Card Detection – Poker Group

Jake Milanowski, William Thesken, Brendan Boewe, Jared Brown, Chirag Sirigere

CSSE463 Image Recognition

20 August 2021

Abstract

Image recognition shows numerous applications to detection of card ranks and suits. Our proposed model attempts to identify cards in natural environments through image processing and classification. To accomplish this, our model uses a region-based convolutional neural network (R-CNN) to crop important card identification regions from natural images containing cards. Then, one of three different classifiers identify the rank and suit of the card found. Our results proved promising in the identification of cards, with the highest of the three classification models producing an accuracy of 77%. However, full classification from natural images to single cards proved difficult as R-CNN crop outputs ranged in quality. As such, our report lends validity to the use of numerous models (SVM, CNN, and transfer) in card classification but warrants future research in the cropping of cards in natural environments to facilitate classification.

1 Introduction

The prevalence of playing cards in society is a continuity throughout the breadth of history. Their applicability to numerous games has leant them value spanning cultural context and language. However, this wide range of card applications – and the even wider range of artistic contexts in which cards are created – has led to many competing card companies and designs.

Given this wide scope of cultural impact, there is sufficient background and reasoning to explore integration of playing cards with image recognition. Card recognition has the potential to span numerous applications, including uses in betting games such as poker and blackjack, identification of cards in venues such as casinos, or consumer novelty in entertainment applications. Additionally, numerous algorithms already hold the capability to perform well in similar games but are limited to digital applications. Image recognition of physical playing cards could help connect digital game-playing artificial intelligences with physical-world contexts.

Playing cards present a complex challenge in the field of image recognition. Each card combines features to culminate in 52 classes of recognition inherent in each deck. Further, differing art between decks and brands make each card a potentially unique recognition task. Current research often uses singular or a small variety of decks for training and testing. This approach, while offering promising results on more common deck types, leaves out the breadth of art present in real world deck usage. To bridge the gap between current models and the real world, more generalizable classifiers are required.

Our solution attempts to rectify this issue by providing an image recognition model that is designed with multiple deck styles in mind. To accomplish this, the model makes three crucial considerations: suit, rank, and training dataset quality. To identify suit and rank, three different models were constructed, a support vector machine (SVM) classifier, a custom convolutional neural network (CNN), and a transfer network model based on the image classifier AlexNet. The training dataset was the final component to aid in classification generalizability. Cards were selected from a variety of different contexts and styles. They contain backgrounds which range from none to natural with varying dimensions and scales.

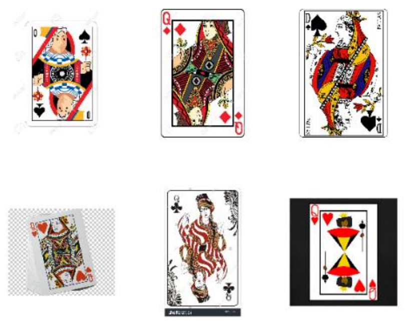


Figure 1: *Queens found in the training dataset with varying features and styles.*

This breadth of training data ensures that the final model accounts for the desired range in inputs and outputs. For later models, modifications were made to the data to increase accuracy. Data was fed into a region-based convolutional neural network to crop images to the corners of cards. This helped decrease the impact of backgrounds and superfluous card designs.

2 Literature Review

Playing card detection has seen numerous attempts throughout the relevant literature. Among these, continuities of procedure have arisen which provide well recognized solutions to similar problems. However, some continuities have found great influence in our proposed solution more than others.

Zheng and Green [1] propose *Playing Card Recognition Using Rotational Invariant Template Matching*. Their recognition model matched images with predesignated templates. While this methodology proved accurate for their uses, template matching was determined to limit generalization too greatly for our model and as such was not used. However, the model also used image cropping and rotation to ensure that only relevant image regions were used for classification. This served as valuable inspiration for the use of our model's R-CNN for cropping. Further, it inspired investigation of image rotation before the topic was scrapped for feasibility reasons. Chen and Shung [2] also propose another method of image standardization, which uses the Hotelling Transform. However, this was not used in our model.

Pimentel and Bernardino [3] in *A Comparison of Methods for Detection and Recognition of Playing Cards* propose two methods of image classification after image standardization. One proposed method uses a probability map of each rank and suit. This was matched similarly in our model during the SVM classification where the highest confidence was used to classify each rank and suit.

Castillo, Goeing, and Westell [4] directly inspired our SVM research by noting the validity of multiclass support vector machine models. This was later found to be inferior in our model to numerous single class SVM models.

3 Process

Several different methods were completed for creating a playing card classification system. Each method had its own unique challenges and difficulties, as well as different setups.

Acquiring correct labeling for each image proved to be difficult with our dataset. To obtain correct labels, both training and testing datasets were organized in 2 separate ways, one for each classifier. The first was organized using 13 folders separated by rank. The same was done for suits. Both testing and training data was organized in this way. This allowed us to create labels for the cards based on the folder names as opposed to manually inspecting and renaming each card image file.

3.1 Region-Based Convolutional Neural Network

Three R-CNNs were attempted on the original dataset after preprocessing the training dataset to be used in the SVMs and CNN transfer learning to determine the card suit and rank. The preprocessing phase consisted of labeling each corner of the six hundred sixty-four cards as a 'corner' as shown in Figure 2. The images are then resized to 227 x 227 pixels for the CNN to handle.

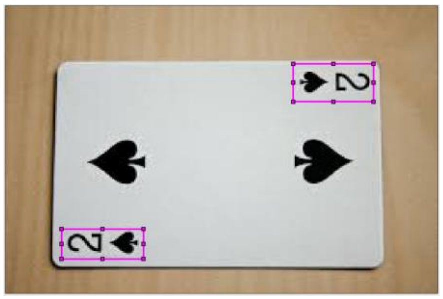


***Figure 2**: Ground truth corner labels of the card.*

Once the preprocessing phase was complete, the dataset was run through an R-CNN with SqueezeNet, a Fast R-CNN with ResNet50, and a Faster R-CNN with ResNet50.

For the R-CNN transfer learning with SqueezeNet, the last two layers were replaced with a new fully connected layer and a new classification layer. A typical transfer learning network is shown in Figure 3.

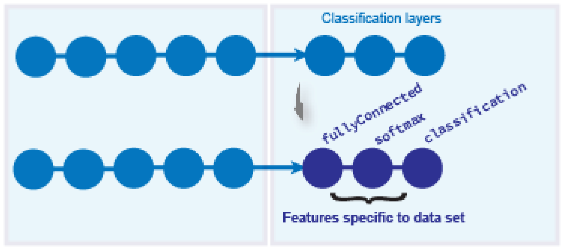


***Figure 3**: Typical transfer learning network model [5].*

In the Fast R-CNN implementation of ResNet50, the activation_40_relu was chosen as the feature extraction layer and an ROI max pooling layer was inserted after the feature extraction layer along with an ROI input layer, as shown in Figure 4 below. The last three modified layers from ResNet50 transfer learning were also included with a new fully connected layer and a box regression layer [5]. Figure 5 shows a typical Fast R-CNN network modification.

The way the Fast R-CNN works is that the algorithm attempts to tighten its prediction of the proposed region of interest, the bounding boxes from the Fast R-CNN's prediction, to accurately classify the corners of the cards using the intersection over union metric shown in Equation 1.

$$\frac{area(A \cap B)}{area(A \cup B)} \qquad \text{(Equation 1)}$$

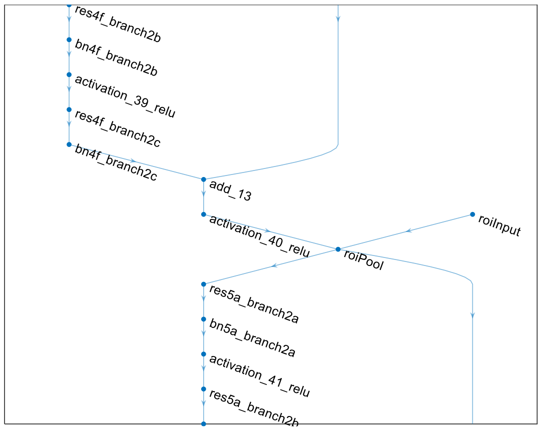


***Figure 4**: The ResNet50 network showing the roiPool and roiInput after the activation_40_relu layer.*

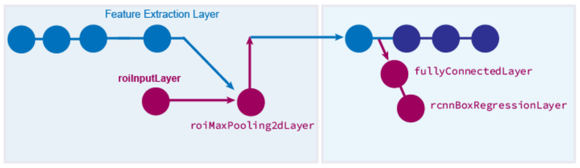


***Figure 5**: Typical Fast R-CNN layer graph [5].*

The third approach was the Faster R-CNN with ResNet50. This network graph has a region proposal network inserted after the feature extraction layer and before the ROI max pooling layer. A typical Faster R-CNN layer graph is shown in Figure 6.

The way the Faster R-CNN works is that the algorithm attempts to predict the bounding boxes on its own in the region proposal network using anchor boxes [5]. Instead of having a sliding window to find the ground truth bounding boxes, a certain number of static bounding boxes are added onto the image to locate the ground truth bounding boxes [6]. An anchor box size of twenty was used for the Faster R-CNN because of its consistent high IoU metric and Figure 6 shows one example of the anchor box to IoU (intersection over union) plot.

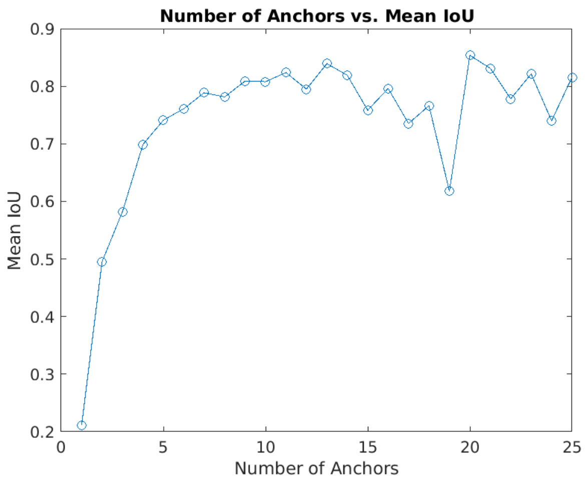


Figure 6: *Mean IoU to size of anchor box.*

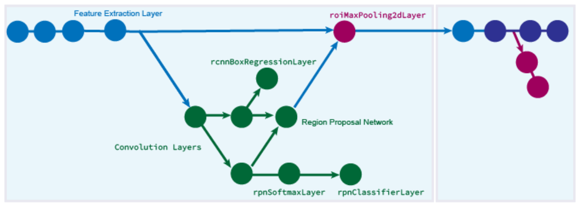


Figure 7: *Typical Faster R-CNN layer graph [5].*

The transfer learning R-CNN with SqueezeNet resulted in a misclassification of the corner bounding boxes, shown below in Figure 7, but had a better accuracy at predicting the corner labels than the Fast and Faster R-CNNs and in less time. The Faster R-CNN implementation took five hours to fully train with ten epochs and the classification of the corner was corrupted by the automatic ROI proposal network. Not only did the Fast R-CNN and the Faster R-CNN not detect the bounding boxes in the correct locations but they proposed bounding boxes that were miniscule in size compared to the ground truth bounding boxes.

Even though the proposed corners by the SqueezeNet algorithm were riddled with false positives like the banner lettering below in the testing dataset, it was still possible to determine the suit and rank of the entire card from the other images

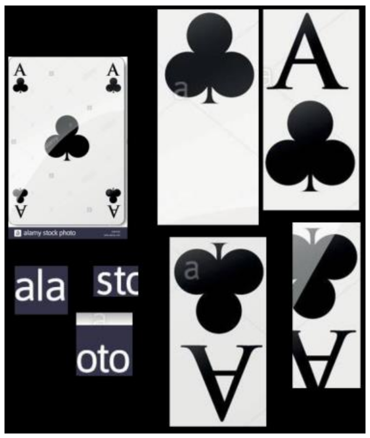


Figure 8: *Top left is the original image, and the other images are proposed corners by the SqueezeNet transfer learning model. Misclassification of the bottom banner in the image as a corner of the card is shown in the bottom left.*

To predict the testing dataset rank and suit, the cropped ground truth images were used on the following algorithms to train and test the algorithms on.

3.2 SVM

The SVM method for playing card classification used feature extraction from a CNN, and then passed the features into the SVM. AlexNet was chosen as the CNN for feature extraction because it is a built-in CNN in MATLAB and because of previous experience using it. Features were extracted from the last fully connected layer ('fc7') in AlexNet.

A multi-class SVM was originally used to classify the data. Images and corresponding labels for all 52 playing cards were passed into AlexNet. Features were extracted and then passed into MATLAB's **fitcecoc()** multi-class SVM function for training. After the multi-class SVM was trained, it was used to classify the data. It was found that using a multi-class SVM to distinguish between 52 different classes achieved a very low accuracy (only around 8%), so the classification process was changed to improve the model.

The classification process was broken apart such that only a binary SVM (MATLAB's **fitcsvm()** function) was needed. To achieve this, the model was broken apart to classify the suit and rank separately. First, the models tested for each suit versus non-suit, then each rank versus non-rank. This resulted in 17 SVM's run back-to-back on the test images (4 SVM's for each suit, and 13 SVM's for each rank, using the process shown in Figure 9). A score was recorded for each rank and suit SVM. The suit and rank SVMs with the greatest scores classified the suit and rank for the card.

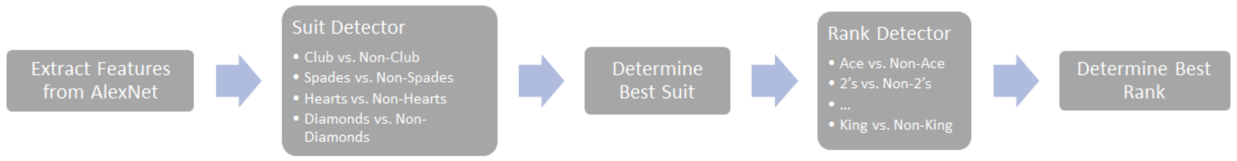


Figure 9*: SVM Flowchart*

The training data for each SVM was divided differently for the suit and rank classifiers to make sure each SVM was properly trained. For the suit classifiers, the training set consisted of the entirety of the available suit images and an equal amount of non-suit images, divided evenly amongst each class. The training images passed into the rank detector were completed the same way, except the non-rank images were broken apart into an equal number of images from all 12 other ranks.

Each SVM used two hyperparameters – the box constraint and kernel scale. These were optimized using MATLAB's **fitcsvm()** 'bayesopt' optimizer, as well as 'auto' 'OptimizeHyperparameters' options. The best estimated values for Box Constraint and Kernel Scale were then passed into the fitting function.

3.3 CNN

Similar to how the previous method used to separate suit and rank classifiers, creation of two custom CNNs was another possible solution to classify the suit and rank of a card. Initially the first CNNs used the entire image of a card, however, to increase performance we would use cropped images that would display only the rank and suit of the card. Those cropped images were then downscaled. The combination of using a crop and downscaling to a 50 x 40 image prevented out of memory errors while training, allowed for faster training of the model, and a higher accuracy over each set. Using a CNN allows for filters in the convolutional layers of the model to collect features of the image automatically. The features get passed to the fully connected layers of the model before outputting a softmax classification.

In the final and most successful iteration of the custom suit model, The Gambler Suit V4, the architecture was based off a CNN designed for determining whether or not a patient qualifies for LASICK eye surgery and was modified to fit the needs of suit classification. The suit classifier takes in a 50x40 image, consists of 36 layers when including dropout, and outputs a classification of the card's suit (Clubs, Diamonds, Hearts, Spades). Further details of the model are shown in the appendix.

The architecture used for the rank classifier was similarly designed and consisted of 24 layers, and outputs a rank for the card. Further details can be found in the appendix.

3.4 Transfer Learning

Transfer learning reuses the feature extraction layers of a prebuilt CNN but replaces the final few layers that classify images. This saves an immense amount of time on training and can result in higher accuracy as prebuilt networks are often trained on millions of images. For this project, AlexNet was used as it was easy to work with and trained quickly. AlexNet is a CNN consisting of 25 layers, the last 3 of which were replaced for this project. Replacing the final 3 layers allowed AlexNet to classify cards instead of the output it was originally trained for. If the features detected in AlexNet's original training data were similar enough to the relevant features distinguishing playing cards, a high accuracy could be obtained.

The final 3 layers were replaced with a fully connected layer, a softmax layer, and a classification layer. In addition, the initial learning rates were set to a low value, but a higher learning rate was used for the final fully connected layer. Using these learning rates allowed for quicker training as less epochs were needed. Essentially, AlexNet did all the feature extraction with its prebuilt layers, then classified with the newly added layers.

Much like the other 2 methods, transfer learning was split into separate classifiers; one for ranks and one for suits. This increased accuracy as it dropped the number of classes from 52 down to 13 and 4 for the rank and suit classifiers respectively. The resulting outputs from each classifier were then concatenated to create the final classification for each card.

AlexNet requires a 227x227x3 input to run, meaning all images in the dataset had to be rescaled to a size of 227x227 with 3 colorbands (RGB). During the resizing, any grayscale images were also converted to RGB color space by duplicating their values across each color band. This was done for all training and testing data.

4 Experimental Setup and Results

4.1 SVM

Using the original card data, the multi-class SVM did poorly, only resulting in around an 8% accuracy. The original data was also passed into the refined SVM method (using sequential SVM's), and performed much better, although still poorly. In Table 1 below, the accuracy for detecting each suit versus non-suit is shown. Overall, the suit detector does very well. In Table 2 below, the accuracy for detecting each rank versus non-rank is shown. Most ranks perform well. However, the overall rank accuracy is brought down by ranks with a lower performance. For the original data, the SVM method resulted in a 23.47% overall accuracy.

Table 1: *Accuracy of Suit Classification Using SVM Method*

Suits	Accuracy (%)
Clubs	86.73
Spades	80.61
Hearts	78.57
Diamonds	79.59
Overall Suits	70.41

Table 2: *Accuracy of Rank Classification Using SVM Method*

Ranks	Accuracy (%)
Aces	77.55
2's	81.63
3's	69.39
4's	72.45
5's	65.31
6's	90.82
7's	66.33
8's	70.41
9's	60.20
10's	82.65
J's	71.43
Q's	79.59
K's	71.43
Overall Ranks	30.61

It was found that the SVM accuracy much worse than the Transfer Learning and custom CNN, so the cropped images were not ever classified using the SVM.

4.2 CNN

A decision was made to first develop a custom model for classifying the suit of the card. However, the validation set results from the initial suit classifiers were not satisfactory and ranged from ~21% - ~27% depending on the model and the hyperparameters. The original data set contained 539 training images and 100 testing images while the cropped image set contained roughly 1400 training images, 100 of which were used for testing. 20% of the training images for each set were used for validation. Using cropped images increased the training and test performance. Training was much faster since it had smaller input images to train on, and accuracy for each set increased as well. To further increase performance of the suit classifier, images and input shape of the model were downscaled. This too increased training speed and improved accuracy.

Finally, the hyperparameters used for training the most successful model were an Adam optimizer with a learning rate of 0.0001, 400 epochs, and loss calculated using sparse categorical cross-entropy. Batch size for training was not used but could have improved training speed for the suit model. Two models were tested, one with the lowest validation loss and the other from the last epoch. Both achieved an accuracy of 90% on the test set, so the one with the lowest validation loss was selected as our final suit model.

Once the suit model was complete, we then created a rank model. To achieve success for this part of the problem, many tweaks to the hyperparameters were needed. For this model, batches were set to 1000. The other hyperparameters used for training were an Adam optimizer with learning rate of 0.0001, an epoch count of 600, and sparse categorical cross-entropy used for loss. Two models were once again tested, one with the lowest validation loss and one from the last epoch. The test accuracy of the lowest validation loss model was lower than the test accuracy of the final epoch model. The lowest validation loss model had a test accuracy of 64% while the final epoch model surpassed it with a test accuracy of 68%.

With the suit and rank models, there is a suit test accuracy of 90% and rank test accuracy of 68% (Table 3), while the rank accuracy is lower than the suit model.

Table 3: *Rank And Suit Test Accuracy Of Custom CNNs*

Model	Accuracy (%)	Loss
Suit	90	1.1068
Rank	68	4.7762

The custom rank CNN still has room for improvement as an accuracy of at least 90% is desired. While the customs CNNs were more accurate than their respective SVMs, the transfer learning models outperformed both.

4.3 Transfer Learning

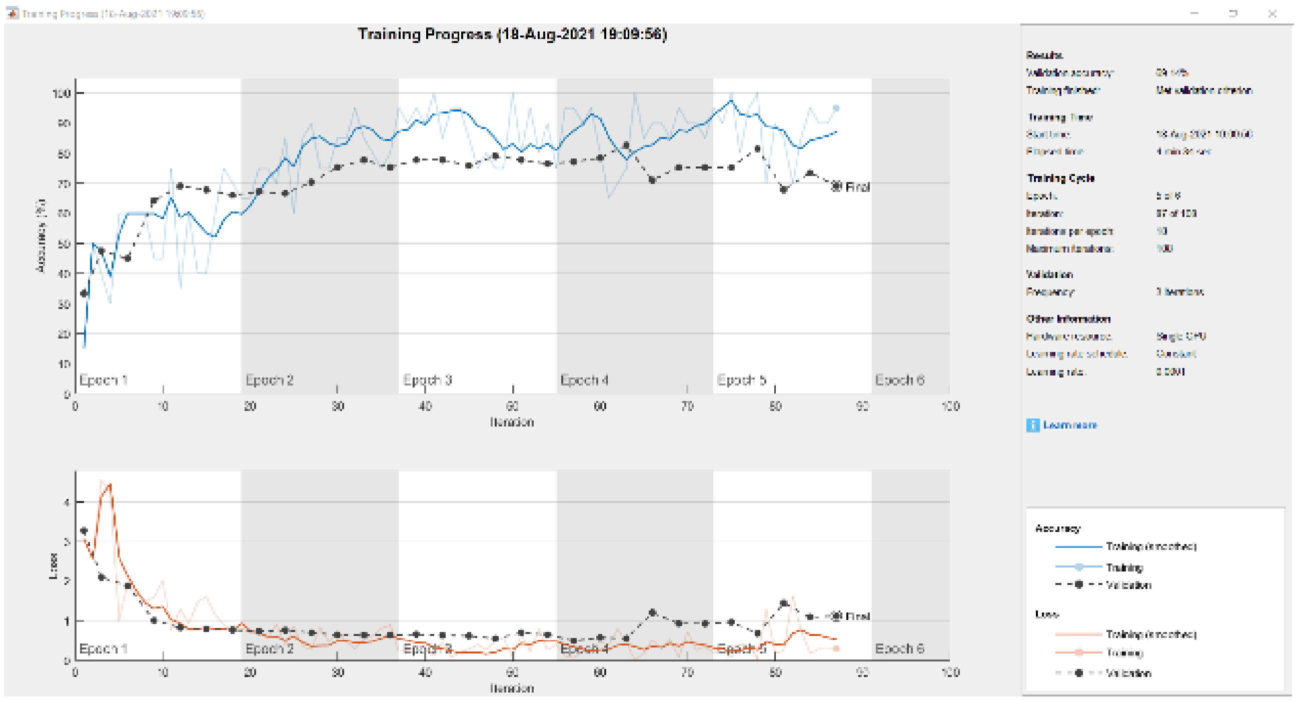
The first dataset used for transfer learning was the original unchanged images. The only change the images underwent was rescaling for input into AlexNet. 30% of the training set was used for validation when training both the suit and the rank CNNs. A minibatch size of 20, validation patience of 10, and initial learning rate of 1x10^{-4} were used when training the suit network. In addition, the maximum allowed epochs were set to 6. When training the rank network, the same options were applied, but the maximum allowed epochs were set to 7.

The resulting accuracies were less than stellar. The suit classifier was roughly 65% accurate while the rank classifier was about 51% accurate for a total combined accuracy of 33.67%.

Using the cropped dataset vastly improved results of both classifiers. For the cropped dataset, 200 of the 1450 test images were partitioned to create the test set and 30% of the remaining training data was used as a validation set. The suit classifier used a minibatch size of 100, validation patience of 10, initial learning rate of 1x10^{-4}, and max epochs of 4. The rank classifier used a minibatch size of 20, but all the other options stayed the same. These resulted in much higher accuracies. The accuracies for all models and data used are summarized in Table 4. Figures from the training process and results are shown as well in figures 10 through 17.

Table 4: *Transfer Learning Accuracies on Test Set.*

	Suit Accuracy	Rank Accuracy	Total Accuracy
Original Data	64%	51%	33.5%
Cropped Data	95%	81%	77%



Results
Validation accuracy: 69.14%
Training finished: Met validation criterion

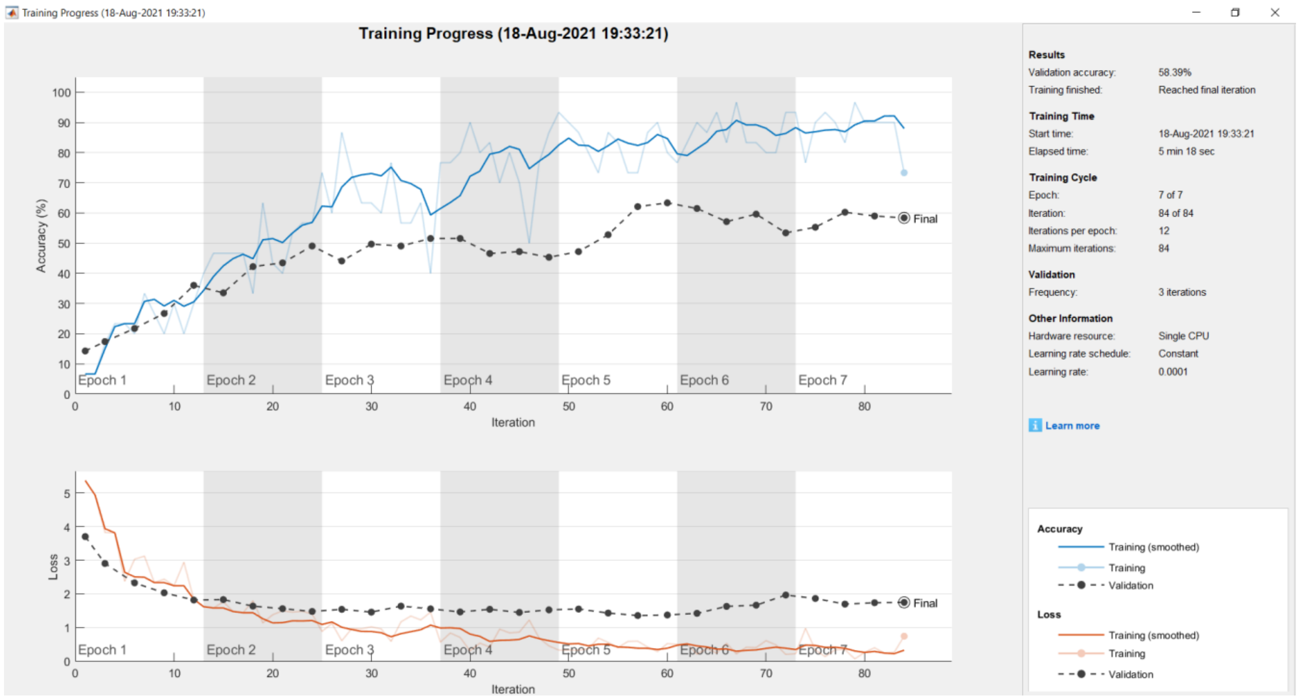
Training Time
Start time: 18-Aug-2021 19:05:[illegible]
Elapsed time: 4 min 34 sec

Training Cycle
Epoch: 5 of 6
Iteration: 87 of 108
Iterations per epoch: 18
Maximum iterations: 108

Validation
Frequency: 3 iterations

Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 0.0001

Figure 10: *Suit transfer learning training results for original data*



Results
Validation accuracy: 58.39%
Training finished: Reached final iteration

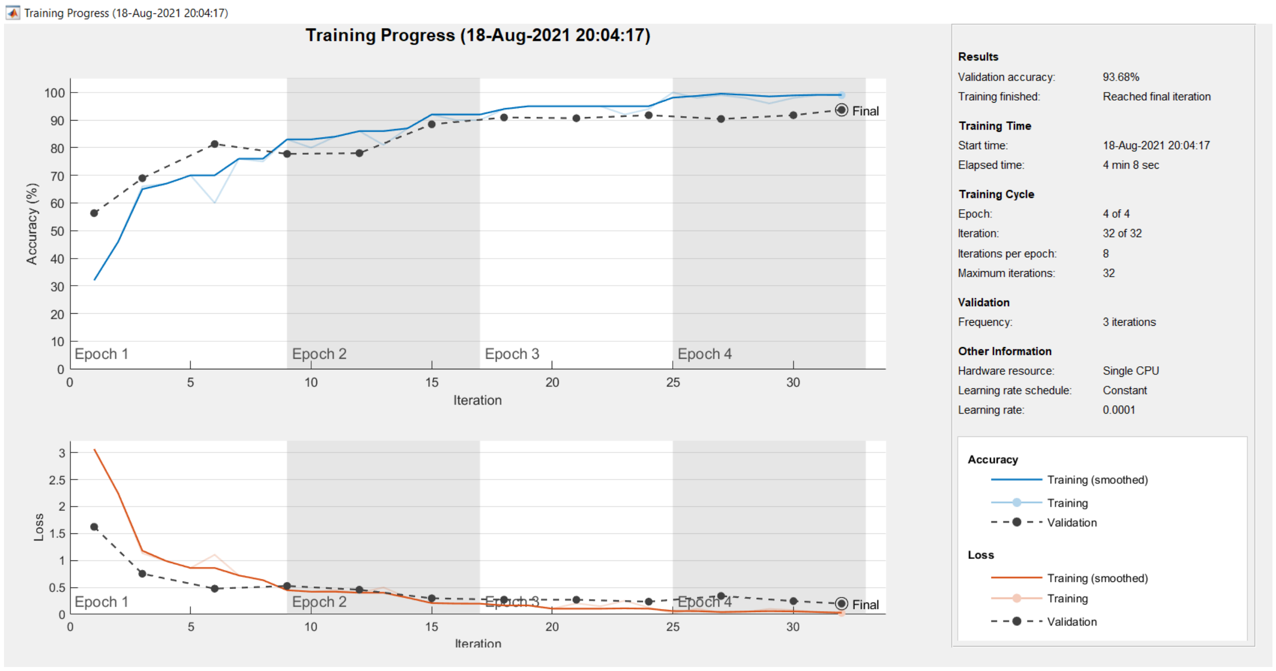
Training Time
Start time: 18-Aug-2021 19:33:21
Elapsed time: 5 min 18 sec

Training Cycle
Epoch: 7 of 7
Iteration: 84 of 84
Iterations per epoch: 12
Maximum iterations: 84

Validation
Frequency: 3 iterations

Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 0.0001

Figure 11: *Rank transfer learning training results for original data*


Training Progress (18-Aug-2021 20:04:17)
Training Progress (18-Aug-2021 20:04:17)
Accuracy (%)
Iteration
Loss
Epoch 1
Epoch 2
Epoch 3
Epoch 4
Final

Results
Validation accuracy: 93.68%
Training finished: Reached final iteration

Training Time
Start time: 18-Aug-2021 20:04:17
Elapsed time: 4 min 8 sec

Training Cycle
Epoch: 4 of 4
Iteration: 32 of 32
Iterations per epoch: 8
Maximum iterations: 32

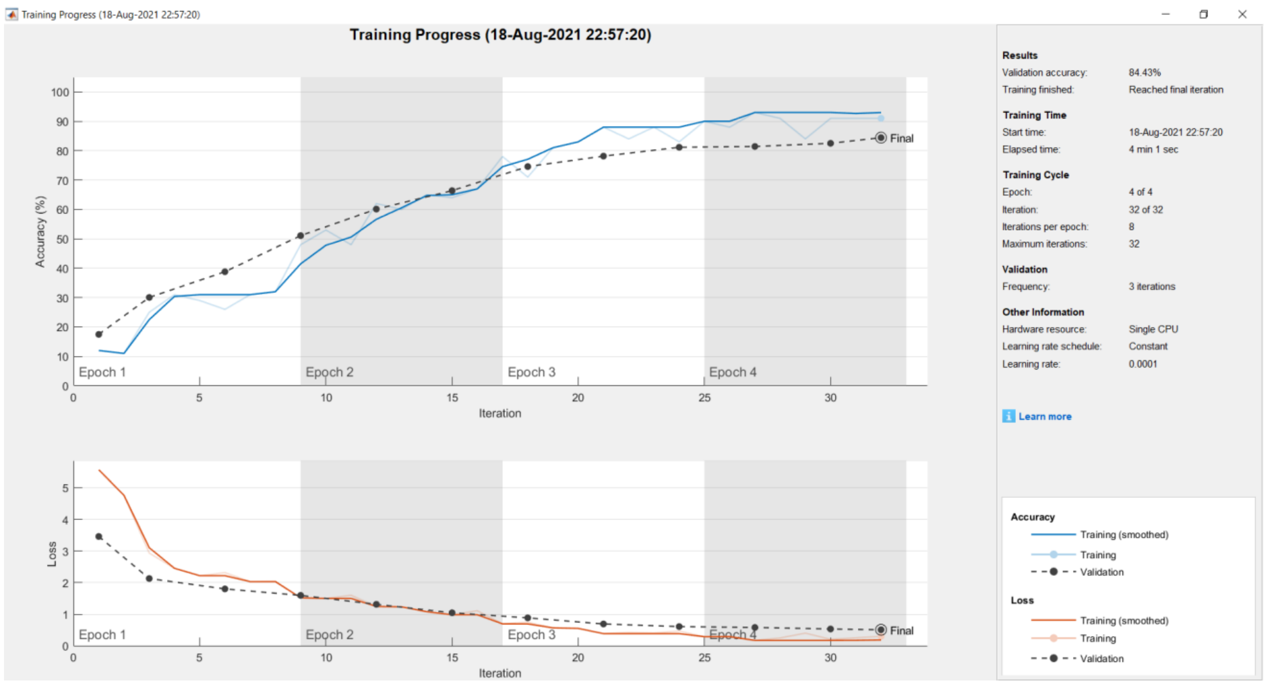
Validation
Frequency: 3 iterations

Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 0.0001

Accuracy
Training (smoothed)
Training
Validation

Loss
Training (smoothed)
Training
Validation

Figure 12: *Suit transfer learning training results for cropped data*


Training Progress (18-Aug-2021 22:57:20)
Training Progress (18-Aug-2021 22:57:20)
Accuracy (%)
Iteration
Loss
Epoch 1
Epoch 2
Epoch 3
Epoch 4
Final
Learn more

Results
Validation accuracy: 84.43%
Training finished: Reached final iteration

Training Time
Start time: 18-Aug-2021 22:57:20
Elapsed time: 4 min 1 sec

Training Cycle
Epoch: 4 of 4
Iteration: 32 of 32
Iterations per epoch: 8
Maximum iterations: 32

Validation
Frequency: 3 iterations

Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 0.0001

Accuracy
Training (smoothed)
Training
Validation

Loss
Training (smoothed)
Training
Validation

Figure 13: *Rank transfer learning training results for cropped data*

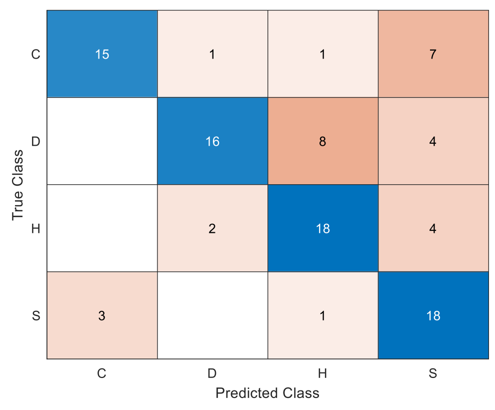


Figure 14: Suit transfer learning confusion matrix for original data

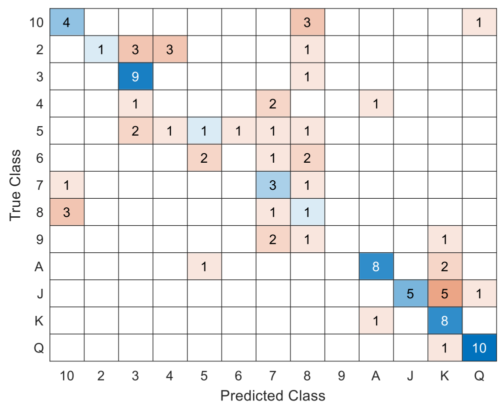


Figure 15: Rank transfer learning confusion matrix for original data

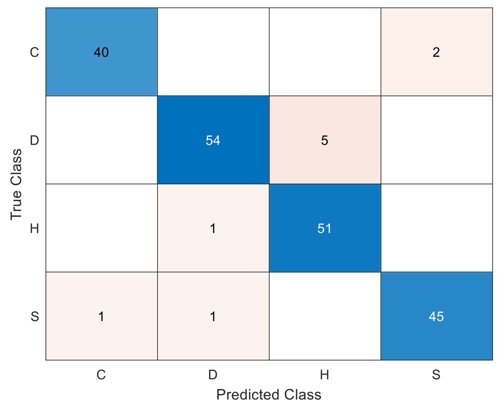


Figure 16: Suit transfer learning confusion matrix for cropped data

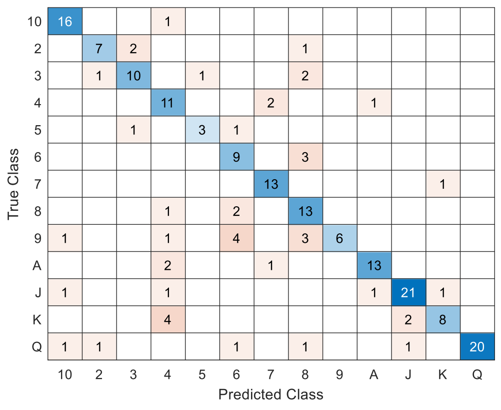


Figure 17: Rank transfer learning confusion matrix for cropped data

5 DISCUSSION

Our models produced admirable results individually, which lends credibility to the application of each of their algorithms to the task of playing card recognition. However, taken together, the model produced is best described as volatile.

To discuss this volatility, it is most important to look at the classifier input, produced using the image cropping R-CNN.

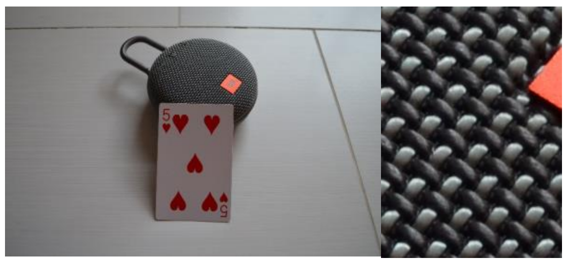


Figure 18: Original image and detected cropping region.

The original image in Figure 18 was completely misclassified as a corner. This was a problem in the SqueezeNet transfer learning R-CNN's feature extraction layer and the training images. In Figure 19, the last convolutional layer is shown indicating that the features extracted from the convolutional layer were not enough to classify a card's corner. Because the backgrounds vary among the card images, the detector is unable to determine the corners. Figure 20 shows the center of a card being detected as a card corner as well.

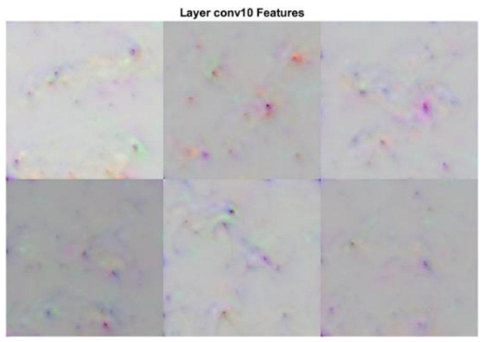


Figure 19*: Final convolutional layer of the SqueezeNet transfer learning network.*

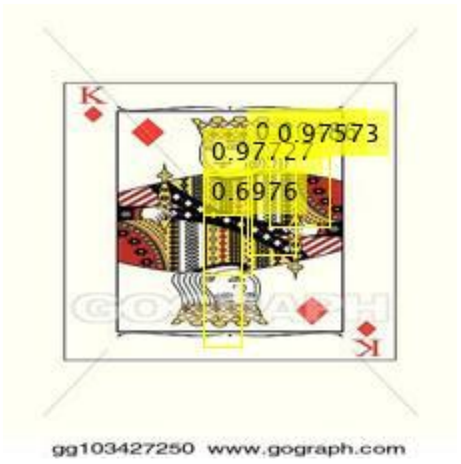


Figure 20*: Test dataset image of an entire image being misclassified.*

When images are cropped correctly, all three models perform well in their classification of individual features yet vary in their quality of combined feature classification. The support vector machine model is the clearest example of this. Individual rank classifiers ranged in accuracy from 60% to 90%. However, overall rank classification was only 30% accurate. This is likely due in fact to the number of categories being classified. The likelihood that any classifier results in a false positive is considerably higher than any single classifier. Should that false positive receive a higher score than the correct class, the image is classified incorrectly.

While this issue was amplified by the numerous classifiers required in the support vector machine model, it remained a constant between all models to a lesser extent. The custom CNN model produced suit and rank accuracies of 90% and 68% respectively. However, did much poorer in combined classification. The final and most successful model achieved suit and rank accuracies of 95% and 81%. However, even it only arrived at a 77% total classification accuracy.

Despite this, each model provides a good baseline of classification with understandable errors. Many misclassifications confound similar ranks or suits of the same color.

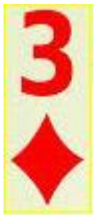


Figure 21*: 3 of diamonds misclassified by transfer learning as 8 of diamonds*

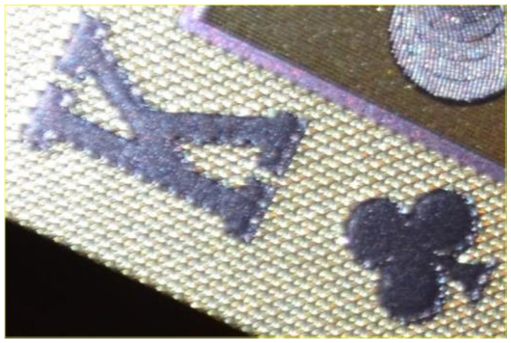


Figure 22*: King of clubs incorrectly classified by transfer learning as king of spades*

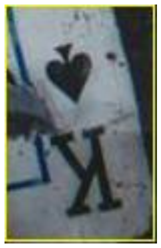


Figure 23*: King of spades correctly classified by transfer learning*

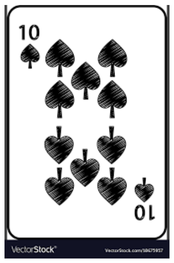


Figure 24*: 10 of clubs incorrectly classified as a spade using SVM's*

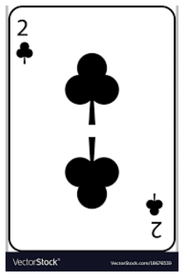


***Figure 25**: 2 of clubs incorrectly classified as a 7 using SVM's*

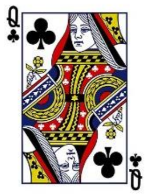


***Figure 26**: Queen of clubs correctly classified as both clubs and a queen using SVM's*

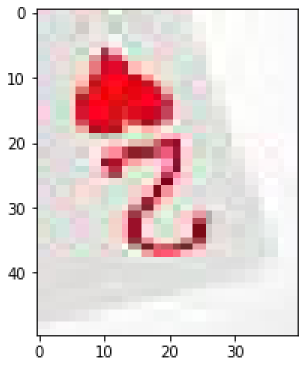


***Figure 27**: 2 of Hearts misclassified as a diamond using the custom CNN.*

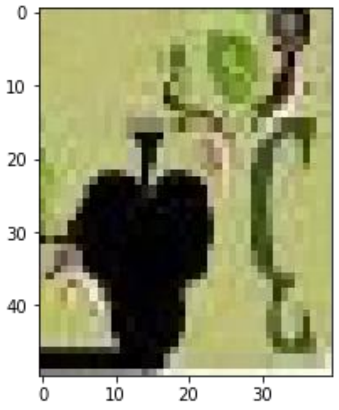


***Figure 28**: 3 of clubs misclassified as an 8 using the custom CNN.*

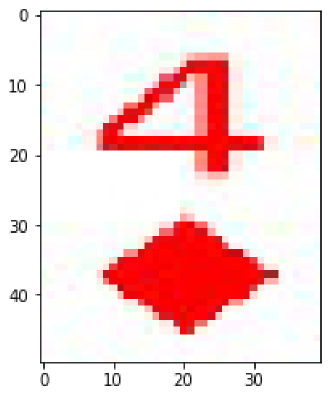


***Figure 29**: 4 of Diamonds correctly classified by both the custom CNNs.*

Other errors resulted from greedy cropping, which occurred infrequently enough in the training set to not be covered in testing entirely. However, some images still managed to be classified through greedy crops.

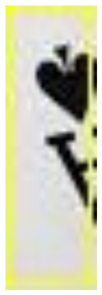


***Figure 30**: Ace of spades correctly classified by transfer learning*

6 Conclusion and Future Work

The models produced provided sufficient results to justify their use in generalizable card classification when corner images are given. However, insufficient results from our image cropping R-CNN prevents the models from being able to classify entire cards in natural environments. Despite this, the outputs from the R-CNN model still contain numerous features which may be used for classification. A better implementation, which utilizes multiple inputs from the R-CNN might be successful in meshing the image cropping and corner classification models.

There are many possible ways of improving card classification. Improvements to Hough transforms and masking could allow for better isolation of the card making it easier for the model to classify cards in different situations, such as occlusion and confusing environments. Increasing the dataset through augmentation or manually would also lead to improvements in classifying cards as the model would have more images for the 13 ranks it has to classify. Template matching or object detection could be implemented so that cards could be identified and located in an image and could allow a program to determine which cards are in hand and which are on the table. A long-term outcome of further topic exploration could be developing a suggestion bot for card games like poker or blackjack using game playing artificial intelligences. Finally, the system could be implemented in a mobile application, that could allow a user to determine best moves on the go.

7 Key Challenges and Lessons Learned

The final versions of each of our models underwent numerous revisions amidst troubles, errors, and incorrect assumptions. As is usual, these ranged from the domain of data collection to the realm of classification. Below are the most prominent of these challenges and any valuable knowledge gained from each.

7.1 Dataset Issues

The most prominent issue, spanning the totality of the project, was the failure of our dataset to meet our needs. This can best be broken down into a few key issues.

The first and most crucial problem was the repeated labeling and spelling errors. Numerous images were either labeled incorrectly, or the label was correct but misspelled. This required us to manually sift through the entirety of our dataset to ensure that our networks were not training on incorrect data. This took numerous hours to correct and was caught later into the project, resulting in large amounts of wasted time with data issues. The most important takeaway from this is to ensure the dataset is correct before starting any other work, so that corrections can be made, or another can be chosen.

Another problem which plagued the dataset was the suboptimal organization of the data. From the beginning, the only signifiers of an image's class were stored in .csv files. MATLAB's image datastore took in classes from files. At the point, our group decided to detect rank and suit separately, this became a larger problem, as both had to be parsed from this single file. The solution to this was the complete reorganization of the data into classification files by hand.

The final issue our dataset faced was its limited size. This became an issue particularly during support vector machine training as often a single rank only had a handful of data points to use. Other models in similar domains had dataset sizes of 54,600 cards [4]. This helped those models to learn more precisely and reduced the chance of overfitting. The solution arrived on was using hand-taken images to supplement the original dataset. In the future, it would be wise to look for larger datasets, particularly when training is to be done between numerous classes.

7.2 Image Datastore

As a result of the dataset issues our group encountered early in the project, we required a method of reading in classes from a .csv file. The .csv file was looped through, and its values were added to the image datastore used for the training and test sets. This was done quickly and seemed to have few drawbacks. However, due to type misclassification, training functions refused to read in the label matrix that the file had produced in the image datastore. This caused numerous days of confusion, and as types are not explicit in MATLAB, it was time consuming to find.

7.3 Hough Transform

Our literature reviews noted that the Hough transform was a useful algorithm for isolating cards and their constituent parts. Further, it could be seen to provide a straightforward method of image rotation to ensure standard training inputs. This prompted an exploration of the algorithm's applicability to the project. Two different models were constructed to do this. One attempted to find rectangles of varying size in the image which resembled a card. This method failed due to its long computation times and therefor its inability to scale to even moderately sized datasets. The second method detected the lines in the image before finding two opposing lines which signified the edges of a card. This was much faster and could use the power of built in MATLAB functions. However, this failed at providing the accuracy needed to train or test classifiers. As a result, both functions were scrapped.

This example serves as a good instance of when to cut your losses and move on. Numerous hours were poured into developing the algorithm, which yielded no impact on the result. As such, it illustrates that moving past an idea is sometimes the best option when nothing is providing results and improvements are minor between revisions.

8 REFERENCES

[1] C. (. Zheng and R. Green, "Playing Card Recognition Using Rotational Invariant Template Matching," *University of Canterbury Department of Computer Science and Software Engineering,* 2007.

[2] W.-Y. Chen and C.-H. Shung, "Robust Poker image recognition scheme in playing card machine using Hotelling transform, DCT and ren-length techniques.," *Digital Signal Processing,* vol. 20, pp. 769-779, 2009.

[3] J. Pimentel and A. Bernardino, A Comparison of Methods for Detection and Recognition of Playing Cards, Instituto de Sistemas e Robotica, Instituto Superior T ´ ecnico.

[4] M. Castillo, B. Goeing and J. Westell, "Computer Vision for Card Games," *Stanford University,* 2016.

[5] MathWorks, "Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN," [Online]. Available: https://www.mathworks.com/help/vision/ug/getting-started-with-r-cnn-fast-r-cnn-and-faster-r-cnn.html. [Accessed 19 August 2021].

[6] MathWorks, "Anchor Boxes for Object Detection," [Online]. Available: https://www.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html. [Accessed 19 August 2021].

9 APPENDIX

9.1 CUSTOM CNN ARCHITECTURES

Suit CNN
3 3x3 relu filter layers with 50x40x3 input
8 3x3 relu filter layer
8 3x3 relu filter layer
0.1 Dropout
32 3x3 filter relu layer
32 3x3 filter relu layer
0.1 Dropout
64 3x3 filter relu layer
64 3x3 filter relu layer
0.1 Dropout
128 3x3 relu filter layer
128 3x3 relu filter layer
128 3x3 relu filter layer
0.1 Dropout
256 3x3 relu filter layer
256 3x3 relu filter layer
256 3x3 relu filter layer
0.1 Dropout

512 3x3 relu filter layer
512 3x3 relu filter layer
512 3x3 relu filter layer
0.1 Dropout
flatten layer
500 neuron relu layer
500 neuron relu layer
0.1 Dropout
500 neuron relu layer
500 neuron relu layer
0.1 Dropout
500 neuron relu layer
0.1 Dropout
500 neuron relu layer
500 neuron relu layer
4 Softmax Layer output

Rank CNN
3 3x3 relu filter layers with 50x40x3 input
8 3x3 relu filter layer
8 3x3 relu filter layer
0.1 Dropout
16 3x3 relu filter layer
32 3x3 relu filter layer
0.1 Dropout
64 3x3 relu filter layer
0.1 Dropout
128 3x3 relu filter layer
128 3x3 relu filter layer
128 3x3 relu filter layer
0.1 Dropout
256 3x3 relu filter layer
256 3x3 relu filter layer
256 3x3 relu filter layer
0.1 Dropout
512 3x3 relu filter layers
512 3x3 relu filter layers
512 3x3 relu filter layers
0.1 Dropout
Flatten Layer
1000 neuron relu layer
13 Softmax Layer output