GO TERRITORY DETECTION ALGORITHM

Jake Adam, Matt Moran, Calvin Weaver, Varun Maheshwari

CSSE463 Image Recognition

Email: adamj@rose-hulman.edu, moranmt@rose-hulman.edu

weavercf@rose-hulman.edu, maheshv@rose-hulman.edu

Abstract

The goal of this project is to implement a method that, given a completed game of Go, classifies board territories as regions controlled by black, regions controlled by white, or as neutral regions. This can be difficult since most empty intersections on a Go board are influenced by several surrounding stones. The relative influence of either type of stone is difficult to compute manually, while the high variability of Go games precludes preset pattern and/or template-matching algorithms. Thus, an optimal algorithm would match manual territory classification accuracies with minimal reliance on preset rules. The paper presents an integrated approach combining neural network and image processing techniques for Go territory classification.

To perform territory detection, Smart Game Format (SGF) files containing information about completed Go games were converted to grayscale PNG images to encode board states, and fed into a CNN. The output of the CNN was then filtered using connected component analysis to further improve accuracy.

Multiple CNNs were made, each correcting for flaws in previous implementations. Originally, the post-CNN filtering was done using morphological operations but was ultimately implemented using connected component analysis.

Overall the detection algorithm performed very well with a best single-game accuracy of 99.723%. The overall accuracy across the entire test set was 96.816%, and the lowest single-game accuracy was 82.825%.

1.INTRODUCTION

Go is a board game invented over 2500 years ago in China and is believed to be the oldest board game still played to this day [1]. Go has been a hot topic in artificial intelligence (AI) research. For instance, recent advancements in developing highly-skilled Go-playing AI such as Google's AlphaGo and AlphaGo Zero are considered historic achievements in AI history. In fact, AlphaGo's success highly motivated the start of this project.

Figure 1: Image of a Go game [2]

Go is primarily decided on the number of territories captured by either player. Consequently, territory classification is a crucial aspect in Go. However, the complexity and variability of Go render manual territory classification methods inefficient and template-matching based algorithms intractable.The proposed algorithm aims to maximize efficiency while delivering high accuracy territory classification.

Position evaluation on a complete 19x19 board is one of the hardest tasks in artificial intelligence. [3]Traditionally, position evaluation programs have relied on 'huge, static knowledge bases derived from the programmers' Go skills and knowledge' [3]. This leads to overly complex programs that find it difficult to generalize. Further, attempting to brute force position evaluation calculations is unfeasible, since the estimated number of possible board positions is around 10^{172} combinations [4]. Consequently, a method capable of applying learned parameters based

on large training datasets will likely be able to overcome this problem.

The proposed method utilizes an image-to-image regressional convolutional neural network, for which input and output images are grayscale images. In input images: white, black, and gray pixels represent white stones, black stones, and unplayed intersections respectively. In output images: white, black, and gray pixels as well correspond to white, black and neutral territories respectively.

2. LITERATURE REVIEW

When first attempting to tackle this problem, the team wanted to see what kind of projects and work have been done with machine learning for Go games. The first resource found providing knowledge in this area was “AI techniques for the game of Go” by E. van der Werf [5]. Van der Werf investigated many different algorithms for scoring and analyzing the territory of Go games as a means for developing an algorithm to interpret the state of a game in real-time. The basic territory classification algorithms that the paper investigated are: explicit control, direct control, distance-based control, influence-based control, Bouzy’s method and multi-layer perceptron neural networks. While this is a great resource, the paper focuses primarily on evaluating the winning player of a given Go game and the performance impact of each algorithm. This project focuses more on determining the territory classification of an in-progress Go game, rather than the final score or the speed of a given algorithm. The paper also fails to talk about the relationship between the total number of turns taken in a Go game and the accuracy of the classifier algorithm as the game progresses.

Another resource we found during research was "Optical Game Position Recognition in the Board Game of Go" [6]. This paper was about performing detection on a live-stream or video of a Go game in order to find the final score. Although the author did not get it working in real-time, they managed to use Hough Transform to detect the different edges of a Go board and its stones. They first transformed an image of a Go game into grayscale and then used Hough Transform to detect the edges of the board and the grid lines of the board. The average color value of 5x5 pixel grids throughout the image is taken in order to determine if the grid space on the board is covered by a white stone, black stone, or a brown neutral space. Similarly, we converted our images to grayscale images so we could find the different colors of stones, with gray representing a neutral space instead of brown. We performed a similar color average based on the gradient that the CNN outputted and used those averages to create a clearer, solid color image to more accurately measure the accuracy of the classification.

A few other resources were also looked at that, in the end, did not contribute to the classification process. “Evaluation of Strings in Computer Go Using Articulation Points Check and Seki Judgment” by H. Park & K. Kang [7] is one such resource that discusses how to handle life and death situations within a game of Go. Life and death situations are the hardest part of classifying territories but the methodology used is too unique to be applied to the classification used in this project.

3.PROCESS

3.1 Gathering the Data Set

Before territories can be classified by a CNN, a data set had to be put together to act as the input to the CNN. The data comes from a website called Dragon Go Server (DGS) [8] and is stored in SGF files. A small program was written to loop over the completed game listings from DGS [8] and to download the SGF file associated with each game. As a starting dataset, 200 games were downloaded. All 200 games have been completed within the past three years, and the players in each game rank 5000 and above. Later in the project, this dataset was expanded to include 1000 games, 107 of which were removed due to having an invalid board size. This yielded a dataset of 893 unique games. Later, during the storing and saving of the images, all input and output images were flipped three different times to give us a final dataset of 3572 images. The test image dataset we use, is around the last 10% of the total dataset, which is 358 images. A sample SGF file has been provided in Figure 2 below. The SGF files contain a full record of the entire game and the final territory classification of the game, using letters A-S to denote the coordinates of each piece. The SGF files also contain information such as player ranks and game duration, however this was not included as part of the dataset for this project.

```
(
;FF[4]GM[1]
AP[DGS:1.20.3]
PC[Dragon Go Server: https://www.dragongoserver.net/]
DT[2019-12-24,2020-01-18]
GN[hightrees-Pmcguire-1301258-20200118]
SO[https://www.dragongoserver.net/game.php?gid=1301258]
PB[Patrick McGuire (Pmcguire)]
PW[chrise (hightrees)]
BR[2k]
WR[3k]
XM[223]
GC[Game ID: 1301258
Game Type: GO (1:1)
Rated: Y

White Start Rating: 4k (+33%) - Elo rating 1733
Black Start Rating: 2k (-17%) - Elo rating 1883
White End Rating: 3k (+13%) - Elo rating 1812
Black End Rating: 2k (+20%) - Elo rating 1920]
OT[30 days with 1 day extra per move]
RU[Japanese]
SZ[19]
KM[6.5]
RE[W+1.5]
;B[pd]
;W[dp]
;B[pq]
;W[ec]
;B[pk]
;W[mc]
```

Figure 2: Sample SGF File

3.2 Converting SGF to PNGs

The trainingImageGen.m function was written to parse SGF files into images that can be interpreted by our algorithm. The function is capable of extracting two images from each SGF: a completed game before territory classification and a completed game after both players have agreed on the final territory of the game. The latter represents the goal of our algorithm and can be used as test comparison data. The completed game before classification represents the input to our algorithm. White pixels represent white stones, black pixels represent black stones, and gray pixels represent spaces with no stones and neutral territory. The image is 19x19 pixels and in grayscale. An example of what the trainingImageGen.m function outputs is shown in Figure 3. trainingImageGen.m was run on all SGF files to create PNG files for both the input and output SGF files. The input images show stone positions at the end of a Go game, shown in the left image of Figure 3. Output images show the final territories of a Go game which is what our expected output will be when compared to the CNN classification. An example of an output image is shown in the right image of Figure 3.

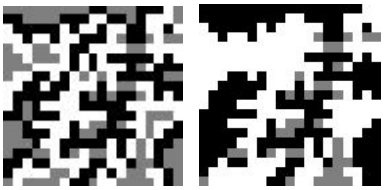

Figure 3: Stone positions at the end of a Go game (Left), the final territories of the Go game (Right)

3.3 Storing and Saving the PNGs

After the SGF files are parsed into PNGs, we needed to create a script that would quickly store each input and output PNG that was created into their own respective folders. The script storing_images.m creates two 3D arrays, one for the input images and one for the output images. These arrays are initialized by using an array of zeros as placeholders for the size. How the input array was created is shown below in Equation 1.

```
inputs = zeros(19,19,numel(srcFilesTrain));  [1]
```

Since the board size is 19x19, we need to store each pixel in a 2D array but since we need to test our classification on thousands of images, the third dimension was added to store each game ID so each image has an ID number with it. Storing_images.m uses a for-loop that calls the trainingImageGen.m function for each SGF file and stores the input and output image data in their respective arrays. Once all SGF files have been converted to PNGs, there are two more for-loops that are run. One for-loop is for storing the input images into a folder called input_images and the other is storing output_images into a folder called output_images. These for-loops also contain an if statement checking to see if the dimensions of the game are different than 19 x 19. Some games of Go are conducted on smaller boards when a less experienced player is playing. We want to ensure that all of our data images are the same size for a simpler classification process.

After each image is checked if it is a valid size, the image is flipped vertically, horizontally, and both to provide four different images for one game file. This increases our dataset size by four times to have more precise classification.

3.4 CNN Selection Overview

In developing a method based on deep learning techniques, two types of convolutional neural networks (CNNs) were initially considered: semantic segmentation CNNs, and image-to-image regressional CNNs. Semantic segmentation networks classify each pixel of an image as an instance of one of multiple preset classes. On the other hand, image-to-image regressors develop feature sets containing information relevant to the input image, with learnable weights determining the influence of a

specific feature on the image output. In both architectures, deconvolutional layers decode extracted information into output images.

While semantic segmentation networks present an intuitive and clear approach to territory classification, the team recognized that such networks would find it challenging to extract the global context of a full Go board, especially because such information is difficult to retain as network depth increases. Additionally, hue, texture and shape features that are commonly used by semantic segmentation networks are negligibly relevant for the purposes of Go territory classification. In fact, the primary determinant of territory classes is the relative influence of surrounding stones on a specific area. Furthermore, the team's research indicated that semantic segmentation networks had primarily been applied in instances wherein data had specific patterns. For instance, in one implementation, semantic segmentation was applied to classify features of a road. Throughout the network's training data, objects such as buses, cars, and lights had unique shapes, hues, and in some cases, spatial locations. This indicated to the team that semantic segmentation was better suited to applications wherein objects possessed low feature variability. However, the complexity and variability of Go board states would preclude any possibility of a low variability dataset.

On the other hand, past applications of image-to-image regressors in denoising, relighting and colorization indicated to the group that regressional structures are better suited to the variability of Go board states. Specifically, encoder-decoder structures were selected because of their ability to capture positional and strategic information effectively, as explained in the sections below.

3.5 CNN Structure Attempt 1

Our first approach to classifying territories was to pass the input game state data through a convolutional / deconvolutional neural network. The goal was to have the network generate a full-size grayscale output image with the pixel intensities marking the classes that each board space belongs to. To accomplish this, we designed a simple linear network architecture that combined several consecutive convolutional layers with several consecutive "deconvolutional" layers. Deconvolutional layers (also known as "transposed convolutional layers" in MATLAB) essentially perform the reverse operation of convolutional layers in that they upscale an image based on a set of learned filters (rather than downscaling the image). This feature was leveraged to create a network capable of downscaling an input image, deriving meaningful data about a Go game state and then upscaling the data to produce a full-resolution classified image. This approach was adapted from a MATLAB manual that described a process through which a series of deconvolutional layers was used to produce a denoised image [9]. Our first network architecture can be seen in Figure 4.

Initially, our network failed to train using any of the conventional training algorithms supplied by MATLAB. After extensive testing, we realized that this was due to an excessively low learning rate. Through trial and error, we found the optimal learning rate to be 0.01 and the best training algorithm to be MATLAB's Adaptive Moment Estimation (ADAM).

Once the training algorithm had properly converged, we were able to evaluate the network's accuracy. After applying the rounding and connected component filters to the output images (described in sections 3.8 and 3.9), the convolutional / deconvolutional network had an overall accuracy of 92.997% on our 358 test image dataset. This accuracy is represented by the number of correctly labeled pixels (i.e. a single instance of assigning the correct pixel class) versus the total number of classified pixels across the entire dataset. See Appendix Figure A.1 for the complete test breakdown of the convolutional / deconvolutional network.

While the accuracy was impressive, the convolutional / deconvolutional network had a number of weaknesses. Most notably, the network had a relatively low confidence in the center of territories. Generally this didn't cause any issues, however there were a few observable regions where the network struggled so much that it misclassified huge patches of otherwise obvious territory. This was likely due to the extensive depth of the territorial regions in these specific images. These classification issues can be observed in the difference images in Appendix Figure A.2.

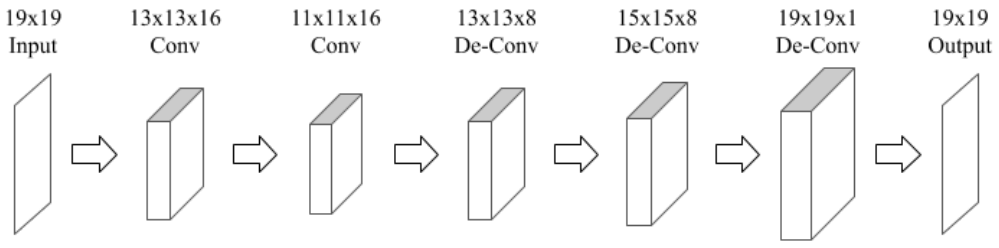

Figure 4: (Approach #1) Fully Convolutional/Deconvolutional Neural Network Architecture

3.6 CNN Structure Attempt 2

Our second approach to classifying territories was to pass the input game state data through a neural network in a manner similar to our attempt in the previous section, however for this attempt we used a drastically different network architecture. This approach was inspired by MATLAB's ability to dynamically mix and match various neural network layer types, and in particular the program's ability to link fully connected layers together with convolutional and deconvolutional layers. In an attempt to explore the efficacy of different network architectures, we trained a network with a single fully connected layer sandwiched between a high-resolution convolutional and deconvolutional layer (see Figure 5 below). The idea was that this network would do a better job of analyzing the strategic implications of individual stones on the Go board, and make territory classification decisions with a greater knowledge depth than the previous architecture.

In practice, the network failed to achieve an accuracy that was higher than our former architecture. It reached a peak accuracy of only 83.421% on our full test image dataset. See Appendix Figure A.3 for the full test results of this network. There are a number of possible causes for this high error rate, but the most likely is perhaps the lack of bandwidth through which the network could propagate the exact pixel boundaries of each territory on the board. As part of our tests, we experimented with fully connected layers of a much higher resolution (i.e. 300+ nodes), however this proved problematic to train and we were unable to get networks of this size to converge properly on our laptops.

Ultimately the extra bandwidth was unnecessary, however, because the strengths of the smaller fully connected network was found to compliment those of the previous structure. The fully connected architecture was observed to be extremely good at classifying territory centers with a high confidence. Cases such as these extended even to areas where the convolutional / deconvolutional network suffered, such as extremely deep territories. The test images shown in Appendix Figure A.4 exemplify this strength. Similarly, the weaknesses of the fully connected network structure were found to mirror those of the previous structure; the fully connected network was much weaker around territory edges.

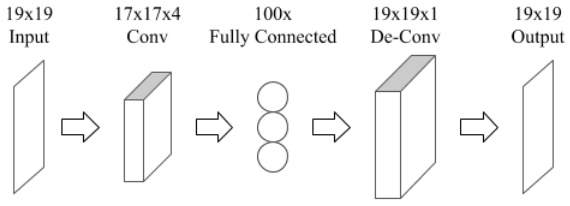

Figure 5: (Approach #2) Fully Connected Layer-Centric Neural Network Architecture

3.7 Final CNN Structure Attempt

Our final approach to classifying territories was to pass the input game state data through a neural network in a manner similar to our first two approaches, however this attempt was designed to combine the network architectures from the two previous attempts so as to leverage the strengths of each. This final network was designed with two branching paths: a path with smaller convolutional and deconvolutional filters separated by a single fully connected layer, and a path with several consecutive large convolutional and deconvolutional layers. The idea was that the fully connected layer path would handle the larger "strategic decisions" of the Go game's territories (building on our second attempt's ability to consistently handle territory centers correctly). Similarly, we hoped that the larger convolutional / deconvolutional path would precisely manage local information (building on our first attempt's ability to handle territory boundaries well). The resulting network structure can be seen in Figure 6 below. After a few trials, we concluded that the best way to merge the output data from the two paths was to concatenate both paths with a high 3rd dimensional output and then merge the 3rd dimension with a convolutional layer. This approach was chosen over simply adding two 19x19x1 outputs from each path in an attempt to preserve as much information flowing out of each path as possible. In practice, this yielded a small accuracy increase on our test dataset of about 0.5%.

The combined network architecture performed astoundingly well on our test image dataset, yielding a 96.171% per-pixel accuracy. Compared to the other two network attempts, this network produced the most confident overall values after its initial training run, and generally performed much better overall in all areas of the image. The network struggled a little with classifying neutral territories, however these were determined to be unpredictable (due to the random nature of a Go game's progression) and the error was deemed acceptable for the purposes of this project.

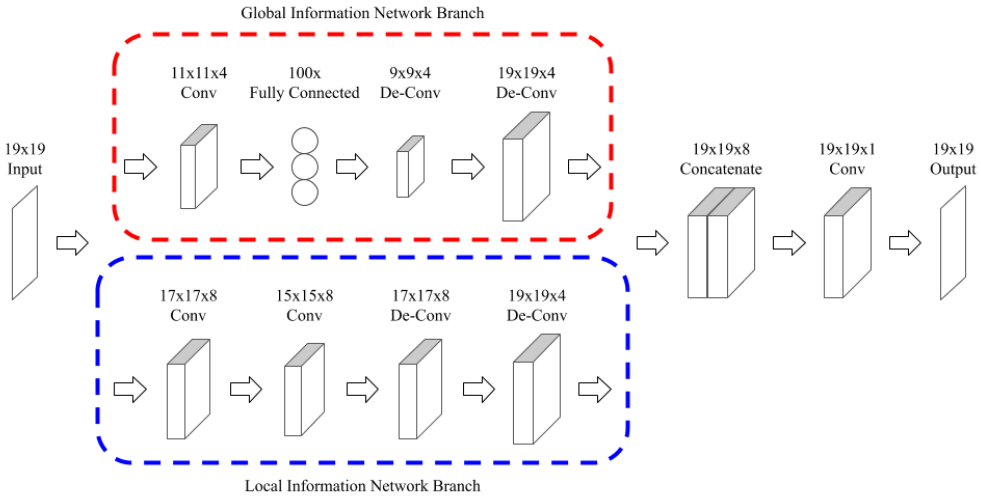

Figure 6: (Approach #3) Combined Branching Neural Network Architecture

After identifying the most promising network structure, we used one of Rose-Hulman's CSSE department servers to train the network for 40 epochs with a mini-batch size of 80 images. The network was trained using a single Nvidia Tesla K80 (one of the two available cards) and took 10 minutes and 54 seconds to complete all 1400 iterations. The training graph for this network can be observed in Figure 7. This additional training yielded a network that was able to obtain a 96.816% accuracy on our test image dataset. The detailed test results for this iteration of the network can be observed in Appendix Figure A.5. Similarly, the network's classification of several test images can be seen in Appendix Figure A.6.

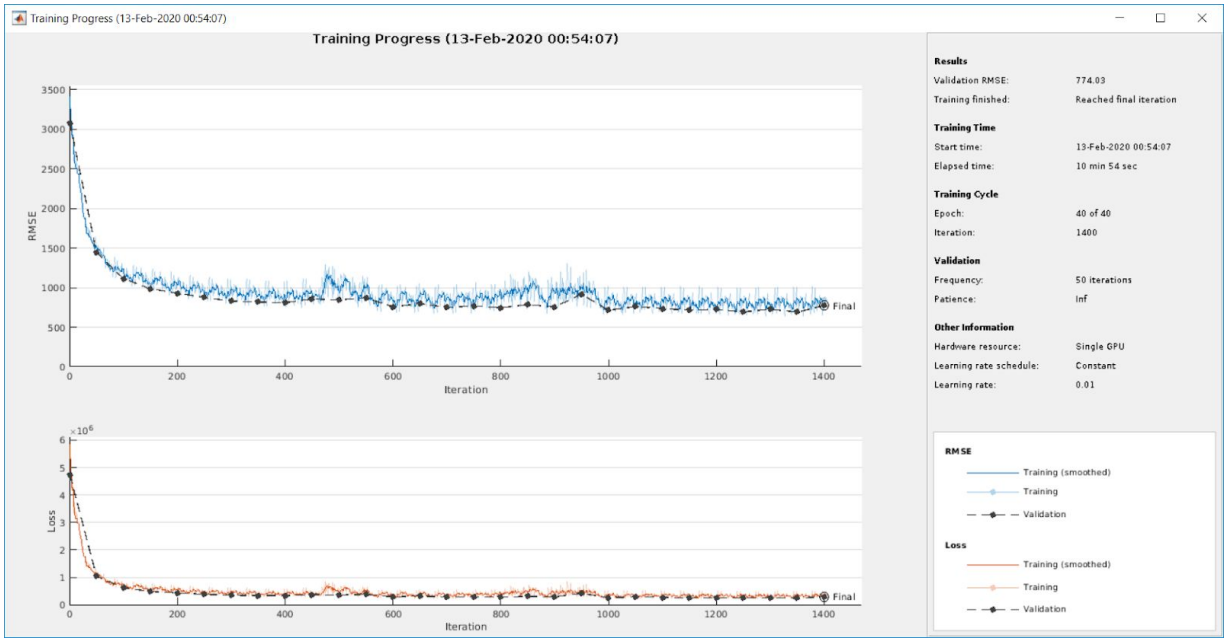

Figure 7: Combined Branching Neural Network Architecture Training Graph

3.8 Post CNN Filtering using Rounding

In order to properly judge the accuracy of our network, we needed to implement a filter to match the network's output to the three valid Go board classes (black, white or neutral). After classification, the output returns an image with gradients shown below in Figure 8. The gradients prevent us from measuring accuracy because not every single pixel is a solid black, gray, or white pixel so we lack a value to compare to for accuracy.

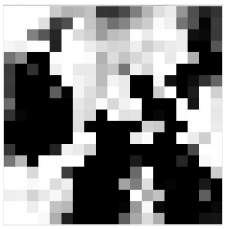

Figure 8: Final CNN Output Before Rounding

For this we implemented a simple rounding filter; rounding is what allows accuracy to be computed since the pixels will have definite values to be worked with. Currently the output is represented in grayscale with values from 0-255. If the CNN was fully confident the pixel should be black, it was given a value of 0 and 255 for white stones. To perform rounding the 0 to 255 scale was changed to a 0 to 1 scale. Any pixels above 0.5 are considered white and any below 0.5 are considered black. Unfortunately this does not account for the gray pixels. Therefore a rounding margin of 0.03 was added to pad the values, to determine a black or white pixel better along with a margin for gray pixels. The rounding margin's threshold, 0.5, had a value of 0.03 added and subtracted to it. If a pixel had a value of 0.53 or greater, it would be set to completely white. Completely black was any pixel less than or equal to 0.47. Anything within the middle of that range was set to a gray pixel which represents a neutral position. The result of rounding is shown below in Figure 9. Performing rounding on the CNN output allows the image to more closely resemble the expected output.

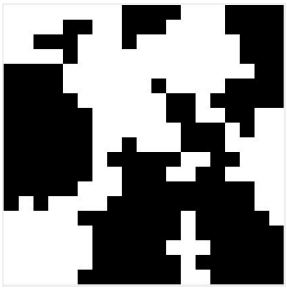

Figure 9: Rounded CNN output

3.9 Post CNN Filtering using Connected Components

With rounding complete, the image is now ready to be filtered again. The second form of filtering involves using connected component analysis to improve the algorithms accuracy. Equation 2 shows the MATLAB command to perform connected component analysis.

```
cc = bwlabel(rounded, 4);          [2]
```

The command finds groups of similar components. In this case, the white pixels that are connected are assigned a number to group them. If the size of a region was less than 8 pixels, the group would be absorbed by the pixels surrounding it. An example output of the filter is shown in Figure 10 below. If the region size was set too small, not enough of the misclassified territories would be absorbed by the correct territory. If this value was set too high, correctly classified territories would be wrongly absorbed. The optimization process showing the region size and associated accuracies are shown in Table 1 on the right. A region size of 8 was chosen due to it giving the highest accuracy per image. A region size of 9 and 10 were also considered, but they resulted in almost identical accuracies. To avoid omitting a correctly classified territory, the region size should be kept as small as possible. Keeping the region size at 8 avoids overfitting for specific images and allows the algorithm to work for more general cases. The connected component analysis was performed on both white and black pixels. When comparing the filtered output to the rounded output, the filtered output is obviously much closer to the expected output.

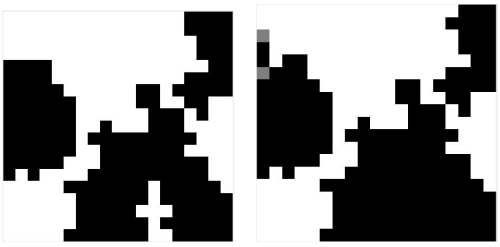

Figure 10: Filtering Output (Left), Expected Output (Right)

Region Size	Accuracy (%)
1	96.51
2	96.63
3	96.67
4	96.69
5	96.72
6	96.77
7	96.80
8	96.82
9	96.82
10	96.83

Table 1: Accuracy Associated with the Connected Component Region Size

4. LESSONS LEARNED

Multiple attempts to develop a functional neural network allowed the group to understand factors including architecture, layer selection, and layer specifications that must be accounted for in designing a neural network. For instance, a non-branching architecture limited the bandwidth of extracted information, with feature sets weighting global or local information disproportionately depending on filter sizes and number. Additionally, the effects of network depth were also better understood only after extensive trial and error. Overly deep networks are highly complex, entail longer training times, and may find it difficult to converge as relevant information extracted in earlier layers becomes increasingly ‘diluted’ in later layers. On the other hand, shallow networks allow for faster debugging, and faster training times. However, an exceedingly shallow network may find it difficult to generalize as only superficial or partial information is extracted. Thus, a balanced architecture must be found in order to be able to extract and weight global and local information optimally. In the team’s case, this balance was realized through the branching

architecture that captured positional and strategic information in two separate sub-networks before combining their outputs.

The group was also able to develop a strong understanding of layer specification selection. Extensive utilization of in-built MATLAB tools such as *analyzeNetwork* and *Deep Network Designer App* helped the group develop an intuition of the resizing effects of convolutional, deconvolutional, and fully connected layers. Further, experimentation with layer types including batch-normalization, dropout, pooling, and un-pooling layers allowed the group to understand the impact of these layer types. For instance, in the initial design stages prior to data augmentation, dropout layers were used to reduce overfitting. Batch-normalization layers were also experimented with, although the relatively shallow architecture meant that gains in terms of speed, performance and stability were negligible. Lastly, in initial attempts-- especially in those incorporating non-branching architectures-- the group attempted to implement pooling and unpooling layers to account for global positional information. However, these attempts were quickly discarded because of the network's failure to converge under these conditions.

Connected component analysis was not the only post-CNN filtering attempt.. Morphological operations were also performed on each image to try and eliminate the misclassified territories. Unfortunately, performing erosion on a 19x19 image is not very effective. Any operation performed made details and edges of correctly classified territories disappear. At first, this was thought to stem from the image only being 19x19. To try and fix this before erosion happened the image was resized at a scale of 100 times more resulting in a 1900x1900 image. Again, unfortunately, when erosion was performed the same result happened. It was at this point the group decided to abandon the idea of using morphological operations. The eroded image compared to the rounded image is displayed below in Figure 11.

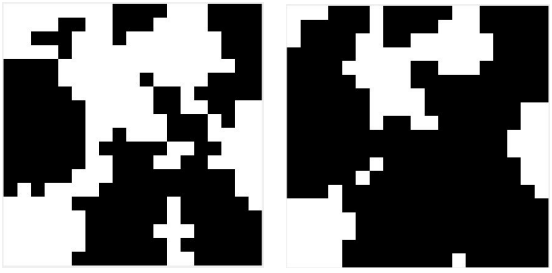

Figure 11: Rounded Image (Left), Eroded Image (Right)

The results of erosion when performed on the entire test set are shown in Figure 12. The result window includes a confusion matrix of predicted colors and their actual colors along with the percentages. In the top right grid section of the window is the best case results with the worst case grid underneath. The bottom left grid just shows the overall accuracy on the entire test dataset along with any perfect scores that were found. For the best and worst case grids, the leftmost image is the output of the classification, the top right image is the input, and the bottom right image is the expected output. Comparing the input image to the classification output shows which neutral spaces were captured. The classification output is compared to the expected output to get an accuracy value. As can be seen, erosion made the accuracy worse than the rounded output of the CNN. Running erosion on the entire test set confirmed a new filtering process was required.

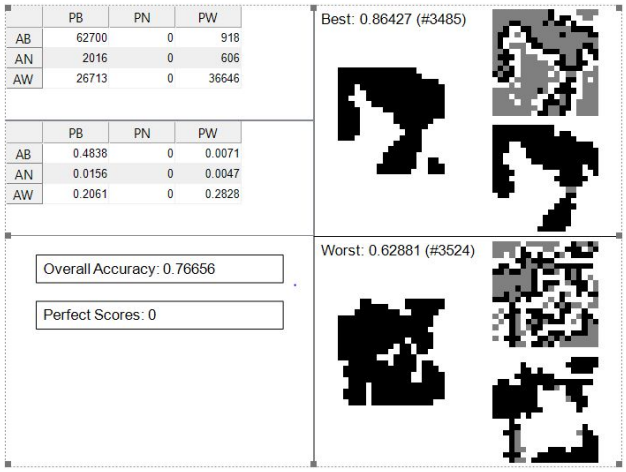

Figure 12: Result Window with Eroded Images

5. RESULTS

The initial attempts i.e. deconvolutional / convolutional network and fully connected network produced results with disparate accuracies in central zones of territories, which are typically more complex and strategic zones.

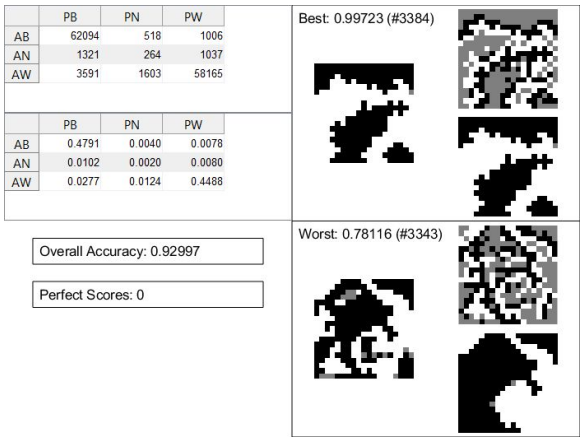

Figure 13: Convolutional/Deconvolutional Network Results

The purely convolutional/deconvolutional network failed to accurately predict territories in central zones of territories because of the absence of a fully connected layer to combine information and create strategic dependencies. As seen in the results included in Appendix Figure A.2, the network output indicated low confidence in central regions of territories. Indeed, the network's worst case output i.e. lowest accuracy test output of 78.116%, evidences this hypothesis. For the worst case test, regions in crowded central zones were entirely misclassified, with a black territory in the lower right zone predicted to be surrounded by white territory--an impossible outcome given the size of the black territory seen in this case. This can be seen in Figure 13 above. The deconvolutional / convolutional network performance is summarized in the table and graphics also included in Appendix Figure A.1 along with Figure 13. This network has an overall accuracy of 92.997%.

The fully connected architecture network remedied the purely deconvolutional / convolutional network's errors in central zones of territories, but failed to accurately predict border regions of territories. This is likely because the fully connected layer was able to encode positional dependencies but could not retain information for the more isolated, higher resolution border regions. In other words, the fully connected layer was simply not large enough. However, an increase in the layer's size would have detrimentally impacted network complexity and training times. This aspect of the network's performance is clear to see in its test outputs, visualized in Appendix Figure A.4.

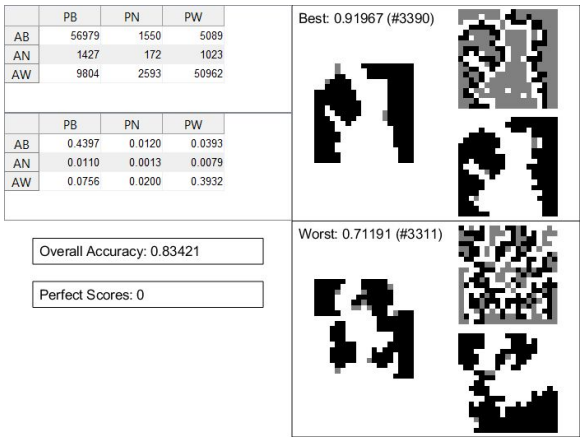

Figure 14: Fully Connected Network Results

The network's overall performance is summarized in Figure 14 above or Appendix Figure A.3. This network has an overall accuracy of 83.421%.

Contrasting the deconvolutional / convolutional network and fully connected network outputs, another interesting aspect of network behavior becomes evident. The fully connected network captures general territory shapes well, while the deconvolutional / convolutional network captures localized zones better. Clearly, the fully connected network must have extracted global information better than the deconvolutional / convolutional network. This aspect of the networks' behavior provided the basis for the combinatory branched structure.

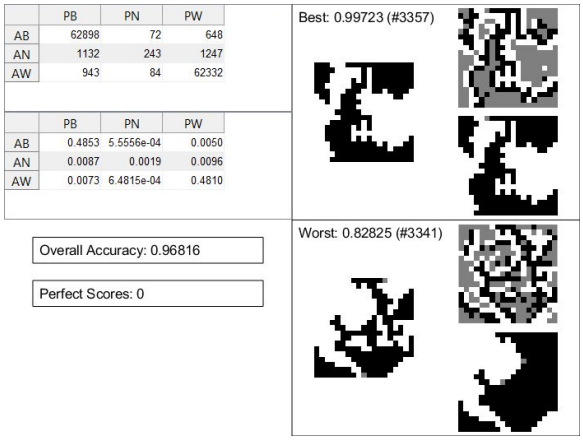

Figure 15: Combined Branching Network Results

The branched network, combining the above structures, performs best. This is because of its ability to extract and combine global and localized board state information. Arguably, the network's primary flaw is in its ability to predict neutral territories, a feature of its performance that is expanded on in later sections. The network's overall performance has been summarized in Figure 15 and Appendix A.5. This network's overall accuracy is 96.812%.

6. DISCUSSION

This algorithm has many strengths, the first and most obvious is its ability to obtain very high accuracies when classifying territories. Another strength is the algorithm's ability to not overfit the data. Even the worst case scenarios had accuracies at least in the 80 percent range which is still relatively good for a machine learning algorithm.

The biggest problem the algorithm had was classifying neutral territory and can be seen within the best case scenarios. The algorithm was never able to get 100% accuracy due to neutral territory misclassification. This is highlighted in Figure 13. These problems came from rounding which will be explained more in section 7.2.

Figure 13: Result Window Displaying Misclassified Neutral Territory

Another item to note is in all three CNN structures, the best and worst case result was a different image each time. These images also are not flipped versions of each other. The reason this has occurred is most likely due to the fact that each structure has a strength and it finds the best and worst image for that strength. The first structure has a strong confidence with the center of territories whereas that is the second structure's weakness and the third structure combines the two. With these different strengths and weaknesses, different best and worst images will appear.

7. CONCLUSION AND FUTURE WORK

7.1 Conclusion

After reviewing the results of this project, we conclude that our custom CNN can quite accurately detect territories in a game of Go. The convolutional / deconvolutional network produced a high accuracy of 92.997% overall. From there we found that the fully connected network only provided us with an overall accuracy of 83.421% but returned greater confidence values on territory centers which the convolutional / deconvolutional did not. This led to making the decision of combining both of these structures to provide the best accuracy possible for classifying Go territories. We proceeded to take this the next step further and add post-classification filtering which omitted misclassified territories of certain sizes. In the end, the combined branching network resulted in a final overall accuracy of 96.816%. This network combined with the post-CNN filtering produced a best case accuracy of 99.723%. We believe that it

would be possible to get an accuracy of 100% if we trained the network better on how to score neutral spaces. This topic is talked about more in-depth in 7.2 Future Work below.

7.2 Future Work

There are many things that could improve this project if the team was able to work on it longer. One thing that would be really cool to implement would be a score calculator to tell the winner of each game, the total number of stones under the players control, and highlights the territory boundaries. This would be a nice feature to add since one of the goals of this project is to create something the Go community could use to make scorekeeping more efficient. Already, the algorithm produces high accuracy, making it as a whole, very reliable. This addition would be the only thing limiting a pair of players from using this algorithm to determine a winner at the end of a game.

If given a much longer time to work on this project, for example a couple of months to a year, the goal would be to get 100% accuracy when classifying territories. In the best seen scenario, the classifier missed one gray (neutral) spot. The neutral positions are interesting because the classifier was not always sure which territory to classify them as or if they should remain a neutral position. One thing that could be changed is the rounding margin used. The only problem is that some neutral spaces were sometimes given more confidence than actual, correctly classified spaces. Simply changing the rounding margin will not produce the 'obvious' result making this a more complex problem. A possible solution would be to perform some analysis on the input images to find possible patterns that could be used specifically for neutral space classification. Again this would be difficult since most spaces in a game are neutral before the CNN classifies them.

8. REFERENCES

[1] "Go (game)", *En.wikipedia.org*, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Go_(game). [Accessed: 18- Feb- 2020].

[2] *Image of a Go game*. 2020. [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/thumb/2/2a/FloorGoban.JPG/1024px-FloorGoban.JPG. [Accessed: 21- Feb- 2020].

[3] E. C. V. D. Werf, H. J. V. D. Herik, and J. W. Uiterwijk, "Learning to score final positions in the game of Go," *Theoretical Computer Science*, vol. 349, no. 2, pp. 168–183, 2005.

[4] "Go and mathematics", *En.wikipedia.org*, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Go_and_mathematics. [Accessed: 18- Feb- 2020].

[5] E. van der Werf, "AI techniques for the game of Go" , *psu.edu*, 2005. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.81.1906&rep=rep1&type=pdf. [Accessed: 26- Jan- 2020].

[6] T. Musil, "Optical Game Position Recognition in the Board Game of Go", *Tomasm.cz*, 2014. [Online]. Available: http://tomasm.cz/imago_files/go_image_recognition.pdf. [Accessed: 26- Jan- 2020].

[7] H. Park & K. Kang, "Evaluation of Strings in Computer Go Using Articulation Points Check and Seki Judgment" AI 2005: Advances in Artificial Intelligence, 2005. [Online]. Available at: https://books.google.com/books?id=-o8HCAAAQBAJ&ppis=_e&lpg=PA198&ots=pzJGtb1kl1&dq=machine%20learning%20life%20and%20death%20situations%20go%20game&pg=PA204#v=onepage&q=machine%20learning%20life%20and%20death%20situations%20go%20game&f=false. [Accessed 26 - Jan - 2020].

[8] "DGS - Dragon Go Server", *Dragongoserver.net*, 2020. [Online]. Available: https://www.dragongoserver.net/. [Accessed: 18- Feb- 2020].

[9] "Prepare Datastore for Image-to-Image Regression", *mathworks.com*, 2020. [Online]. Available: https://www.mathworks.com/help/deeplearning/examples/image-to-image-regression-using-deep-learning.html. [Accessed 21- Feb- 2020]

9. APPENDIX

9.1 Convolutional / Deconvolutional Neural Network Performance

	PB	PN	PW
AB	62094	518	1006
AN	1321	264	1037
AW	3591	1603	58165

	PB	PN	PW
AB	0.4791	0.0040	0.0078
AN	0.0102	0.0020	0.0080
AW	0.0277	0.0124	0.4488

Overall Accuracy: 0.92997

Perfect Scores: 0

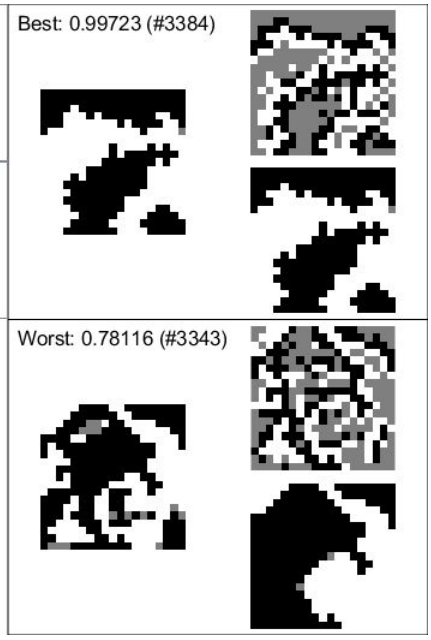

Figure A.1: Convolutional/Deconvolutional Network Results (Complete Test Dataset)
(Using Network Output Rounding and Region Filtering)

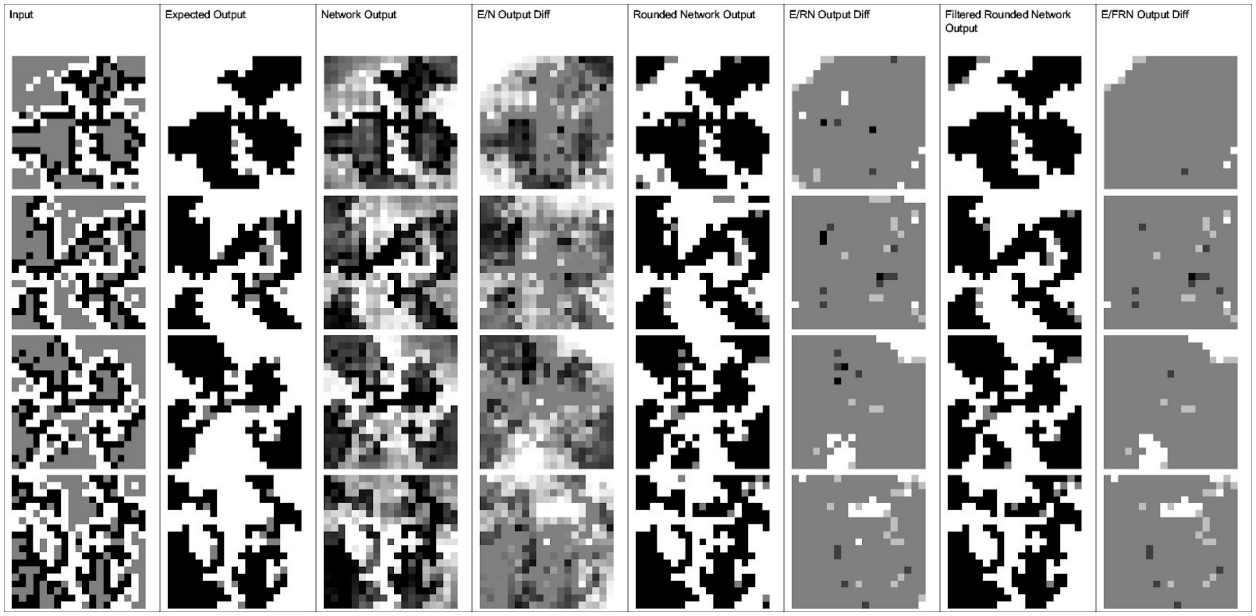

Figure A.2: Convolutional / Deconvolutional Network Image Classification Breakdown
(From Left to Right) System Input (1), Expected Output (2), Convolutional/Deconvolutional Network Output (3),
Expected and Network Output Difference (4), Network Output (Rounded to Match Classes) (5),
Expected and Rounded Network Output Difference (6), Network Output (Rounded and Region-Size Filtered) (7),
Expected and Filtered Rounded Network Output Difference (8)

9.2 Fully Connected Neural Network Performance

	PB	PN	PW
AB	56979	1550	5089
AN	1427	172	1023
AW	9804	2593	50962

	PB	PN	PW
AB	0.4397	0.0120	0.0393
AN	0.0110	0.0013	0.0079
AW	0.0756	0.0200	0.3932

Overall Accuracy: 0.83421

Perfect Scores: 0

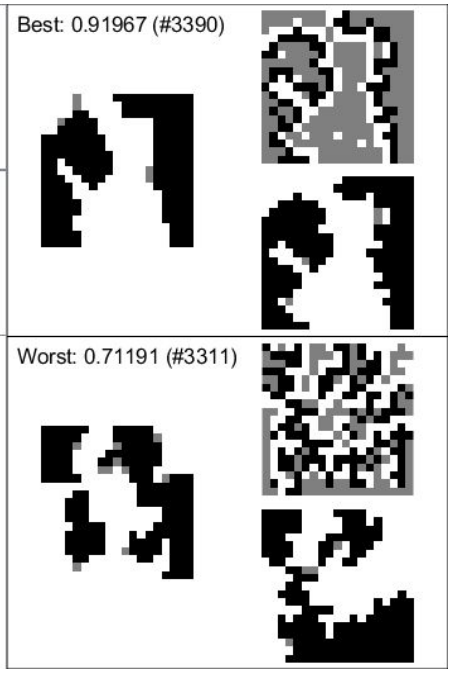

Figure A.3: Fully Connected Network Results (Complete Test Dataset)
(Using Network Output Rounding and Region Filtering)

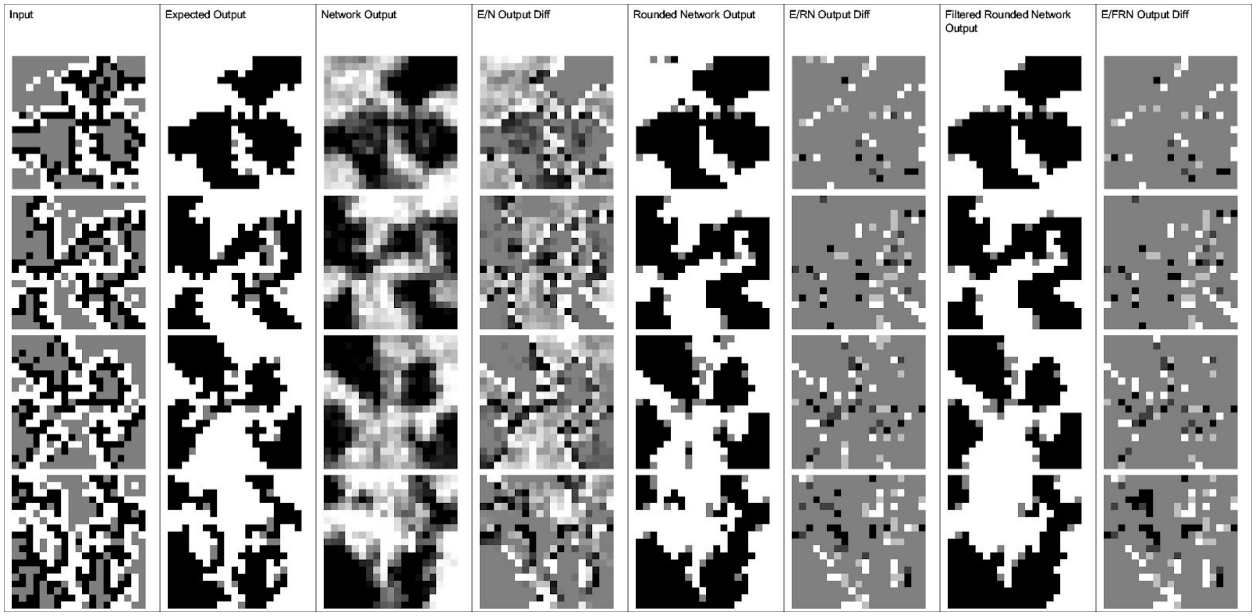

Figure A.4: Fully Connected Network Image Classification Breakdown
(From Left to Right) System Input (1), Expected Output (2), Fully Connected Network Output (3),
Expected and Network Output Difference (4), Network Output (Rounded to Match Classes) (5),
Expected and Rounded Network Output Difference (6), Network Output (Rounded and Region-Size Filtered) (7),
Expected and Filtered Rounded Network Output Difference (8)

9.3 Combined Branching Neural Network Performance

	PB	PN	PW
AB	62898	72	648
AN	1132	243	1247
AW	943	84	62332

	PB	PN	PW
AB	0.4853	5.5556e-04	0.0050
AN	0.0087	0.0019	0.0096
AW	0.0073	6.4815e-04	0.4810

Overall Accuracy: 0.96816

Perfect Scores: 0

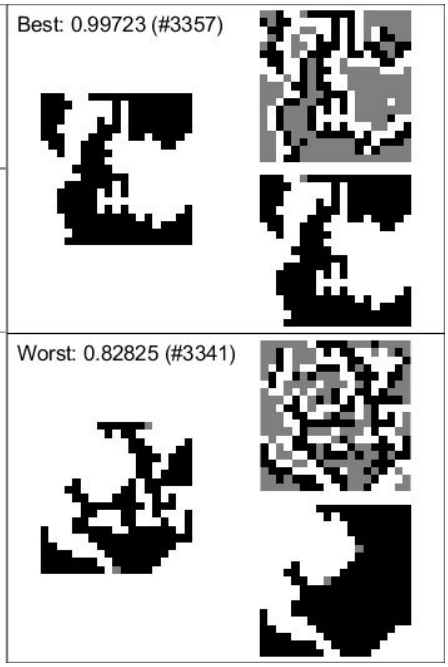

Figure A.5: Combined Branching Network Results (Complete Test Dataset)
(Using Network Output Rounding and Region Filtering)

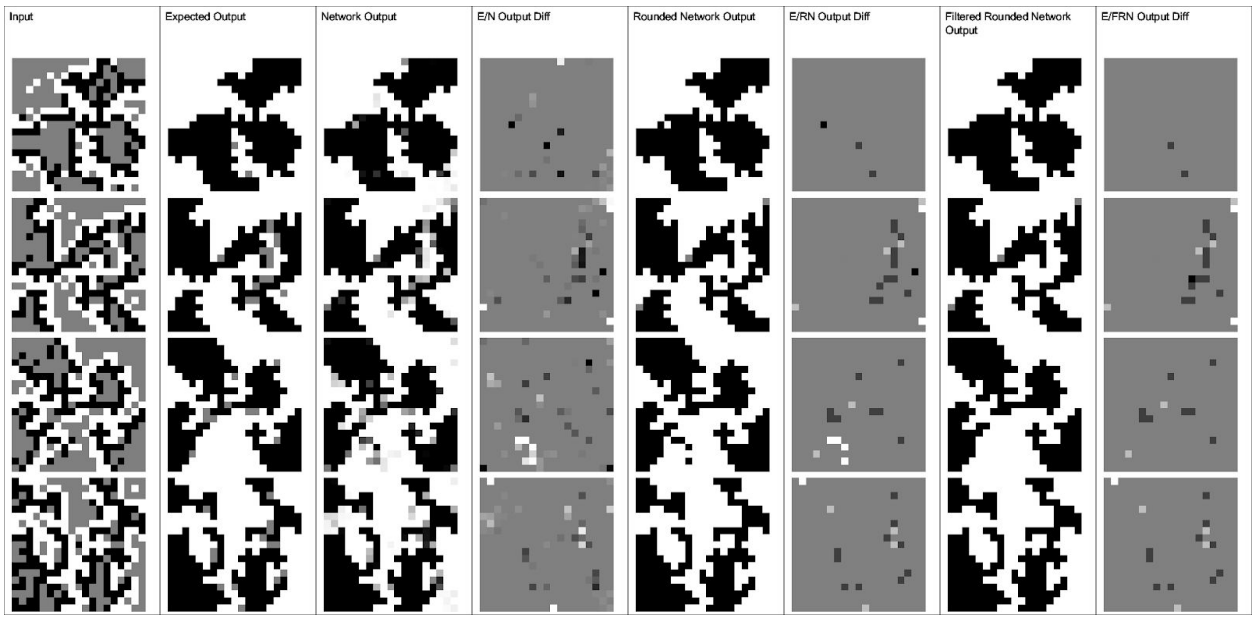

Figure A.6: Combined Branching Network Image Classification Breakdown
(From Left to Right) System Input (1), Expected Output (2), Branching Network Output (3),
Expected and Network Output Difference (4), Network Output (Rounded to Match Classes) (5),
Expected and Rounded Network Output Difference (6), Network Output (Rounded and Region-Size Filtered) (7),
Expected and Filtered Rounded Network Output Difference (8)