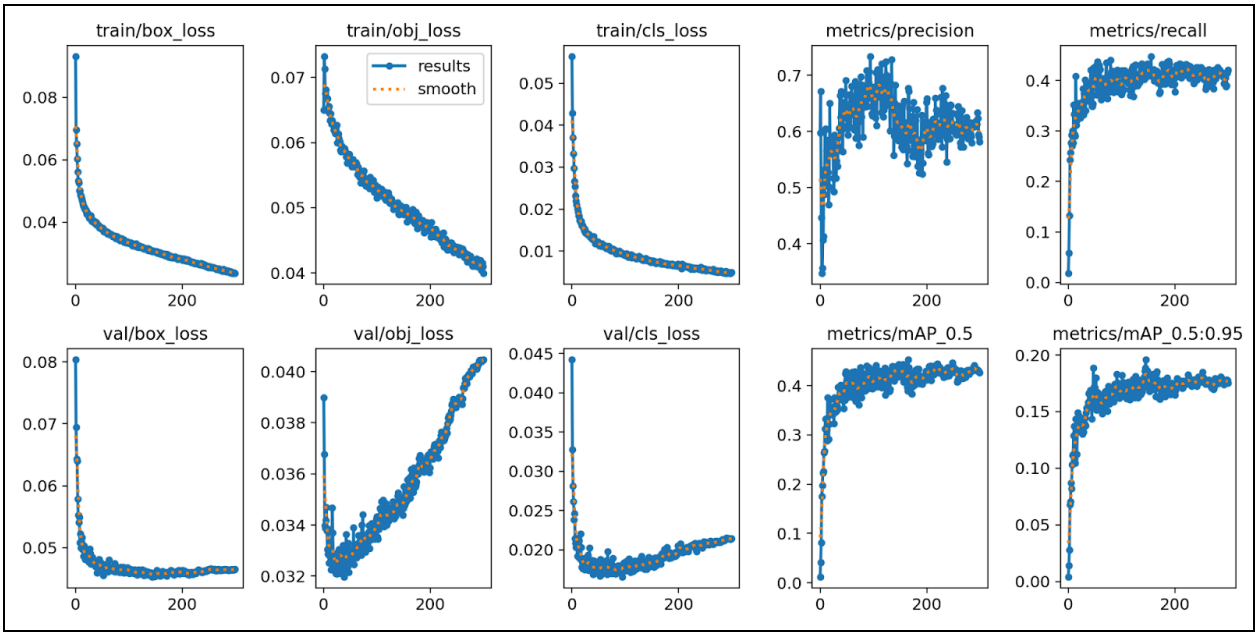
Blue dots show data points for each class. Light blue line is the linear trendline.



The evolution of loss and accuracy metrics over 300 epochs can also be seen in the following graphs. As expected, accuracy generally improved over epochs while loss decreased. For object loss and class loss on the validation set, overfitting was observed, but this should not have been a problem due to the highest fitness model being selected.

Figure 11: YOLOv5s Loss and Accuracy Metrics over Epochs

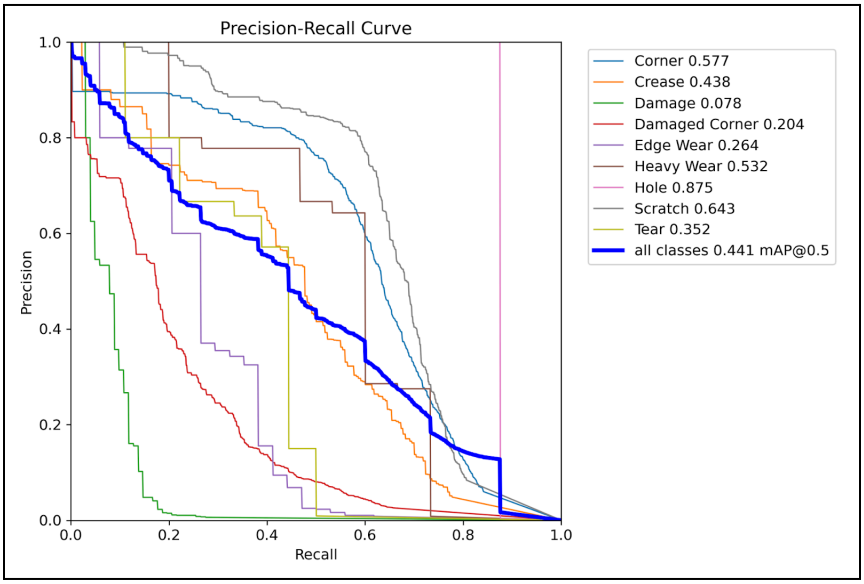
Precision, recall, map50, and mAP50-95 given as estimations of classification and bounding box accuracy. Loss provided for box, object, and class detection tasks. Blue curves are raw results, while orange shows results after smoothing.



A precision-recall curve for the data in Table 4 is given in the following graph. Curves for each class are given, with some exhibiting linear trade-offs (i.e. all classes, crease), while others (i.e. corner, heavy wear) exhibit increase area under the curve (AOC), and others exhibit reduced AOC (i.e. damage, damaged corner).

Figure 12: YOLOv5s Precision-Recall Curve for each Class

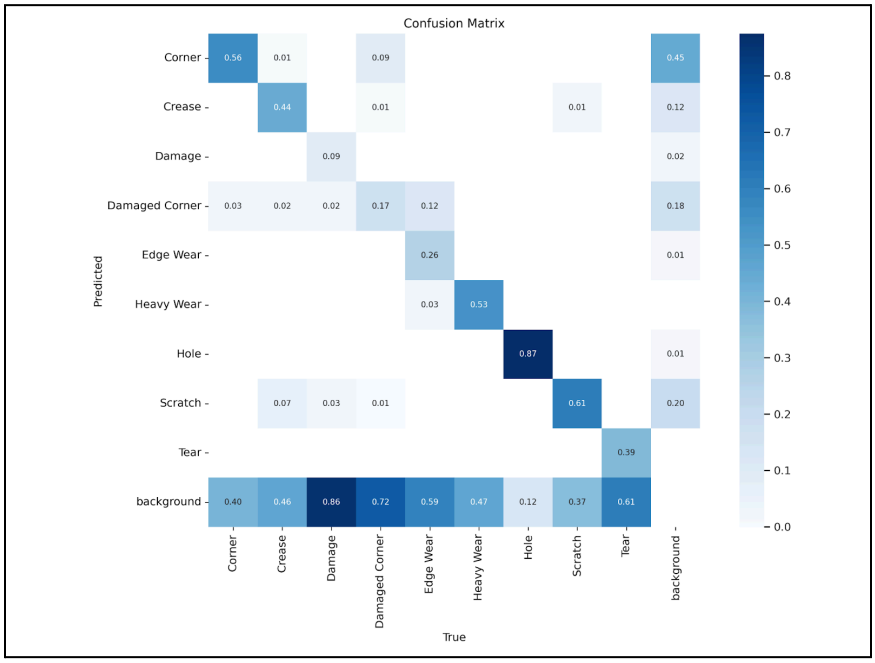
The corresponding curve for each class is given in the adjacent key



A confusion matrix for the data in Table 4 is given in the following graph. All classes experience confusion with the background. Damaged corner and tear experienced significant confusion with each other and crease and damage. Other classes experienced relatively low confusion with classes other than background.

Figure 13: Confusion Matrix for each Class Prediction and True Label

The color key for the intensity of confusion rate is on the right side (darker is more, lighter is less confusion).



The following table shows accuracy metrics for each model size tested. Generally, increasing model size led to increased accuracy and greater inference time.

Table 5: YOLOv5 Accuracy Metrics and Inference Time across Model Sizes

Precision, recall, map50, and mAP50-95 given as estimations of classification and bounding box accuracy. Sizes are in increasing order

Size	Precision	Recall	mAP50	mAP50-95	Inference Time (ms)
Nano	0.518	0.365	0.346	0.158	5.4
Small	0.512	0.444	0.395	0.171	6.8
Medium	0.552	0.390	0.419	0.201	14.9
Large	0.586	0.393	0.400	0.196	23.7
XLarge	0.589	0.429	0.419	0.200	44.7

The following table shows accuracy metrics for each model version tested. YOLOv5s had greater recall, mAP and lower inference time while YOLOv8s had greater precision.

Table 6: YOLOv5s vs. YOLOv8s Accuracy Metrics and Inference Time

Precision, recall, map50, and mAP50-95 given as estimations of classification and bounding box accuracy. Models are in order of increasing recency.

Model	Precision	Recall	mAP50	mAP50-95	Inference Time (ms)
YOLOv5s	0.512	0.444	0.395	0.171	6.8
YOLOv8s	0.574	0.352	0.350	0.154	9.1

The following table shows accuracy metrics for the card detection task.

Table 7: YOLOv5s Card Detection Accuracy Metrics and Inference Time

Precision, recall, map50, and mAP50-95 given as estimations of classification and bounding box accuracy.

Class	Instances	Precision	Recall	mAP50	mAP50-95
Card	134	0.929	0.985	0.980	00.902

Discussion

Each of the individual models had a relatively high accuracy when considering the ability to distinguish and identify cards. The first model to discuss is the text extraction model. The results indicate that EasyOCR is notably more accurate, with fewer misclassifications as well. This is expected, as the Tesseract OCR engine is heavily reliant on well-prepped, clean, and high-quality images which are difficult to achieve with Pokémon Cards, due to the different types of photos taken, as well as the different designs and colors on cards. Challenges for this model mostly resulted from the fact that different Pokémon Cards have vastly different colors, lighting effects, etc. that make a standardized preprocessing difficult to apply. The different aesthetic choices of cards, i.e, color gradients for backgrounds, holographic effects, etc. add noise to the images as well, and affect OCR accuracy. Generally, improvements can be made to preprocessing, and parameters for fuzzy matching can also be adjusted.

For the feature extraction utilizing template matching and VGG-16, it was found that the template matching actually had a higher accuracy prior to data color augmentation, but had a drastically lower accuracy after. VGG-16 had a slightly lower accuracy prior to data color augmentation, but remained relatively consistent after. Thus, if one model were to be chosen here, we would choose to apply the VGG-16 model. Both models can be used in conjunction to improve accuracy overall and ensure correct classification. However, this may add length and complexity to the overall identification pipeline. The main challenge for both template matching and VGG-16 when applied to the cropped set symbols seems to be low resolution of card images. Even when taking a relatively high resolution photo up close, the set symbols are small and will usually have a low resolution. Applying VGG-16 to the entire card had a much lower accuracy of 74%. This was most likely attributed to the similarity of certain cards across various sets. High overfitting was also observed.

The ResNet50 model generally maintained consistently high accuracy, with the weakness and type features having the highest accuracy. Set and rarity had lower accuracies. However, these were still above 90%. It should be noted that the set extraction from VGG-16 is indeed higher. Thus, in the final pipeline, we deduce that the ResNet50 model should only be utilized for weakness, types, and rarity features. It is believed that this model could however be improved with a larger dataset.

The final model is the object detection / defect detection model with YoloV5. Overall, this model demonstrates a wide range in the level of precision and accuracy across the various defect classes. Particularly noteworthy was the high precision achieved for the “Heavy Wear”, “Corner”, and “Scratch” classes. However, this model does exhibit varying levels of performance, with lower precision and recall observed for classes such as “Damage” and “Hole”. Generally, though, this class does establish a preliminary baseline for defect detection and can be further improved on, by adding more instances of the underrepresented classes to the dataset. There did not appear to be a relationship between the mAP of the classes and their instance counts ($R^2 = 0.030$), but since multiple classes were underrepresented to begin with, these results maybe not be reliable. For training, accuracy generally improved over epochs while loss decreased, as expected. For object loss and class loss on the validation set, overfitting was observed, but this should not have been a problem due to the highest fitness model being selected. A precision-recall curve for each class showed variation in the AOC for each curve with some exhibiting linear trade-offs (i.e. all classes, crease), while others (i.e. corner, heavy wear) exhibit increase area under the curve (AOC), and others exhibit reduced AOC (i.e. damage, damaged corner). A confusion matrix showed all classes experience confusion with the background, while damaged corner and tear in addition experienced significant confusion with each other and crease and damage. A table with the accuracy metrics and inference times for each model size was produced, and generally increasing model size led to increased accuracy and greater inference time. These results were roughly in line with previous studies on the COCO dataset as far as the relative performance changes across sizes.[11] Not depicted, however, was the drastic increase in training time for the larger models, even on the Rockfish cluster. A table with the accuracy metrics and inference times comparing YOLOv5s and YOLOv8s was produced and interestingly, there did not appear to be performance increase with YOLOv8s. YOLOv5s had greater recall, mAP and lower inference time, while YOLOv8s had only greater precision. Further testing of the impact of model versions would be warranted to support this conclusion. Finally, a table showing the accuracy metrics for YOLOv5s trained on a card detection dataset were shown. This model had significantly higher accuracy metrics than the one trained on the defect detection dataset, likely because it only had one class with many instances while the defect detection dataset had multiple classes with varying representation. Comparing to other work, the same dataset was hosted on Roboflow was used to train a Roboflow 2.0 Object

Detection (Fast) model, which got an mAP of 36.2%, precision of 52.3%, and recall of 35.0%.[8] The YOLOv5s model trained in this study had roughly comparable performance at an mAP of 39.5%, Precision of 51.2%, and recall of 44.4%. Comparing to DeepCornerNet, which, of note, was a classification model instead of object detection and used baseball cards, slightly lower precision and recall for the corners was found: 0.717 and 0.650 respectively for this study's YOLOv5s, and 0.77 and 0.74 respectively for their VGG-16 Model-2A. For card detection, other models also got high performance, recall, and mAP of around 90%.[14]

Future work includes creating a final pipeline for card identification will utilize the best models for text extraction and feature extraction in conjunction with each other to identify trading cards. Thus, the EasyOCR model will be utilized for text extraction, VGG-16 for set symbol extraction, and the ResNet50 model for extraction of other features. Each of these models should act as a "filter", and will be checked against each other to ensure highly accurate classifications. For object detection/defect detection, the YOLOv5x model could be further trained to be more accurate. Eventually, it is hoped to improve this model to a point that basic defect extraction can be used to help determine the value of a card. Perhaps with a dataset of graded Pokémon card images, it will be possible to improve the model to eventually automatically grade cards. Other tasks would include an object detection CNN for name, HP, moves bounding boxes to compare with traditional CV approach. This would require us to make our own dataset since such a dataset could not be found presently. A comparison of traditional CV card detection with that of the object detection CNN could be performed by running the CV card detection on the same test set and calculating bounding boxes and IOU. Classification could be expanded to other attributes in addition to the weakness, type, set, and rarity studied here. These might include resistance, element, name, etc.

Conclusion

Overall, the development of the various models for the extraction and grading of Pokemon trading cards demonstrates promise for the field of authentication and organization. This project merges traditional computer vision techniques with deep learning methodologies to a relatively high degree of success. We still need to merge these models together to create a cohesive automatic identification system, but the individual models themselves show high promise and effectively establish a baseline for further refinement and integration. The object

detection model showed comparable performance compared to previous work and showed the importance of model size, version, and dataset class representation on overall accuracy. Thus, the next steps would involve enhancing the interoperability of these models to ensure that they can work synergistically. We also plan to continue addressing the limitations identified in each model to further improve accuracy and precision.

References

1. Sakaji, H., Kobayashi, A., Kohana, M., Takano, Y. & Izumi, K. Card Price Prediction of Trading Cards Using Machine Learning Methods. in *Advances in Networked-based Information Systems* (eds. Barolli, L., Nishino, H., Enokido, T. & Takizawa, M.) 705–714 (Springer International Publishing, Cham, 2020). doi:10.1007/978-3-030-29029-0_70.
2. Nahar, L., Islam, Md. S., Awrangjeb, M., Verhoeve, R. & Tuxworth, G. DeepCornerNet: A Deep Learning Approach for Automated Corner Grading in Trading Cards. in *2023 International Conference on Digital Image Computing: Techniques and Applications (DICTA)* 24–31 (2023). doi:10.1109/DICTA60407.2023.00013.
3. Hernandez, J. C. S. MTG CARD GRADER USING IMAGE CLASSIFICATION AND DETECTION.
4. Hashemi, N. S., Aghdam, R. B., Ghiasi, A. S. B. & Fatemi, P. Template Matching Advances and Applications in Image Analysis. Preprint at https://doi.org/10.48550/arXiv.1610.07231 (2016).
5. Wang, T., Wu, D. J., Coates, A. & Ng, A. Y. End-to-End Text Recognition with Convolutional Neural Networks.
6. Pokemon Card Database. https://www.kaggle.com/datasets/stevenu/pokemon-card-database.
7. Pokemon TCG All Cards 1999 - 2023. https://www.kaggle.com/datasets/adampq/pokemon-tcg-all-cards-1999-2023.
8. PokeFrontBackDamage Dataset > Overview. *Roboflow* https://universe.roboflow.com/pokemonfyp/pokefrontbackdamage.
9. data augmentation · ultralytics/yolov5 · Discussion #10469. *GitHub* https://github.com/ultralytics/yolov5/discussions/10469.
10. The ‘fitness’ function in train.py. · Issue #2303 · ultralytics/yolov5. *GitHub* https://github.com/ultralytics/yolov5/issues/2303.
11. Fig. 3. YOLOv5 different model sizes, where FP16 stands for the half... *ResearchGate* https://www.researchgate.net/figure/YOLOv5-different-model-sizes-where-FP16-stands-for-the-half-floating-point-precision_fig3_354846944.

12. pkb_mul Dataset > Overview. *Roboflow* https://universe.roboflow.com/thomasdlmp/pkb_mul.
13. OBB with yolov5 ? · Issue #9908 · ultralytics/ultralytics. *GitHub* https://github.com/ultralytics/ultralytics/issues/9908.
14. TCG Dataset > Overview. *Roboflow* https://universe.roboflow.com/tcg/tcg-kzyeu.