Rock Paper Scissors Using Machine Learning and Image Processing

Faizan Mansuri
Dept. of Electrical and Computer Engineering
University of British Columbia
Vancouver BC, Canada
faizanmansuri27893@gmail.com

***Abstract*—Image Processing has been a topic of great research in the past few decades. Using Machine Learning along with Computer vision results into many applications, which can be for entertainment, educational research or health and social benefits to improve human life. This paper describes the implementation details of developing rock, paper and scissors game using python and OpenCv on Raspberry Pi. This project uses image processing features like hue distance and gray scale thresholding to identify the gesture and play the game. With only 3 labels to choose from the project is implemented using Support Vector Machines and Principal Component Analysis, where the model determines the gesture in real-time based on the trained classifier. It is a robust model and can be extended to develop different apps for mobile phones.**

***Index Terms*—Computer vision, Hue Saturation Value Model, OpenCv, Principal Component Analysis, Raspberry Pi, Support Vector Machine.**

I. Introduction

IMAGE processing is an interesting topic in recent research field. This project involves use of hardware and software interacting with each other to develop a game using Pygame and OpenCv. Rock paper Scissor is a gesture-based game played widely by kids under the age of twelve to fifteen years. The game is played between two players. Each player shows a gesture from a set of rock as fist, paper as palm and scissor as a figure of two with their fingers. Based on their gesture rock wins over scissor, scissor wins over paper and paper wins over rock. On each win the player gets a point and lastly the player with maximum points win. This project implements the rock paper scissor game using Raspberry Pi as the processing unit, picamera as the camera module with a fixed green background to carryout the background subtraction in order to achieve the gesture recognition. I have implemented the complete game along with designing the pipeline to achieve the gesture recognition in real-time when a player plays against the computer. The computer gestures are determined randomly and the computer gets trained with each player moves making it difficult and challenging for the player to win.
The basic challenges in the project were to determine the image processing algorithms to obtain the feature extraction, and to design a support vector machine with higher confidence.

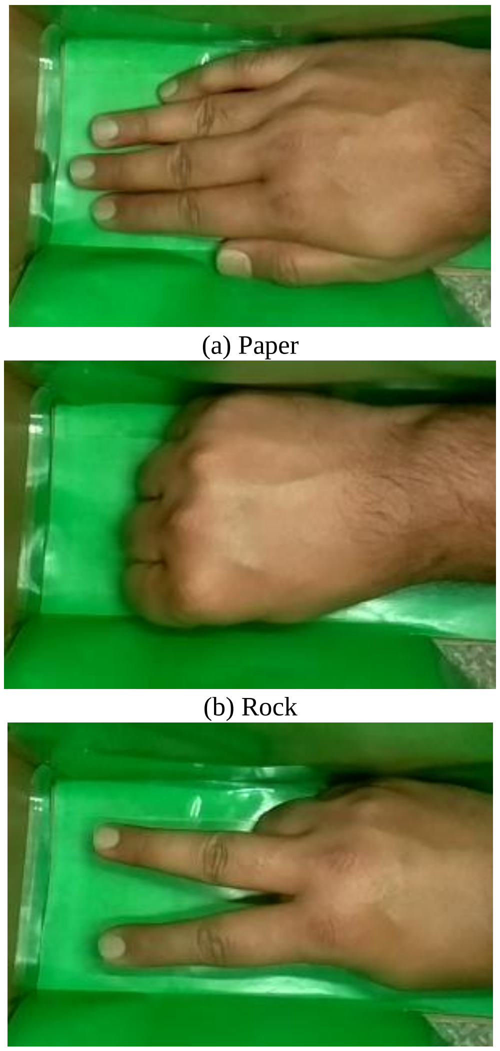

(a) Paper
(b) Rock
(c) Scissors

Fig. 1. Gestures used to Play Rock paper Scissors game

.

II. DATA GATHERING

This project involves the use of Support Vector Machine which determines the labels and classifies the gesture. To achieve this, I was required to gather a dataset with different types of hand gestures and classify them into subsets of rock, paper and scissor and then train the support vector machine using that. For my project, I developed a capture.py script which used the picamera in the same project environment and capture images with labels. I took help of my friends and neighbours to obtain and create datasets of hand gestures from people with different hand sizes, hand color and hands with certain deformities or anomalies to make my model more robust inorder to get an error free game.

My setup included a box of a fixed height with camera attached on the top and light sources which creates perfect image capture environment. Based on the capture script, when a user puts his hand under the camera and clicks a button, mainly ***r*** for rock, ***p*** for paper and ***s*** for scissor on the keyboard, the script saves the labelled image in the image folder with respective folder name so that the images gets separated based on labels which facilitates the training of the model.

I used about 600 rock images, 589 paper images and 650 scissor images to train the support vector machine.

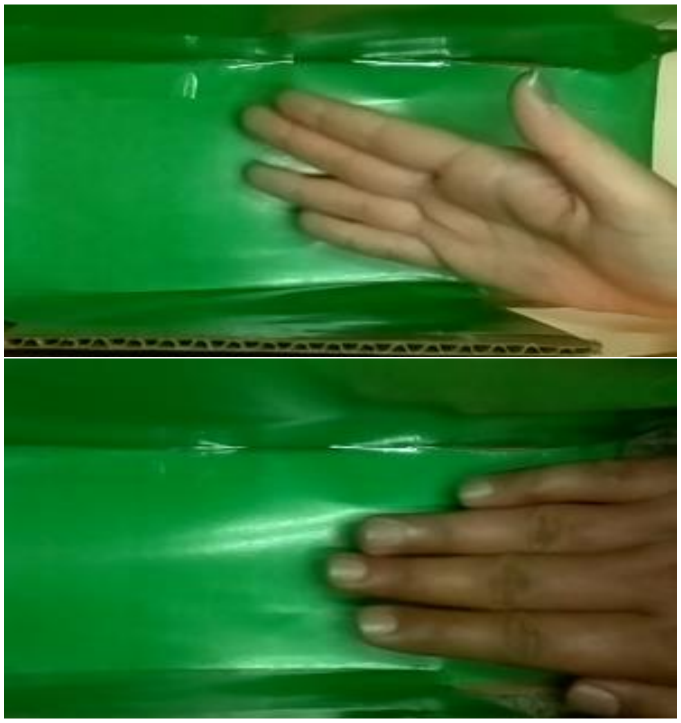


Fig. 2. Different Images used to train the Support Vector Machine for Paper Label.

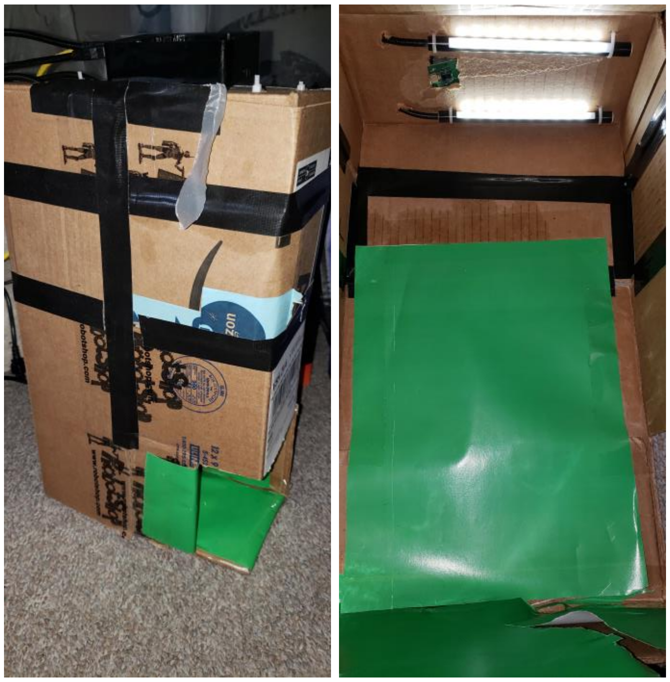


Fig. 3. Experimental Setup and Internal Image of the setup

III. EXPERIMENTAL SETUP

I used python libraries namely numpy and scikit learn to create a support vector machine trained with classifier and features extracted using image processing tools. The processing unit used is Raspberry Pi.

A. *Hardware Specification*

The project is designed for Raspberry Pi [1]. Raspberry Pi is a mini computer board with a linux operating system and GPU to run the game and run the image processing tools. Raspberry Pi Model 3B+ was used in this project. It is compact, high processing power and low cost making it best suitable for a low-cost game with image processing. Raspberry Pi supports python programming language and OpenCv which is an open source computer vision library has a python support making it best suited for this project.

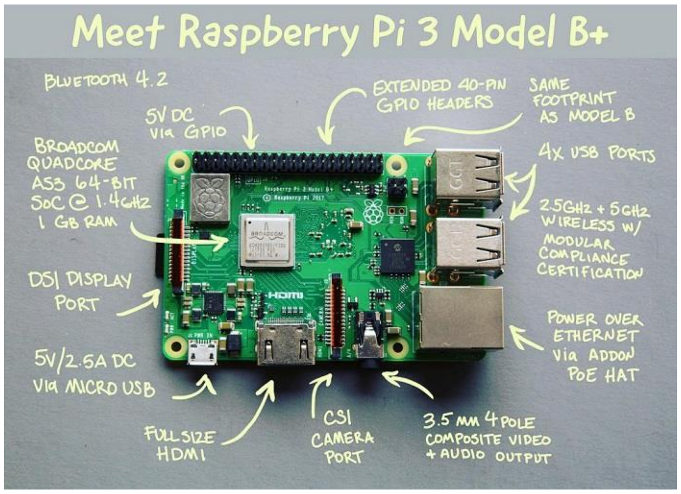

Meet Raspberry Pi 3 Model B+
Bluetooth 4.2
5V DC via GPIO
Extended 40-pin GPIO headers
Same footprint as Model B
Broadcom Quadcore A53 64-bit SoC @ 1.4GHz 1 GB RAM
4x USB ports
2.5GHz + 5GHz wireless w/ modular compliance certification
DSI display port
Power over Ethernet via addon PoE HAT
5V/2.5A DC via micro USB
Full size HDMI
CSI camera port
3.5mm 4 pole composite video + audio output

Fig. 4. Raspberry Pi 3B+ Mini Computer Board

Some of the Raspberry Pi 3B+ model:

- Quad core 64-bit processor clocked at 1.4GHz
- 1GB LPDDR2 SRAM
- Dual-band 2.4GHz and 5GHz wireless LAN
- Bluetooth 4.2 / BLE
- Higher speed ethernet up to 300Mbps
- Power-over-Ethernet capability (via a separate PoE HAT)

The camera module used in this project is PiCamera Version 2 [2].

Fig. 5. PiCamera attached to 3B+ RPI

Camera Specifications:

- 8-megapixel native resolution high quality Sony IMX219 image sensor
- Cameras are capable of 3280 x 2464 pixel static images
- Capture video at 1080p30, 720p60 and 640x480p90 resolutions
- All software is supported within the latest version of Raspbian Operating System
- 1.12 µm X 1.12 µm pixel with OmniBSI technology for high performance (high sensitivity, low crosstalk, low noise)
- Optical size of 1/4"

B. *Model Description*

This model comprises of a processing pipeline with image capture, followed by image feature extraction, then training and classification. Image capture is carried out as explained above in the data capture section.
Image feature extraction is carried out in three steps:

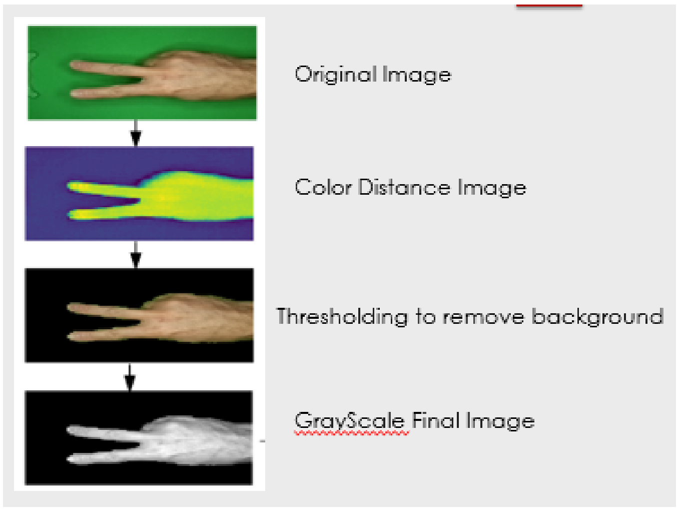


Fig. 7. Feature Extraction Pipeline

1) *Hue Distance*

Image color can be expressed as a Hue Saturation Value Model. Hue is the type of color namely, green, yellow, etc. Saturation is the brightness of the color, that is, amount of grey, or white or black in the actual color, e.g., light green and dark green. Value provides the lightness value of the color in HSV model [12]. This project makes use of the hue value to determine the gesture from the background with hue distance:

$$hue_{distance} = \min(abs(h1 - h0), 360 - abs(h1 - h0)),$$

where h1 and h0 are hue values of two datapoints on the image between which the distance needs to be calculated.

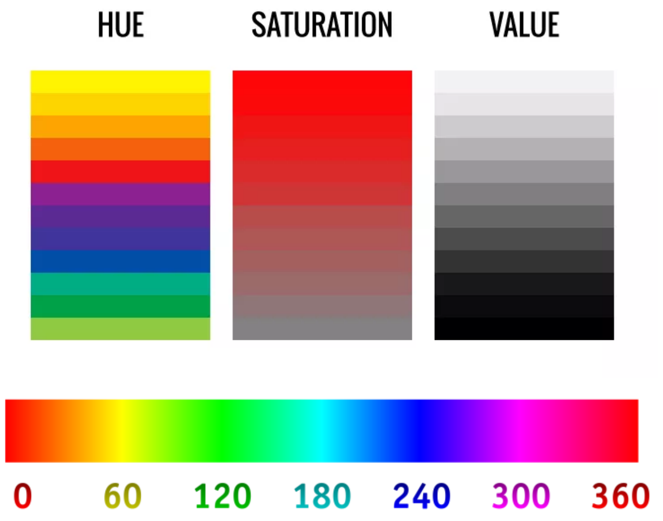


Fig. 8. Hue Saturation and Value Model

2) *Thresholding of Background*

In-order to achieve feature extraction, I used a histogram thresholding on the hue distanced image. I calculated the background hue value and using thresholding cleared the background color and made it black by setting the pixel value to 0. This helps in determining the gesture explicitly. The thresholding step generates an image with the hand on a black background.

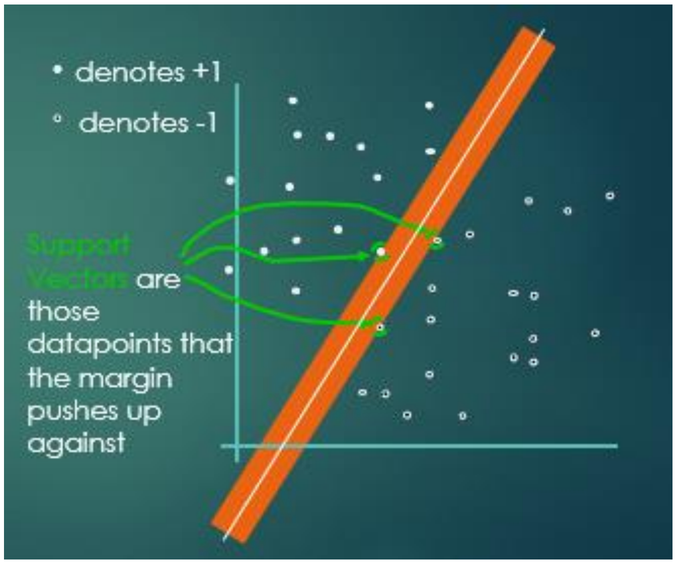
3) *Generate Grayscale Features*

This step is used to simply convert the image from the thresholding step into a grayscale image vector. This grayscale image vector is then fed into the support vector machine to train the classifier.

IV. SUPPORT VECTOR MACHINE

Support Vector Machine is a set of Supervised Learning Algorithm. It is used to determine a model which can separate a set of unknown inputs into the respective labels based on model classifier.

Support Vectors can be visualized as lines on a graph which can easily divide and segregate different data-points into different labels. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces [3].

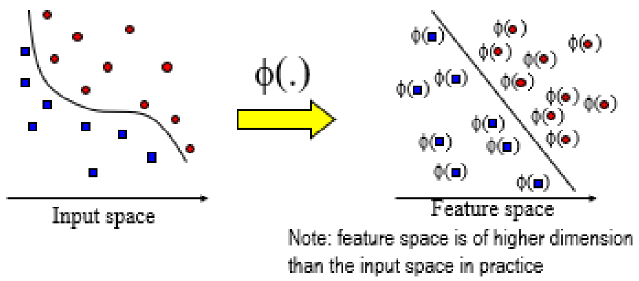

denotes +1
denotes -1
Support Vectors are those datapoints that the margin pushes up against
Fig. 9. Support Vector Machine

The data-points in this project are features which makes the vector machine non-linear and classifying non-linear dataset on a support vector machine is tricky, and time as well as resource consuming. The robustness and reliability of the predictions is reduced when a simple SVM is used to classify non-linear data. Hence, I have used a Kernel namely, *Radial Basis Function – Kernel*, to help convert the non-linear classification dataset into a linear classifier set which can then be used to train the SVM to give a robust and deterministic prediction of the label of a new data point.

Radial Basis Function [4] is given by equation,

$$K(x, y) = e^{-||x-y||^2 / (2\sigma)^2},$$

Where, x and y are feature inputs and σ is a free parameter used to determine effective mapping of non-linear space to linear data space.


ϕ(.)
Input space
Feature space
Note: feature space is of higher dimension than the input space in practice
Fig. 10. Radial Basis Function Kernel Mapping

The features from the grayscale image are used to train this SVM using scikit-learn python library. The feature space size for each image is 60,000 individual pixel features. This made the training model very slow and ineffective since, it became time-consuming and resource heavy. This would have been a limiting factor for a game as the game needs to be fast and robust as well as real-time. Hence, to tackle this feature expansion problem, I used Principal Component Analysis to determine main and distinct features from the image and only use those features in training the SVM. Using PCA, I reduced the dataset size of 60000 features/image to 40 features/image. This helped in reducing the time consumed in validating and predicting a label as well as reduced the training time of the model. Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The number of principal components is less than or equal to the number of original variables [5].

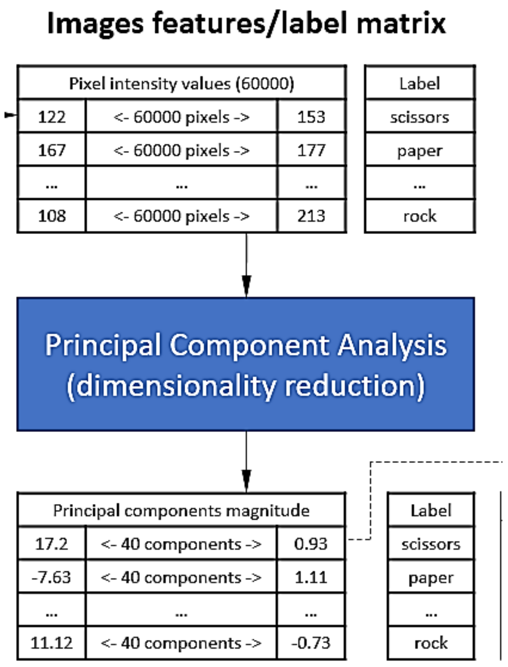

Images features/label matrix
Pixel intensity values (60000)
122
<- 60000 pixels ->
153
167
177
...
108
213
Label
scissors
paper
rock
Principal Component Analysis (dimensionality reduction)
Principal components magnitude
17.2
<- 40 components ->
0.93
-7.63
1.11
11.12
-0.73

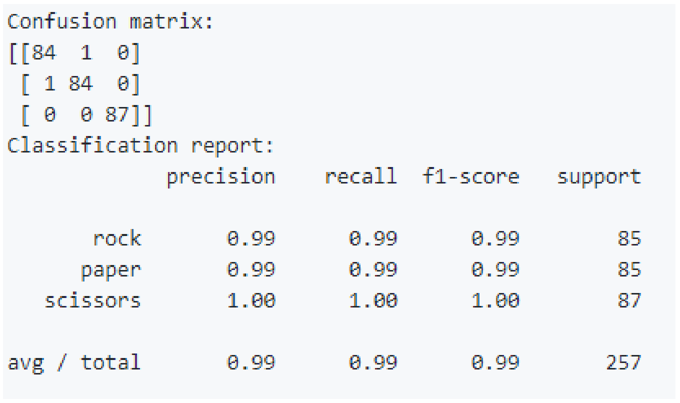
Fig. 11. Feature Space Reduction using PCA

These 40 components are then used to train the model SVM. The training was done on an UBUNTU 16 Linux machine, with 4GB RAM and Intel i5 Processor @ 2.4GHz. I created an equivalent environment on Ubuntu to train the model so that it can be easily ported to the Raspberry Pi without any further modifications. It took about 75 minutes to train the SVM against a dataset of ~600 paper images, ~580 rock images and ~650 scissor images. To train the model, we simply run the train.py script from the code supplied along with this report. And observe, the logs for training output. Once, the training is finished, a .cfl pickle classifier file is generated which is used by the game to predict the new input images.

V. RESULTS

A. *Training*

Post training, I used classification report [6] and confusion matrix [7] to determine the efficiency of the trained model on the test data set. Using the above data set, and testing the model against a set of input images, namely, ~85 rock images, ~85 paper images and ~87 scissor images, I received a classification report of 99% precision.



```
Confusion matrix:
[[84  1  0]
 [ 1 84  0]
 [ 0  0 87]]
Classification report:
             precision    recall  f1-score   support

       rock       0.99      0.99      0.99        85
      paper       0.99      0.99      0.99        85
   scissors       1.00      1.00      1.00        87

avg / total       0.99      0.99      0.99       257
```

Fig. 12. Confusion Matrix and Classification Report

Classification Report consists of:

- ***Precision*** = ability of the classifier not to label as positive a sample that is negative, tp / (tp + fp) where tp is the number of true positives and fp the number of false positives
- ***Recall*** = ability of the classifier to find all the positive samples, tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives
- ***F1-score*** = mean of recall and precision
- ***Support*** = number of occurrences of each class in test set

B. *Principal Component Analysis*

After reduction in feature space, it was necessary to test the effectiveness of the model with reduced space. Hence, I developed a Jupyter Notebook to plot graphs with just 2 principal components and their classification model.

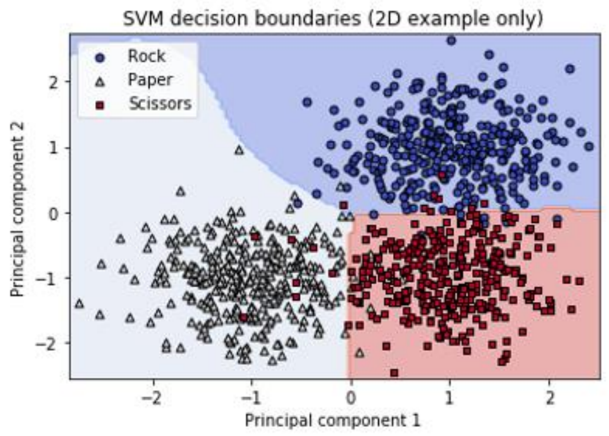


Fig. 13. Classification Plot of 2 Principal Components

It can be seen that the model is pretty robust with few outliers in a 2 Principal Component Classification, hence, I used 40 components to get a very robust machine. I also used the 40-feature vector to regenerate the image to visually examine, if the generated image could be classified manually as either rock, paper or scissor.

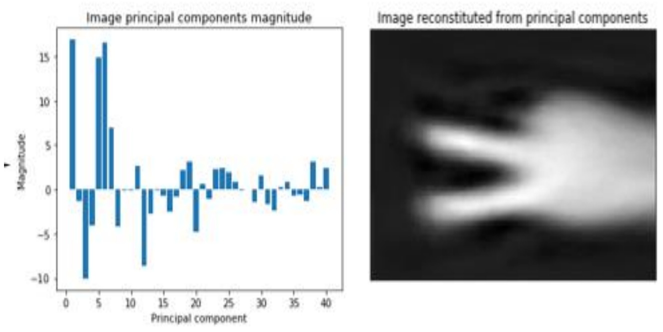
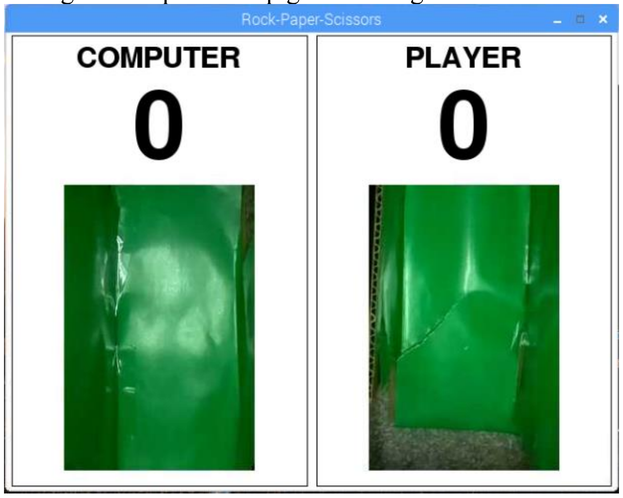


Fig. 14. Image reconstructed using Principal Components

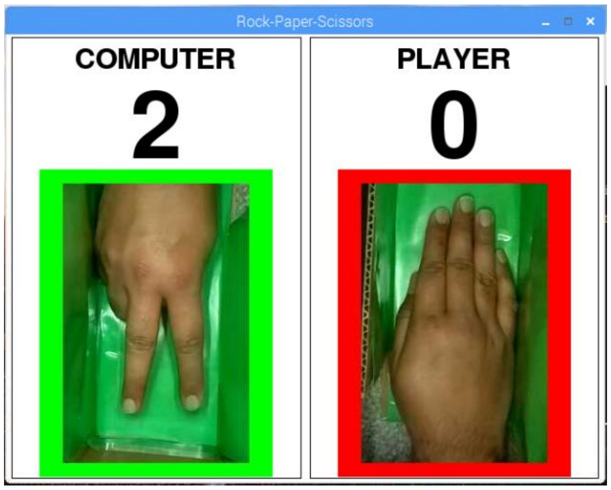
Based on the image generated, I could confirm that image still has features prominent enough to distinguish the gestures, such as, scissors in the given Figure 14 with only 40 features as opposed to 60,000 features in original image.

C. Game

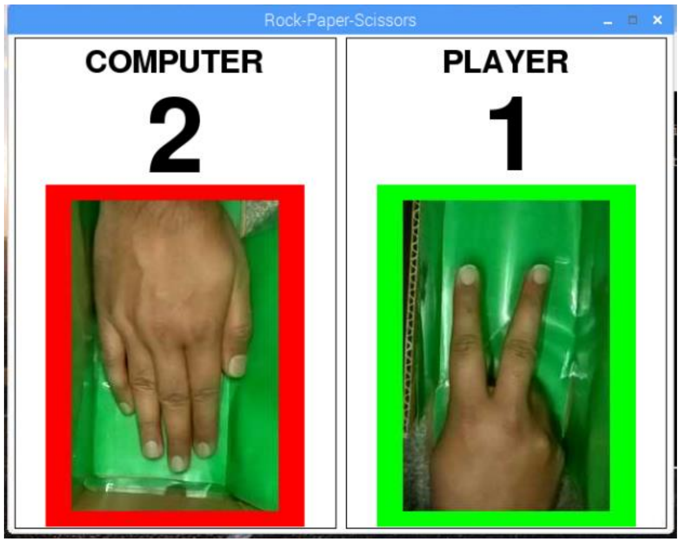
The game uses PyGame library to develop the user Interface. It starts with initializing the camera with white gains such that the additional brightness and reflection constant of the model environment is taken care of and subtracted from the image so as to get a sharp and crisp gesture image.



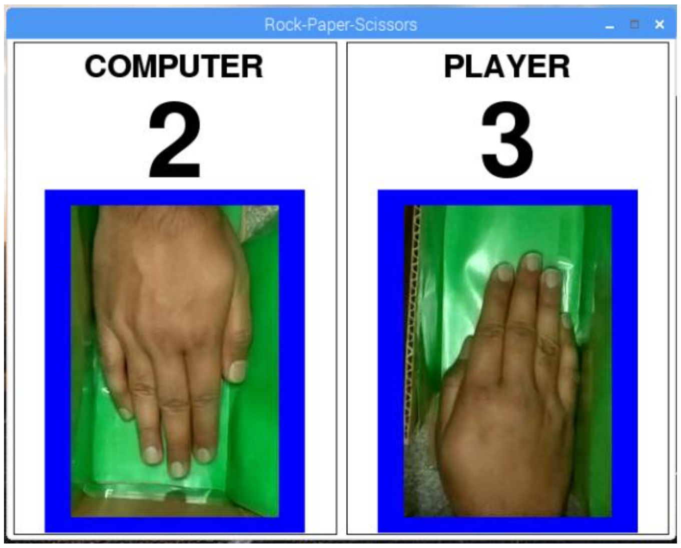
(a) Game UI Initial Screen



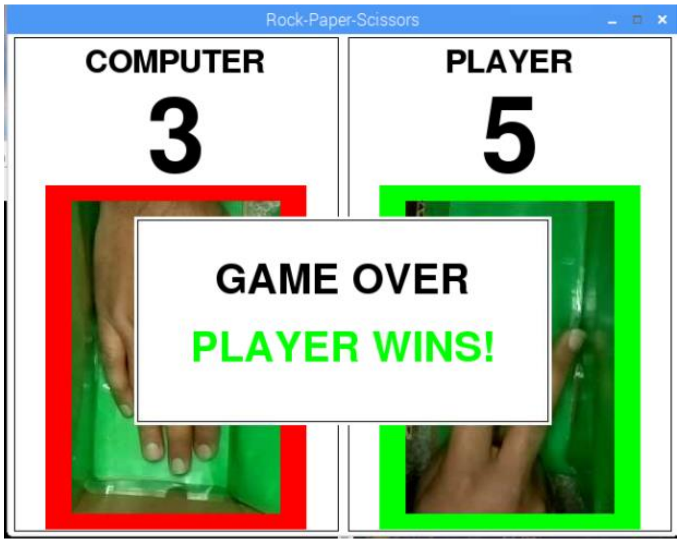
(b) Game UI Player Lost



(c) Game UI Player Won



(d) Game UI Tie



(e) Game UI Player Lost

Fig. 15. Game User Interface

VI. Conclusion

This project provides an application of Machine Learning and Image Processing in action. It involves use of hardware to run the machine learning and image processing stack. The game designed works efficiently for the setup description provided but starts to give inconsistent results for different background as the algorithm used in the experiment is background dependent. As an advancement in the project, I tried images without a background and training the model for those images resulted into a weaker classification report precision. It was visible from this experiment that using hue value for a background independent task was a problem as the thresholding based on hue value was inconsistent. To solve this, one can use an edge detection technique instead of a hue distance technique as the gestures for the rock paper scissor game are quite distinct and using canny edge detector or any high pass filter could easily make the project background independent. Due to the time constraint I couldn't complete this advanced feature in the project but with the intuition given above, I can add future work of adding the edge detection technique along with hue value distance technique to train the classifier such that for background independent environment where the background color is not fixed and the light conditions are not ambient, the model could still perform due to the edge detection classification. This classifier model as well as the game could then be used in designing a gesture recognition game for any mobile device with a rear camera. This would be a better form factor for the game as it would become available to a bigger section of people.

In this project, I was successfully able to design a pipeline which could become a base pipeline for any future gesture recognition-based games with developers designing other games and using my training scripts to train their models for their image sets. This project also tries to appreciate the hardware computational power of a mini computer board like Raspberry Pi.

The major complications for this project were to determine the Kernel function to be used along with the sigma value for mapping the feature vectors from non-linear state to linear classifier. The other complication was to design the user interface as I am not familiar with the Pygame library. But this project provided me with an opportunity to work with different scientific libraries and Pygame library for Python and OpenCv. I was able to achieve a 98% precision rate on the predictions from the Support Vector Machine using 40 feature components from the grayscale image using Principal Component Analysis.

Acknowledgment

I would like to that Dr. Rafeef Garbi, and T.A. Tajwar Aleef for providing guidance throughout my project. I would also like to thank my niece who inspired me to select rock paper scissor game as my project. And every single volunteer who helped me acquire the dataset to train the support vector machine.

References

[1] https://www.digikey.ca/en/maker/blogs/2018/meet-the-new-raspberry-pi-3-model-b-plus

[2] https://www.raspberrypi.org/products/camera-module-v2/

[3] **AN INTRODUCTION TO SUPPORT VECTOR MACHINES** (and other kernel-based learning methods) N. Cristianini and J. Shawe-Taylor Cambridge University Press 2000 ISBN: 0 521 78019 5

[4] https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

[5] Abdi, Hervé and Lynne J. Williams. "Principal Component Analysis." *Encyclopedia of Biometrics* (2009).

[6] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

[7] https://scikit-learn.org/0.17/modules/generated/sklearn.metrics.confusion_matrix.html

[8] http://www.kernel-machines.org/

[9] http://www.support-vector.net/

[10] http://www.kernel-machines.org/papers/tutorial-nips.ps.gz

[11] https://scikit-learn.org

[12] https://en.wikipedia.org/wiki/HSL_and_HSV