Computer Move Suggestions from Physical Game Images

Xiluo He
Stanford University
450 Serra Mall, Stanford, CA 94305
xiluohe@stanford.edu

Zach Witzel
Stanford University
450 Serra Mall, Stanford, CA 94305
zachwitz@stanford.edu

Abstract

In this paper, we discuss how we constructed computer-vision programs which are able to recognize the board state of physical versions of the games Set and Scrabble. The programs rely on a combination of traditional computer-vision techniques as well as convolutional neural networks. Once the board state is recognized, the programs are then able to analyze the game and suggest optimal moves. In order to construct our programs, we also developed our own datasets of images of Scrabble and Set games from a simulated, realistic looking environment through rendering on Blender. We demonstrated that CNNs can successfully recognize different areas of Set and Scrabble boards. We hope these programs will help people both save time when looking for computer assistance for their games as well as contribute to players' improvement through ease of use in postgame analysis.

1. Introduction

The popularity of computer analysis programs for board games has increased in recent years as artificial intelligence can outperform most humans in games from chess to go. These programs allow players to improve their abilities by learning from the computer's move suggestions. However, while many board game solvers are available online, it is still difficult to quickly run computer analysis as the digital board has to be edited as the physical board changes. This becomes tedious when multiple boardstates are to be analyzed. Building off of previous works on converting physical chess board positions to digital versions [6, 15], we aimed to convert images Set and Scrabble game states into digital versions that could be easily used for computer analysis.

Set is a card game consisting of 81 unique cards with 12 random cards placed down at a time on a table. Each card has 4 features – shape, color, fill, and number – with each feature having 3 options (eg. color could be red, purple, or green). The objective of the game is to find a set of 3 cards with each feature being either all the same or all different (eg. a set can consist of 3 cards with all the same color and shape and all different number and shading).

Scrabble is a board game consisting of a 15x15 game board and lettered game tiles. Each player has 7 tiles in their personal rack and take turns placing them onto the board, branching of already placed tiles such that valid words form both horizontally and vertically. The objective of the game is to gain points by making more words that build off of existing words on the board.

There are many challenges in converting physical game states to digital ones. For instance, there is a lack of datasets available for this task, especially for individual pieces or cards. There are also typically a large number of classes with only subtle differences between some classes. The model also needs to be able to compile the results of individual pieces on a game board which requires understanding of the game board itself with explicit references to an empty board.

Performance with digitalizing Set and Scrabble boards can provide a baseline into how other board games can perform. Because many others have already created board recognition programs for chess, we sought to explore the potential in digitizing other games. Set and Scrabble were chosen because they have clear objectives, obvious classes, and are not heavily dependent on luck. Specifically, our goal was to take an image of a physical game of Set or Scrabble to then convert it into a digital representation of the board state such that the computer could analyze it and suggest a move.

To obtain a model of the game pieces, we begin with self created synthetic datasets of Set and Scrabble boards. The datasets are then processed using various computer vision techniques to create a new dataset of Set cards and Scrabble tiles. The input to our algorithm are these images. We then use a CNN to output a predicted card or tile type. We match this prediction with the original board image to recreate the physical board. We also show examples of computer analysis programs used on the digital boards.

In this work, we explore a physical game board image to

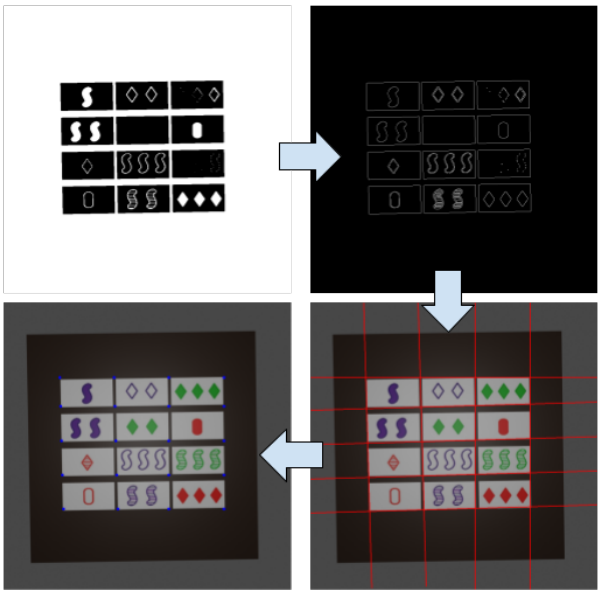

Figure 1. Bilateral filter, Canny edge detection, Hough lines transform, and point cluster applied on Set image to detect Set cards

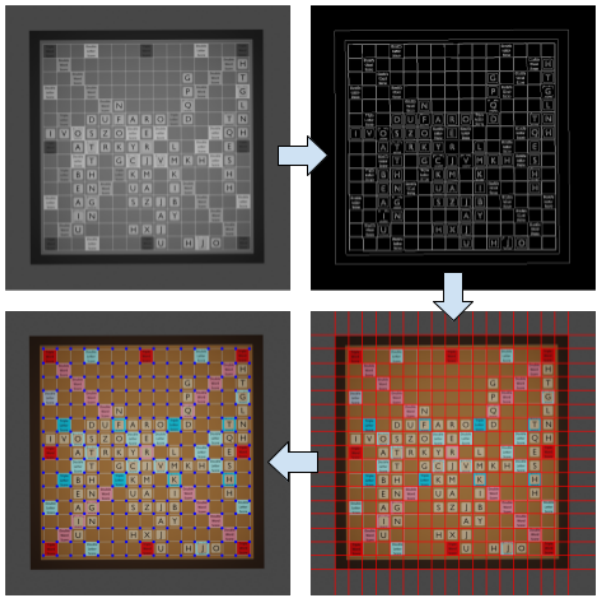

Figure 2. Grayscaling, Canny edge detection, Hough lines transform, and point cluster applied on Scrabble board image to detect Scrabble grids

digital game board pipeline that uses both traditional computer vision techniques as well as CNNs. We also create synthetic datasets for Set and Scrabble. Our method achieved 95.58% accuracy on Set and 93.10% accuracy on Scrabble.

2. Related Work

2.1. Synthetic Datasets

Synthetic datasets have been used in machine learning models and proved to perform only slightly worse than real-life datasets in tasks such as classification as the realism of synthetic images does not need to be perfect to achieve its purposes [8, 13]. It is commonly used in medical applications due to privacy concerns and in other applications where there is a lack of resources to obtain data [5, 16]. Blender is a 3D computer graphics software that allows for 3D renderings of scenes and has a Python API that allows for editing of a Blender model's scene and objects. It has been used to create datasets and allows for isolated experiments where irrelevant details are abstracted away in the data [3].

2.2. Chessboard Detection

Chessboard detection shares many similarities with Scrabble and Set and has been well studied and documented due to the checkerboard pattern's role in camera calibration [1]. Image transformations and morphological operations are the main methods used for chessboard detection [9]. Usually, the Canny edge detector [2] is applied on a grayscale image for the Hough lines transform to detect the gridlines [15]. The intersection points of these lines define the corners of the chessboard which can be used to segment the board into individual squares. Previous techniques also include merging similar lines, clustering intersection points, and using heap maps to probabilistically determine where the board is located [4]. Line detection remains as the most popular method for chessboard detection, but is very sensitive to noise and can typically only be done with domain knowledge of the environment (lighting, angle, focal length, etc). This is because line detection lacks the ability to take into account context, mid-level and high level information, and local information outside of the threshold [6]. Neural networks have been shown to be promising, but in general it remains difficult to enforce all the rules required to define an edge [7].

Chess piece recognition typically relies on support vector machines (SVMs) or convolutional neural networks (CNNs). SVMs are trained as binary one-vs-rest classifiers and each classifier is run over the testing image [6]. Given enough training data and a relatively low number of classes, CNNs outperform SVMs. For datasets where the piece colors do not match the board colors, color segmentation has also proven to work well.

3. Methods

3.1. Board Detection and Cropping

Our approach for board detection during testing is the same as cropping the board to generate the training dataset. At a high level, our approach for board detection for both

Set and Scrabble can be seen in Figures (1) and (2) and consists of: preprocessing the image, using Canny edge detection, using Hough line transform, finding the vertices of the board, and finding each subarea of the board where individual pieces could reside.

Canny edge calculates the local gradient of the image to detect edge intensity and performs non-maximum suppression, double threshold, and edge tracking by hysteresis to thin out the lines. Hough line transform identifies lines on binary images by transforming the edges into points in a new image space defined by the slope and intercept of that edge. These points are turned into lines in the new space and if many lines cross a certain point, then that point is transformed back into a line in the original space because it means there are many lines with that slope and intercept.

The first step is to apply Gaussian blur and a bilateral filter to reduce noise and produce a suitable input for Canny edge detection. Then, the Hough transform was applied with a low threshold to ensure every grid line was captured. This resulted in extraneous lines that were later eliminated. The lines were separated into horizontal and vertical lines and both subsets of lines were processed by location to eliminate extraneous lines. Hough transform orders the lines by descending strength, so the remaining lines were processed in order with each line added to the list of valid lines if it was not similar to any previous lines. This discarded almost all duplicate lines but occasionally discarded a useful grid line. The intersections of the remaining vertical and horizontal lines were used to find the vertices of each square on the board. The intersection points were then clustered by euclidean distance to remove redundant points and sorted before being normalized such that each row had the same y-coordinates.

3.2. CNN Classification

Once we divided up the game state into component tiles or cards, we used CNNs to classify what kind of object was in each box.

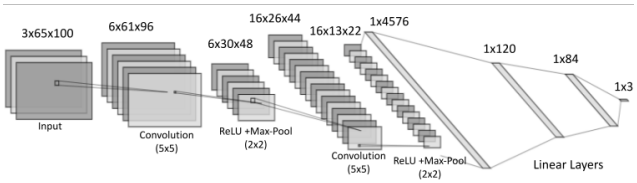

Figure 3. Set NN Model Architecture

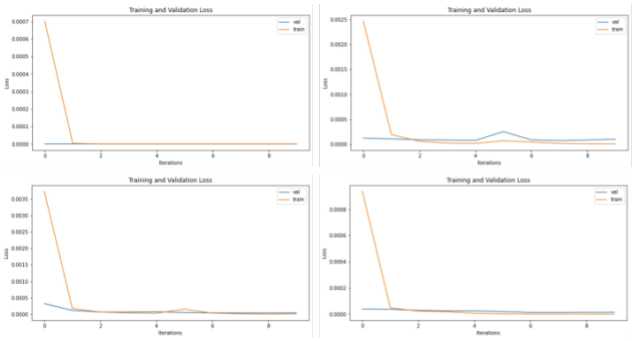

Figure 4. Set Loss Graphs (From top left to bottom right: Color, Shape, Fill, Number)

For the Set card recognition, we had 4 different CNNs, 1 for each of the 4 properties a card has. We chose to do this because classification problems become significantly more difficult to categorize accurately with a large number of possible classes. If we just used 1 CNN, we would have had $3^4 = 81$ total classes, which is hard to train properly. This way, each model could focus on it's own specific property, and they were able to learn them well. Each of the 4 CNNs had the same architecture with 2 convolutional layers and 3 fully connected layers. The first convolutional layer converts an input channel size of 3 (RGB image) to 6 channel outputs with a kernel size of 5. Then there is a ReLU activation and a max pool layer with both a kernel size and stride length of 2. The next convolutional layer takes in the 6 channels outputted from the previous one and has 16 output channels of its own with a kernel size of 5. Then there is another ReLU activation and max pool. Then we go into the 3 linear layers. The first linear layer has an input size of 4576 and output of 120, the second has an input of 120 and output of 84. After each of those layers we apply another ReLU activation. Finally, a third linear layer is applied taking in an input of 84 and outputting 3 for each of the 3 classifications each set card property can take.

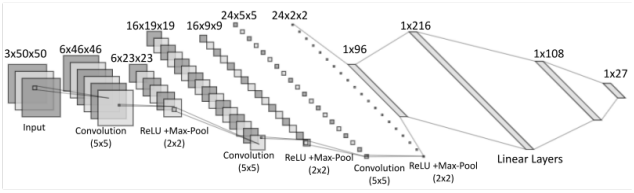

Figure 5. Scrabble NN Model Architecture

For the Scrabble tile recognition, we used just 1 CNN as we were only trying to classify by letter. This made the model a little difficult to train as we had 27 classes (26 letters and a blank space) for the model. This CNN had 3 convolutional layers. The first took in a channel size of 3 and outputted a channel size of 6, the second took in 6 and outputted 16, and the third took in 16 and outputted 24. All convolutions used a kernel size of 5. After each convolu-

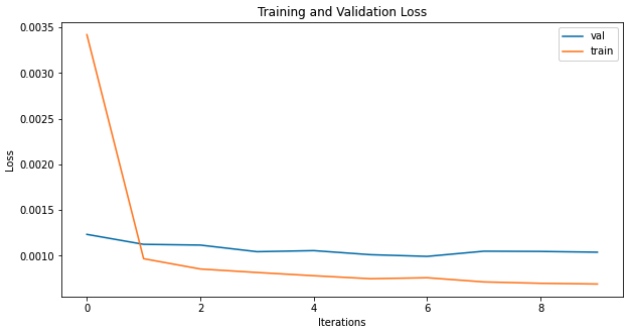

Figure 6. Scrabble Loss

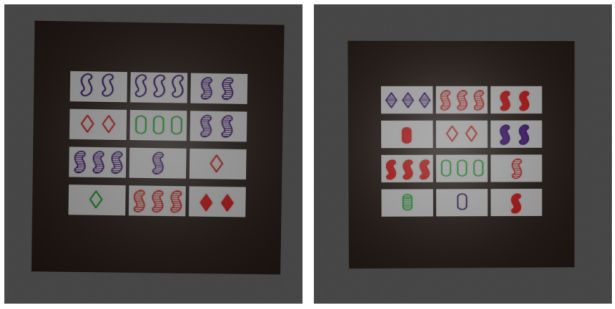

Figure 7. Example of a training (left) and testing (right) image of Set cards

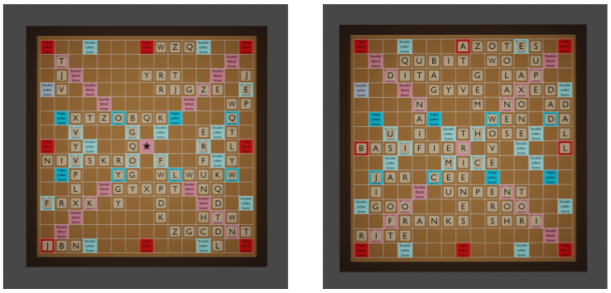

Figure 8. Example of a training (left) and testing (right) image of Scrabble board

tion there was a ReLU activation and max pool of kernel size and stride 2. Then there were 3 fully connected layers. The first had an input size of 96 and output of 216, the second took in 216 and outputted 108, and the third took in 108 and outputted 27 for the 27 possible classes. After both the first and second linear layers there was a ReLU activation as well.

3.3. Game Analysis and Move Recommendations

For Set, once we had the CNNs recognize each card, we wrote a brute force searching algorithm to find all the possible sets from the 12 cards on the table. It did so by just looping through every group of 3 cards, and if the group met the feature requirements, displayed the cards making up the set.

For Scrabble, we knew where each tile was relative to each other from the original cropping and subdivision part of our process. As such, once we classified what was in each square, we were able to reconstruct the entire Scrabble board. From there, we had a section where one can type in what letters they have in their rack (as a move requires both knowing the board state and the letters someone can play). Then, using a pre-made Scrabble solving program [11] we could give a new move recommendation.

4. Data

4.1. Game Board Dataset

For both Set and Scrabble, we constructed images of boards with pieces using Blender with varying camera angles and lighting conditions. The angles and lighting were made to mimic a person taking a picture of a game board with a phone camera and the background was set to be gray. There is randomization in the lighting arrangement and camera tilt in order to account for the random variation in taking a picture in real life. Both datasets consist of a 80/20 split for training and testing data. The images were rendered at 1024x1024 at 100% resolution.

Set: For each training image, a 4x3 grid of Set cards placed on a dark table was generated. The cards were randomly selected from the 81 potential cards with replacement. For each testing image, the cards were selected without replacement. This allowed us to have an approximately equal distribution of cards appear in the training set and the testing set matched the actual game conditions. The card textures were taken from an online image [10].

Scrabble: For each training image, Scrabble tiles were randomly placed in a 15x15 grid with around 30% of the board filled. This was chosen based on the amount of tiles available in a real Scrabble match. For each testing image, we used an online dataset of Scrabble game layouts and constructed images of the board in Blender [14]. The tiles were placed on an image of a Scrabble board which lays on a dark table [12].

4.2. Game Piece Dataset

The objective of processing the game boards is to find a mapping of the grid that a piece could be on and the image of each grid during training. Our approach for cropping the board during training is the same as identifying the board layout during testing as discussed in the previous section as this process will be the one of the steps during testing. We check each set of cropped images for validity by checking the number of cropped images and comparing the sizes of the cropped images. While we had to discard some boards due to the algorithm detecting incorrect cropping areas, the size of our datasets were large enough for our model to train

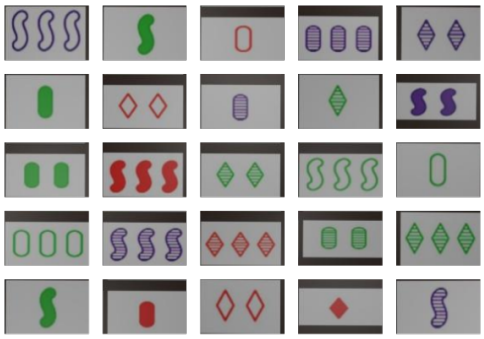

Figure 9. Example of a cropped Set cards

Figure 10. Example of cropped Scrabble tiles and spaces

well on, with about 60,000 samples for each. The dataset size also resolved problems involving piece mislabeling and misaligned cropping.

Set: Each game board was cropped into 12 images with 96.32% of the boards successfully cropped. We had a roughly equal distribution of each of the 81 classes.

Scrabble: Each game board was cropped into 225 images with 94.33% of the boards successfully cropped. Approximately two-thirds of the images belonged to the "blank" class due to the nature of the game. The remaining letter classes had relatively equal amounts of data.

5. Experiments

5.1. Experiment Set Up

For all of our convolutional neural network models, we chose the AdamW optimizer with a learning rate of 0.001 and weight decay of 0.1. Since we did not want the model to converge too quickly to a suboptimal solution, we picked a smaller learning rate. We used the Cross Entropy loss function as this is a classification problem. We used a batch size of 128 to decrease training time and trained for 10 epochs. We found that having a large batch size did not significantly impact our results. We prevented overfitting of our training data by setting the number of epochs to be low, having a large dataset, and adding a standard weight decay.

5.2. Results

The metrics we used are accuracy, precision, recall, and F1 score. Accuracy measures the percent of images that are labeled correctly. Precision measures the proportion of positive identifications that is actually correct. Recall measures the proportion of actual positives that was identified correctly. F1 score is the harmonic mean of precision and recall, which is useful when there is an uneven class distribution.

They are calculated as follows (TN = True Negative, TP = True Positive, FN = False Negative, FP = False Positive):

$$\text{Accuracy} = \frac{\text{TN} + \text{TP}}{\text{TN} + \text{TP} + \text{FN} + \text{FP}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{F1} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

The CNNs for both the Set and Scrabble classifiers are exceptionally strong. The Set classifier has an overall accuracy of 99.23% and an F1 of 99.89%, even when considering all 4 properties. Similarly, the Scrabble CNN has an accuracy of 98.70% and F1 of 97.38%. While this is slightly lower than the Set version, it's still very accurate. This is also an expected result as it's harder to classify letters given them being more complex than set pieces with 26 categories. Combined with the failure rate on cropping, we have an overall accuracy of 95.58% on Set and 93.10% accuracy on Scrabble.

Table 1. Set CNN Label Accuracy, Precision, Recall, and F1 Percentage by Property

Property	Acc	Prec	Rec	F1
Diamond	99.92	99.79	99.64	99.87
Oval	99.74	99.92	99.74	99.64
Squiggle	99.64	99.86	99.69	99.75
Red	100.00	100.00	100.00	100.00
Green	100.00	100.00	100.00	100.00
Purple	100.00	100.00	100.00	100.00
Blank	98.38	99.84	99.83	99.87
Striped	99.79	98.38	99.79	99.79
Solid	99.79	99.10	99.04	99.83
1 Piece	99.97	100.00	99.97	99.97
2 Pieces	100.00	99.97	100.00	99.97
3 Pieces	99.97	99.99	99.99	99.97
Overall Shape	99.76	99.86	99.69	99.75
Overall Color	100.00	100.00	100.00	100.00
Overall Fill	99.32	99.11	99.53	99.83
Overall Number	99.98	99.99	99.99	99.97
All Properties	**99.23**	**99.74**	**99.81**	**99.89**

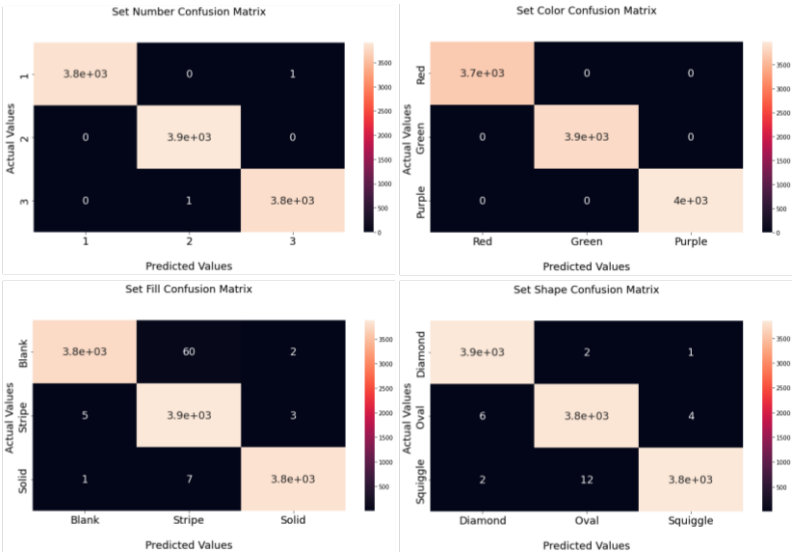

Figure 11. Confusion Matrices for Each Set CNN
(Number, Color, Fill, Shape)

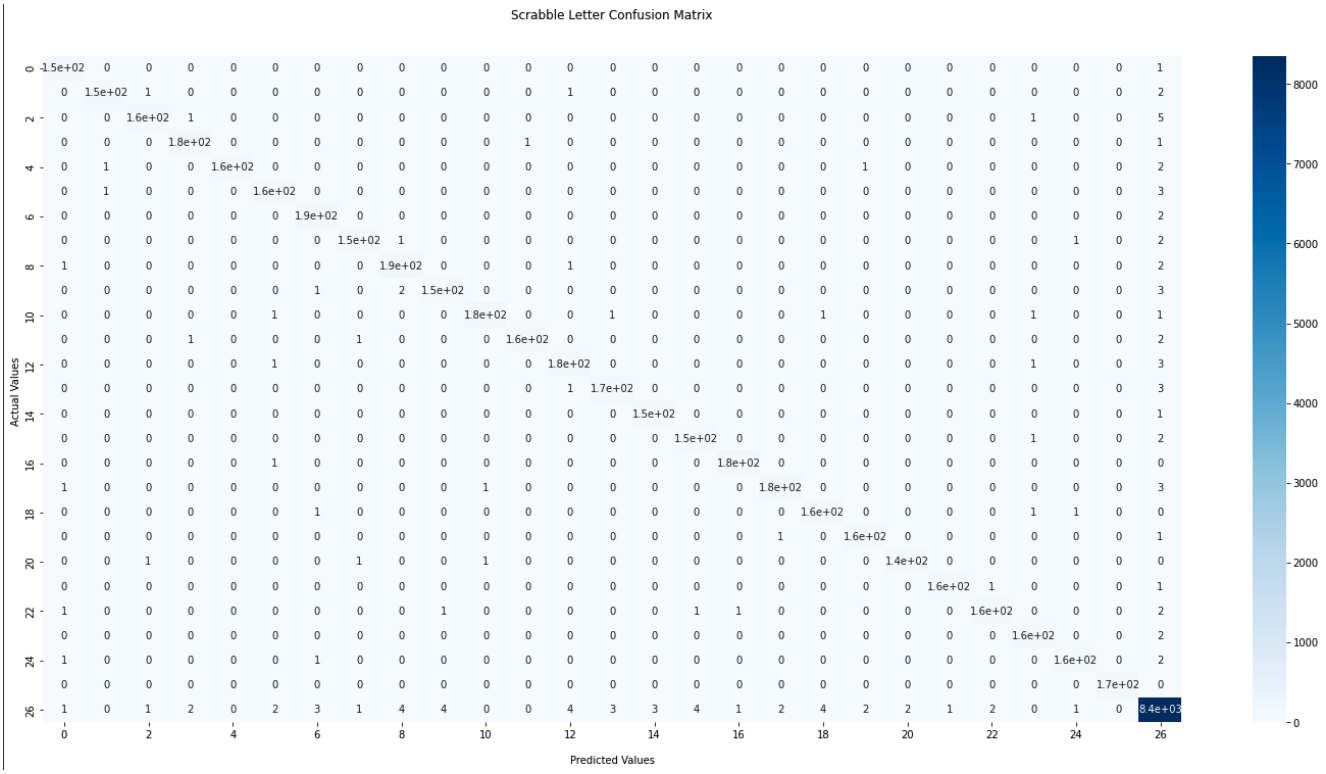

Figure 12. Confusion Matrix for Scrabble CNN
Each Letter is Associated Index Starting from A at 0
Index 26 is the Blank Tile

For the Set color CNN, we got everything 100% correct. This isn't all that surprising because color is probably the simplest property to detect given that we are feeding the CNN all 3 RGB channels as inputs. Contrastingly, while still very accurate, there were 60 miscategorized blank filled cards which the model instead thought were striped. This makes sense in being the most common error as determining the fill of the objects might be difficult due to the how blurry

some of the images are. It can be easy to mistake the stripes for just blur or fuzz on what appears to be a card with blank fills. We can also see this in the confusion matrices. Note for the confusion matrix for the Scrabble CNN model, due to the large number of images in the blank class, the colors for the other classes become muted. Nonetheless, we can see that the diagonal is darker, indicating that the model performs well. Our precision score indicates that there were some false positives. Namely, the blank class contains some images of blank squares with words on them, leading the model to incorrectly predict the class.

Table 2. Scrabble CNN Label Percentage Correct by Letter

LETTER	ACC	PREC	REC	F1
A	99.33	97.01	97.01	97.01
B	97.45	99.37	95.81	97.56
C	95.78	95.10	93.15	94.12
D	98.92	98.03	97.39	97.70
E	97.55	98.20	98.80	98.50
F	97.52	94.23	95.45	94.84
G	98.94	95.48	98.26	96.05
H	97.39	97.87	97.18	96.10
I	97.89	98.49	96.08	98.56
J	96.08	94.33	97.08	96.82
K	97.25	99.34	95.73	97.52
L	97.53	98.14	94.05	96.05
M	97.30	96.73	95.48	96.10
N	97.66	98.84	98.28	98.56
O	97.35	96.82	96.82	96.82
P	98.00	97.40	98.68	98.04
Q	99.45	98.28	98.84	98.56
R	97.36	97.45	96.23	96.84
S	98.19	98.11	98.11	98.11
T	98.74	99.43	97.74	98.58
U	97.97	97.60	97.60	97.61
V	98.73	98.76	98.76	98.76
W	96.36	97.26	97.93	97.59
X	98.79	100.0	98.01	99.00
Y	97.59	98.22	96.51	97.36
Z	100.0	97.66	97.09	97.38
BLANK	99.44	99.22	99.54	99.38
Overall	**98.70**	**97.68**	**97.10**	**97.38**

5.3. Qualitative Analysis

For the Scrabble dataset, the saliency map for letters are concentrated near the center of the image. This makes intuitive sense as the main areas that the model focuses on match the shape of the letter. For blanks, the model focuses on the edges and bottom of the image. This is likely due to the point multiplier grids that have text in the center, with all of them having the same words near the bottom (ie. letter score and word score). Additionally, the saliency map for the K and A images also look kind of a K and A themselves. This makes sense as the model would be trying to recognize the letter from their shape, and thus the saliency would outline the same general shape as the letters themselves.

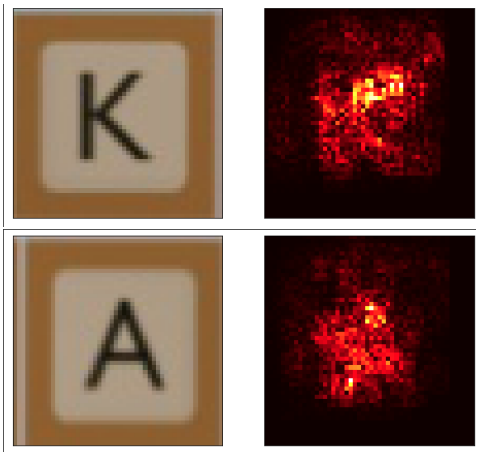

Figure 13. Saliency Map for K and A Scrabble Pieces

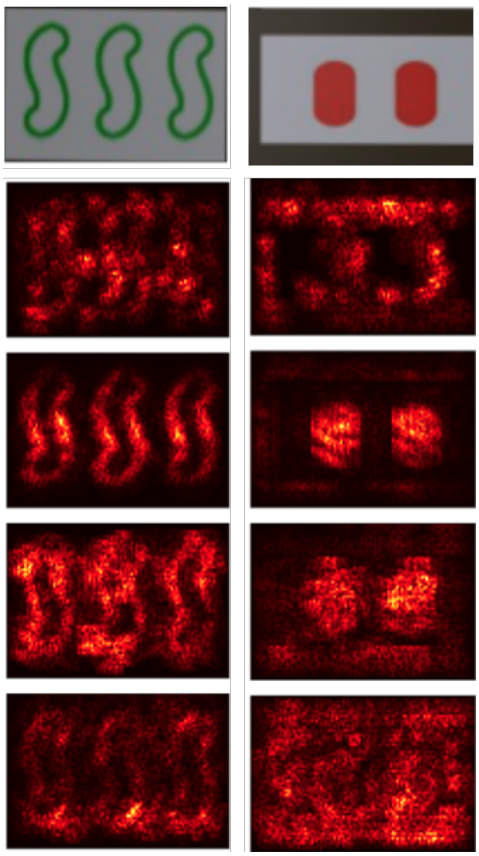

Figure 14. Saliency Map for Each Set CNN (Shape, Color, Fill, Number)

For the Set dataset, we can see that for the shape CNN, the saliency map shows a natural focus on the general shape of the object. For example, the saliency map for squiggle shows 'S' forms that are indicative of a squiggle shape. It is also concentrated around the edges since edges determine shape. For the color CNN, there is a clear focus on the part of the shape that is colored. For shapes that are not filled

in or are striped, the saliency map shows a focus on the edges and stripes, while it lights up all the way for solid cards. We see this in the saliency maps as well with the color CNN lighting up only on the edges of the squiggles because that is the only place containing green in the image, while the solid ovals are entirely lit up because they are a complete red. Similarly, the fill saliency map focuses on the colored portions of the shape. Interestingly, it does not focus solely on the inside of each shape as the information around the edge of the shape seems to be enough to indicate fill. Finally, there is no strong intuitive indication of what the CNN is focusing on for count. This is likely due to the CNN needing to understand every shape whereas the other properties could be inferred from only one shape. That is to say, for counting, an object being present or not either way is indicative of the count. Therefore, the model will not have easily understandable focus points as it isn't necessarily looking for something obvious.

5.4. Failure Modes

In cropping the boards, poor lighting conditions and angles can cause misalignments in detecting the edges of the boards. Furthermore, extraneous lines in the image (eg. the edge of the board) can also detect faulty corner points. In Scrabble, this typically means many letters with vertical lines are placed in the same column which the algorithm deems as more important than the another real line. In Set, when certain shapes are lined up, the algorithm might recognize the edges of those shapes as a line.

In general, the cropping algorithm is quite sensitive to changing perspectives, which makes generalization harder. Because of this, we can also have failures in the CNN as well. While the overall accuracies for both the Set and Scrabble classifiers are extremely high, if the cropping gives a bad image to classify the CNNs will fail. For example, a crop only giving the left-half of an O would make it indistinguishable from a Q, or the top-half of an X making it look quite similar to a Y. Additionally, blank tiles can contain words and the center blank tile contains a star which makes them difficult to categorize. In Set, while there are 81 unique cards, there are only 4 different properties with 3 configurations in each property which makes the cards very similar. Failures on Set are typically with the fill property since the fill area differs depending on the other properties.

6. Conclusion/Future Work

Overall, our models performed as expected, and that is to mean they did well. The CNN classification was extremely accurate due to the size of our datasets, and the models' ability to recognize multiple patterns can be seen through saliency maps. Most of our errors come from the cropping algorithm where the algorithm can not always distinguish between lines on the game pieces and the lines of the game board. As described in previous works, ideal lighting conditions are necessary for this task as the game pieces are very similar and the game board can be very detailed. The use of synthetic datasets allowed us to generate enough good quality data for our model to perform well.

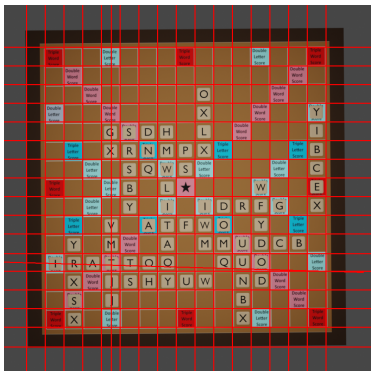

Figure 15. Example of Cropping Failure on Scrabble

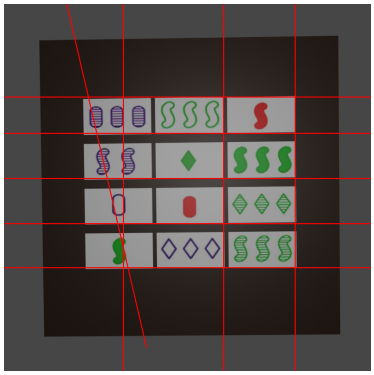

Figure 16. Example of Cropping Failure on Set

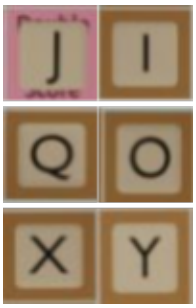

Figure 17. Pairs of Mutually-Confused Letters

In the future, we aim to explore more advanced methods of board state detection such as using heat maps or neural networks to segment the image as that is our current bottleneck. We also would like to try using bounding box techniques for object recognition. This would allow us to move beyond games with a standard layout (as Set, Scrabble, and chess are all played on grids). Working with card games such as Blackjack would be ideal paths to explore because they aren't played on with any set locations, but the standardization of playing cards would make them good candidates to still be able to detect.

Contributions

Xiluo He: On the technical side, I worked on the Blender models and scripts to generate the dataset as well as the board cropping/detection. I also worked on half of the paper (Introduction, Related Works, Dataset, and parts of Method, Experiments, and Conclusion).

Zach Witzel: For the project, I came up with the overall project architecture as well as designed, trained, and tested all the CNNs and related material. I also wrote the code to analyze their results. Additionally, I wrote the code for solving the Set games and integrating the project components to have a single workflow from image file to move output. I also worked on half the paper (Abstract, Introduction, Experiments, and parts of Methods, and Conclusion).

Public Code Used:
https://github.com/andrewleeunderwood/project_MYM
https://github.com/norvig/pytudes/blob/main/ipynb/Scrabble.ipynb
https://github.com/sunnynevarekar/pytorch-saliency-maps/blob/master/Salienc_maps_in_pytorch.ipynb

References

[1] Saurabh B. Convert a physical chessboard into a digital one. https://tech.bakkenbaeck.com/post/chessvision. Accessed: 2022-06-02. 2

[2] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986. 2

[3] João Cartucho, Samyakh Tukra, Yunpeng Li, Daniel S. Elson, and Stamatia Giannarou. Visionblender: a tool to efficiently generate computer vision datasets for robotic surgery. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 9(4):331–338, 2021. 2

[4] Maciej A. Czyzewski, Artur Laskowski, and Szymon Wasik. Chessboard and chess piece recognition with the support of neural networks, 2020. 2

[5] Ashish Dandekar, Remmy AM Zen, and Stéphane Bressan. A comparative study of synthetic dataset generation techniques. In *International Conference on Database and Expert Systems Applications*, pages 387–395. Springer, 2018. 2

[6] Jialin Ding. Chessvision: Chess board and piece recognition. https://web.stanford.edu/class/cs231a/prev_projects_2016/CS_231A_Final_Report.pdf. Accessed: 2021-12-06. 1, 2

[7] P. Dollar, Zhuowen Tu, and S. Belongie. Supervised learning of edges and object boundaries. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1964–1971, 2006. 2

[8] Christos Manettas, Nikolaos Nikolakis, and Kosmas Alexopoulos. Synthetic datasets for deep learning in computer-vision assisted tasks in manufacturing. *Procedia CIRP*, 103:237–242, 2021. 9th CIRP Global Web Conference – Sustainable, resilient, and agile manufacturing and service operations : Lessons from COVID-19. 2

[9] Soh Matas, L. M. Soh, J. Matas, and J. Kittler. Robust recognition of calibration charts. In *In IEE 6th International Conference on Image Processing and Its Applications*, pages 487–491, 1997. 2

[10] Nerdist. The card game that puzzled mathematicians for decades, 2016. [Online; accessed June 02, 2022]. 4

[11] norvig. Scrabble solver. https://github.com/norvig/pytudes/blob/main/ipynb/Scrabble.ipynb. Accessed: 2022-06-02. 4

[12] Printablee. Printable large scrabble board and tiles. https://www.printablee.com//postpic/2021/04/printable-large-scrabble-board-and-tiles.jpg. Accessed: 2022-06-02. 4

[13] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3234–3243, 2016. 2

[14] Cross Tables. Annotated games. https://www.crosstables.com/annolistself.php. Accessed: 2022-06-02. 4

[15] Andrew Underwood. Board game image recognition using neural networks. https://towardsdatascience.com/board-game-image-recognition-using-neural-networks-116fc876dafa. Accessed: 2021-12-06. 1, 2

[16] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. PATE-GAN: Generating synthetic data with differential privacy guarantees. In *International Conference on Learning Representations*, 2019. 2