Annex D

NANYANG POLYTECHNIC

Diploma in

Mechatronics Engineering

Final Year Project Report

Project Title:

Development of the Control System of an Outreach Robot

Project Supervisor:

Dr Edwin Foo

Project Student:

Oei Zheng Yong (173505G)

Table of Contents

Abstract

In the recent years, the demand and use of robots is increasing rapidly. Hence, future engineers need to be cultivated and one of the ways is to have children develop interest in robotics through interacting with robots. To do this, the software of robot that can play Scissor Paper Stone game with the user is developed. The software interacts with the user through Ubuntu OS, GUI and video streams from connected webcam. It crops and processes the frame of the video stream to a binary image with skin extraction to get the move of the user when the software indicates to put a move to the user. It feeds the binary image to the trained Convolutional Neural Network model to predict the move and gives the result of the game by comparing with its own chosen gesture. This software requires user to put their gesture's front view clearly, lacking flexibility. More can be done to improve this software.

Introduction

Robotics is going to be a big part of our lives in the future. The demand for robotics (like a robot arm) in most of the working industries is increasing as the years goes by. People tend to rely on robotics as it has many advantages that will make things easier.

Advantages include:

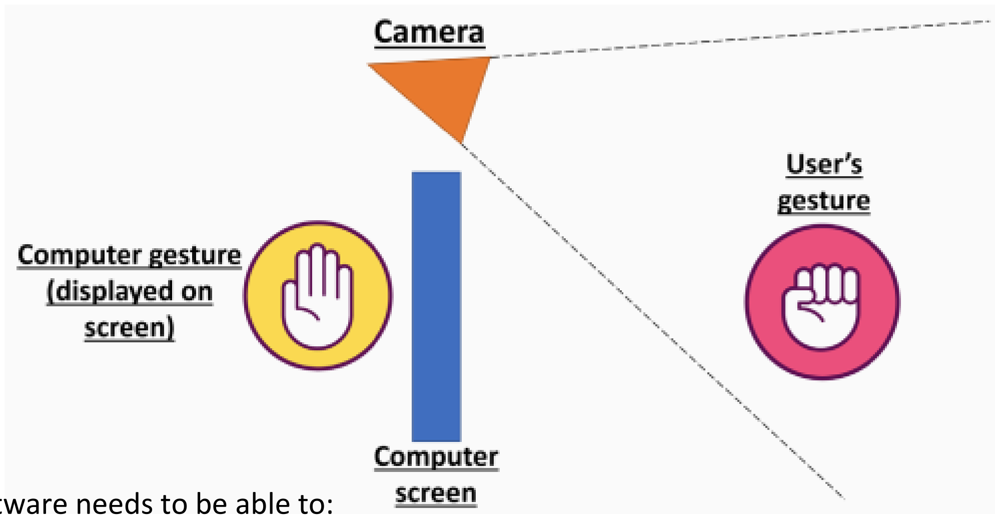
- High productivity with long operation hours
- Resolve shortage of skill labour issue
- Flexible operations
- Consistent quality in production
- Able to work in hazardous environment to human
- And more…

Deep learning is a machine learning technique which allows the AI to recognize gestures, signs and even faces. It is being used in many of the industries in the real world right now. Examples like, driverless cars which the AI will detect road signs, voice recognition for control and even CCTV facial recognition which the police used. This game software uses deep learning to detect the user’s hand gesture which will be explained further through the report.

Due to the increase demand in robotics, this will lead to lack of engineers to develop, design and troubleshoot. Hence, we will have to start developing interest in children which will lead to them pick up subjects related to robotics and even allow them to start careers in the robotics industries. This would increase the number of engineers overtime, coping with the increasing demand. One of the best ways to target children is through games. Therefore, this game software is developed for children to play with.

Objectives

This project is about developing an engaging and interactive game software that allow the user to play scissors, paper, stone on. This software uses deep learning and webcam to get the gesture the user is putting out and will respond the user with a gesture using a robotic arm. The arm uses solenoid to extend the fingers which depicts scissors, paper or stone depending on wat the AI responds with.



The software needs to be able to:

- Recognise the gesture of the user (Scissor, Paper, Stone) through a webcam with more than 90% accuracy.
- Tell the user the gesture that the software has predicted.
- Respond to the gesture of the user with another gesture from the software within 1 second.
- Have a Graphics User Interface, GUI, of the game that is engaging, easy to understand and use for the user.
- Run on the Nvidia Jetson Nano to make the whole thing more portable
- Work together with the robotic arm by displaying the same gesture the AI gives on the screen and the robotic arm

Project Scope

Key Specification

This project uses the following to run:

- Software
  - Ubuntu 18.04
  - Python 3.7.3
  - Tensorflow 1.13.1
  - Keras 2.2.4
  - OpenCV 4.0.0
  - Pygame 1.9.6
- Hardware
  - Logitech HD Pro Webcam C910
  - Nvidia Jetson Nano
  - Adafruit Crickit Hat
  - 4 Relay module 2PH63003A
  - Robot arm

Software

Operating System

Ubuntu 18.04 is the operating system used to run the software. The software can also be run on Windows 10 however, initializing the webcam takes some time. This makes the software hang momentarily hence, running the software on Ubuntu is preferred.

Programming Language

Python is used to write all the programs for the software. Python is an object oriented, general purpose and high-level programming language. It has a simple syntax that allow easier troubleshooting and less typing, hence it was used.

Software libraries

Deep learning

- **Tensorflow**
  - Tensorflow is a free and open source library for developing and training Machine Learning (ML) model. It is used to train the Convolutional Neural Network model to recognise the user gesture.
- **Keras**
  - Keras is an open source high-level neural network API written in Python. It is capable of running on top of Tensorflow, hence used to train the Convolutional Neural Network model (refer to Concepts) to recognise the user gesture using Tensorflow backend. Using Tensorflow directly is possible but was not done as writing the program using Keras is much easier.

Computer Vision

- **OpenCV**
  - OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. It is used to get the video stream from the webcam, draw figures into the video and process the video frames to feed them to the CNN model.

Graphics User Interface (GUI)

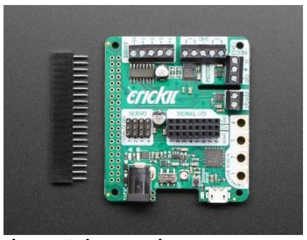
- **Pygame**
  - **Pygame** is a cross-platform set of Python modules for writing video games. It includes Python libraries for computer graphics and sound. It is used to create the GUI for the Scissor Paper Stone game.

Hardware

- **Logitech HD Pro Webcam C910**
  - **Logitech HD Pro Webcam C910** (1080p) is used to get the video stream for the GUI, processing of image and getting the frame with the gesture of the user.

- **Nvidia Jetson Nano**
  - **Nvidia Jetson Nano** is a small and powerful computer for running multiple neural networks like image classification, gesture detection and many more. This is device is chosen to run the game on because it is small in size and consist of a GPU.

- **Adafruit Crickit Hat**
  - **Adafruit Crickit Hat** is a peripheral board connected to the Jetson Nano via the 2x20 GPIO connector which allows access to motors, servos and solenoids which are being used for the robot arm.

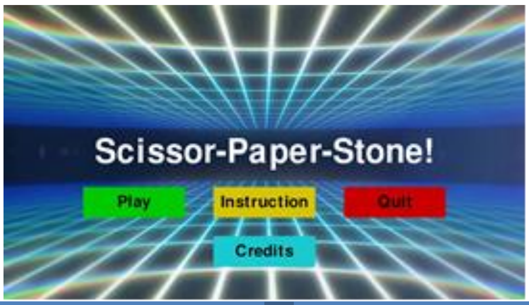


- **4 Relay module 2PH63003A**
  - **4 Relay module 2PH63003A** is used to connect the 3 solenoids to the Adafruit Crickit hat which allow me to be able to control the 3 solenoids separately.
- **Solenoid**
  - **Solenoid** is used to actuate the movement of fingers on the robot arm

Major Deliverables

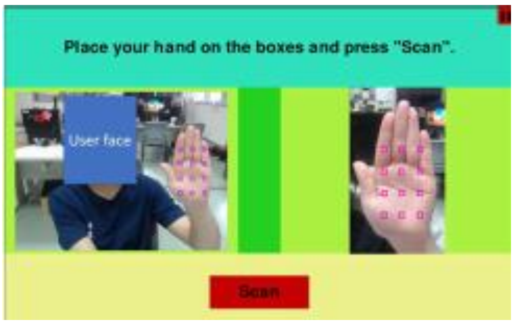
The software will run the game in this manner:

The software will open the Main Menu. The ‘Play button is pressed.



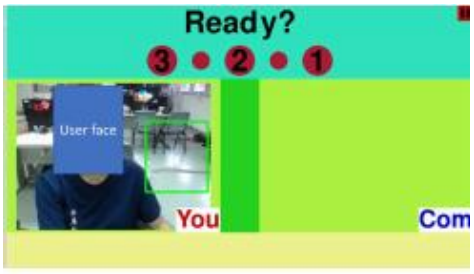
The software will open a screen for the user to choose the difficulty they want to play in

The software will open Scan screen. There will be instruction on where the user’s hand is to be placed and the “Scan” button is pressed when user has placed their hand.

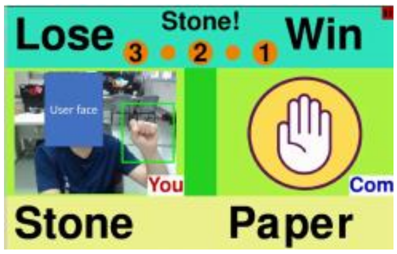


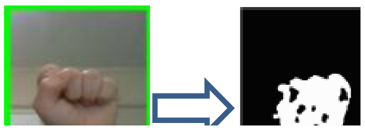
When the ‘Scan’ button is pressed, the last video frame before the button is pressed is converted from RGB to HSV. Hue and saturation histogram of the skin is calculated using the colour in the pink rectangles

The software will open Game screen. There will be words and circles to help user gauge the timing to put their move.

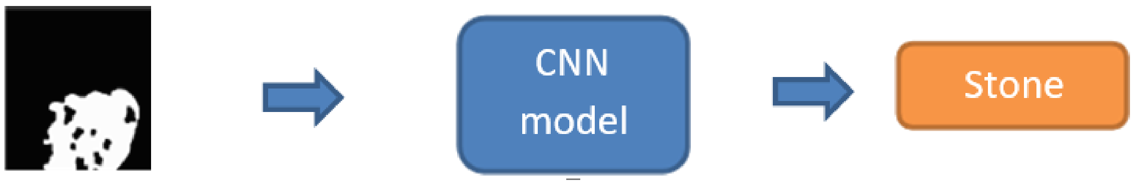


When the user is expected to put a move in the green box, the software crops the frame at the green box, process it into binary image with only the skin present as white.





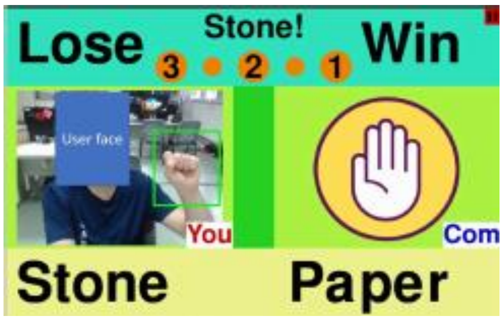
The binary image is fed to the CNN model to predict the gesture.



The software choose a move based on the different difficulties (Scissor, Paper, Stone)

The software compares the gestures between the software and the user to see who won the match

The software displays the results and the moves for both sides with the frame that was used for gesture prediction.



<u>Gantt chart</u>

S/N		Week	1	2	3	4	5	6	7	8	9	10	11	12
1	Research on programs required to run the game Learn more about linux OS and python programming	Planned	🟥	🟥										
		Actual	🟦	🟦										
2	Implement the required deep learning libraries on the Jetson Nano to run the	Planned			🟥									
		Actual			🟦	🟦								
3	Improve and optimise the interface of the game	Planned				🟥	🟥							
		Actual				🟦	🟦							
4	Retrain Keras model to improve accuracy of predictions	Planned					🟥	🟥						
		Actual						🟦	🟦	🟦				
5	Research and carry out the ways to speed up the game	Planned						🟥	🟥	🟥				
		Actual							🟦	🟦				
6	Set up the robot arm and program it	Planned								🟥	🟥	🟥		
		Actual								🟦	🟦			
7	Add in programs for different difficulties to make the game more interesting	Planned									🟥	🟥		
		Actual									🟦	🟦		
6	Improving the programs and final testing	Planned										🟥	🟥	
		Actual										🟦	🟦	
7	Presentation	Planned												🟥
		Actual												🟦

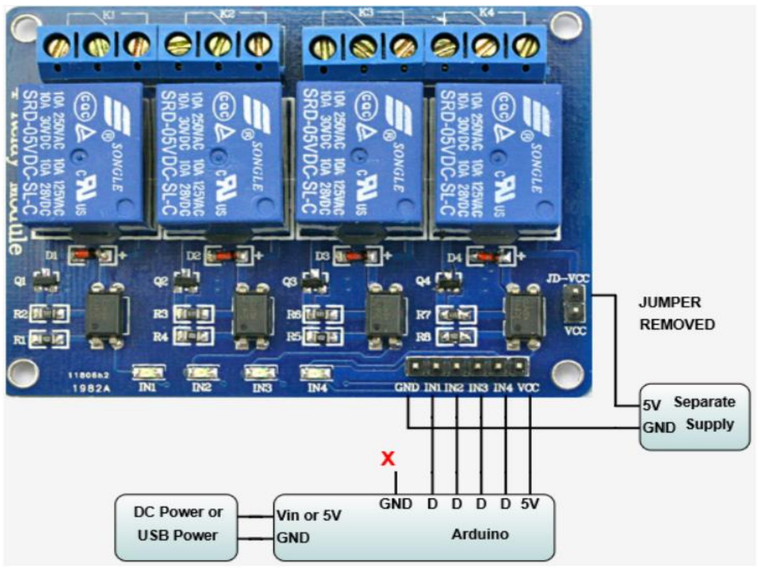
Project Implementation

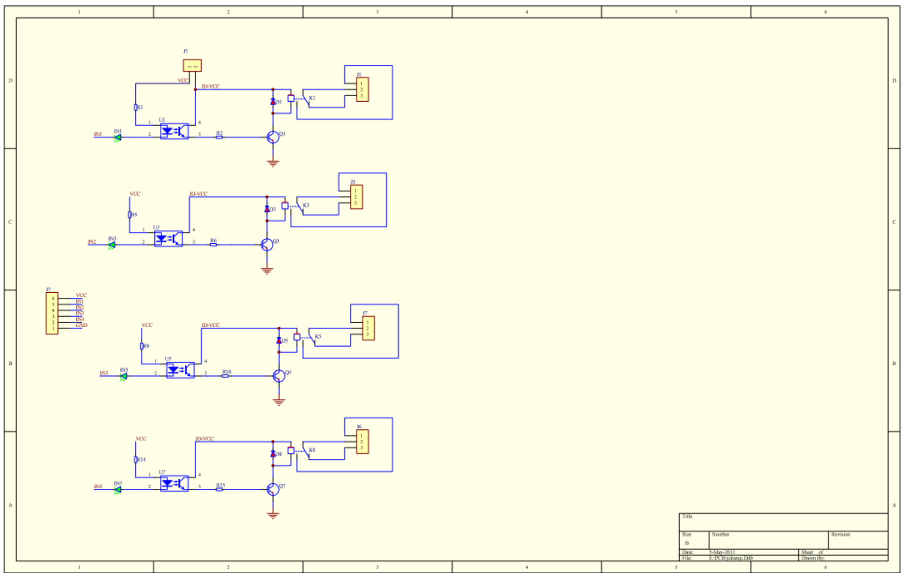
Hardware Development

The Adafruit Crickit hat is connected to the Nvidia Jetson Nano using the standard 2x20 GPIO connector which allowed me to control the solenoids on the robot arm.

After connecting both the Crickit hat and the Jetson Nano together, I needed to connect the Crickit hat to the solenoid on the robot hand. The connections are based on the images.

The connections are based on the image below:





Software Development

Program for the robot arm

The solenoids can be controlled by using the following python commands.

```
from adafruit_crickit import crickit

crickit.drive_1.frequency = 5000

crickit.drive_1.fraction = 0.0  # all the way off
crickit.drive_1.fraction = 1.0  # all the way on
```

However, this only controls one of the solenoid and the robot arm consists of 3 of solenoid. Hence, the following part of the program is added in to be able to control all 3 of the solenoid at the same time and reset them after every match.

```
if rand_gesture == 'Scissor':
            crickit.drive_1.fraction = 0.0  # all the way off
            crickit.drive_2.fraction = 1.0  # all the way on
            crickit.drive_3.fraction = 0.0  # all the way off
        elif rand_gesture == 'Paper':
            crickit.drive_1.fraction = 1.0  # all the way on
            crickit.drive_2.fraction = 1.0  # all the way on
            crickit.drive_3.fraction = 1.0  # all the way on
        elif rand_gesture == 'Stone':
            crickit.drive_1.fraction = 0.0  # all the way off
            crickit.drive_2.fraction = 0.0  # all the way off
            crickit.drive_3.fraction = 0.0  # all the way off

        if time.time() - start > 3:
            crickit.drive_1.fraction = 0.0  # all the way off
            crickit.drive_2.fraction = 0.0  # all the way off
            crickit.drive_3.fraction = 0.0  # all the way off
            break
```

Program for the different modes

To make the game more fun and interesting, I have added 3 more game modes with different difficulties each being more and more difficult to win against.

The first mode is named **easy**. This mode has a fair chance for the user to win as the AI gesture given is random. This is programmed using the following codes below:

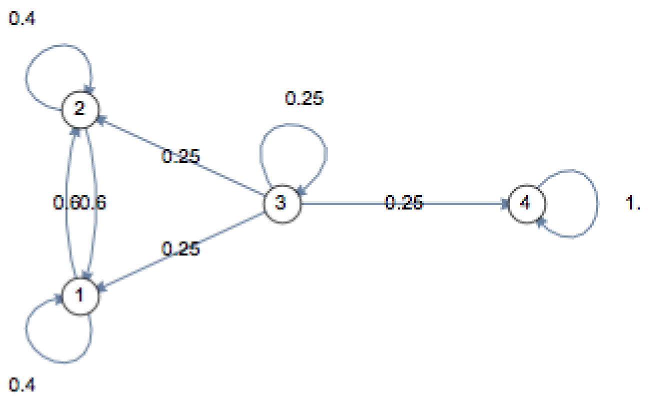
```
def easy_gesture():
    gesture_int = random.randint(0, 2)
    if gesture_int == 0:
        gesture = 'Scissor'
    if gesture_int == 1:
        gesture = 'Paper'
    if gesture_int == 2:
        gesture = 'Stone'
    return gesture
```

The second mode is named **intermediate**. In this mode, the AI will first choose a random gesture like in the easy mode. After that, instead of showing the results of who won the round, it is given another chance of rerolling another gesture to put out if the AI loses. However, it only has a 50% chance of rerolling another gesture to make it fairer for the user but at the same time also making the user have a lesser chance of winning. The program is shown below:

```
def intermediate_gesture(gest_ture):
    gesture = ["Stone", "Paper", "Scissors"]
    continuePlaying = True
    prevGesture = ""
    reroll = ""
    choice = 3
    start = time.time()

    if gest_ture == 'Stone':
        gesture_integer = 0
    elif gest_ture == 'Paper':
        gesture_integer = 1
    elif gest_ture == 'Scissor':
        gesture_integer = 2
    elif gest_ture == 'Nothing':
        gesture_integer = 2
    elif gest_ture == '???' :
       gesture_integer = 2


    machine_int = random.randint(0, 2)
    if machine_int == 0:
        machine_gesture = 'Stone'
    elif machine_int == 1:
        machine_gesture = 'Paper'
    elif machine_int == 2:
        machine_gesture = 'Scissor'

    if gest_ture == machine_gesture:
        result = 'Draw'
    elif gest_ture == 'Scissor' and machine_gesture == 'Paper':
        result = 'You Win!'
    elif gest_ture == 'Stone' and machine_gesture == 'Paper':
        result= 'You Lose'
    elif gest_ture == 'Paper' and machine_gesture == 'Scissor':
        result = 'You Lose'
    elif gest_ture == 'Stone' and machine_gesture == 'Scissor':
        result ='You Win!'
    elif gest_ture == 'Paper' and machine_gesture == 'Stone':
        result = 'You Win!'
    elif gest_ture == 'Scissor' and machine_gesture == 'Stone':
        result = 'You Lose'
    elif gest_ture == 'Nothing':
        result = 'You Lose'
    elif gest_ture == '???':
        result = 'You Lose'

    if result == "You Win!":
        reroll = random.randint(0, 1)
        if reroll == 0:
            machine_int = random.randint(0, 2)
        elif reroll == 1:
            machine_int == machine_int
    elif (result == "Draw"):
        if reroll == 0:
            machine_int = random.randint(0, 2)
        elif reroll == 1:
            machine_int == machine_int
    elif (result == "You Lose"):
        machine_int == machine_int

    if machine_int == 0:
        machine_gesture = 'Stone'
    elif machine_int == 1:
        machine_gesture = 'Paper'
    elif machine_int == 2:
        machine_gesture = 'Scissor'

    return machine_gesture
```

For the third mode, it is the **Hard** mode. In this mode, the AI uses Marcov chains to make predictions. Marcov chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. In this case, the 3 event is when the user choose Scissors, Paper or Stone. Data from previous match of the game are collected and stored in a matrix. Based on all of these data, it is able to calculate the possibility of what the next event would be, be it the user choose Scissors, Paper or Stone. It will then choose the event with the highest possibility and choose the gesture that will counter and win the user. For example, if there is a probability of 0.85 where the user is going to choose paper, the machine will process this and will give out the gesture of Scissors.

The image below is an example of a Marcov chain.



This is the program for the hard mode of the game:

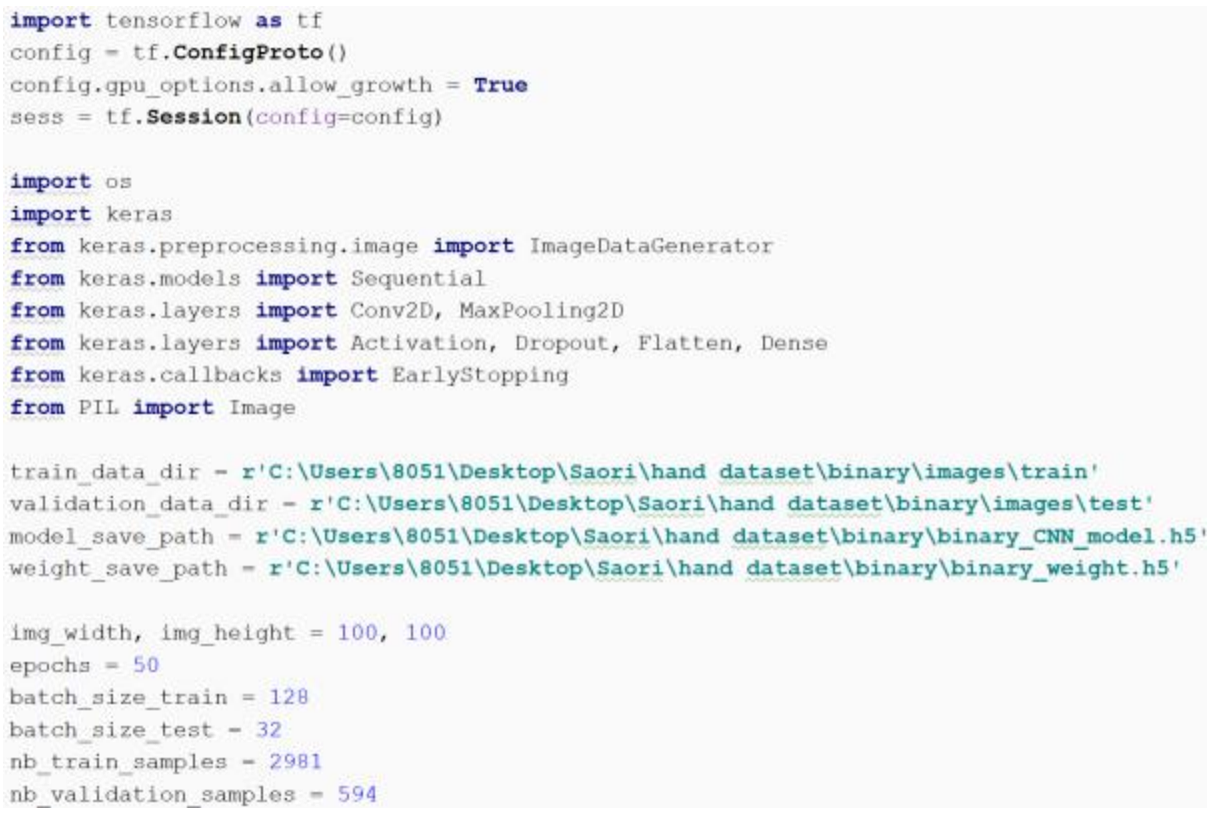
```python
def hard_gesture(gest_ture):
    global probabilitiesRPS
    choices = ["Rock","Paper","Scissors"]
    choi = ['r','p','s']
    continuePlaying = True
    prevGesture = ""
    result = ""
    choice = 3
    probRock = 0
    probPaper = 0
    probScissors = 0



    machineChoice = random.randint(0, 2)
    if gest_ture == machineChoice:
        result = 'Draw'
    elif gest_ture == 'Scissor' and machineChoice == 'Paper':
        result = 'You Win!'
    elif gest_ture == 'Stone' and machineChoice == 'Paper':
        result= 'You Lose'
    elif gest_ture == 'Paper' and machineChoice == 'Scissor':
        result = 'You Lose'
    elif gest_ture == 'Stone' and machineChoice == 'Scissor':
        result ='You Win!'
    elif gest_ture == 'Paper' and machineChoice == 'Stone':
        result = 'You Win!'
    elif gest_ture == 'Scissor' and machineChoice == 'Stone':
        result = 'You Lose'
    elif gest_ture == 'Nothing':
        result = 'You Lose'
    elif gest_ture == '???':
        result = 'You Lose'

    if gest_ture == 'Stone':
        gesture_integer = 0
    elif gest_ture == 'Paper':
        gesture_integer = 1
    elif gest_ture == 'Scissor':
        gesture_integer = 2
    elif gest_ture == 'Nothing':
        gesture_integer = 2
    elif gest_ture == '???' :
        gesture_integer = 2

    prevGesture = gesture_integer

    transMatrix = buildTransitionProbabilities(prevGesture,gesture_integer,result)
    machineChoice = random.randint(1, 100)
    probabilitiesRPS[0] = transMatrix[prevGesture][0]
    probabilitiesRPS[1] = transMatrix[prevGesture][1]
    probabilitiesRPS[2] = transMatrix[prevGesture][2]
    rangeR = probabilitiesRPS[0] * 100
    rangeP = probabilitiesRPS[1] * 100 + rangeR
    if (machineChoice <= rangeR):
        machineChoice = 1
    elif (machineChoice <= rangeP):
        machineChoice = 2
    else:
        machineChoice = 0

    if machineChoice == 0:
        machinechoice = 'Stone'
    elif machineChoice == 1:
        machinechoice = 'Paper'
    elif machineChoice == 2:
        machinechoice = 'Scissor'

    print("Your winning transition matrix is:\nr: %s\np: %s\ns: %s\n" % 
(tMatrix[0],tMatrix[1],tMatrix[2]))
    print("Your losing transition matrix is:\nr: %s\np: %s\ns: %s\n" % 
(tMatrixL[0],tMatrixL[1],tMatrixL[2]))
    print("Your tying transition matrix is:\nr: %s\np: %s\ns: %s\n" % 
(tMatrixT[0],tMatrixT[1],tMatrixT[2]))

    return machinechoice



def buildTransitionProbabilities(pC,c,winloss):
    global buildTMatrix
    global buildTMatrixL
    global buildTMatrixT
    choi = ['r','p','s']

    if winloss == "Win!":
        for i, x in buildTMatrix.items():
            if ('%s%s' % (choi[pC],choi[c]) == i):
                buildTMatrix['%s%s' % (choi[pC], choi[c])] += 1
    elif winloss == "Tied!":
        for i, x in buildTMatrixT.items():
            if ('%s%s' % (choi[pC],choi[c]) == i):
                buildTMatrixT['%s%s' % (choi[pC], choi[c])] += 1
    else:
        for i, x in buildTMatrixL.items():
            if ('%s%s' % (choi[pC],choi[c]) == i):
                buildTMatrixL['%s%s' % (choi[pC], choi[c])] += 1

    return buildTransitionMatrix(winloss)

def buildTransitionMatrix(winlosstwo):
    global tMatrix
    global tMatrixL
    global tMatrixT

    if winlosstwo == "Win!":
        rock = buildTMatrix['rr'] + buildTMatrix['rs'] +buildTMatrix['rp'] 
#number of gesture that appeared
        paper = buildTMatrix['pr'] + buildTMatrix['ps'] +buildTMatrix['pp']
        scissors = buildTMatrix['sr'] + buildTMatrix['ss'] +buildTMatrix['sp']
        choi = ['r','p','s']
        for row_index, row in enumerate(tMatrix):
            for col_index, item in enumerate(row):
                a = int(buildTMatrix['%s%s' % (choi[row_index],choi[col_index])])
                if (row_index == 0):
                    c = a/rock
                elif (row_index == 1):
                    c = a/paper
                else:
                    c = a/scissors
                row[col_index] = float(c)
        return (tMatrix)
    elif winlosstwo == "Tied!":
        rock = buildTMatrixT['rr'] + buildTMatrixT['rs'] +buildTMatrixT['rp']
        paper = buildTMatrixT['pr'] + buildTMatrixT['ps'] +buildTMatrixT['pp']
        scissors = buildTMatrixT['sr'] + buildTMatrixT['ss'] +buildTMatrixT['sp']
        choi = ['r','p','s']
        for row_index, row in enumerate(tMatrixT):
            for col_index, item in enumerate(row):
                a = int(buildTMatrixT['%s%s' % (choi[row_index],choi[col_index])])
                if (row_index == 0):
                    c = a/rock
                elif (row_index == 1):
                    c = a/paper
                else:
                    c = a/scissors
                row[col_index] = float(c)
        return (tMatrixT)

    else:
        rock = buildTMatrixL['rr'] + buildTMatrixL['rs'] +buildTMatrixL['rp']
        paper = buildTMatrixL['pr'] + buildTMatrixL['ps'] +buildTMatrixL['pp']
        scissors = buildTMatrixL['sr'] + buildTMatrixL['ss'] +buildTMatrixL['sp']
        choi = ['r','p','s']
        for row_index, row in enumerate(tMatrixL):
            for col_index, item in enumerate(row):
                a = int(buildTMatrixL['%s%s' % (choi[row_index],choi[col_index])])
                if (row_index == 0):
                    c = a/rock
                elif (row_index == 1):
                    c = a/paper
                else:
                    c = a/scissors
                row[col_index] = float(c)
        return (tMatrixL)
```

As for the **expert** mode, is just a fun add-on whereby the user does not stand a chance of winning as the gesture the AI chooses will be based on the user's gesture. For example, if the user uses paper, the AI will put out a gesture of Scissors respectively.

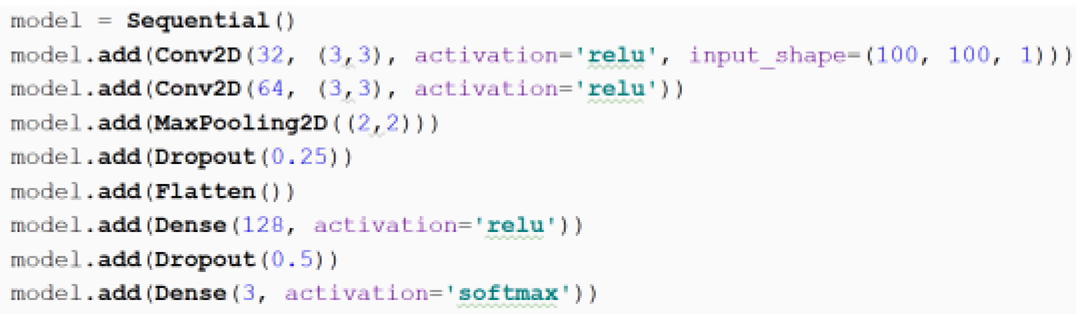
```
def expert_gesture(gest_ture):

    if gest_ture == 'Stone':
        machinechoice = 'Paper'
    elif gest_ture == 'Paper':
        machinechoice = 'Scissor'
    elif gest_ture == 'Scissor':
        machinechoice = 'Stone'
    elif gest_ture == 'Nothing':
        machinechoice = random.randint(0, 2)
        if machinechoice == 0:
            machinechoice = 'Scissor'
        if machinechoice == 1:
            machinechoice = 'Paper'
        if machinechoice == 2:
            machinechoice = 'Stone'
    elif gest_ture == '???' :
        machinechoice = random.randint(0, 2)
        if machinechoice == 0:
            machinechoice = 'Scissor'
        if machinechoice == 1:
            machinechoice = 'Paper'
        if machinechoice == 2:
            machinechoice = 'Stone'

    return machinechoice
```

Retrain the Keras model to improve accuracy

- The images that are trained into the model by the previous student was only with hand gestures from 1 angle. Hence, I have gathered more images with different backgrounds, angles and people and retrained the first model.
- The program I have used for retraining the model was done by the previous student on this project and it is the following below:



```
import tensorflow as tf
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
sess = tf.Session(config=config)

import os
import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import EarlyStopping
from PIL import Image

train_data_dir = r'C:\Users\8051\Desktop\Saori\hand dataset\binary\images\train'
validation_data_dir = r'C:\Users\8051\Desktop\Saori\hand dataset\binary\images\test'
model_save_path = r'C:\Users\8051\Desktop\Saori\hand dataset\binary\binary_CNN_model.h5'
weight_save_path = r'C:\Users\8051\Desktop\Saori\hand dataset\binary\binary_weight.h5'

img_width, img_height = 100, 100
epochs = 50
batch_size_train = 128
batch_size_test = 32
nb_train_samples = 2981
nb_validation_samples = 594
```



```
model = Sequential()
model.add(Conv2D(32, (3,3), activation='relu', input_shape=(100, 100, 1)))
model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D((2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1. / 255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size_train,
    class_mode='categorical',
    color_mode='grayscale')

validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size_test,
    class_mode='categorical',
    color_mode='grayscale')
```

```
model.fit_generator(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size_train,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size_test)

model.save_weights(weight_save_path)
model.save(model_save_path)
```

Outcome:

- The software now is able to detect hand gesture from more angles and the consistency for the prediction has increased.

The training program will be run on jupyter notebook. The following command below is to open jupyter notebook.

-conda activate tf1.13

-jupyter notebook

Concepts

The game was developed on a PC which is big and not portable hence it is being transferred and run on the Nvidia Jetson Nano which is small with dimensions of only 70x45mm. However, due to its small system memory of 4GB, the game tends to run much slower on launch and sometimes crash. Hence, I've done some research and found out some solutions that can solve this problem.

1. Converting the Convolutional Neural Network model to a tflite model.
    - Tensorflow Lite is a set of tools made to run Tensorflow models on mobile phones, embedded and IoT devices with low system memory just like the Jetson Nano. It enables on-device machine learning interface with low latency and a small binary size which will decrease the system memory the game is taking up hence speeding up the game.

It is converted by the following program below:

```
import tensorflow as tf
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
tflite_quant_model = converter.convert()
```

The command input into the game to run this model instead is:

```
interpreter = tf.lite.Interpreter("CNN/pruned_keras.tflite")
interpreter.allocate_tensors()
input details = interpreter.get_input_details()
output details = interpreter.get_output_details()
model = converted_model.tflite
```

Outcome:

- It sped up the time for the game to launch and the game does not crash
- However, the accuracy of the model is always stuck at 40% which does not hit the targeted 85% accuracy

Hence, this method was not used.

2. Post-training quantization
    - This is a process of reducing the size of the Tensorflow lite model. It quantizes weights to 8-bits of precision from floating-point and dequantizes them during runtime to perform floating point computations.
    - However, even though this will speed up the process, this could have some accuracy implications

    This is being done by using the program below:

```
import tensorflow as tf
converter =
tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
tflite_quant_model = converter.convert()
```

Outcome:

- The accuracy of the model dropped to 40% and it always give the same results of stone no matter what gesture is being given.

3. Magnitude-based weight pruning with Keras
    - Weight pruning will eliminate the unnecessary values in the weight tensor. This will set the neural network parameters value to 0 to remove low-weight connections between the layers of the Convolutional Neural Network model.

    These are the steps and programs required to prune the Keras model:

Train a MNIST model with Keras without pruning

```
#Batch normilization
import os
import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense,
BatchNormalization
from keras import backend as K
from keras.callbacks import EarlyStopping
from PIL import Image
from time import time
from tensorflow.python.keras.callbacks import TensorBoard

# dimensions of our images.
img_width, img_height = 100, 100
```

```
train_data_dir = r'C:\Users\8051\Desktop\Saori\hand
dataset\binary\images\train'
validation_data_dir = r'C:\Users\8051\Desktop\Saori\hand
dataset\binary\images\test'
model_save_path = r'C:\Users\8051\Desktop\Saori\hand
dataset\binary\binary_model.model'
weight_save_path = r'C:\Users\8051\Desktop\Saori\hand
dataset\binary\binary_weight.h5'
nb_train_samples = 2981
nb_validation_samples = 594
epochs = 50
batch_size_train = 128
batch_size_test = 32
# if K.image_data_format() == 'channels_first':
#     input_shape = (1, img_width, img_height)
# else:
#     input_shape = (img_width, img_height, 1)


model = Sequential()
model.add(Conv2D(32, (3,3), activation='relu', input_shape=(100, 100, 1)))
model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D((2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))

tensorboard = TensorBoard(log_dir='log/{}'.format(time()))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)
#    brightness_range=(0.5, 1.5),

# this is the augmentation configuration we will use for testing:
# only rescaling
test_datagen = ImageDataGenerator(rescale=1. / 255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size_train,
    class_mode='categorical',
    color_mode='grayscale')

validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size_test,
    class_mode='categorical',
    color_mode='grayscale')

train_history_adam = model.fit_generator(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size_train,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size_test)

callbacks=[tensorboard]
model.save_weights(model_save_path)
model.save(weight_save_path)
```

Train the pruned model layer by layer

```
# Backend agnostic way to save/restore models
# _, keras_file = tempfile.mkstemp('.h5')
# print('Saving model to: ', keras_file)
# tf.keras.models.save_model(model, keras_file, include_optimizer=False)

# Load the serialized model
loaded_model = tf.keras.models.load_model(weight_save_path)

epochs = 4
end_step = np.ceil(1.0 * nb_train_samples / batch_size_train).astype(np.int32)
* epochs
print(end_step)

new_pruning_params = {
      'pruning_schedule': sparsity.PolynomialDecay(initial_sparsity=0.50,
                                                   final_sparsity=0.90,
                                                   begin_step=0,
                                                   end_step=end_step,
                                                   frequency=100)
}

new_pruned_model = sparsity.prune_low_magnitude(loaded_model,
**new_pruning_params)
new_pruned_model.summary()

new_pruned_model.compile(
    loss=tf.keras.losses.categorical_crossentropy,
    optimizer='adam',
    metrics=['accuracy'])
```

Train the pruned model

```
callbacks = [
    sparsity.UpdatePruningStep(),
    sparsity.PruningSummaries(log_dir=logdir, profile_batch=0)
]
new_pruned_model.fit(train_generator,
          batch_size=batch_size_train,
          epochs=epochs,
          verbose=1,
          callbacks=callbacks,
          validation_data=(validation_generator))
```

```
score = new_pruned_model.evaluate(validation_generator, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Results of the pruned model

```
final_model = sparsity.strip_pruning(new_pruned_model)
final_model.summary()
```

Compare the original and the pruned model

```
import tempfile
import zipfile

_, new_pruned_keras_file = tempfile.mkstemp(".h5")
print("Saving pruned model to: ", new_pruned_keras_file)
tf.keras.models.save_model(final_model, new_pruned_keras_file,
include_optimizer=False)

# Zip the .h5 model file
_, zip3 = tempfile.mkstemp(".zip")
with zipfile.ZipFile(zip3, "w", compression=zipfile.ZIP_DEFLATED) as f:
    f.write(new_pruned_keras_file)
print(
    "Size of the pruned model before compression: %.2f Mb"
    % (os.path.getsize(new_pruned_keras_file) / float(2 ** 20))
)
print(
    "Size of the pruned model after compression: %.2f Mb"
    % (os.path.getsize(zip3) / float(2 ** 20))
       )
```

Convert the model to a tflite model

```
import tensorflow as tf
converter =
tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
tflite_quant_model = converter.convert()
```

__**Outcome:**__

- The following is printed out:
  Saving pruned model to:
  C:\Users\8051\AppData\Local\Temp\tmp1dsfcvzt.h5
  Size of the pruned model before compression: 72.09 Mb
  Size of the pruned model after compression: 41.37 Mb
- This showed that the size of the model is reduced after pruning
- However, the results of the prediction is the same as converting the model to tflite model with an accuracy of 70%

4. Freezing of Keras model and convert into TensorRT model
    - TensorRT quantizes the model from 32-bit floating point to 16-bit floating point which is efficient in reducing memory consumption. It also fuses the layers and tensor together which further optimizes the use of GPU memory and bandwidth.
    - The program being used is the following below:

Freeze Keras model

```
import tensorflow as tf

from tensorflow.python.framework import graph_io

from tensorflow.keras.models import load_model



# Clear any previous session.

tf.keras.backend.clear_session()


save_pb_dir = './model'

model_fname = './model/model.h5'

def freeze_graph(graph, session, output, save_pb_dir='.', save_pb_name='frozen_mod
el.pb', save_pb_as_text=False):

    with graph.as_default():

        graphdef_inf = tf.graph_util.remove_training_nodes(graph.as_graph_def())

        graphdef_frozen = tf.graph_util.convert_variables_to_constants(session, gr
aphdef_inf, output)

        graph_io.write_graph(graphdef_frozen, save_pb_dir, save_pb_name, as_text=s
ave_pb_as_text)

        return graphdef_frozen


# This line must be executed before loading Keras model.

tf.keras.backend.set_learning_phase(0)


model = load_model(model_fname)


session = tf.keras.backend.get_session()


input_names = [t.op.name for t in model.inputs]
```

```
output_names = [t.op.name for t in model.outputs]


# Prints input and output nodes names, take notes of them.

print(input_names, output_names)


frozen_graph = freeze_graph(session.graph, session, [out.op.name for out in model.
outputs], save_pb_dir=save_pb_dir)

import tensorflow.contrib.tensorrt as trt


trt_graph = trt.create_inference_graph(

    input_graph_def=frozen_graph,

    outputs=output_names,

    max_batch_size=1,

    max_workspace_size_bytes=1 << 25,

    precision_mode='FP16',

    minimum_segment_size=50

)

graph_io.write_graph(trt_graph, "./model/",

                     "trt_graph.pb", as_text=False)
```

Outcome:

- The model was unable to make any predictions.

Integration and Testing

Folder Structure

To run the Scissor Paper Stone Game, several folders are created to organise the program.

- Scissor Paper Stone Game
  - CNN

    (Contains everything related to CNN model)
    - images

      (Contains images used for training)
      - test
      - train

        (Contain images used for validation and training of CNN model respectively)
    - binary_training.py

      (Program to run the training of CNN model)
    - gathering_binary_image.py

      (Program to take binary images for CNN model training)
    - binary_CNN_model.h5

      (CNN model used for the game)
  - Music

    (Contains music files and program to play the music during game)
    - Credits
    - Draw
    - Game
    - Instruction
    - Intro
    - Lose
    - Win

      (Contain music file to play during the game at certain screen)
    - music.py

      (Contains functions to load and play music during game)
  - Pictures

    (Contains pictures used in the game and the program to load it)
    - Credits
    - Tutorial

      (Contain pictures used for Credits and Instruction Screen)
    - background.png
    - hand.png
    - paper.png
    - scissors.png
    - stone.png

(Pictures for default background, for Scan screen, and for representing computer's gesture during the game respectively.)

pictures.py

(Script to load all pictures in the Pictures folder)

Sound

(Contains sound files and program to play the sound effect at certain event)

NFF-ethno-confirmation.wav
NFF-funny-cancel.wav
NFF-gong.wav
NFF-lose.wav
NFF-menu-04-a.wav
NFF-menu-a.wav
NFF-prompt-soft.wav
NFF-select-04.wav

(Sound files for the game)

sound.py

(Contains function to load and play sound)

Utils

(Contains function and values to be used frequently in the Main program)

color.py

(Script that contain variables stored with RGB values for font and shape colours used in the game)

utils.py

(Contain function that are used frequently)

main.py

(Main Program of the game)

Steps to run the game software

Switch on the 3 of the switch, 1 for the Jetson Nano, Adafruit Crickit Hat and the power supply

Open up the terminal on the Jetson Nano

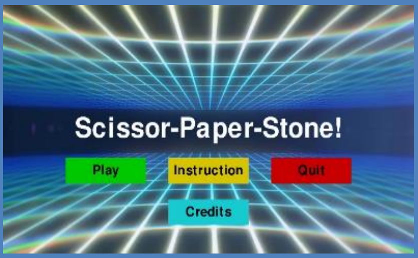
Path it to the folder that has the game in it by using this code in the terminal:

```
cd/Desktop/Scissor Paper Stone Game/Main
```

Run the program by:

```
Sudo python3 main_keras.py
```

The game will launch and this screen would appear



Key Technical Issues & Solutions

1. Program was not able to detect gestures that are not the front view.
    a. **Description:**
    When the users are playing the game, they tend to display the hand gesture at a certain angle or sideways as naturally we do not display the front view of our hand when we are playing Scissors, Paper, Stone with someone else in real life. Hence, the AI is not able to detect what gesture is that.
    b. **Solution:**
    I have captured around 1200 images for each gestures by different people, on different backgrounds and gestures with a different angle instead of only the front view. After that, these images are being trained onto the Convolutional Neural Network model which tallies up to a total of 2200 images of each gesture being trained into it.

2. The robot arm had some problems with the movements of the fingers
    a. **Description:**
    As solenoids are being used to actuate the movements of the fingers, there was a problem where the solenoids are not able to push the fingers out to actuate the movement.
    Solution:
    The springs for the fingers had been weaken by cutting them shorter. This reduces the force required to actuate the fingers.

Conclusion

Accomplishments

- Added a way for the program to be able to record and save the binary image of the user’s hand when playing the game
- These images were used to retrain the convolution Neural Network model which helped to improve the accuracy of gesture recognition
- Added 4 different difficulties to the game to make it more interesting and challenging for the user
- Wired and programmed the robot arm to be able to display the AI’s gesture to make the game more interactive
- Tried different solutions to improve the speed of the game which were stated above. However none of them worked

Further Enhancement

Problems:

- The game still requires time to load in the Convolution Neural Network model when it first launched and doing the first prediction
- Lighting of the environment still affects the binary images that are captured by the program to be fed into the CNN model to do predictions which will still affects the accuracy
- The second solenoid on the robot arm still has a little problem as it is unable to properly push the fingers of the arm out

Suggested improvements:

- The bearing on the robot arm is not in a very good condition causing the problem with the fingers not being able to move consistently

Appendices

Reference

May 7, 2015. Finger Tracking with OpenCV and Python.

http://www.benmeline.com/finger-tracking-with-opencv-and-python/

(Last Accessed: 3 June 2019)

Chin Huan Tan. Real-time Finger Detection.

https://becominghuman.ai/real-time-finger-detection-1e18fea0d1d4

(Last Accessed: 3 June 2019)

Francois Chollet. Building powerful image classification models using very little data.

https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

(Last Accessed: 3 June 2019)

Introduction to PyGame.

https://pythonprogramming.net/pygame-python-3-part-1-intro/

(Last Accessed: 1 June 2019)

Displaying images with PyGame.

https://pythonprogramming.net/displaying-images-pygame/

(Last Accessed: 1 June 2019)

Displaying text to PyGame screen.

https://pythonprogramming.net/displaying-text-pygame-screen/

(Last Accessed: 1 June 2019)

Drawing objects with PyGame.

https://pythonprogramming.net/drawing-objects-pygame-tutorial/

(Last Accessed: 1 June 2019)

Drawing Objects and Shapes in PyGame.

https://pythonprogramming.net/pygame-drawing-shapes-objects/

(Last Accessed: 1 June 2019)

PyGame Buttons, part 1, drawing the rectangle.

https://pythonprogramming.net/pygame-buttons-part-1-button-rectangle/

(Last Accessed: 1 June 2019)

PyGame Buttons, part 2, making the buttons interactive.

https://pythonprogramming.net/making-interactive-pygame-buttons/

(Last Accessed: 1 June 2019)

PyGame Buttons, part 3, adding text to the button

https://pythonprogramming.net/placing-text-pygame-buttons/

(Last Accessed: 1 June 2019)

PyGame Buttons, part 4, creating a general PyGame button function.

https://pythonprogramming.net/pygame-button-function/

(Last Accessed: 1 June 2019)

PyGame Buttons, part 5, running functions on a button click.

https://pythonprogramming.net/pygame-button-function-events/

(Last Accessed: 1 June 2019)

Sounds and Music with PyGame.

https://pythonprogramming.net/adding-sounds-music-pygame/

(Last Accessed: 1 June 2019)

Converting model to Tensorflow lite

https://www.tensorflow.org/lite/convert

(Last Accessed : 15 August 2019)

Quantization of model

https://www.tensorflow.org/lite/performance/post_training_quant

(Last Accessed : 15 August 2019)

Freezing Keras model to run on Jetson Nano

https://www.dlology.com/blog/how-to-run-keras-model-on-jetson-nano/

(Last Accessed : 15 August 2019)

Pruning Keras model

https://www.dlology.com/blog/how-to-compress-your-keras-model-x5-smaller-with-tensorflow-model-optimization/

(Last Accessed : 15 August 2019)

Different difficulties for the game software

https://github.com/goelp14/RockPaperScissors

(Last Accessed : 15 August 2019)