BLACKBEARD

Project Report

Team Jaqen

Nicolas Morant, Huayu Tsu,

DGMD E-14, Harvard University, Fall 2021

December 7th, 2021

I. DESCRIPTION & GOAL OF THE PROJECT

We wanted to build Blackbeard, a world class blackjack AI, that trains your ability to play blackjack from a wearable device in real time. The challenges Blackbeard would face are as follows: obtain real time data via a Raspberry Pi camera, read user commands via gesture recognition, detect cards using object recognition, detect card numbers using CNN model to train on digits classification, recommend best betting amount based on card counting method, and recommend best action based on optimal strategy. In addition, it will need to be easily wearable on a pair of so users can use it for training anywhere.

This wearable device would enable to improve training of people new to blackjack or even players who want to learn how to improve their decision throughout the plays. Furthermore, Blackbeard could be adapted to other applications that would require object detection and gesture recognition. For the counting feature, we can see this technology working in places like pharmaceutical companies, libraries, factories, and many more. Regarding the gesture, this could be used in AR, gaming, drone control, etc. The possibilities are close to endless, and this would give a new edge to these industries.

Disclaimer: The device is intended for academic use or training purposes only. Even though card counting is not illegal under Federal, State, or Local law, we don't support or recommend it.

II. TEAM ORGANIZATION

Our team was composed of two team members: Nicolas, and Jack. We each had our own list of features and tasks to work on (Figure 1). We decided to develop separately each feature to enable us to take advantage of our own schedule flexibility in order to optimize the speed of the development. In addition,

we scheduled weekly meetings to update each other on our advancements and make further decisions accordingly. In the last few weeks of our project, meetings increased at a rate of three times per week in order to accommodate for integration of the features.

Nicolas Morant	Huayu (Jack) Tsu
Object detection	Gesture Detection
Hardware integration	Blackjack Strategy
GPIO/MQTT	Pipeline

Figure 1. Team Members & Roles

III. SYSTEM ARCHITECTURE

Software & Packages:

- Python
- Darknet
- YOLO
- OpenCV
- MediaPipe
- Tensorflow
- Paho MQTT
- GPIO Zero
- NumPy
- GitHub
- Pandas
- Anacond

Hardware:

- Raspberry Pi Zero W
- Raspberry Pi Camera Board v2 - 8 Megapixels
- Raspberry Pi Zero v1.3 Camera Cable
- 64 – 128 GB Micro SD Card
- PowerBoost 1000 Basic - 5V USB Boost @ 1000mA from 1.8V+
- Lithium-Ion Polymer Battery - 3.7v 1200mAh
- Diffused 5mm LEDs (Red, Yellow, Green, Blue, and White)
- 1.14" 240x135 Color TFT Display board including two tactile buttons

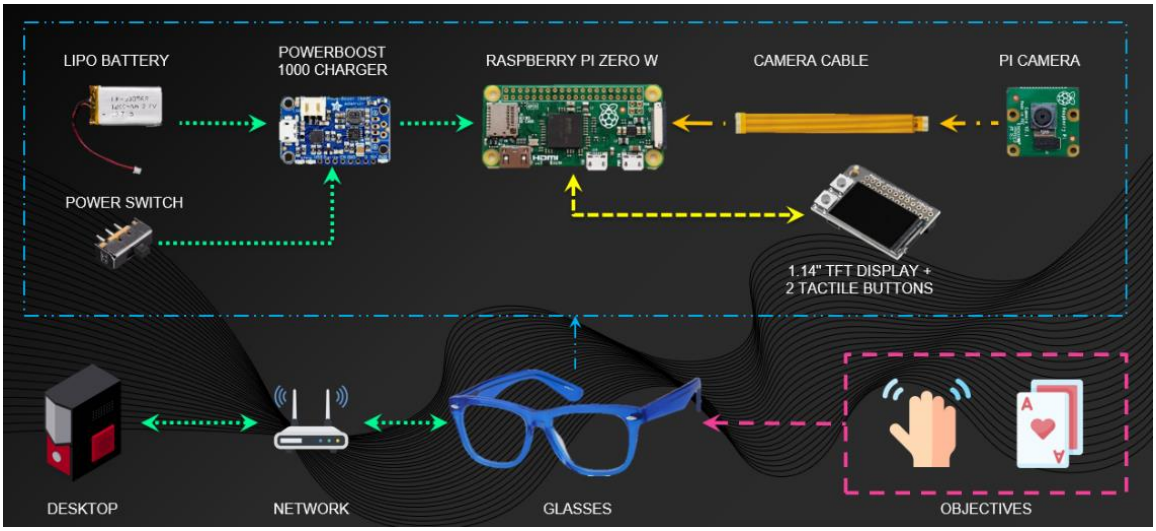

Figure 2. Blackbeard Architecture

IV. FEATURES

1. CARD DETECTION

For the card detection feature, we wanted to detect the card top left and bottom right corners where the number and symbol are located. We used a dataset from Roboflow that contained 24,240 images of playing cards with different backgrounds, position, etc. Using this dataset enabled us to create our own custom model right away. Prior to training the model we had to create a text file for each of the images in the dataset. We use the pre-trained model from YOLOv4-tiny. The most important one was setting up the .cfg file which is for the configuration. As a quick overview we had 52 classes, 64 training batch, 16 subdivisions, and a width/height of 416x416, with a maximum batch of 500,000 iterations. We ended up stopping the model training as we found no improvements between 150,000 to 180,000 iterations and chose the 170,000 iterations weight file as our final model. This resulted in successfully detecting every single card at a confidence level over 90% and on average being at the 99% level with speeds on average of 0.02 seconds. The card detection pipeline is returning a list of the detected cards to the blackjack strategy feature.

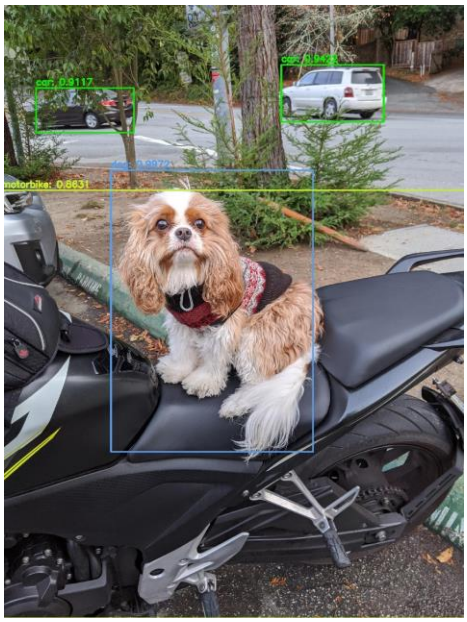

Figure 3. Objects Detection Using Yolov3

Figure 4. Cards Detection Using Custom Model

- **Status: Completed**

- **Key Related Files:**
  - object_detection_func.py – object detection pipeline
  - yolov4-tiny-blackbeard.cfg – model config file
  - blackbeard.names – classes names
  - blackbeard_yolov4-tiny-obj_170000.weights – model weights

2. GESTURE DETECTION

We wanted to develop a gesture classifier that can understand the various blackjack gestures so the user can implicitly interact with the device. We trained Blackbeard to recognize “hitting”, “standing”, “splitting”, and “reset”. We leveraged Google’s MediaPipe to generate the hand landmarks. Since MediaPipe’s model isn’t a classifier, we trained a deep neural network classifier on top using the landmarks to classify different gestures. We converted the TensorFlow model to TensorFlow lite to improve the

inference speed for real time inference since TensorFlow lite model is designed for underpowered devices like IOT to achieve fast inference speed.

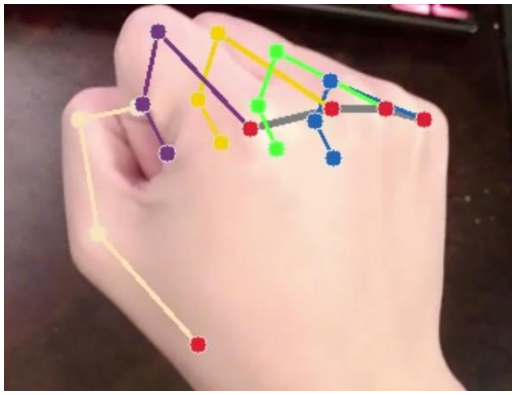

Figure 5. MediaPipe Landmarks

- **Status: Completed**

- **Key Related Files:**
  - gesture_init.py – gesture detection pipeline
  - model_training.ipynb – train model
  - gesture_data_collection.ipynb – generate training data

3. BLACKJACK STRATEGY

In order for Blackbeard to understand blackjack, we had to code the blackjack environment first with the action space available and game rules. Blackbeard takes information from the environment like card detection and gesture recognition to assess the game state and the optimal action. Blackbeard uses a simple Hi-Low system, which assigns negative values to low cards that are favorable to dealers, and positive values to high cards that are favorable to players. By keeping track of previously detected cards, Blackbeard can assess the house vs player advantage and recommends the optimal action to the player.

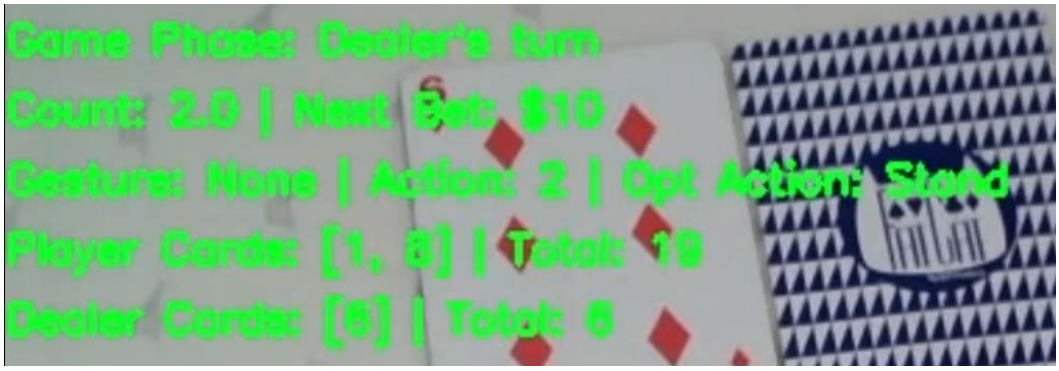

Figure 6. Blackbeard debug view

- **Status: Completed**

- **Key Related Files:**
  - blackjack.py – blackjack game logic

4. REMOTE GPIO

We implemented a remote connection between our desktop to the GPIO of the Raspberry Pi Zero W by using the GPIO Zero library. This enabled us to remotely control things such as LEDs and buttons. We used this feature to create a "Start" and "Stop" button for Blackbeard. This was in addition the power switch we added on the power boost charger board.

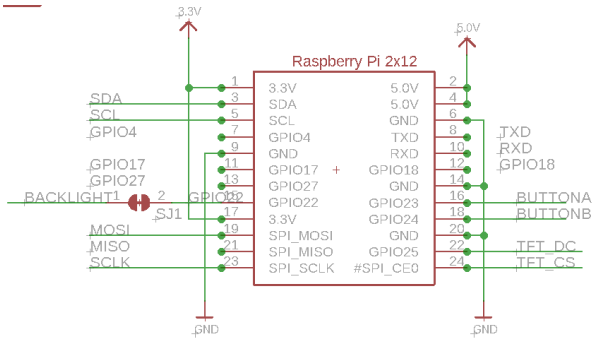

Figure 7. 1.14" TFT Display Board Including Two Tactile Buttons

The ability to control and receive information when a button is pressed can be very useful in developing a better interface for the user. Also, we were able to develop multiple functions to control LEDs and the

blinking of these. This could be implemented in a next prototype in order to potentially give directions, notifications to the user.

- **Status: Completed**
- **Key Related Files:**
  - gpio_func.py - remote GPIO functions

5. UI WITH MQTT

For our user interface (UI), we wanted something where we could give multiple different information to the user in order for them to understand what is happening as well as let them know the directions, recommendations that Blackbeard is giving them. For this, we used the MQTT protocol thanks to the paho-mqtt python library. It is a very lightweight publish and subscribe message transport protocol that has a small code footprint and bandwidth use. This was perfect for application in order to not overload the Raspberry Pi and the bandwidth usage as we already had our RTSP feed and remote GPIO. We were able to display to the user the count, next suggested bet, detected hand gesture, suggested action, detected player cards with their total value, and the dealer cards with their total value. In addition we were also displaying information such as the stage of the starting and closing steps including things like “Press Start”.

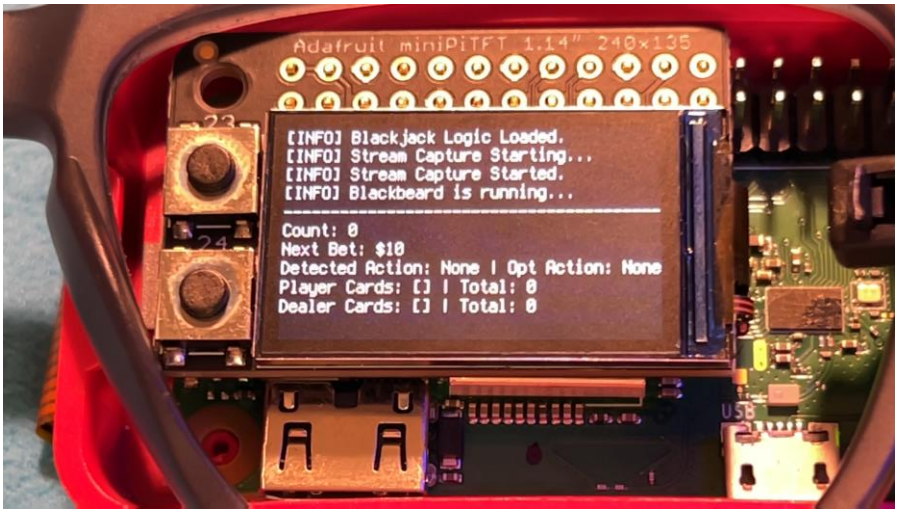

Figure 8. UI with the MQTT Output

- **Status: Completed**

- **Key Related Files:**
  - mqtt_client.py - client script to run on Raspberry Pi
  - mqtt_func.py – functions to publish messages

V. LESSONS LEARNED

We learned that AI/LM in IoT faces many challenges from the hardware side as well as the software side. The initial hardware we planned on using, Raspberry pi Zero, was too underpowered to run machine learning inference at a reasonable speed. Our solution of using RTSP with another device resolved the inference speed issue, but introduced the lag issue from RTSP. Working through these problems and finally got a working prototype was a rewarding experience.

VI. REFERENCES

- https://google.github.io/mediapipe/
- https://github.com/DevGlitch/botwizer
- https://pjreddie.com/darknet/
- https://github.com/AlexeyAB/darknet
- https://www.eclipse.org/paho/
- https://gpiozero.readthedocs.io/en/stable/remote_gpio.html
- https://universe.roboflow.com/augmented-startups/playing-cards-ow27d/1
- https://learn.adafruit.com/adafruit-mini-pitft-135x240-color-tft-add-on-for-raspberry-pi