Playing Card Classifier

Team members: Yang Hu
Ali Jahangirnezhad
Sonia Saravanan

Demo Presentation Link (videos included):
https://docs.google.com/presentation/d/1TVU9nxy17iW3zI4Y2phvjlFU48QrJESxoJkdN_HOqkg/edit#slide=id.gc6f90357f_0_13

Goal:

In this project, our goal is to detect a playing card from a camera feed, and classify its Rank and Suit in real time.

First, a set of references are extracted from images of the deck of the cards we are trying to classify. Then, the Suit and Rank symbols are extracted from each variation (4 suits and 13 ranks). These are stored as a set of references to compare the input to. The input is taken from a camera feed, and the algorithm is run for every frame on the camera. Using our work, we want to be able to classify the cards in the deck of cards, correctly.

The Process:

Populate the reference set:
A set of cleanly scanned cards are previously analyzed, and their rank and suit is extracted using the same extraction algorithm used in the project (extractor.h)

Isolate individual cards:
Every frame of the camera feed is analyzed using canny edge detection and finding contours, playing cards are detected (using the largest contours, in order to bypass the smaller contours around other elements on each card).

Extract Rank and Suit information:
Each incoming frame is converted into grayscale image and passed through Canny edge detection with adaptive threshold. After that, we find all contours in the top-left one-fourth of the image and generate polygons surrounding each contour. Finally, the contours representing the rank and suit images are filtered out and used to generate final sub-images used for matching.

Match with the reference set:
The extracted rank and suit images are converted to grayscale, blurred, and then thresholded to get binary B/W. Then, the absolute difference (absDiff) between the two images is calculated as a measure of difference between two images. The reference (rank or suit) that has the lowest difference, is chosen as the rank (or suit). There is also a safety maximum-difference put in place, as without this any meaningless contour would also be classified as something, so if the difference is more than an absolute value, it is meaningless for our classification.

Classes and relevant Fields and Public Methods:

Extractor:
A class to extract rank and suit from a card.

Fields:
Rank: The extracted Rank number or letter from image of a playing card
Suit: The extracted Suit symbol from image of a playing card

Methods:
Extractor(bool showResult); Constructor of the Extractor class. The variable showResult is to indicate whether we want the results displayed or not.

Bool extract(Mat input): This method takes in an image (a flat image) of a playing card, and extracts the corner Rank and Suit symbols and saves them as class fields.

Matcher:
A class to calculate the difference between input and reference images, and to decide on a good match for each extracted rank and suit.

Important Fields:
```
// threshold for converting to binary
const int BW_THRESHOLD = 128;
// an error threshold for Rank - if the error exceeds this amount
// then the classifier has failed to classify the image
double MAX_ERROR_RANK;
// an error threshold for Suit - if the error exceeds this amount
// then the classifier has failed to classify the image
double MAX_ERROR_SUIT;
// rank images previously extracted, as reference.
```

```
// read in the same order as NAMES
Mat ranks[13];
// suit images previously extracted, as reference.
// read in the same order as SUIT_names
Mat suits[4];
```

Methods:

Matcher(double maxErrRank = 2000, double maxErrSuit = 1000): Constructor for the class, takes in the maximum tolerated absdiff value for clarifying. Any error above the input errors is not a successful classification.

readTrainFiles(); Reads and populates the reference arrays of images for suits and ranks.

Double matchValue(const Mat& test, const Mat& input): This method calculated the absolute difference between the black and white images of input and test images, in order to be used for classification. The value this function reserves is a metric in quantifying the difference between images.

String getBestMatch(const Mat& input, bool isRank): Finds the best match for the input image from the reference images. The boolean “isRank” indicates whether we are comparing ranks, or suits (in order to have the proper output, and to use the same method for both purposes of classifying suits and ranks). This method uses the max errors previously defined.

Accomplishments:

We were able to successfully establish a pipeline where every frame of the camera feed is analyzed using edge detection and finding contours, playing cards are detected (using the largest contours, in order to bypass the smaller contours around other elements on each card). Then the rank and suit is extracted using Extractor, passed on to the Matcher, and the results are shown.

Lessons Learned:

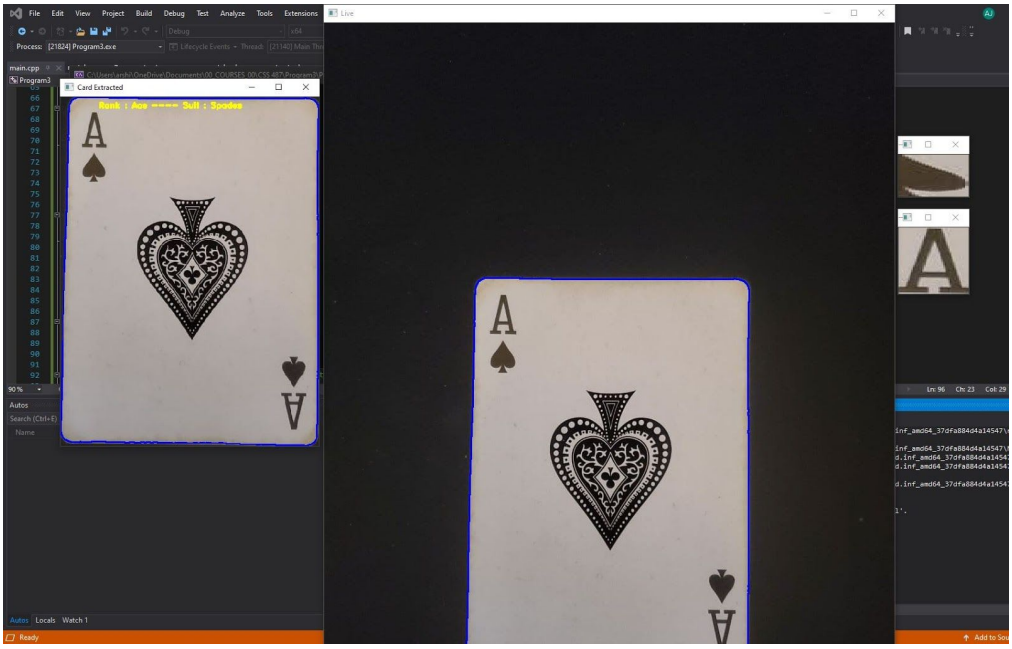
Without using convolutional learning algorithms, classifiers such as the one implemented in our project, are prone to errors caused by (among other things) camera quality, camera noise, hand shake, lighting, background images, etc.
Since a classifier like ours works analytically, which is to say it is designed to compare a given input to a set of references, any significant deviation from this set of references would result in a possibly wrong output.
In addition, we got to experience firsthand the effects and outcomes of using a camera feed with OpenCV, and how the dynamics of using a camera (instead of a stationary image) changed how our algorithms worked (things like random contours popping up, etc.).
What we took away from this project is in short “How to use OpenCV and Computer Vision algorithms in order to achieve a specific task”. It gave us a perfect understanding and experience in working with OpenCV without a set of instructions to follow. This helped us not only understand the concepts better, but also to understand the real ramifications of using them.

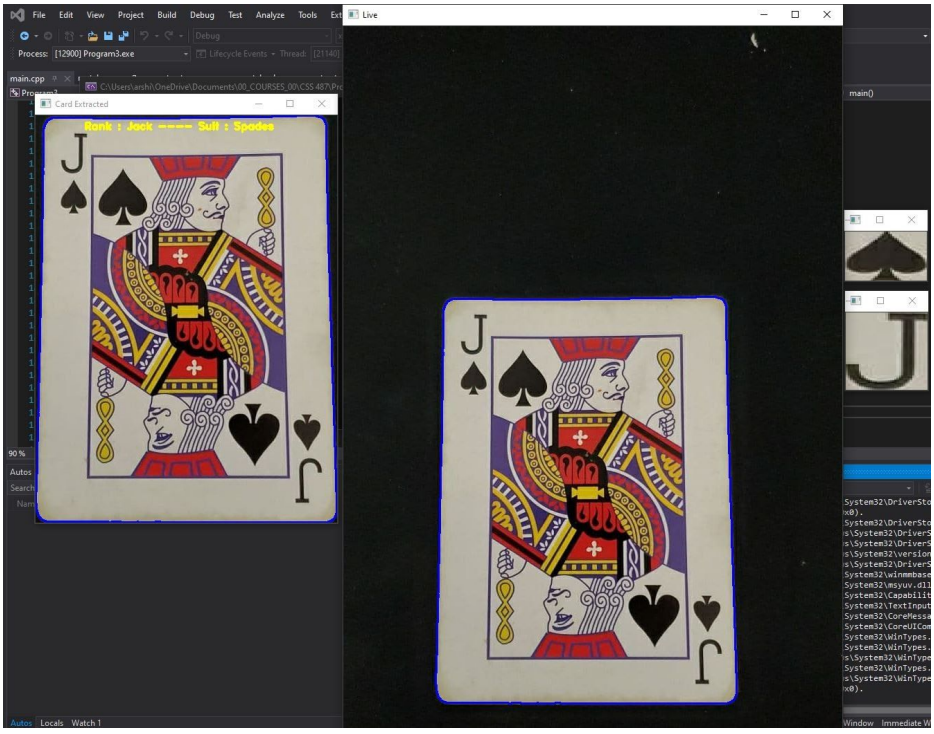
Results:

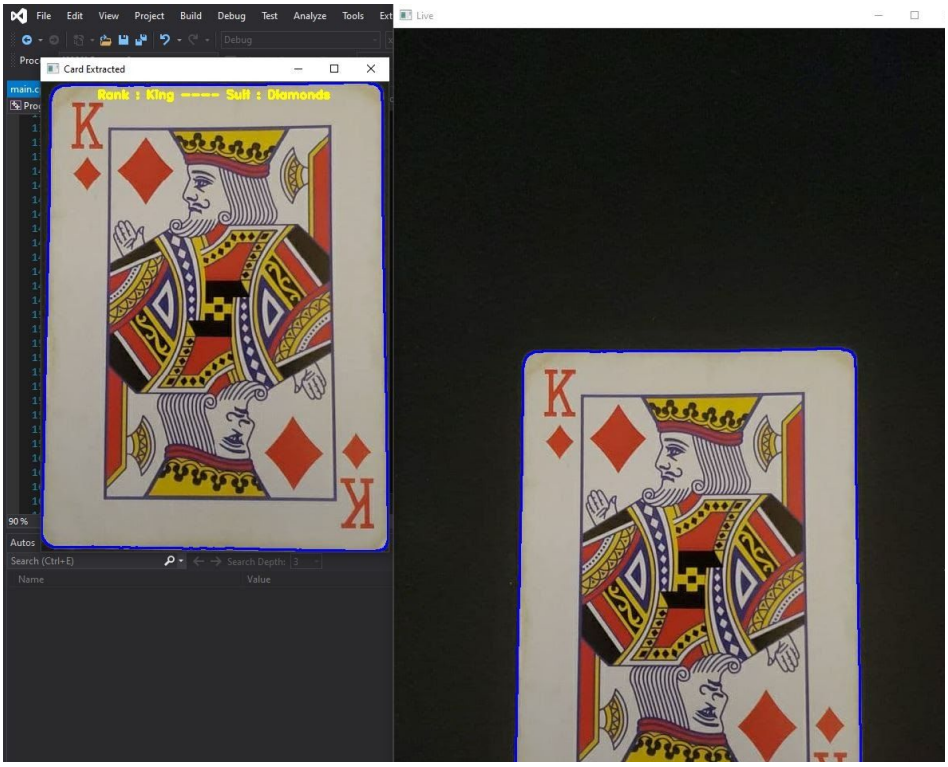
With a high quality camera, the results are almost perfect. The biggest miss that happens in this algorithm is when it mistakes Diamonds for Hearts. This is because of the absDiff algorithm, and that Diamonds and Hearts take about the same area of pixels with the same color intensity, that makes the absDiff return values close to each other for these. But, if the card is set in alignment with the frame, and the camera is perpendicular to the card, it is correctly classified, as you can see in the next images.

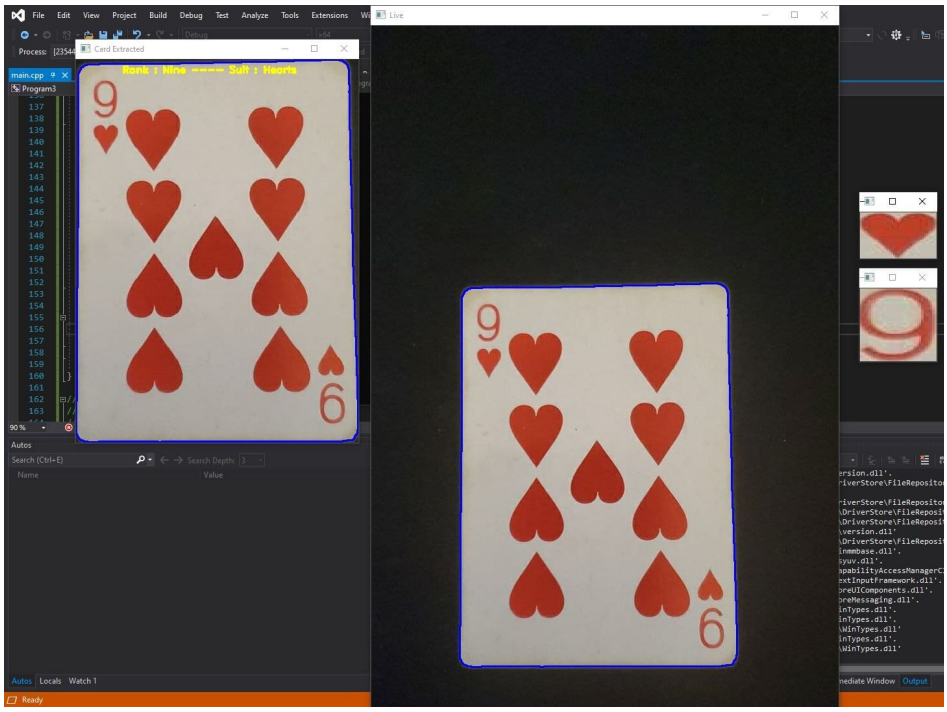
Another important result was the use of 0 in 10 for classifying 10s. Instead of scanning the whole number of “10”, we used the uniqueness of number “0” in order to classify 10. (last page shows a sample miss of misclassification of Diamonds)

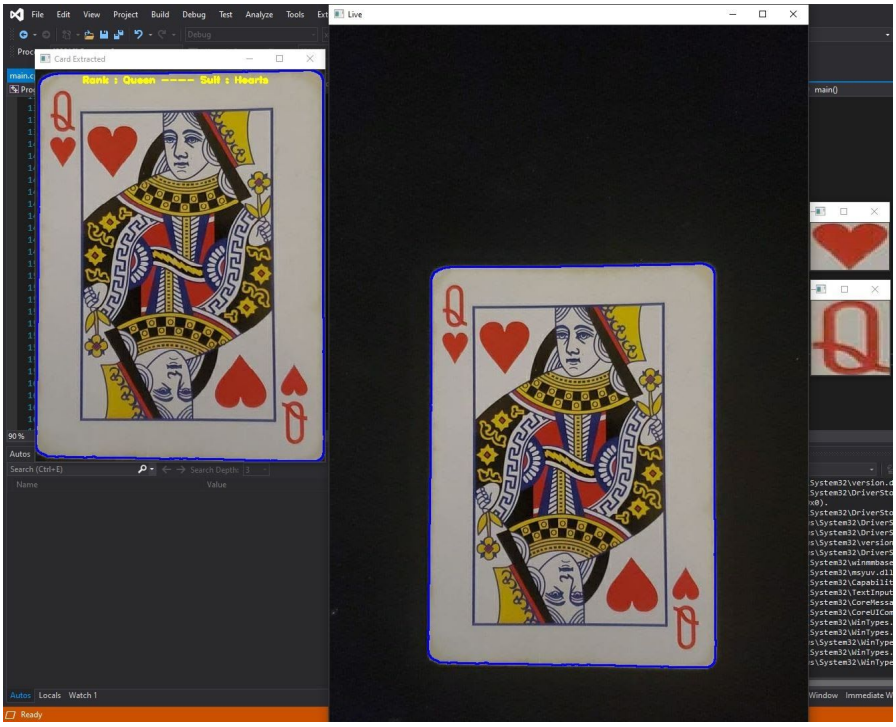
Hits:

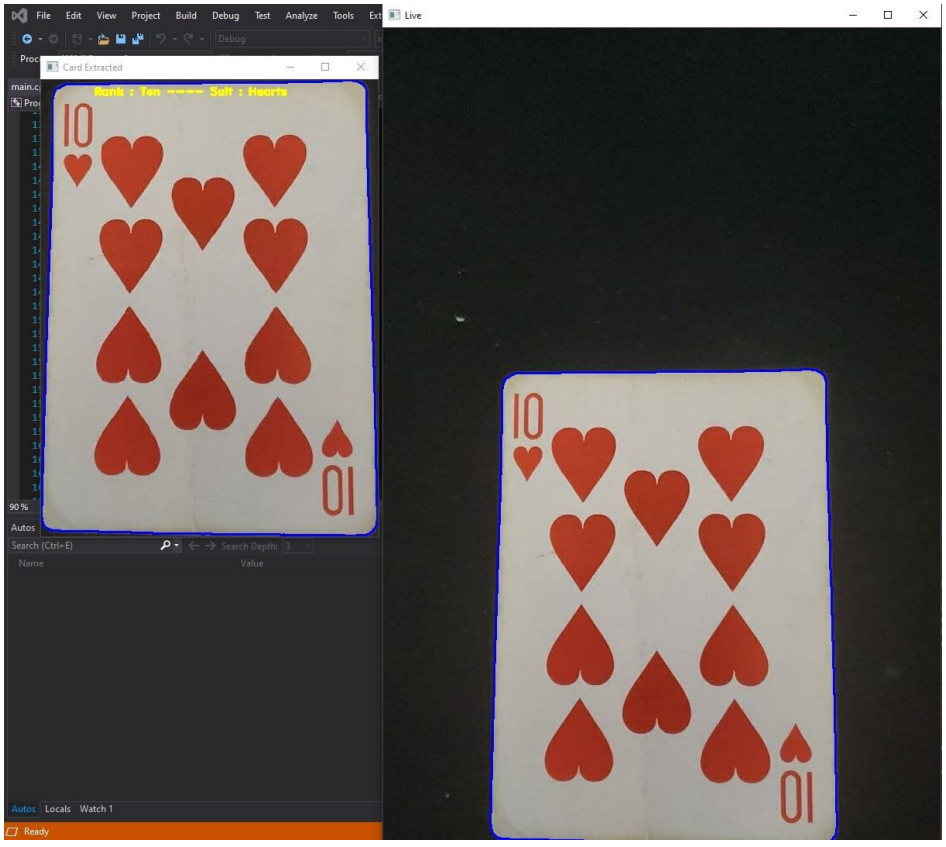

Card Extracted
Rank : Ace ---- Suit : Spades
Live

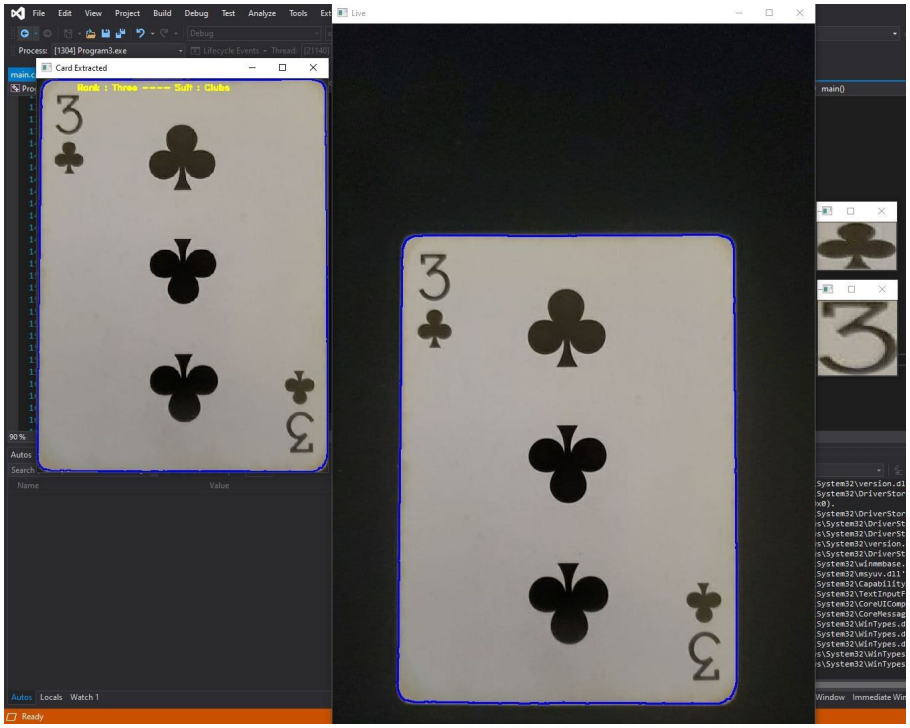

Card Extracted
Rank : Jack ---- Suit : Spades
Live


Card Extracted
Rank : King ---- Suit : Diamonds
K
K
Live
K
K

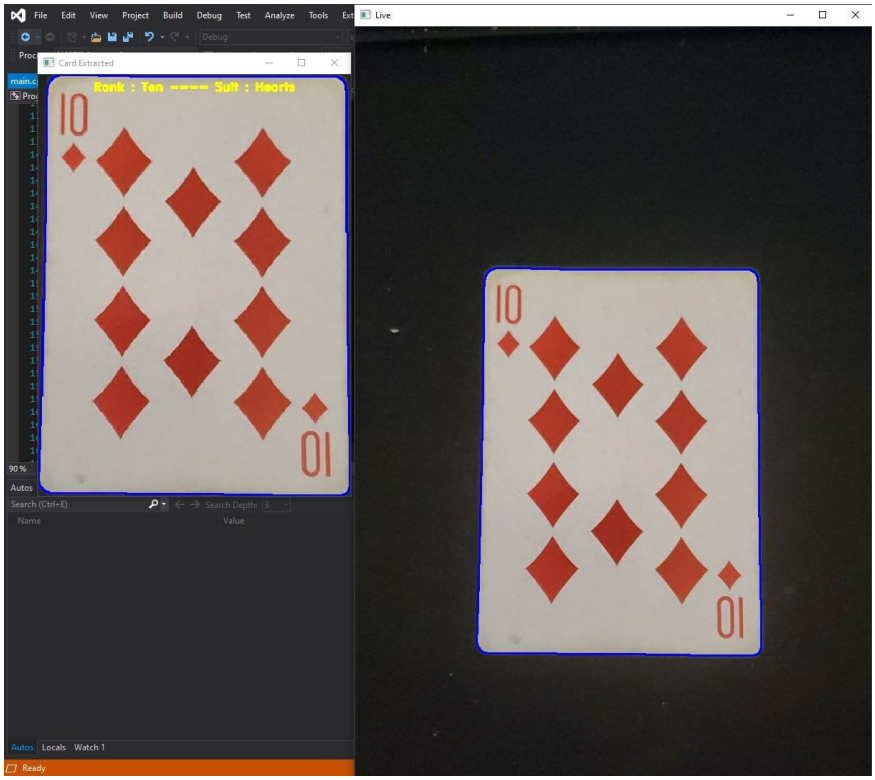

Card Extracted
Rank : Nine ---- Suit : Hearts
9
6
Live
9
6


Card Extracted
Rank : Queen ---- Suit : Hearts
Live


Card Extracted
Rank : Ten ---- Suit : Hearts
Live


Card Extracted
Rank : Three ----- Suit : Clubs
Live

Sample Miss:


Card Extracted
Rank : Ten ----- Suit : Hearts
Live