Tiny Towns Scorer

Alex Owens
Virginia Tech
Blacksburg, Virginia, USA
xelasnewo7@vt.edu

Daniel Schoenbach
Virginia Tech
Blacksburg, Virginia, USA
danielschoenbach@vt.edu

Payton Klemens
Virginia Tech
Blacksburg, Virginia, USA
danielck@vt.edu

Figure 1. A game state from the board game Tiny Towns alongside the recognized digital game state. Fourteen of the sixteen tokens were identified correctly.

Abstract

In the board game Tiny Towns, a player's score is determined by pieces arranged into a 4x4 grid. We propose a novel computer-assisted model for calculating a player's score using computer vision. By combining object detection, feature matching, and homography, we can detect and classify each piece on a player's game board. This digital representation of the game state can then be used to calculate a player's score. We introduce the problem, detail initial experiments, and present a final model. The model achieves high accuracy on clear images, but poorly on images taken at distance or at oblique angles.

Keywords: OpenCV, computer vision, board game, Tiny Towns, classification, scoring, RetinaNet, homography

1 Introduction

"Board games" is a broad category encompassing a wide variety of games played by many people. Most board games have physical pieces or tokens whose composition and arrangement determine the current state of the game. Purely digital versions of many board games have also been published, which have the advantage of enforcing rules and calculating scores automatically. However, the experience of playing a purely digital game often lacks the social benefits of physical play. To preserve these benefits, we focus on physical play and use computer vision to gain some of the benefits of digital play. We propose a multi-part model for calculating scores in the board game Tiny Towns [5].

In Tiny Towns, each player has their own game board with 16 spaces arranged into a 4x4 grid. As the game is played, these spaces are filled with tokens of different shapes and colors representing buildings or resources. A player's score is determined entirely by the type and arrangement of pieces on their board at the end of the game. This makes Tiny Towns ideal for computer-assisted scoring as no extra work is needed while playing.

In the first part of the proposed model, we acquire an image of the physical game state consisting of a player's game board and tokens. We then use a neural network based on the RetinaNet [6] architecture and trained on gameplay images to detect game objects in the image. Next, we perform feature matching between a reference image of the game board and the detected board using features generated by ORB [9]. Using a projective transformation derived from the matches, the model transforms the detected game objects to an overhead perspective. Lastly, the model partitions the transformed area into a 4x4 grid, effectively creating a digital

version of the physical game state. The player's score can then be calculated from the digital game state according to the game rules.

2 Background

When finishing a board game, manually tabulating scores and determining a winner can be a cumbersome task. The excitement of the game's play rarely extends to the game's scoring. Furthermore, players may miss a relevant piece or configuration of pieces, forget a scoring rule, or make an arithmetic error. Human scoring also takes a nontrivial amount of time— time that could be spent playing another game!

The most common physical scoring aids are pads of disposable paper scoring sheets provided with the game. However, these only ameliorate the underlying issues with human scoring, not eliminate them. Additionally, scoring pads are a finite resource; they will eventually run out.

While less-studied than other computer vision problems, the idea of scoring board games is not completely novel. Some commercial applications have been published [2], and similar challenges have been attempted academically as well. In particular, Waymann [10] explored the detection and positioning of building-shaped game pieces on a hexagonal grid. However, their color-based classification model failed to achieve a viable level of accuracy.

3 Data

3.1 Collection

To study this problem, we created a dataset of images of Tiny Towns game boards, tokens, and game states. To accomplish this, we played a game of Tiny Towns and photographed our boards every few turns (Figure 2). We also took several

Figure 2. Experimental setup.

photographs of each Tiny Towns piece on the game board by itself, as well as created a reference scan of the game board. Photos were taken using hand-held smartphones at 3 angles: top-down, front-isometric, and side-isometric. Some pictures focused purely on the game state, while others included unrelated background elements. Image orientations were corrected manually following collection.

3.2 Annotation and Class Imbalance

To label the data, we used the Computer Vision Annotation Tool (CVAT) [3]. For each image, we drew rectangular bounding boxes around the game board and each token on the game board. These annotated images comprised the complete dataset. In total, there were 226 images containing 1,436 objects across 13 classes. Since our dataset consisted of only

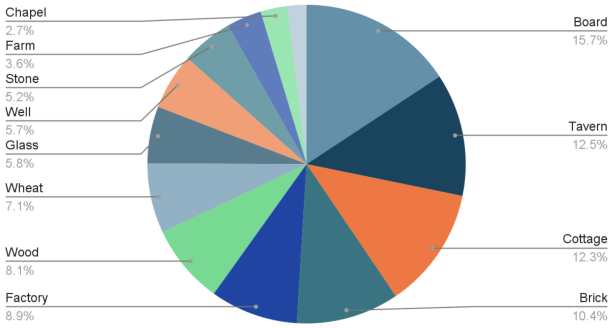

Figure 3. Dataset class breakdown.

a single game of Tiny Towns, there were differences in the number of images collected for each type of token, leading to a class imbalance. For example, some pieces comprised as many as 10-12% of tokens, whereas others were as few as 2-4% (Figure 3).

3.3 Train/Test Split

We first split the complete dataset into two categories: images from the played game and images of individual pieces taken afterward. We combined the images of individual pieces and 80% of the game state photos to construct a training dataset. The other 20% of the game state images were used for validation.

3.4 Augmentation

When training a neural network, a relatively small number of images in the dataset could lead to overfitting. To combat this, we used a preprocessing technique called data augmentation on the training dataset to create new images from existing ones. Rather than training on raw images, the model was trained on an infinite "pipeline" of generated training data.

When working with single-class images, augmentation does not affect image labels. In object detection, however, labeled images contain bounding boxes that must be preserved through the augmentation process. Otherwise the extra data gained will be labeled incorrectly and corrupt the training process.

Our augmentation process was limited to techniques with no or easily-corrected impact on the bounding boxes. The

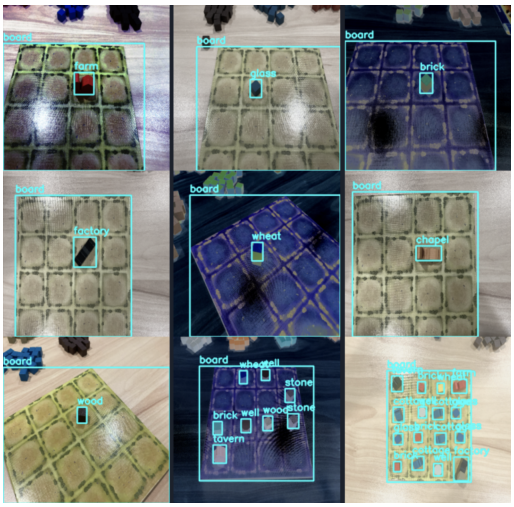

Figure 4. Examples of data augmentation.

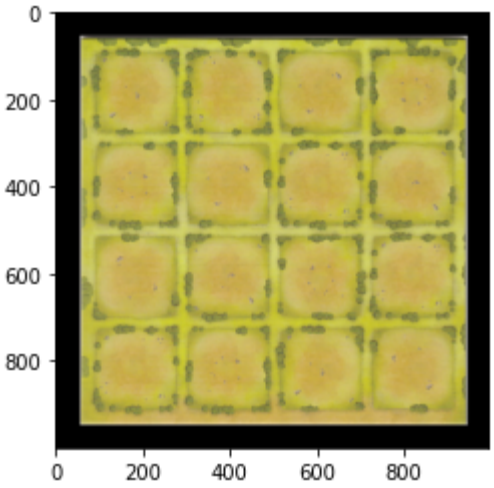

Figure 5. A scanned Tiny Towns game board.

two data augmentation layers used were RandomFlip and RandAugment [4]. RandomFlip flips images horizontally and vertically, but we chose to solely do horizontal flips. RandAugment chooses from a pool of distortions (adjusted brightness, contrast, color, etc.) and randomly applies zero or more to the image. The original RandAugment also includes geometric distortions like shear, but those were excluded to preserve bounding boxes. Examples of the augmented dataset can be seen in Figure 4.

4 Experimentation

4.1 Identification and Homography

The first part of the model must identify the game board within the "test" image. To do this, we use a scanned reference of the game board (Figure 5) as a "query" image and identify matching points of interest between the two images. We then construct a *homography*, or projective transformation, between the images. The transformation takes points from the plane of the board in the test image and sends them to the plane of the board in the query image. The board in the transformed test image then has the same overhead composition as the board in the query image.

4.2 Manual Corner Detection

To humans, the corners of the game board are easily identifiable as points of interest. We began by manually selecting the corners of the game board in a test image and using those 4 coordinates with the scikit-image transform module to perform homography. We saw good results with this method as can be seen in Figure 6. However, asking users to manually select the corners is undesirable, so we attempted to find the corners with Harris Corner Detection next.

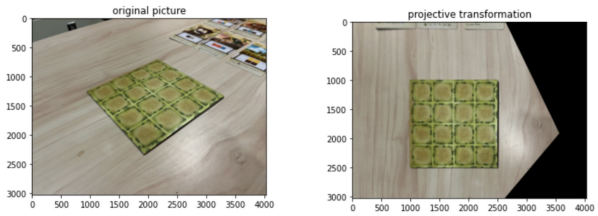

Figure 6. An example of projective transformation using manual corner detection.

4.3 Harris Corner Detection

We attempted to use Harris Corner Detection, but our observations and tests led to poor results. Instead of successfully finding corner points, our test image would result in looking glitched with a large mass of dots as seen in Figure 7. These results were less than promising, so we decided to move on from corner detection.

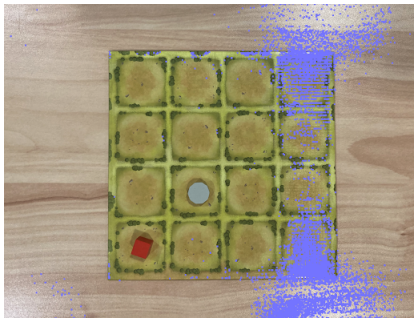

Figure 7. The "corners" found on a test image after utilizing Harris Corner Detection.

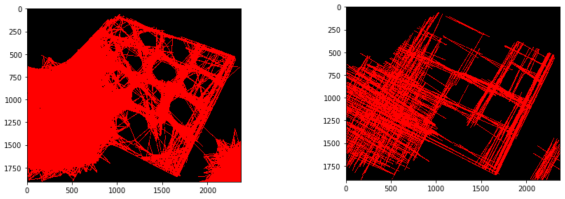

Figure 8. The left half displays the lines found in a single board image; the right half displays the perpendicular and parallel lines found after analysis.

4.4 Hough Line Detection

We then leveraged prior knowledge about the game board, namely that it is a 4x4 grid. Using this, we tried to detect lines and find the most likely 4x4 grid in the image. First, we identified the most likely lines via Hough Line Detection [8] as seen in the left half of Figure 8.

After this, we found the subset of lines that are all parallel and perpendicular to each other, as shown in the right half of Figure 8. We then grouped them by similarity to find the 8 largest groups, corresponding to the 8 lines of the grid. These groups then form the board.

This approach had some drawbacks, such as being disrupted by reflections on the game board. We ran out of time before achieving sufficient accuracy using this approach, but would consider it for future work.

4.5 ORB

Our last attempt expanded the definition of interesting points, or *features*, rather than continuing to rely solely on corners and edges. Several algorithms for identifying and constructing these features have been developed. We used the ORB algorithm [9] to create lists of feature points in the query and test images.

For each feature in the query image (Figure 9), we used a brute force algorithm to find its two nearest neighbors in the test image, as measured by Hamming distance. We then filtered this list using Lowe's ratio test [7], keeping only features whose distance to their 1^{st}-nearest neighbor is less than 70% of the distance to their 2^{nd}-nearest neighbor. These remaining "good" matches (keeping only the first neighbor) are then used to construct a projective transformation.

4.5.1 Multiple Boards. While experimenting with ORB, we discovered that the model had a harder time matching the reference features to test images that were taken at an angle. Hypothesizing that reference images of the board taken at an angle would have better matches, we expanded the model to use several query images rather than a single one (Figure 10). The reference image that resulted in the most matches was used to construct the transformation (Figure 11).

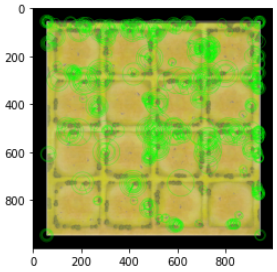

Figure 9. Features selected via ORB.

Since the query images were no longer uniform in perspective, this means the transformation did not always produce the desired overhead view. To resolve this, we created secondary transformations between the overhead reference and all other query images. By composing the transformations between test image and query image, and between query image and overhead reference, we could match angled test images to angled query images, but transform them to an overhead perspective. Anecdotally, however, the "multiple board" enhancement did not perform substantially better than using a single reference image and was not included in the final model's feature matching component.

4.5.2 Distraction. Although feature matching with ORB worked well for many images, we identified some where ORB allocated too many features to describing background elements rather than the game board. In these cases, the board could not be identified because too few features describing the game board were available for matching to the reference image. If the image was first cropped to remove the distracting elements, the issue was resolved.

5 Final Model

5.1 Motivation

Hoping to achieve better accuracy than ORB, we then sought to employ a convolutional neural network (CNN), leading to our final model. Our final model takes advantage of both a neural network based on the RetinaNet [6] architecture, as well as the ORB feature matching developed earlier. The code and data will be available at https://github.com/MagicShoebox/vt-cs4664-tiny-towns-scorer.

5.2 RetinaNet

RetinaNet is a one-stage object detector, combining a backbone network and two task-specific subnetworks. The backbone network consists of a Feature Pyramid Network (FPN) built on top of the ResNet50 architecture. The two task-specific subnetworks are a classification subnet and a box regression subnet. The classification subnet predicts the probability of the presence of a certain object, while the regression subnet outputs the object location with respect to a bounding box if it exists [6].

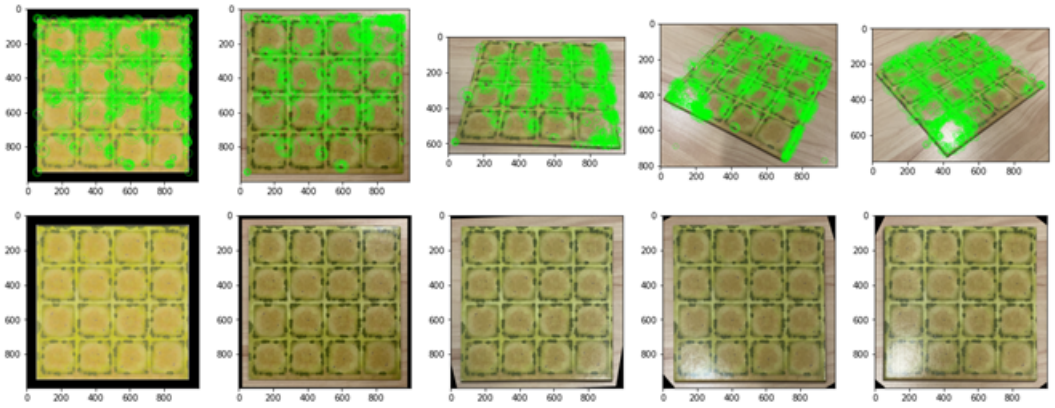

Figure 10. Several query images with their ORB features and transformations to overhead reference.

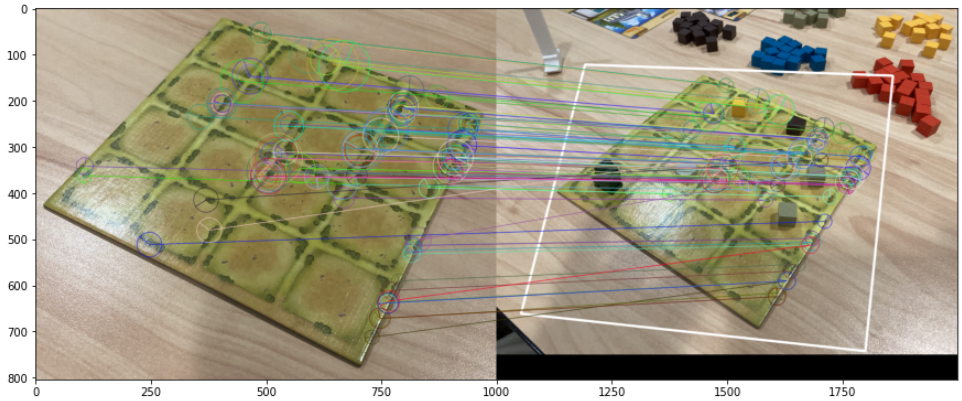

Figure 11. ORB features from an angled query image matched to test image features. The white box outlines the edges of the query image under the transformation.

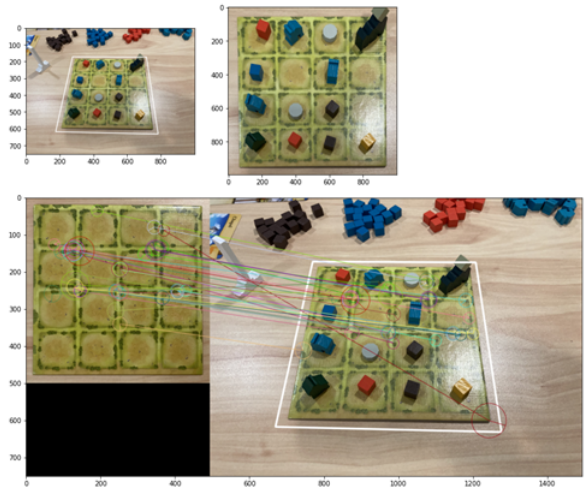

Figure 12. An overhead test image (upper left), matching features (bottom), and transformed result (upper right).

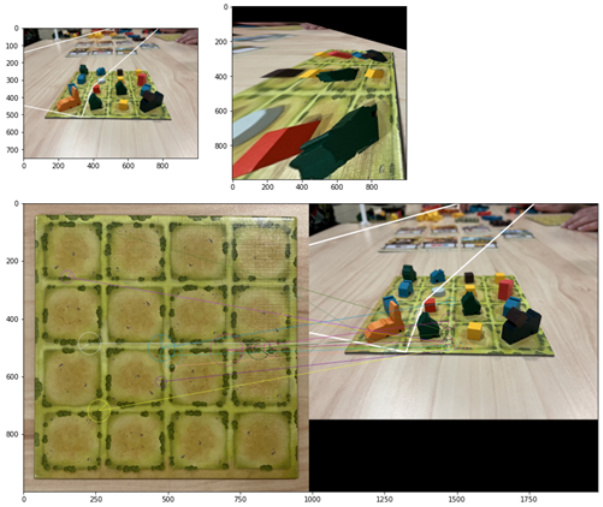

Figure 13. The model failed to match enough features in this angled test image and generated an unusable transformation.

We selected RetinaNet because of its high accuracy and simplicity of training. Many other object detection networks have complicated training procedures. In contrast, RetinaNet uses a novel "focal loss" function to achieve high accuracy in a reasonable training time while still using typical neural network training processes.

5.3 Training

We use KerasCV [11] to implement and train the network. First, the ResNet50 portion of the network is loaded with weights from a version of ResNet50 pretrained on ImageNet. This portion of the network's weights are frozen and never updated. To train the other parts of the network, we construct a training pipeline: annotated images are resized to 512x512, shuffled in groups of 128, and then batched into groups of 16. These batches are then augmented using the techniques discussed earlier to artificially expand our dataset. Then model was then allowed to train for 128 epochs before the validation loss failed to improve for 10 epochs and training was stopped.

Figure 14. Objects and their classes detected by the trained neural network. Some game pieces have been detected as multiple objects.

5.4 Projective Transformation

The trained neural network outputs bounding boxes, classes, and a confidence score for detected objects in the image. A visualized example of these predictions (without the confidence scores) can be seen in Figure 14. This is an achievement, but to construct the digital game state we must still identify the grid arrangement of the objects.

To begin, the next part of the model identifies the detected bounding box for a game board with the highest confidence score. The model then slightly expands the box and extracts the corresponding region of the image. Since the extracted region contains only the board and pieces, this resolves the issues discussed in 4.5.2.

Finally, the model uses OpenCV [1] to perform ORB-based feature matching as described in 4.5, using 5000 features. Provided a sufficient number of good matches are found, we again use OpenCV to construct a projective transformation to an overhead perspective.

5.5 Constructing the Grid

Using homography, the center points of the remaining bounding boxes are transformed into the reference perspective. In this perspective, the points are easily partitioned into a 4x4 grid. Since the network may predict multiple objects within one tile on the grid, we select the object with the highest confidence score. This grid of classified objects is the final output of the model, visualized in Figure 1. A flowchart of the complete model can be seen in Figure 15.

6 Results

6.1 COCO Metrics

We evaluated the neural network portion of our model using COCO metrics commonly used in object detection tasks.

In general, precision refers to the number of true positives out of all the positives that were predicted by the model, while recall measures the percentage of true positives out of all actual positives in the dataset. These definitions do not perfectly describe COCO metrics, but provide good intuition for understanding the results.

Both metrics provide valuable insight for us. A high precision rate for tokens would infer that our model's classification of those tokens is good. In contrast, a high recall score would indicate that our model is good at detecting all tokens that are actually within the image. As seen in Figure 16, for the game board and the well, our model does a good job in both precision and recall metrics which is likely due to the unique shape of the tokens. Overall though, our model has a low precision metric and an average recall metric. Our model is able to do a decent job at picking up a majority of the tokens in the image, but doesn't have consistency with its classification.

6.2 Accuracy

To evaluate the complete model, we compared the grid of class predictions output by the model to the grid of actual objects present in the image. This leads to a simple accuracy metric: the number of spaces correctly identified divided by the total number of spaces (16). However, the detected grid could be a rotated version of the physical grid. Since the board's orientation has no effect on scoring in Tiny Towns, we considered the best-possible orientation for the detected grid.

Unfortunately, we were unable to calculate the model's accuracy on all the images in the dataset since they were not

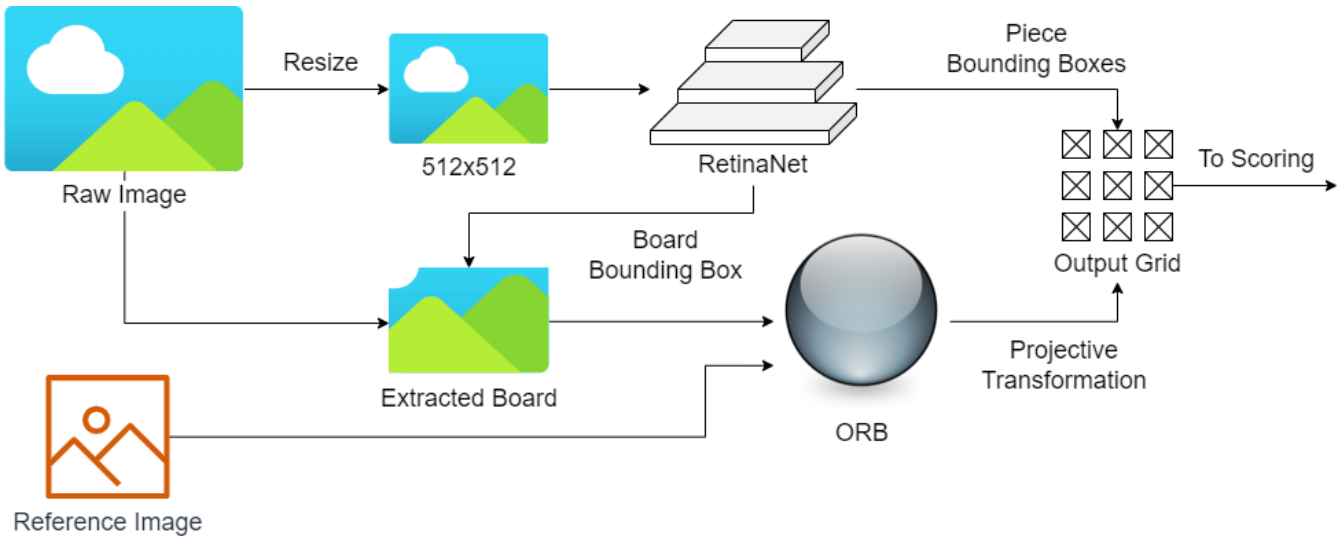

Figure 15. Our final model for Tiny Towns Scorer.

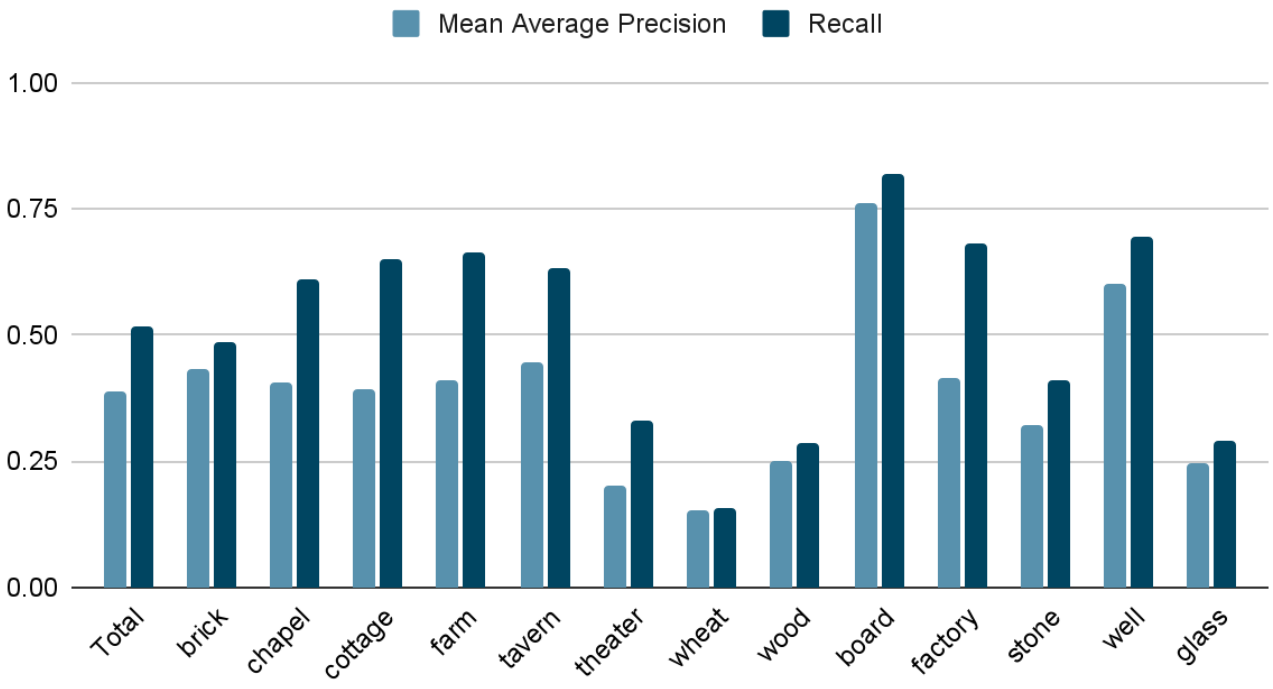

Figure 16. COCO metrics across all game images.

labeled with their complete grid. Further, we could not derive the correct grid from the object annotations since the board position was only annotated with a bounding rectangle, and deriving the board position from said rectangle is one of the problems the model is attempting to solve!

Instead, we evaluated the accuracy of the model on the 10 images in the dataset that were taken at the conclusion of the played game. This is justified since the model is intended to only be used on a player's final game state to calculate their score. These results can be seen in Table 1.

The model performed well on some images, but very poorly on others. On examination, we discovered that even among the poorly-handled images, the neural network part of the model tended to produce good predictions. Only image 0279's poor accuracy was a result of poor object detections.

Image	Correct	Accuracy
frontal/IMG_0278.jpeg	2	13%
frontal/IMG_4556.JPG	14	88%
frontal/IMG_6211.jpg	3	19%
side_angle/IMG_0280.jpeg	1	6%
side_angle/IMG_4557.JPG	14	88%
side_angle/IMG_4558.JPG	14	88%
side_angle/IMG_6212.jpg	3	19%
top_down/IMG_0279.jpeg	8	50%
top_down/IMG_4555.JPG	14	88%
top_down/IMG_6210.jpg	15	94%

Table 1. Number of grid spaces correctly identified in images taken at the end of the game.

In the other cases, the ORB features were mismatched, producing a bad projective transformation similar to the effect seen in Figure 13. These bad transformations distorted the network's detections into highly inaccurate output grids.

7 Conclusion

Based on these results, our model was somewhat effective and holds promise, but is not yet accurate enough for a real-world environment. Additionally, the model is incomplete: it does not take the final planned step, calculating a player's score using the detected grid. Beyond this obvious addition, several avenues exist for potential improvements to the model.

First, more images could be collected and added to the dataset. Class imbalances in the existing dataset may have skewed some object detections, as evidenced by the model's confusion between the "glass" resource and the similarly-colored "cottage" building. Correcting these imbalances, as well as having more varied grid arrangements in the training data, could improve the detections in the neural network part of the model.

The most significant detriment to the model's performance, however, was poor matching of ORB features when identifying the board's exact position. Anecdotally, ORB features seem to "focus" on foreground objects– a poor choice when trying to detect the "background" game board. A different approach that relied on line detection or some sort of "most-likely-grid" algorithm using the classified points could produce substantially better results. Alternatively, if the dataset annotations were supplemented with the corner points of the boards, perhaps the backbone portion of the model's network could be trained to identify the corner positions directly via regression.

Finally, the RetinaNet architecture has some inherent drawbacks in its classification method with respect to our problem. Currently, generated bounding boxes for the same class that overlap significantly are suppressed into one. However, that doesn't prevent the network from predicting two different pieces contained in a single tile, which is an invalid game state. In the future, this could be solved by first detecting pieces as a single class and then using their bounding boxes to classify them precisely.

Acknowledgments

Anuj Karpatne, for the instruction of CS 4664.
Luke Wood, for authoring an object detection tutorial and contributions to KerasCV.

References

[1] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[2] Serg Buslovsky. 2019. Ticket to Ride Score Calculator. https://serg.buslovsky.com/ttrsc/

[3] CVAT.ai Corporation. 2022. CVAT. https://www.cvat.ai/

[4] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. 2019. RandAugment: Practical data augmentation with no separate search. *CoRR* abs/1909.13719 (2019). arXiv:1909.13719 http://arxiv.org/abs/1909.13719

[5] Alderac Entertainment Group. 2022. https://www.alderac.com/tiny-towns/

[6] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal Loss for Dense Object Detection. (2017), 2999–3007. https://doi.org/10.1109/ICCV.2017.324

[7] David G. Lowe. 2004. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision* 60, 2 (2004), 91–110. https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

[8] Peter E. Hart Richard O. Duda. 1972. Use of the Hough Transformation To Detect Lines and Curves in Pictures. *Commun. ACM* 15, 1 (1972). https://dl.acm.org/doi/10.1145/361237.361242

[9] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. 2011. ORB: An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision*. 2564–2571. https://doi.org/10.1109/ICCV.2011.6126544

[10] Lukas Waymann. 2018. *Scoring Board Games with Computer Vision*. Bachelor's thesis. University of Bayreuth, Bayreuth, Germany. https://meribold.org/scoring-board-games-with-computer-vision.pdf

[11] Luke Wood, Zhenyu Tan, Stenbit Ian, Scott Zhu, François Chollet, et al. 2022. KerasCV. https://github.com/keras-team/keras-cv.

Submitted 8 December 2022