Final Project Report for CS 175, Spring 2018

Project Title: Rock Paper Scissor AI

Project Number: Group 24

Student Name(s)
Kyle Ingwersen, 62486692, kingwers@uci.edu
Lee Phan, 37738425, lhphan@uci.edu
Lei Pan, 64422004, leip1@uci.edu

1. Introduction and Problem Statement (1 or 2 paragraphs)

[This can be similar to what you wrote in your proposal or progress report]

This project involves detecting valid rock, paper, scissors moves done in real time. A camera was used to record a person making gestures for the game. The player would shake their fist three times and play either rock, paper, or scissors when their fist reaches the lowest point for the third time. The classifier watched this in real time. For every frame, a label of “None”, “Rock”, “Paper”, or “Scissors” is displayed. “None” is produced if the user hasn’t shaken their fist three times. “Rock”, “Paper”, or “Scissors” is displayed after the three fist shake is seen. The challenge of the project will be to find a balance between differentiating fist shakes from playing rock and to allow for playing at different speeds.

Two convolutional Neural Networks (CNN) models were used in order to accomplish real time move detection. One model is used to detect the moment a valid move has occurred, and a second model was utilized to classify the valid moves as "Rock", "Paper", or "Scissor". Using these approach, we were unfortunately unable to produce desirable results.

2. Related Work: (1 or 2 paragraphs)

A group from Slovak University of Technology attempted to solve the project's problem with the use of Kinect camera [1]. By tracing the right hand's movement and comparing each frames, they were able to determine if the hand was moving up or down by subtracting the distance between the right hand's points (Fig 1). After the end of the third downward motion, classification of "Rock", "Paper", or "Scissor was done by calculating the number of convex defects in the hand, where 0-1 defects means rock, 1-4 means scissor, and more than 4 means paper (Fig. 2). The results of their experiment showed several issues. The 3 fist shake detection heavily relied on hand tracking, which could fail at multiple points during the shaking motion. Distance is also an issue because far distance meant low resolution hand image, which lead to inaccurate counting of the convex defects. Another problem encountered was the fact that they were unable to distinguish between rock and scissor when 1 convex defect is seen (Fig. 3).

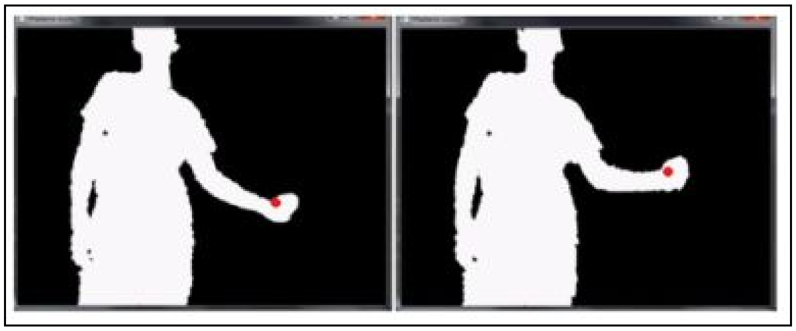
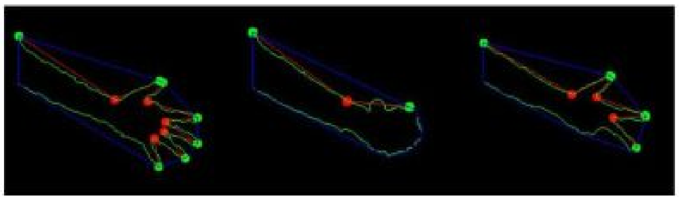
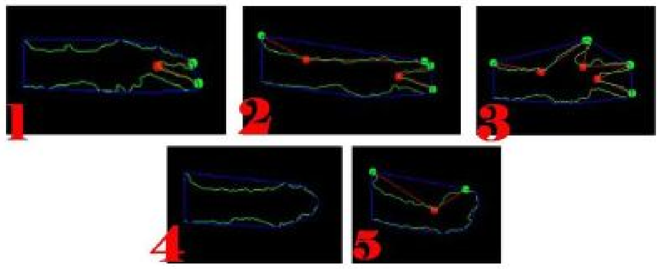


Fig. 1 Determining up or down movement by taking vertical difference on hand between two frames

 **Fig. 2 Counting convex defects (Red = convex defects, Green = convex hull)**	 **Fig. 3 Problem distinguishing imgs with 1 convex defect (img 1 and 5)**

Our project steered away from using hand tracking to detect valid moves by using CNN with a series of frames. The distance problem found in the discussed experiment was solved by utilizing CNN classification with a dataset. This approach is less susceptible to blurry images, and it does not have any issue distinguish between rock and scissor.

3. Data Sets

A new data set was built by using a webcam to record a video of a person doing the valid rock-paper-scissor with a white static background. 50 rock, 50 paper, and 50 scissor gestures have been collected using a single person's hand. Then the frames were extracted from the videos (Fig. 4).

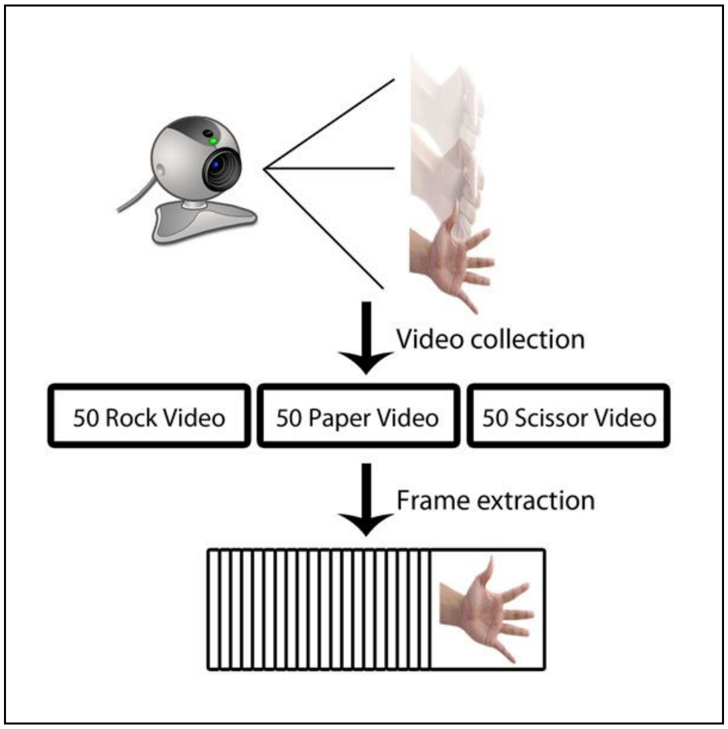


Fig. 4 Collecting dataset

Each frame was assigned one of the following classifications: None, Rock, Paper, or Scissors
Only after the third fist shake ends and before the performer lifts their hand again will the frame be classified Rock, Paper, or Scissors; all other times the frame is classified as None (Fig. 5). The gestures were performed at a variety of speeds to provide flexibility in the trained model. Each gesture has approximately 3000 frames of images. Data preprocessing involves labeling each image according to the above classification. Additionally, if hand is not properly occurred in the image, it is labeled unqualified. Unqualified images are then filtered and will not be used as data input.

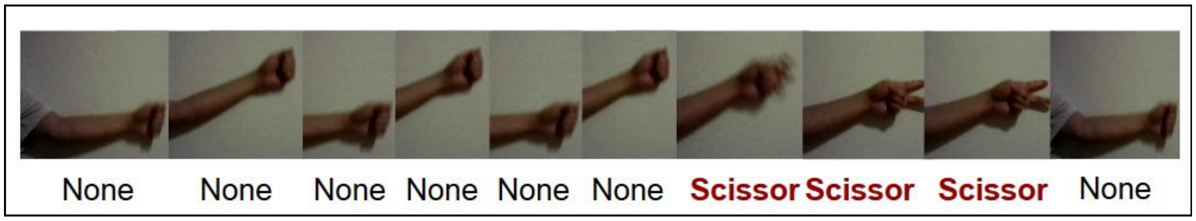


Fig. 5 Labeling of dataset frames

Image augmentation is applied on the dataset to extend the amount of data. The reason why we need image augmentation is that any state-of-art neural network typically have input in the order of millions while the data we collected is in the order of thousands. As with data labeling, each set of 60 frames is treated as one unit and is applied with a series of image augmentations. A set of randomized cropping, rotation, adding brightness and adjusting contrast are applied on the images before they are passed into the CNN model

4. Description of Technical Approach [at least 1 page]

The overall flow of this project can be seen in Figure 6. The underlying application uses a video camera to fill a fixed-size, rotating queue of grayscale matrix image frames. At every frame, one convolutional neural network will view every frame in the queue and determine if three fist-shakes have been made; if true, a second convolutional neural network will classify the most recent frame as either “Rock”, “Paper”, or “Scissors”.

Our decision to split the model into two networks is based on a concern for complexity. Each network solves a different problem in a different dimension - the former across images, the latter within an image - and we believed that attempting to combine these problems into one large network would force its complexity to be proportional to the product of the complexities of each of the component parts. Given we want to run this model in real time, we didn’t consider this to be a practical approach.

Initially we decided to use recurrent neural network (RNN) in the first model as the task involves dealing with time-series information. Through research, we were able to build and train a prototype RNN model. In analyzing the performance between CNN and RNN, it turned out that the accuracy of RNN was worse, despite taking a significantly longer time to train. We searched for an explanation for these results and came to the conclusion that CNN fits better in the context of image classification, even if the task involves dealing with a time series problem. The underlying reason that CNN can do a good job on this problem is that if we take a batch of 45 frames as one unit of input data, the time-series information is then embedded into the input itself.

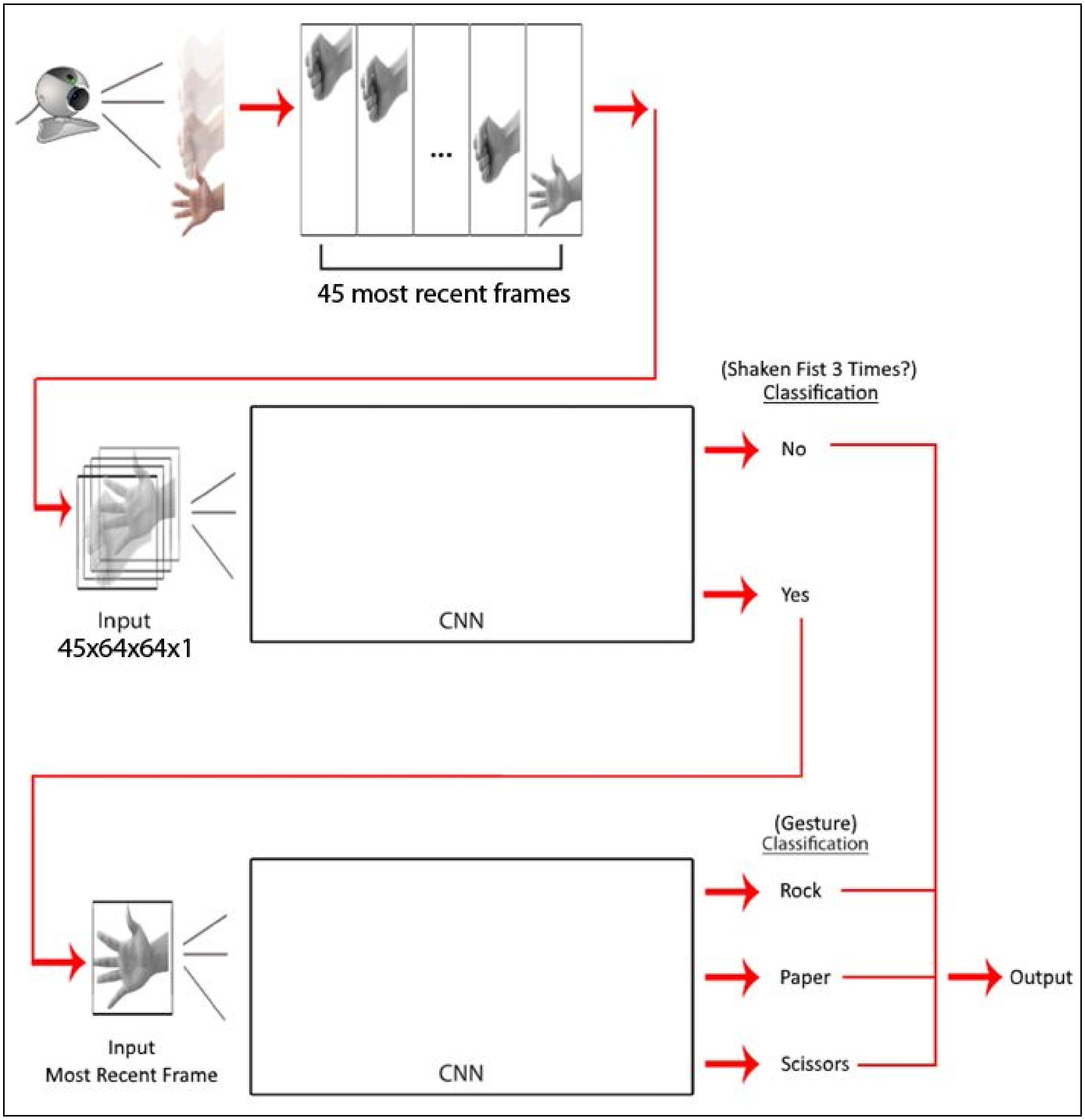


Fig. 6 Overall flow of this project

4. Software [at least ½ a page]
a) Code written

Code	Description
Video collection script	Runs webcam and writes video files for data collection
Main program that detects player's move in real time	UI created that runs webcam to detect player's move in real time. It redisplay what the webcam is capturing. It also uses trained models to display the appropriate labels generated by player's movement.
FPS synchronization	Synchronizes frames rate between dataset collecting program and main detector program
Augmentations	Applies random cropping, flipping horizontally, rotation, brightness, and contrast to dataset
"Model1"	Convolutional Neural Network model built to identify when the user has shaken their fist three times (true or false)
"Model2"	CNN built to classify move that was played (rock, paper, or scissor).

b) Outside Software/Code

Software/Code	Description
Virtual Dub	Program that takes in a video file as input and outputs each frame as image files.
Tkinter library	Used for UI in video collection program and main player's move detector program
Opencv library	Uses webcam to record and write video files
Template for UI	Skeleton code for UI. It displays a simple UI and displays what the webcam is capturing. https://solarianprogrammer.com/2018/04/21/python-opencv-show-video-tkinter-window
Tensorflow	Neural-Network focused mathematics library that uses dependency trees to perform mathematical operations efficiently. Used to implement neural network models and video augmentation functions.

5. Experiments and Evaluation [at least 1 page]

To test our models, we built two custom k-fold cross-validation functions in Tensorflow, one for each model. One function was designed to build batches of videos, while the other was designed to build batches of images. In both, the k-fold quantity was set to 5, which meant 80% of our data was used for training and the remaining 20% was used for validation. First, we split the entirety of the data (50 shakes rock, then paper, then scissors) into five equal parts. Next, for the video model ("Model1") we built a random permutation of all possible videos in the training set (1, … , n-video_length), or for the image model ("Model2") we built a random permutation of all images (1, ... , n), and iteratively pulled batches from it. These batches were randomly augmented and then used to train and validate our models. We repeated for several epochs, and then repeated for several folds. The results were then displayed in several graphs. Unfortunately, we realized too late that we made a mistake in our implementation: we split the data into fifths before we randomized the inputs, which resulted in the training data not being representative of our validation data. For example, our model might train on rock and paper data, but is validated on scissors data. Our validation results were surprisingly low, and we realized too late that this was the problem. We didn't have time to go back and correct the problem.

To train our models, we simply used a subset of our cross-validation function. Instead of splitting the data into folds and repeating k times, we just used all the data and train once. The data is equivalently randomly augmented before it is inputted into the model. The results of this model were saved to disk where it can be loaded by our real-time applications for real-time prediction.

Our augmentation scheme changed over time following issues with the model and with Tensorflow's interface. Our initial plan was to perform random zoom, crop, brightness, contrast, flip, and rotation on every image or video while retaining red-blue-green color. Unfortunately, there were very few video augmentation libraries, and we ran into a lot of difficulty implementing these. It took an unreasonable amount of time to manipulate existing image libraries to work with videos, particularly Tensorflow. We ultimately got crop, brightness, contrast, and flip to work, but not zoom or rotation. We also later decided to use Canny edge detection as our image input instead of RGB, so brightness and contrast were dropped. In the end, our images were only cropped and flipped before they were edge-detected and trained on. Validation data and inputs to our model in the real-time application were uniformly cropped and edge-detected.

The results we found were very poor, yet we can't explain why. Any specific version of our models we trained got above 90% training accuracy on augmented data within an epoch, yet when we validate the models on practically the same data our accuracy goes to 50% on average - the equivalent of randomly guessing. Granted, as mentioned before, our validation scheme is broken and returns worse results than is actually worth. However, even when we tested the model in the real-time application using the same video that was used to train the model, the results were completely wrong. It's evidently a programming error, but after the time we spent on video augmentation, we haven't had the time to find a solution.

Our current video model ("Model 1") begins with two sets of individual, shared-weight convolutional layers on each image in the video. The convolutional layers are designed to reduce the size of the images while increasing features through filters. Each layer uses l2 regularization. The results of these conv layers are flattened, batch normalized, and put through dropout. Finally, the results are put through a short series of decreasing-dimensionality dense layers with regularization until the final softmax output (true/false). The back-propagation uses cross-entropy and Adam optimizer. We've tried increasing regularization or changing the structure a bit, but the difference in results is negligible compared to the random variation the output has anyway. If we'd realized our cross-validation function error sooner, we'd have made more progress on this front.

We also tried using recurrent neural network practices in our model, but we struggled to build a suitable implementation. Our vision was to have a series of dense networks watch varying widths of the same video from the most recent frame, backward. The fundamental idea behind it was that, in practice, our model's goal was to recognize whether or not a subset of the video contains the three shakes, not whether the whole video contains three shakes. If we could provide a series of varying-width videos for the model to predict from instead of the whole video, it might have produced better results. But after all our troubles, this was rather low on our priority list. We never finished this implementation. We left some traces of our implementation in models.py/model1_alt1().

Our current image model ("Model 2") begins with four convolutional layers that are designed to retain the size of the image while increasing the number of features. It then flattens and is put into several dense layers which end with a softmax (Rock/Paper/Scissors). We didn't put a very large amount of effort into this model because the problem it's trying to solve is relatively easy. In practice, it dangerously overfits and reaches 100% training accuracy on augmented data in a single epoch. Nonetheless, it produces totally wrong results in our real-time model using the same data but un-augmented, most likely due to programming error rather than model error. We've modified the model by changing regularization and dropout, etc., but again, none made a significant impact.

6. Discussion and Conclusion [at least ½ a page]

We experimented to detect a series of motions as well as static gestures. We learned from our experiments that the convolutional neural network (CNN) can do a good job on classifying a single image as long as we feed it with enough amount of data. However, using CNN on a time sensitive task, such as detecting a series of motions, has its limitation. We confronted this limitation by feeding the model with batches of videos, of which the smallest unit is itself a batch of images. The result is surprising. The model achieves an accuracy of 90% on training data, but it dangles at about 50% on the validating data. We suspected that the reason lies in the model being overfitted for the training data, so we applied the regularization to address the issue. The regularization had little improvement. It turns out that the CNN model did not meet our expectation.

As of a future direction of addressing the problem of detecting motions in a time series, we would suggest the research team focus on implementing a new feature for Tensorflow -- to allow the arguments being a clip of video. This will make a complement to the current framework since current APIs only allow batches of images as its argument.

Another possible direction is to combine the power of CNN with RNN. From what we researched, current studies on deep learning mostly rely on either of these two models, rarely are they combined to solve a single problem. It is true that these two models each have distinctive advantages on different tasks, such as CNN fits well in the context of dealing with images and RNN in time-series problems. However, in a problem that meets both conditions, such as motions in a time series, we expect that a mixed model will do a better job.

Overall, we experimented to solve detecting motions in a time series in the field of deep learning, and we ran into some limitation on the current software(Tensorflow) and approaches(CNN, RNN). As a future direction, we propose to extend the functionality of the Tensorflow framework and to infuse the power of one neural network model into the other.

Reference:

[1] Duchoň, František, et al. "USE OF CHILDREN'S GAMES IN ROBOTICS.", 2016