A Baseline for Automated Card Tracking for Bridge Game Broadcasting

Ioannis Christos Karakozis
Princeton University
Advisors: Adam Finkelstein, Greg Humphreys

Abstract

Millions of people play contract bridge worldwide in clubs, tournaments, online and with friends at home, making it one of the world's most popular card games. Currently, there are hired professionals to record the plays performed at high stakes bridge tournament tables so that the game can be broadcast online in a viewer-friendly format. This process is inefficient and costly, as a separate human observer is needed for every bridge table in the tournament.

This project takes the necessary first steps to develop a fully automated system that reliably tracks the plays performed and broadcasts them in a viewer-friendly format. It introduces an original dataset card detection comprised of frames extracted from video recordings of bridge play, it introduces a baseline algorithm for automated card tracking that is developed with traditional computer vision techniques, and it develops a neural network system for card classification achieving high levels of classification accuracy.

The project code is publicly available at this link. To request access to the CARDS dataset, contact the author at ick@princeton.edu.

1. Introduction

Contract bridge, or simply bridge, is a trick-taking card game using a standard 52-card deck. It is played by four players in two competing partnerships, with partners sitting opposite to each other around a table. Millions of people play bridge worldwide in clubs, tournaments, online and with friends at home, making it one of the world's most popular card games [2].

Despite its popularity and highly prevalent tournaments, bridge competition processes for recording competitive games and broadcasting them online are utilizing fairly outdated techniques for achieving their goal. According to bridge master Greg Humphreys, bridge tournament organizations hire individuals to manually record and input the plays made on bridge tournament tables to the online broadcasting system [11]. Since every table requires a separate human observer, this leads to bridge tournament organizations spending a significant amount of money, making this process highly cost-inefficient, despite being prone to frequent human errors [11].

Figure 1. Overhead view of a typical bridge tournament table setup.

Given recent developments in object detection algorithms in the field of computer vision, this process could definitely be automated. The human observers could be replaced with a camera system that would perform exactly the same functionality automatically, without any human intervention other than the initial setup. This would lead to lower costs for the tournament organizations in the long-term, as they would only have to face the one-time fixed cost of purchasing the necessary equipment for such a system. This system would also improve the viewer experience, as a computer system can detect and broadcast the bridge plays almost instantaneously, potentially with a much lower error rate than that of the human observers.

The problem though is not as simple as it might initially seem. If the only items present on the bridge table were the playing cards and green felt, then the problem would be

fairly easy to solve. What complicates things is that there are multiple items on a bridge table, such as special linings to separate the playing area of each player, notepads, lucky charms, personal items the players bring, etc. Most importantly, there is a wall-divider blocking the line of sight between teammates to prevent cheating. This makes the task particularly challenging since it forces the overhead placement of any recording device needed by the proposed computer vision system at a significant distance above the table. This also implies that part of the table is not visible by the overhead camera. A representative example of such a setup can be seen in Figure 1.

The system to be developed has to be able to detect cards when those are present at a sizable distance from the device that will be feeding the system with the overhead view of the table. The system would be able to correctly detect and classify the cards, even in the occurrence of card occlusions, lighting variations, and the presence of items similar to the cards, such as personal items and bridge bidding cards. It should be able to identify which player has played each card and reason through illegal plays the players have made. This system should be able to detect when a bridge hand starts and identify the plays as they are being made, while allocating tricks to the winner of each play. It should finally parse this information and present it to the viewer in a pleasant to the eye format. All of these processes should be fully automated, without the need of human intervention, to tackle the existing inefficiencies in the process.

Due to the high complexity of the task at hand and the limited time available for the delivery of this project, this project focuses on the detection component of the system. It implements a baseline card detection algorithm, introduces the first ever card detection dataset, and performs experiments that confirm that deep learning can be effectively leveraged for card detection.

Section 2 presents related work and background on computer vision and the task of object detection. Section 3 introduces the CARDS dataset and presents the relevant dataset statistics. Section 4 introduces and evaluates a baseline card detection algorithm developed using traditional computer vision techniques. Section 5 presents and evaluated the deep learning system developed for card classification and how this system achieves high levels of classification accuracy, even under restricted amount of data. Section 6 states our conclusions and future work needed to develop the system required to perform all stages of the target task.

2. Related Work

Detection Dataset: Any machine learning technique requires a sufficiently big dataset to leverage big data properties and sufficiently learn the target object classes. For the tasks of object detection and classification, the richest such datasets are ImageNet [4], Pascal VOC [5], and MS COCO [14]. However, none of these datasets include playing card object classes, despite their extensive size. In particular, despite extensive search on existing object detection and classification datasets, there were no datasets with object classes related to playing cards. Thus, the first problem this project is dealing with is the creation of such a dataset that would allow the leverage of machine learning techniques to tackle the target task.

Traditional Object Detection Algorithms: Over the last 20 years, a significant amount of progress has been made on the task of object detection. Traditional computer vision object detection algorithms have utilized sliding windows and feature representations specific to the target objects. Examples of such algorithms are ones based on SIFT features [15], horizontal and vertical filters [19], and the Bag of Words model [1].

Alternative attempts to model the target object classes took the form of template matching algorithms, which aim at deriving a representation of the target object off-line. The resulting set of representations exploits any potential structure in the template distribution of the target object class, so that online matching can be performed using generalized distance transforms [12].

The disadvantage of all of the methods listed was that they could not easily detect objects suffering from significant perspective changes or occlusion of key features needed for detection. This is because, be it high-level templates or low-level features, it is hard to manually capture all possible object representations that can be seen in the wild for a particular object class.

Algorithms that attempted to achieve more generalized representations of the target object were Histogram of Oriented Gradients-based (HoG) algorithms [3]. This technique counted occurrences of gradient orientation in localized portions of an image, forming this way a vector representation of the image. It then used Support Vector Machines to identify the class of objects present in the image. By performing this at various scales and utilizing non-maximum suppression, the objects of the desired class could be identified and localized correctly. Again though, this technique performed to a limited extent in the presence of significant object deformations due to perspective transforms and occlusions [3].

To deal with the problem of deformation, the HoG features algorithm was enhanced in the Deformable Parts Model [6]. This work broke down the desired object class in parts that could be detected independently using HoG features (e.g limbs, torso, and head for a human) and then be put back together to localize an object of the desired class. This model still suffered in the presence of significant perspective transforms, deformations and occlusions, despite its superiority over all previous models.

Regardless of the limitations of traditional object detection algorithms, the most prominent disadvantage of them is their limitation in their ability to generalize to multiple object classes. Most of these algorithms were targeting a particular object class, such as pedestrians [12], human faces [19] or humans in general [3, 6]. Therefore, building upon any of those algorithms for card detection requires specializing the chosen algorithm for the 52 possible combinations of rank-suit observed in a typical deck of cards. Given most cards of the same rank and suit are extremely similar to one another, our baseline algorithm is a specialized to cards version of the Chamfer Template Matching Algorithm [12], which heavily leverages the sharing of common structure and low-intraclass variation among objects of the same class.

Convolutional Neural Networks: Neural networks have succeeded in tackling the inability of traditional object detection algorithms to generalize to multiple object classes without having the need to model each object class separately. State-of-the-art object classification architectures, such as VGG [16] and ResNet [8], have not only outperformed all previous object classification methods. The same goes for the family of R-CNN networks [7], the state-of-the-art object detection deep learning architectures. Therefore, it seems that these architectures are ideal for tackling the problems of card classification and detection, due to how good these models are at representing the target object class in a form that can handle occlusions, deformations and perspective transforms.

The biggest limitation of neural networks is that they require a sizeable amount of data to achieve satisfactory results. Although this can be ameliorated by learning some generic objectness features by pretraining on the ILSVRC classification challenge dataset [4, 10], deep networks still require a significant amount of data to achieve satisfactory generalization performance and avoid overfitting. Therefore, tackling any object detection problem using neural networks is highly dependent on the development of an appropriately large dataset, which motivated the development of the CARDS dataset.

Figure 2. The dummy lays out their cards on the table, making them visible to everyone. In this image, the player at the bottom right corner is the dummy.

3. The CARDS Dataset

Due to the lack of existing annotated card detection datasets, one had to be developed for the purposes of this project. Thus, we are introducing the CARDS dataset.

3.1. Object Classes

CARDS includes annotated images for card detection in the context of bridge. To this purpose, 53 object classes are specified. The 52 basic classes represent the 52 possible combinations for a cards rank {2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace} and suit {Diamond, Hearts, Club Space}. The 53rd class is included because of special rule in bridge called ”the dummy”. In particular, this class is labeled as ”Dummy”.

The dummy in bridge is a player who lays out their cards wide open on the bridge table, making them visible to all other players. Therefore, their cards are visible despite not yet being played. Such a situation is depicted in Figure 2. In the context of generic object detection, this is not a problem, as we would care about localizing every possible card. In the context of card detection for bridge, we only care about detecting cards that have been played and are ”active”, which do not include the cards of the dummy player. This is because the end goal of the algorithm is to detect all plays that are being made by the players, which do not include the unplayed dummy cards. Those visible cards have to be separately labeled to ensure that an algorithm that detects those cards is neither rewarded nor penalized for doing so. Thus, cards that fall within the dummy area are not identified individually, but are collectively localized by a ground truth bounding box and labeled with the ”Dummy” object class. Such an annotation is presented in Figure 4.

In terms of ”active cards”, we only annotate cards that we deem visible and identifiable by a human agent. In other

	2	3	4	5	6	7	8	9	10	Jack	Queen	King	Ace	Total
Clubs	23	88	32	35	13	29	30	18	17	19	42	18	59	423
Hearts	58	28	25	22	47	26	52	18	67	18	68	22	56	507
Diamonds	38	34	28	42	19	46	59	53	43	29	24	31	47	493
Spades	51	23	37	22	27	61	54	22	33	45	36	21	33	465
Total	170	173	122	121	106	162	195	111	160	111	170	92	195	1888

Figure 3. Number of instances per object class in CARDS.

Figure 4. The light blue bounding box corresponds to a "Dummy" class annotation. Notice how the cards inside the bounding box are not individually labeled as they have not been played yet.

words, we draw bounding boxes around the cards that have been played and a human can unambiguously identify their rank and suit. Otherwise, we assume their salient features are too deformed to expect a network to label and learn the correct class from the deformed card. Such significant deformations are typically results of card occlusions and/or significant motion blur.

3.2. Dataset Components

CARDS consist of three sub-components:

1. **Single-Card:** 165 images of single-card instances, at various rotations, scales and lighting conditions. These are considered simple card instances, which would be easy to detect and classify correctly. No occlusion, motion blur, or shadows are present in those images. A single deck of cards is being used for this set of instances, which makes the detection task even easier as all cards of the same rank and suit have exactly the same look and shape. Since there is a single card on each frame, this component contains 165 card instances.

2. **Simple-Bridge:** 574 frames extracted from 29.5 minutes of bridge play recorded at 1080p resolution and 24 fps. A single frame is used per second of the recording to avoid repeated frames. Frames in which no "active cards" are present were manually removed, as it will be detailed in Section 3.3. A typical frame of this component is presented in Figure 2. Multiple cards are present on a table with green felt and no foreign objects are present other than the players themselves. "Dummy" instances can be present. Motion blur, rotations, variations in scale, occlusions and lighting variations are present across the card instance of this components. The frames are extracted from two bridge games at different locations, to make the instances of this component more representative of bridge game setups. The total number of non-dummy card instances is 1129.

3. **Complex-Bridge:** 269 frames extracted from 17.5 minutes of bridge play recorded at 1080p resolution and 24 fps. A single frame is used per second of the recording to avoid repeated frames. Frames in which no "active cards" are present were manually removed, as it will be detailed in Section 3.3. A typical frame of this component is presented in Figure 4. This component is identical to the **Simple-Bridge** component, with only two differences that makes this a harder dataset to classify: 1) foreign objects, such as bidding trays, are present on the table along with the cards, as evident in Figure 4, 2) multiple types of decks were used, which increases the intra-class variation of each rank-suit class of cards. The total number of non-dummy card instances is 594.

In total, there are 1008 frames, with 1888 card instances, if we exclude the "Dummy" class annotations (1.87 "active cards" per frame on average). If we include the dummy annotations, the total number of bounding box instances amounts to 2562. The total annotation time amounted to approximately 15 hours.

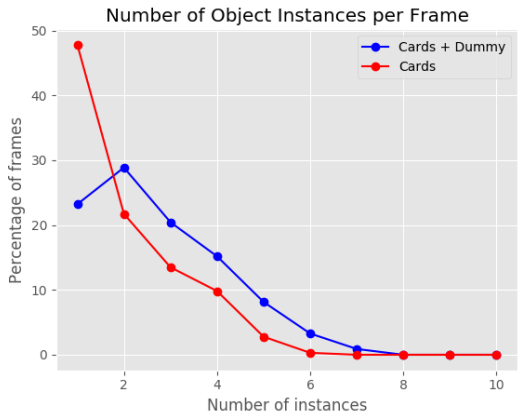

Figure 5. Percentage of frames for each number of card instances present in a frame. Notice how the percentage of frames with more than 5 object instances (blue line) and the percentage of frames with more than 4 card instances (red line) is very small, a consequence of bridge restricting every player to have up to one card "active" at a time.

3.3. Dataset Statistics

The number of instances per rank-suit can be viewed in Figure 3. Note that there is a very even representation for suit, with a standard deviation over average on the number of instances being 0.0679. By contrast, the distribution gets more and more uneven, as you increase the number of classes one considers. The same value for rank is 0.2361 and for rank-suit pairs is 0.4483, which indicate that certain classes are way more prominent than others. For example, the Queen of Hearts has 68 instances while the King of Clubs only has 18. Similarly, while there are only 92 Kings in the dataset, there are 195 Aces. This class bias can lead to skewed training of machine learning algorithms towards specific classes, that can potentially dominate the classifications during test-time, leading to higher test error. Due to the large number of object classes, the only way to mitigate this bias towards specific object classes and ranks, a larger dataset should be developed, Unfortunately, this was not feasible in this context due to the limited time available.

Following the examples of [4, 14], we also report the percentage of frames for each number of card instances present in a frame as a measure of how richly annotated our samples are on average. This is depicted in Figure 5. Note that because each card is unique in a deck, this also corresponds to the number of object classes present in an image. The average number of instances per image are 2.70 if we include the Dummy objects and 1.87 if we do not. These averages are higher than the one for PASCAL VOC [5] but lower than the ones for ImageNet and COCO [4, 14], which indicates our images are moderately rich in number of instances. Note though, that it is extremely unlikely to observe more than 4 instances (if one excludes the dummy) in a frame, since up to four cards can be legally played simultaneously in bridge. If the number of card instances is greater than 4, then some player must have made a mistake which, as evident from Figure 5, this happens very rarely (see red line). Given this inherent restriction in the recording of our data, CARDS is actually very rich in the number of card instances per frame, which is a result of the filtering process described in Section 3.3.

3.4. Annotation Pipeline

The initial plan for the project was to use Amazon Mechanical Turk for the dataset annotation, like it was done in [4, 14], to allow for large scale data annotation and verification. However, due to limited financial resources, delay in the collection of the initial video samples and development of the necessary annotation tools, we had to resort to manually annotating the frames ourselves. Thus, we developed a custom annotation pipeline depicted in Figure 6.

A total of two and a half hours of bridge were recorded in three different locations to capture possible lighting variations in the dataset. However, out of the total play time, only 45 minutes of footage were used for the data extraction. This because bridge players take a lot of time to think, deal, and bid, which leads to a lot of frames in which no active cards are present. Even in the cases in which "active cards" are present in the frame, the time players take to think might be so significant at times, that a fraction of frames with "active cards" had to be removed to prevent certain combinations of rank-suit dominating the dataset. Thus, this resulted in 1008 frames being used, which correspond to approximately 17 minutes of game time, given we only use one frame per second, for the reasons explained in the previous section.

The video samples were collected using a GoPro Hero 5 Session mounted on a tall tripod with an extension that allowed the placement of the camera at a significant height right above the table (The equipment is readily available in the Princeton University Computer Science Technology Office. You can also contact the author for further details). After the footage is collected, parsed into frames and those are filtered into frames in which active cards are present, the frames were annotated using a bounding box annotator. The bounding box annotator used was adapted from the following MIT licensed open source labelling software: https://github.com/puzzledqs/BBox-Label-Tool.

Given the camera tripod was placed right next to the

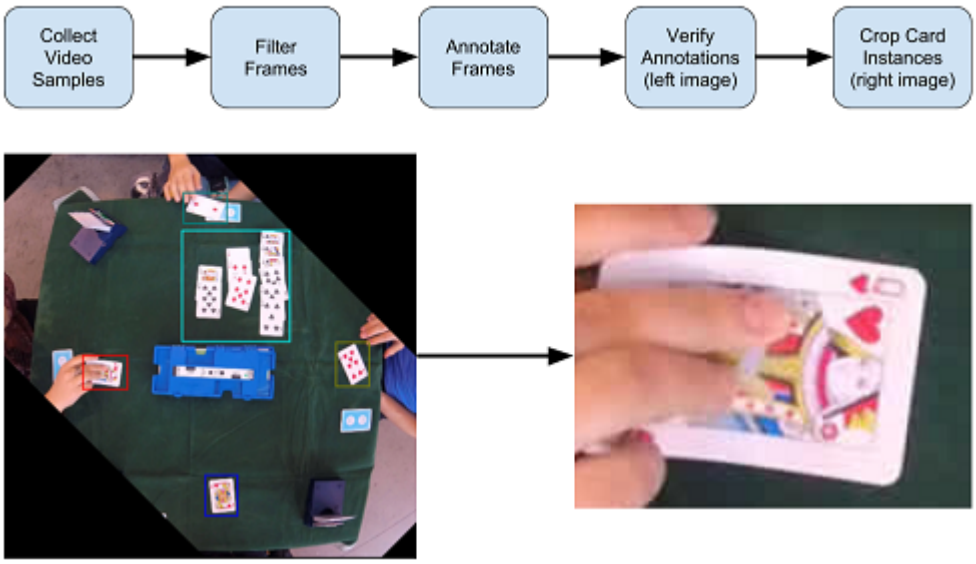

Figure 6. Annotation pipeline for CARDS dataset. We used the bounding box annotations to extract single card instance annotations to form a secondary classification dataset programmatically.

corner of the table to ensure the players experienced the minimum amount of discomfort, all frames were rotated by 45 degrees before annotation to render the table edges parallel to frame boundaries. An example of such a rotation is presented in the left image of Figure 6. This was necessary to ensure more compact bounding box annotations, whose edges are parallel to the boundaries of the image. Post-annotation, which amounted to approximately 15 hours of work, a 2.5 hour verification process followed, in which we ensured no cards were mislabelled or missed and that the bounding box annotations were tight.

The final step was to extract single-card instances from our labelled frames, as evident in the lower half of Figure 6, to form a secondary classification dataset of single card instances. This classification dataset had size of 1888 card instances, with the object class breakdown provided in Figure 3. This extraction was necessary to ultimately train our deep learning classifier, further detailed in Section 5.

4. Baseline Card Detection Algorithm

According to Greg Humphreys [11], the first attempt to integrate an automatic card detector to a professional bridge setup was made in 2011. However, that implementation was operating under very strict restrictions that forced the players to play cards in a particular way that was deemed ”distracting”. Because of that, the system was used for a very limited time by the bridge tournament organizations. Thus, there is no public documentation or performance metrics analysis on the exact computer vision techniques used by the algorithm. Therefore, to understand the system we need to build better, we attempt to reproduce this algorithm.

According to [11], this algorithm required players to detach the card played from their hand and place it an rectangular area that was drawn on the table, similarly to the lines on the bridge table depicted in Figure 1. This way the card was directly visible to the camera and was outlined by auxiliary lines that allowed the algorithm to more easily separate the card from the background. While we will maintain the first restriction in our algorithm, i.e. that the card will not be occluded by any object nor will it be experiencing deformations, we will not be using auxiliary lines, nor guarantee the camera is placed at a fixed height and position. This allows for scale and perspective changes. Thus, our baseline algorithm needs to be robust to sizeable perspective transforms and scale or rotation variations.

More concretely, the simplifying assumptions we make are: 1) occlusions and deformations of the card are not possible, 2) a specific set of cards is being used, 3) a rectangular area of card and background that fully encloses the card can be isolated, 4) the card is placed on felt of uniform color, as is typically the case in professional bridge tournaments. An example of assumption 3 is depicted in Figure 7. The pipeline of the algorithm described in the following subsections is visualized in Figure 8.

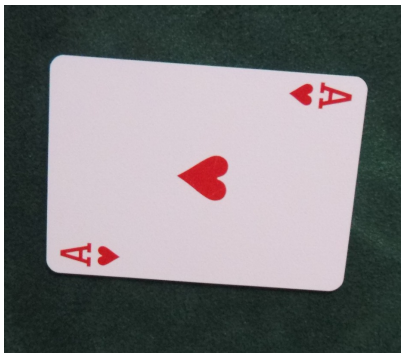

Figure 7. The card is fully enclosed by the frame, leading to no occlusions. However card rotations and perspective transforms are allowed.

4.1. Modelling the Background Felt

Given that most professional bridge tables are covered by felt of uniform color, we developed an algorithm that exploits this environmental element. In particular, we assume that the relatively uniform color channel values of the pixels corresponding to the felt are distributed according to a Multivariate Gaussian distribution with small variance. This approach worked really well for modelling and removing static backgrounds of relatively uniform color in the project I had undertaken in the Princeton Computer Vision course (COS 429) offered in Fall 2017 [9].

The proposed algorithm makes the assumption that all pixels at the boundary of the frame belong to the background, i.e. are part of the felt. The algorithm collects all the pixels on the frame boundary and treats them as a representative sample for the background color distribution. It then parametrizes a k-dimensional Multivariate Gaussian distribution from the sample using maximum likelihood estimation. Using the resulting mean vector and covariance matrix, the algorithm performs a chi-square hypothesis test on every pixel of the to determine whether the pixels belong to the foreground or are part of the background, i.e. the felt. Note that this felt subtraction algorithm can work on uniform felt of any color, as long as that color differs from the colors found on a card (i.e. red, black and white).

Represent the intensity of each pixel with k color channels as the vector $\vec{x} \in R^k$ (in the case of colored pixels, k = 3). We assume the user has access to n background felt pixels to train the model with. The n pixels take color channel values $\vec{x}_1, \ldots, \vec{x}_n$. Thus we can model the background color with a Multivariate Gaussian Distribution $\vec{x}_{BG} \sim N(\hat{\vec{\mu}}, \hat{\vec{\sigma}}^2 I_k)$. For simplicity, we assume the channels of each pixel are independent and, thus, all covariances are zero, rendering the covariance matrix diagonal.

The parameters of the distribution are estimated using the Maximum Likelihood Estimators of the Multivariate Gaussian Distribution. The maximum likelihood estimator of the mean vector is given by:

$$\hat{\vec{\mu}} = \frac{\sum_{i=1}^{n} \vec{x}_i}{n} \tag{1}$$

To be consistent with the channel independence assumption, the variance of each channel is estimated independently by using the Maximum Likelihood Estimator of the simple Gaussian Distribution (since all covariances are zero):

$$\forall j \in \{1 \ldots k\}, \hat{\sigma}_j^2 = \frac{\sum_{i=1}^{n} (v_{i,j} - \hat{\mu}_j)^2}{n} \tag{2}$$

where $\hat{\mu}_j$ and $\hat{\sigma}_j^2$ are the maximum likelihood estimator of the mean and variance of the j-th color channel respectively.

By defining the foreground pixels as pixels that are not part of the felt background, one can use hypothesis testing on the modeled population $\vec{x}_{BG}$. The null hypothesis states that an observed pixel $\vec{x}'$ is well-modeled by $\vec{x}_{BG}$ and, thus, belongs to the background, while the alternative hypothesis states that the pixel is a foreground pixel. For the pixel in question, given a new sample $\vec{x}'$ to be classified, the test-statistic T is given by:

$$T = (\vec{x}' - \hat{\vec{\mu}})^T \cdot (\hat{\vec{\sigma}}^2 I_k)^{-1} \cdot (\vec{x}' - \hat{\vec{\mu}}) \tag{3}$$

The null hypothesis now states that T is distributed under Chi-Squared while the alternative hypothesis states that T is distributed under non-central Chi-Squared. We determine the degrees of freedom from the dimensionality k of the channels vector (DoF = k, as the channel components are assumed independent of one another). Using cross validation, we pick the right confidence level and then use the appropriate threshold values for the test-statistic [17]. This way we determine whether each pixel belongs to the foreground or the background. Thus, we can determine which pixels are part of the felt and subtract them from the frame by thresholding, ending up with a frame in which only the foreground pixels are visible, just like the second frame in Figure 8. To remove any background noise that might have remained, we grayscale the image and despeckle by convolving with a Gaussian filter.

4.2. Card Localization

Having removed all background pixels, we can now easily identify the contours of the card. Contours are defined as curves joining all the continuous points along the boundary of an object, which should typically have the same color or intensity. Due to the boundary of cards being predominantly white and due to having blackened out all the surrounding background pixels, it is very easy to detect

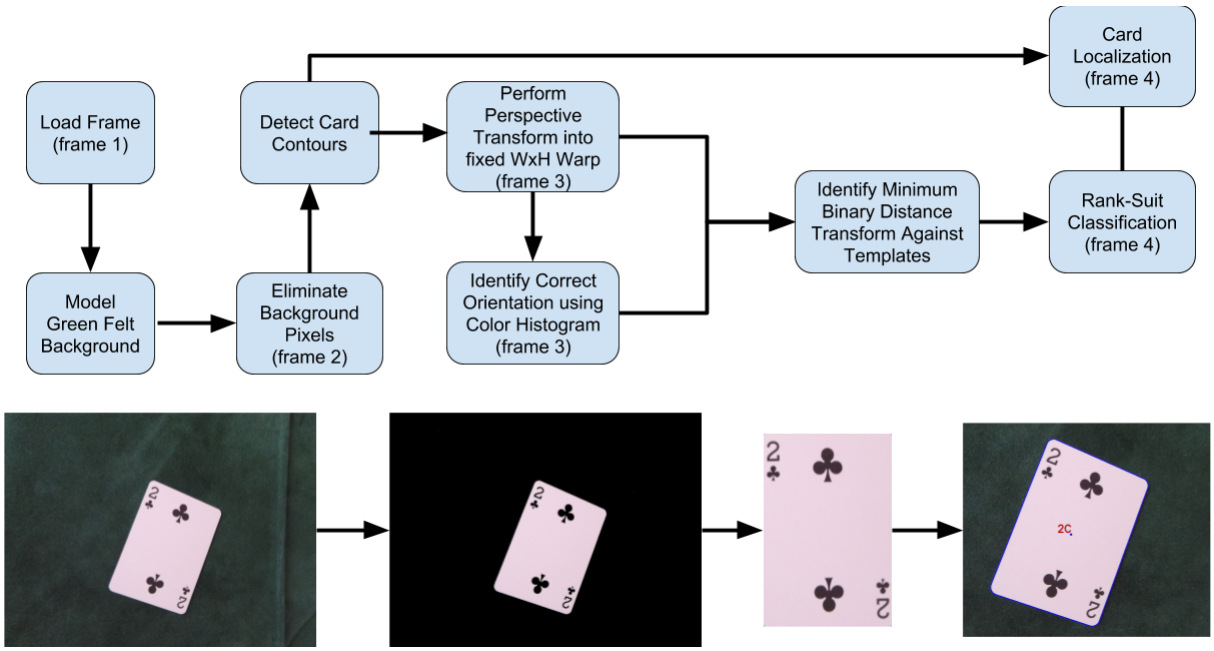

Figure 8. Baseline Card Detection Algorithm. The bottom half of the figure comprises of visualizations of each step of the algorithm.

the contours of the cards as there is a sharp brightness change when one transitions from the black background pixels to the white boundary pixels of the card.

The algorithm being used is the OpenCV implementation of the algorithm described in [18]. Assuming the background has been properly removed, the contour detection algorithm returns the boundary of the card as well as the boundaries of shapes within the card, such as the rank and suit symbols. Then, we determine which of the contours belong to the card outline by applying the following heuristics:

1. The area of the contour is bigger than the minimum expected card size and smaller than the maximum expected card size, which are being set using cross-validation.
2. The contour has exactly four corners.
3. The contour is not fully enclosed by another contour, which would typically imply it is the contour of a symbol inside the card.

Thus, by removing all non-card contours, we end up with a very precise localization of all the cards present in the frame. An example of such a localization is the blue line perfectly enclosing the card in the fourth frame in Figure 8.

4.3. Card Classification

Now that we have localized the card, what remains to be done is identify the card rank and suit. To do so, we perform a perspective transform of the card into a 200 by 300 fixed sized frame, like the one depicted in the third frame from the left in Figure 8. Using this perspective transform into a fixed sized warp, we can easily identify the top-left corner area where we expect to find the rank and suit of the card.

The reason it is important to identify the top-left corner of the warp is because the perspective transform might have resulted in a warp in which the card is in horizontal orientation and not in the vertical orientation it is depicted in the third frame in Figure 8. To check whether the perspective transform warped the card into the correct orientation, we check the color histogram of the top-left corner. By comparing the color histograms of the top-left corners of the two possible warps, we identify the one with the lowest mean brightness value at the bottom 20% of the pixels, ordered by brightness. The lower mean brightness corresponds to the corner with the most black or red pixels, which have lower color intensity than white pixels. By identifying this corner, we can identify the correct orientation and correct the warp if needed. This way, we always end up with a card warp in which the card is in upright orientation.

The next step is to compare the card against the pre-processed card templates of each rank-suit pair. The images that were used to extract those templates are excluded from the CARDS dataset due to them comprising the training

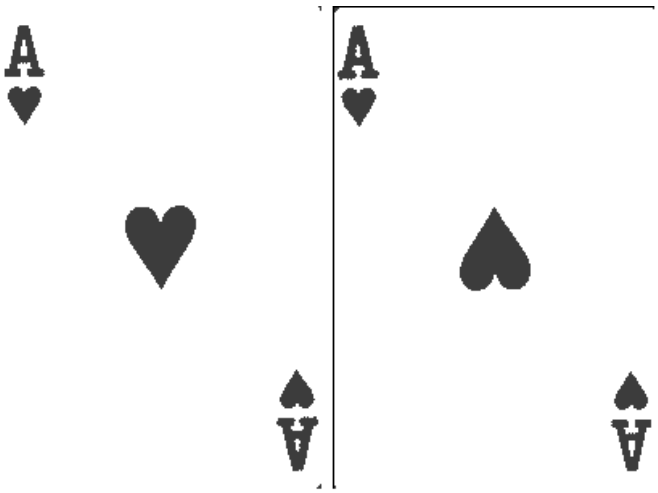

Figure 9. The ace of hearts has two possible orientations due to not being symmetric about the origin. Thus, two templates for this card are needed. Note that, because the contour detection algorithm is not perfectly precise, some boundary background pixels might be collected as part of the card, as evident from the pixels classified as red on the top and left boundary of the template on the right.

set of the baseline algorithm and them being recorded in highly controlled conditions that would be impossible to replicate in practice to ensure high quality of the templates. A template is a 200 by 300 card warp in upright orientation with pixelwise color labels. The pixels are classified as white, black or red, depending on the suit of the card and whether the pixel is part of the foreground elements of the card, such as the rank and suit symbols. Note that certain cards have two possible upright orientations, as they are not symmetric across the x-axis. The Ace of Hearts is such an example and its warp templates are illustrated in Figure 9.

To compare against the warp templates, some preprocessing of our card warp is required. Using the color histogram of the warp corner, we can identify the color of the card: black or red. Knowing the color of the card, we can classify every pixel of the warp as white or non-white (i.e. red or black depending on the determined color). We do so by comparing against a threshold brightness value that was set using cross-validation. However, because noise might be present on the card, we observed certain isolated white pixels are occasionally classified as non-white. To remove such noise from the warp pixel labelling, we use the OpenCV connected components despeckling algorithm.

Even after preprocessing the warps though, we observed that some noise occasionally persisted on the boundaries of the warps, as evident in Figure 9. This is because the contour detection algorithm is not perfectly precise and some boundary background pixels might be collected as part of the card during contour detection. To overcome this limitation of the contour detection algorithm, we curtail the warp and template boundaries by a fixed number of pixels on each side to remove potentially included background that might ruin the template matching. The number of pixels to be curtailed on each side was set using cross-validation.

Now we are finally ready to perform template matching. Since we already know the color of the card from the corner color histogram, we perform template matching only against the templates of the cards with color matching the identified color. The template matching algorithm used is inspired from the Chamfer Template Matching algorithm introduced in [12] that utilizes distance transforms. Specifically, the Chamfer distance transform is defined as:

$$D_{\text{chamfer}}(T, I) = \frac{1}{|T|} \sum_{t \in T} d_I(t) \tag{4}$$

where T is the template warp, I is the card warp and $d_I(t) = 0$ if the pixel t is a background pixel; otherwise, it is equal to the euclidean distance of foreground pixel t of the template to the closest foreground pixel on our labelled card warp. In the context of our algorithm, a background pixel is a white pixel and a foreground pixel is a non-white pixel.

To minimize the impact of noise that our pipeline is not able to remove, we modify the Chamfer matching algorithm by performing a two-sided distance transform:

$$D_{\text{chamfer+}}(T, I) = \sum_{t \in T} d_I(t) + \sum_{i \in I} d_T(i) \tag{5}$$

This way, even if there are some remaining speckles in either the template warp or the image warp, their contribution in the total distance measure will be approximately halved. This indeed lead to better performance during test-time.

By computing the distance transforms against all templates, we can pick the template that minimizes $D_{\text{chamfer+}}(T, I)$, identifying this way the rank and suit of the card. This completes the detection algorithm, as we have both localized the card and identified its rank and suit, leading to a full detection like the one presented in the fourth frame from the left in Figure 8.

4.4. Performance Evaluation

We evaluate the baseline algorithm on the Single-Card and Complex-Bridge CARDS subcomponents. We measure the accuracy of the algorithm in terms of both card

Baseline Model Accuracies		
	Detection	Classification
Single-Card	100.00%	98.18%
Complex-Bridge	29.42%	38.09%

Figure 10. Detection and classification accuracy for baseline card detection algorithm.

detection and card classification accuracy. Card detection accuracy is measured at an Intersection over Union (IoU) threshold of 0.5. The results of our evaluation are depicted in Figure 10.

As initially conjectured, we achieve an extremely high performance in the simple scenarios represented in the Single-Card CARDS component, achieving 100% detection accuracy and 98.18% classification accuracy. This was to be expected as the baseline algorithm was developed with the goal of perfectly handling those simple cases, which are extremely similar to the cases that the algorithm that was commercialized in 2011 was handling [11]. Thus, our goal of replicating the results of the only ancestor to our algorithm have been successful.

However, the performance is not nearly as good when we remove all the simplifying assumptions and we use our algorithm on unrestricted samples of cards from the Complex-Bridge CARDS subcomponent. In particular, the detection accuracy is only 29.42% and the classification accuracy is only 38.09%. Note that the classification accuracy is measured by taking into consideration the cards that were correctly detected. These levels of accuracy, along with the fact that each card detection requires 0.87 seconds, prevents this algorithms from becoming commercially suitable for live bridge play tracking. This is because, not only inference time is not fast enough to perform card detection in an online fashion, but the algorithm seems to be unable to handle card detection in the highly unrestricted setup of bridge play.

To understand why the algorithm is failing, we performed some analysis to identify the characteristics cases that the algorithm is unable to detect correctly and the corresponding modes of failure. We identified the following:

1. Presence of foreign objects (such as a player's hand) on the frame boundary prevent the model from properly modelling the felt pixels, thus leading to very noisy background subtraction.

2. Significant card deformations, in the form of perspective transforms, motion blur, and card occlusions, prevent the algorithm from correctly detecting the card contours and localizing the card, leading to highly distorted card warps that is impossible to classify correctly with template matching.

3. Significant lighting variations across game environments can lead to a wide range of brightness levels of the card pixels, which distorts the color histograms and renders our fixed brightness threshold levels irrelevant. This in turn leads to incorrect color classification, which leads to erroneous suit classification, even if the detection of the card was very precise.

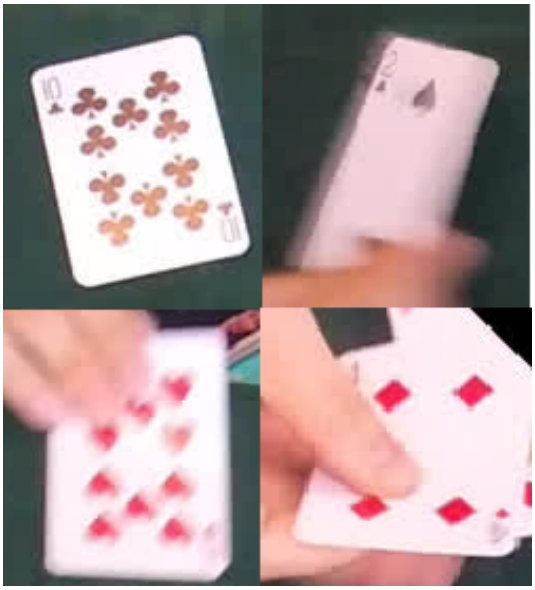

Figure 11. Representative examples of the failure modes of the baseline algorithm. Top-left: Lighting variations lead to black symbols on card appearing reddish, leading to color misclassification. Top-right: Card experiencing significant perspective transform due to player holding it sideways. Bottom-right: Card and background occluded by player's hand and other objects. Bottom-left: Card deformed by motion blur.

Failure modes (1) and (2) were the ones that were encountered most frequently, given that bridge players tend to play their cards really fast, sometimes without even laying them fully flat on the table or fully detaching them from their hand. Failure mode (3) was also encountered, but it occurred less frequently, as a result of flickering light or use of low quality indoor lighting. Figure 11 illustrates some representative examples of those failure modes picked from the Complex-Bridge test set.

Overall, although our algorithm is not able to generalize well to unrestricted setups, we have developed an algorithmic baseline that achieves high detection accuracy in simple, restricted setups that could be replicated in practice if we placed enough restrictions on the way professional bridge players play their cards. According to [11], such restrictions have been attempted in the past with no success though. Thus, we need to consider systems that are able to

handle the hard cases identified above without requiring any restrictions on the way bridge players play.

5. Deep Learning Implementation

Given the recent success of deep learning systems in the object classification and object detection tasks, we decided to investigate whether neural network architectures are suitable for handling the task of card detection. To explore this possibility, we decided to focus on the task of card classification, given that we had more samples for this compared to task of card detection (1888 card instances vs 1008 annotated frames for card detection), which would allow us to get more meaningful results, while still being able to compare with our baseline algorithm.

5.1. Architecture

The network architectures considered were the VGG and ResNet architectures introduced by [16] and [8] respectively. A visualization of the structure of each of those architectures is offered in Figure 12.

These networks were considered due to their exceptional performance on the ILSVRC 1000-class classification task [4]. The main difference between the two is that ResNet has shortcuts connecting the beginning and end of each residual/convolutional block of the architecture [8]. This allows the gradient to backpropagate effectively, even when using very deep architectures. This allows us to use deeper architectures, without worrying for vanishing gradients [8]. Because of this, the fact ResNet trained faster in our experiments, an outcome of it having a single fully connected layers instead of 3 that VGG has as evident in Figure 12, and the fact that ResNet had achieved higher accuracies in both the ILSVRC classification task [8] and our initial experiments, we decided to go with the ResNet architecture for our classification network.

5.2. Training and Validation

For training, we used 80% of the combination of the Single-Card and Simple-Bridge CARDS subcomponents, with the remaining 20% being used for validation. The Complex-Bridge CARDS subcomponent was our test set, so it was not presented to the network until test-time.

During training, a 224x224 crop is randomly sampled from an image, with the per-pixel mean subtracted, as suggested by [8]. We use backpropagation and Stochastic Gradient Descent with mini-batche of size 16 for the weight updates. Using cross validation, we initialize the learning rate at 0.01. The learning rate is divided by 10 every 7 epochs and the network is trained for up to 30 epochs. Our momentum parameter is also set through cross-validation to 0.9. We do not use dropout, following the practice in [8].

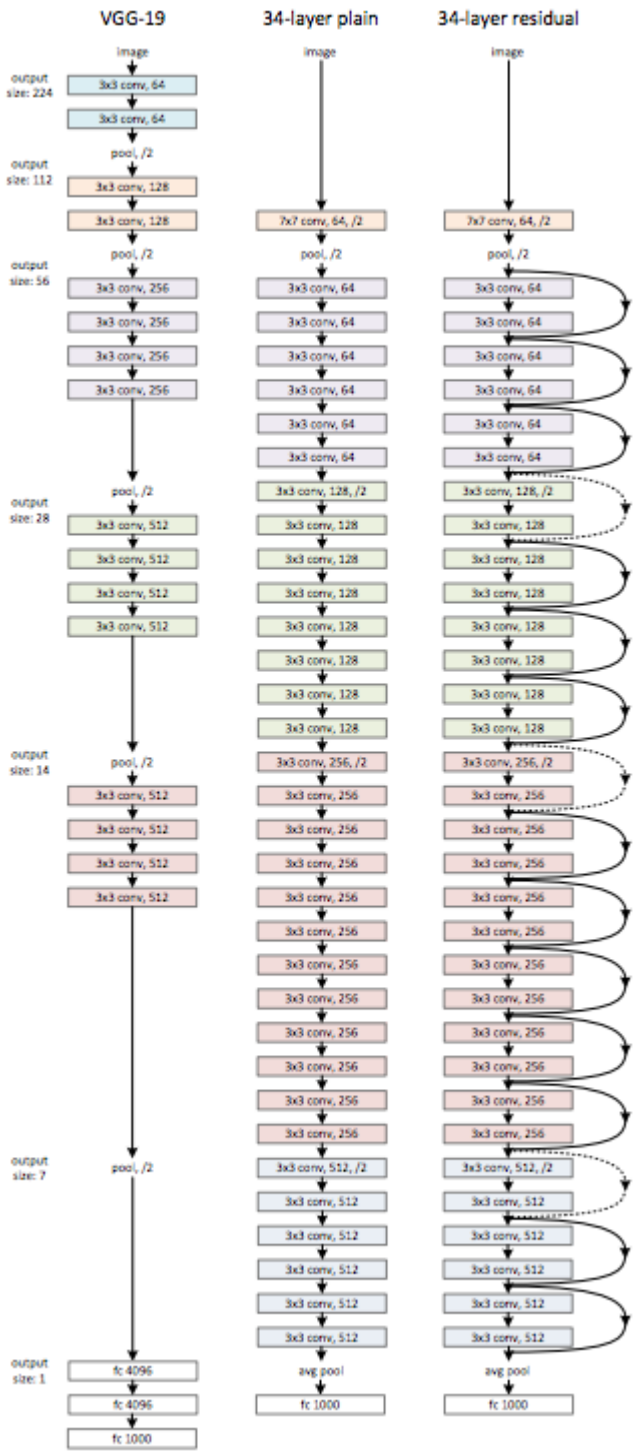

Figure 12. VGG and ResNet architectures. Figure is borrowed from [8].

Since our dataset is of small size, we employ data augmentation techniques to artificially enlarge it, as it has been shown that this can lead to a significant performance

Network Depth - ResNet (fine, 0.01, 7, 0.1, 0.9) - rotate+jitter				
	Train Accuracy	Validation Accuracy	Training Time	Epochs
ResNet18	0.8952	0.8645	4m 28s	30
ResNet34	0.9922	0.9377	6m 51s	30
ResNet50	0.9951	0.9524	8m 47s	30
ReNet101	0.9941	0.9414	12m 42s	30

Figure 13. Training and validation accuracy for various depth levels.

boost [13]. However, when performing data augmentations we have to ensure we preserve the structural invariances of our objects, which in this case are all playing cards from a standard 52-card deck. In particular, one of the most distinct characteristic of playing cards is the fact that the rank and suit of the card can always be found on the top-left and bottom-right corner. For this reason, we cannot use the horizontal and vertical flip techniques suggested by [13], as they violate this invariance. Instead, the data were augmented using rotations, random 224x224 crops and color jitter. All of these transformations preserve the card invariance just described. This artificial enhancement of the training set led to a significant boost performance, as conjectured by [13]. Thus, in all future experiments, we used the augmented training set for training.

To identify the best architecture depth, we train and validate ResNet architectures of various depth levels. For this experiment, the network weights are initialized using transfer learning on the ILSVRC classification task, as described in [10], and only the last few layers of the network are fine-tuned during training. The training and validation levels of accuracy of this experiment can be found in Figure 13. The results seem to indicate that the best tradeoff between performance and training time is achieved by the ResNet34 and ResNet50 architectures. We decided to go with the latter because it achieved a higher validation accuracy at a very small training time increase from ResNet34.

During this experiment, we observe two things. First, the loss converges really fast to a really low level during training, as evident in Figure 14. Although the eventual convergence of the network to a low level of loss is a desired behavior, the fact it happens so fast, over the course of fewer than 10 epochs, is an indicator that we might be overfitting to the training set. This hypothesis is reinforced from the fact our training accuracy is consistently 4%-5% higher than our validation accuracy. This difference should not be that big given both training and validation samples originate from the same CARDS subcomponents. Interestingly

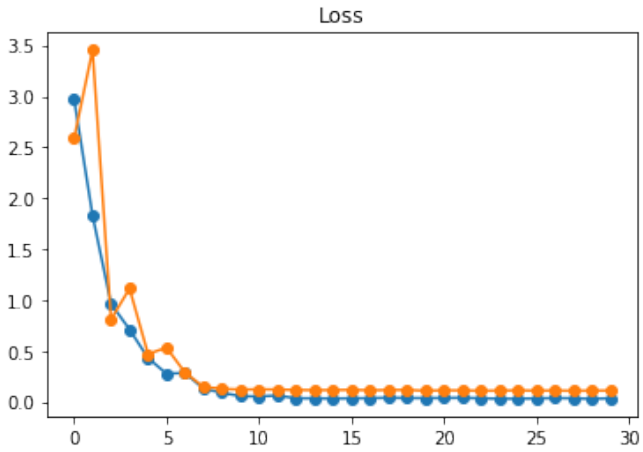

Figure 14. Loss over number of epochs for ResNet50 training. The blue line represents the loss during traing and the orange line represents the loss during validation.

enough, similar behavior was exhibited with all 4 architecture depths we experimented with. This is an indicator we are not making the most out of the architectures we use due to having a small training set. The best way of working around this would be to expand CARDS.

5.3. Performance Evaluation

We test our neural network implementation on the Complex-Bridge subcomponent to be able to perform a meaningful comparison against our baseline. We use the ResNet50 architecture with three different types of initialization. **Random Init** is our model with its weights randomly initialized. **Pretraining** is our model with its weights initialized using transfer learning on the ILSVRC classification task, as described in [10]. **Fine-tuning only** is also initialized the same way. The difference between the last two models is that **Pretraining** is fine-tuned across all layers, while we fine tune only the last few layers of **Fine-tuning only** during backpropagation. All models are trained with the same (learning rate, decay period, decay rate, momentum) parametrization, which is specified at the top fo the table in Figure 15. All models take advantage of the data augmentation techniques described in section 5.2.

Figure 15 presents the results of our performance evaluation across all three models. In particular, we observe that transfer learning is essential, else our model attains very low levels of accuracy. In particular, **Random Init** achieves worse accuracy levels than our baseline implementation. This is very likely because our dataset is not large enough to allow the network to learn generic objectness features that have been pre-learnt by training the first few layers on ImageNet [4, 10]. Thus, we conclude transfer learning is indeed essential for this task.

Initialization Analysis - ResNet50 (0.01, 7, 0.1, 0.9)					
	Train Accuracy	Validation Accuracy	Test Accuracy	Training Time	Epochs
Random Init	0.2831	0.3323	0.0697	16m 4s	30
Pretraining	0.999	**0.9231**	**0.8673**	16m 13s	30
Fine-tuning only	0.999	0.8974	0.7126	8m 43s	30

Figure 15. Test-time evaluation and weight initialization comparison using ResNet50.

If we focus on the other two models, we see that we achieve 86.73% and 71.2% precision, with **Pretraining** achieving the better performance. However, in both cases the test-time accuracy is significantly lower than the validation and training accuracy. This validates our hypothesis that we are overfitting, which is expected when training deep networks on relatively small datasets, like the CARDS dataset. However, it is very important to keep in mind that our test set is harder than our training and validation set, due to Complex-Bridge containing card instances from various decks and richer table environments, which leads to card instances that are harder to classify than the ones found in Single-Card and Simple-Bridge. Thus, we naturally expect lower test accuracy than validation accuracy. This implies that although the there is a 13% drop in the accuracy between training and testing on our best performing model, the effective accuracy drop is smaller than that, as a portion of 13% is a result of increased difficulty in the testing instances. Thus, the degree of overfitting is smaller than it initially looks, despite still being significant.

Comparing against our baseline implementation, we see a huge performance increase from 38.09% classification accuracy to 86.73%. Not only that, but the inference time of our deep learning implementation is significantly lower. Specifically, the per-frame classification rate on a CPU is 0.19 seconds. Therefore, deep learning outperforms traditional computer vision techniques in accomplishing this task both in terms of time-efficiency and in terms of lower generalization error.

Nevertheless, the deep learning system is not ideal. In particular, we have identified two failure modes that are the main sources of error: 1) our deep learning model has a hard time distinguishing between the highly similar figure cards and 2) our deep learning model is not as good at classifying correctly object classes that have very few training instances compared to similar object classes. Examples of (1) are illustrated by the red highlights in Figure 16. The model seems to predominantly predict Queen of Diamonds over Queen of Hearts, Queen of Spades over Queen of Clubs, Jack of Diamonds over Jack of Hearts, and Jack of Spades over Jack of Clubs, which results in this huge discrepancy in classification accuracy within each of those object class pairs. This is very likely due to instances of each pair of classes being very visually similar to each other at low resolution, making it really hard for the model to tell the difference for each other. For example, if one compare the Queen of Hearts to the Queen of Diamonds, one observes that there are very few fine details on the figure that make the two cards distinct.

An example of the second model weakness is the pair of the 9 of Clubs, with only 9 instances and significantly lower accuracy than the rest of the classes with rank 9, and the 2 of Diamonds, with only 12 instances and a lower classification accuracy than the rest of the classes with rank 2, both highlighted blue in Figure 16. However, we also observe odd levels of low classification accuracy for classes with a large number of instances in the training set, such as the 10 of Hearts, with 66 instances but 0% accuracy. We unfortunately can not explain this behavior, other than using the fact that each object class has approximately 9-10 instances in the test set on average due to the small size of CARDS, which could have led to a non-representative test-time evaluation for some object classes.

6. Conclusions and Future Work

This project is a very important first step towards solving the problem of automated card tracking for bridge game broadcasting. Our work specifically made three major contributions towards the completion of this task. We introduced the CARDS dataset, the first ever card detection and classification dataset that can serve as a valuable source of data to train future models on and great baseline to improve upon for the creation of a larger, more general card detection and classification dataset. We developed an algorithmic baseline using traditional computer vision techniques that achieved high detection accuracy in simple, restricted setups that could be replicated in practice given enough restrictions on the way professionals play. Finally, we trained a deep learning implementation that converges fast to high card classification accuracy (86.73%), despite the fact it is trained on a restricted amount of data. Thus,

ResNet50 - Classification Accuracy per Object Class													
	2	3	4	5	6	7	8	9	10	Jack	Queen	King	Ace
Clubs	100%	100%	89%	48%	100%	100%	43%	78%	100%	56%	0%	100%	100%
Hearts	100%	82%	90%	80%	100%	88%	100%	95%	0%	11%	0%	95%	92%
Diamonds	85%	100%	75%	0%	67%	60%	75%	100%	100%	100%	96%	75%	100%
Spades	90%	0%	100%	100%	100%	97%	25%	100%	100%	90%	67%	30%	100%
Training Set - Instance Count per Object Class													
	2	3	4	5	6	7	8	9	10	Jack	Queen	King	Ace
Clubs	16	20	23	14	12	27	23	9	13	10	38	16	57
Hearts	45	23	21	19	41	16	48	14	66	17	19	18	18
Diamonds	12	23	18	27	15	29	27	32	32	20	14	10	22
Spades	41	22	36	14	26	25	50	21	32	35	30	11	27

Figure 16. Class-based breakdown of test-time performance of the ResNet50 model.

we showed the problem at hand is definitely tractable and we have setup the framework necessary for the final steps to be taken towards its solution.

The next steps to be taken would be to train a card detection network using the R-CNN family of architectures, which are the state of the art systems for object detection [7]. To avoid overfitting, we would also need to expand CARDS. The best way to scale up would be to crowd-source annotations using Amazon Mechanical Turk. Since we have already implemented a bounding box annotation suite, this should not be a particularly challenging task. To improve detection performance further and potentially speed up inference, we can enhance the detection system with bridge game heuristics that would allow the system to be aware of the bridge game rules and the predetermined deal of the hands. This would enable the system to predict the next legal play without necessarily running all frames through the network. Once the detection network is fully trained, all that is left is integrating it with the online broadcasting system and test it in real time. This implies that inference times for detection need to be fast enough to allow for live streaming by the broadcasting system. Only a detection system with high accuracy levels and low inference times would be fully commercializable.

7. Acknowledgements

This work would have not been possible without the contribution of many admirable individuals. I would like to first thank my two advisors, Adam Finkelstein and Greg Humphreys, whose insights and experience taught me a lot about the fields of computer vision and bridge and enabled me to push through major roadblocks in my project by realizing alternative paths that allowed me to achieve much better results than I would have otherwise. I would like to thank Amber Lin, Shun Lam, Nathan Finkle, Andreea Magalie, and Thomas Fair for offering to play bridge for me so that I could collect the necessary samples to form the CARDS dataset, without which this project would have been impossible. I would like to thank my dear friends, Grace Guan, Matthew Li, Gordon Chu and Parker Kushima, who are there to back me up and cheer me up whenever things look grim. Finally, I would like to thank my beloved parents and siblings without the love and care of whom I would not be where I am right now.

References

[1] N. M. Ali, S. W. Jun, M. S. Karis, M. M. Ghazaly, and M. S. M. Aras. Object classification and recognition using bag-of-words (bow) model. In *2016 IEEE 12th International Colloquium on Signal Processing Its Applications (CSPA)*, pages 216–220, March 2016.

[2] Contact Bridge. Contact bridge — Wikipedia, the free encyclopedia, 2018. [Online; accessed 18-April-2018].

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, June 2005.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[5] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, Jan. 2015.

[6] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645, Sept. 2010.

[7] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[9] S. He and Y. Karakozis. Complex background subtraction using gaussian models. 2017.

[10] M. Huh, P. Agrawal, and A. A. Efros. What makes imagenet good for transfer learning? *CoRR*, abs/1608.08614, 2016.

[11] G. Humphreys. Series of interviews conducted by Ioannis Christos Karakozis; February, 2018.

[12] I. Katz and H. Aghajan. Multiple camera-based chamfer matching for pedestrian detection. In *2008 Second ACM/IEEE International Conference on Distributed Smart Cameras*, pages 1–5, Sept 2008.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[14] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[15] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004.

[16] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[17] M. Software. Chi square table, 2018.

[18] S. Suzuki and K. Abe. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

[19] P. Viola and M. J. Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57(2):137–154, May 2004.