Punch-A-Thon
CS4242 Final Report

Igor Kochanski - 23358459
Emily Domini - 23362235
Ciaran Whelan - 23370211

1 INTRODUCTION

We aimed to develop a punch-the-bag game called “Punch-A-Thon” similar to the popular arcade game “Punch-Out!”. Our goal was to create an application that used multiple inputs to form an interesting system that can be played as a game. It used the built-in microphone and accelerometer of an Arduino Nano 33 BLE Sense along with FaceOSC to calculate a score based on the maximum acceleration of the player’s fist, their shout, and their facial expression. To score a high amount of points, the player is required to punch fast, shout loudly, and make an angry face.

2 RESULTS OF OUR RESEARCH

2.1 Possible Applications in the Martial Arts Industry

There are a wide range of applications for our project within World Boxing and UFC industries. It can be utilised in tracking the form, speed, and precision of an athlete's attack. It can also give data on where an athlete can lose performance and how long they can defend themselves for. This is useful both for athletes and coaches as the data received can be used to eliminate weaknesses in the fighter and gives continuous updates on their improvement. Coaches could potentially assess opponents’ strengths and weaknesses in the same manner, helping to create strategies for their fighters.

Our game can be used to provide information to judges during a professional fight, allowing for more in-depth analysis and enabling them to make more informed decisions. The system’s use as a video assistant referee (VAR) of sorts is another possible application in MMA. While we don’t have video footage of each punch, the directional acceleration of the punch can be of use to judges in determining whether or not a foul was conceded.

2.2 Existing Applications

We have found that similar applications exist, however none exactly like ours. There is an AI punching bag called BHOUT which tracks the user's energy, strength and technique. It gives feedback on the person's performance, specifically on areas for improvement and injury prevention.

Most AI models developed for boxing use a classification algorithm to identify the type of punch delivered, for example left jab, right hook, uppercut, etc.. They use gyroscope values as well as x, y, and z acceleration from an inertial measurement unit (IMU) built into the boxer’s glove to track the direction and acceleration of the punch. These factors are monitored and assessed by coaches to determine their fighter’s dynamics of change. The statistics provided allow them to see if their strategies and approaches to training are effective or if adjustments are necessary to develop the boxer’s skills further. However our system is unique in that we incorporate not only punch statistics, but also shout scores and emotion expressed. We created a game which can be useful for martial arts training, rather than a system solely for improving the fighter.

Other similar applications include win prediction algorithms for boxing and MMA matches. DeepStrike, for example, is a system developed by Jabbr.ai last year with a multitude of functions. It has been used by the public to calculate the odds of wins and losses and place bets on fighters. Much like our model, it has potential as assistance to boxing judges in calling fouls and awarding points.

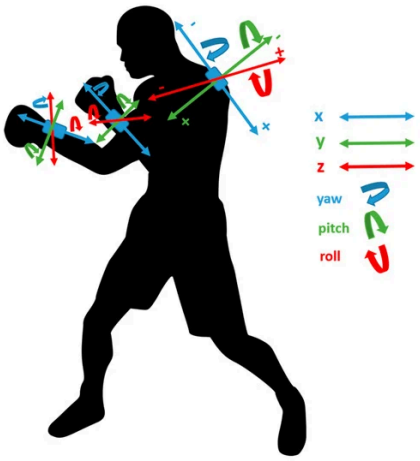

Figure 1: IMU Axis Orientation

2.3 The Use of Face Recognition and its Dangers

Face recognition is a form of visual pattern recognition based on the identification of people using facial feature data, such as the size, location, and distance between different parts of the face. Images are collected, processed, and stored by machines and can then be used to match personal data to the person’s photograph. Research on this area of technology began in the 1950s, however little progress was made until the 1990s. Nowadays facial recognition technology is implemented everywhere, from mobile phone security, airport passport control, event and hotel check-ins, and, most controversially, advertising.

The reason for this controversy is that face data collected by advertising companies is often done so without explicit consent. They are intended to gather data about target markets and audiences for products and media, so that specific advertisements can be directed towards you and companies can reach more customers. The ethical issue with this practice surrounds the fact that these images of our faces taken from social media, security scans, and CCTV are not always encrypted and

so can be accessed by anyone and distributed anywhere around the world. Many see this as a breach of privacy laws as their faces are free to be sold online or used to create provocative content known as DeepFakes. DeepFakes are AI-generated photos and videos where people's faces and voices are manipulated to create fake scenarios and speeches. They are often used to spread misinformation, particularly during political campaigns, in the form of "fake news". This is where artificial videos are procured featuring political figures spreading propaganda and offensive, untrue statements. With recent advancements in AI, it's becoming increasingly difficult to differentiate between artificially generated content and real, authentic interviews.

Where this topic links to machine learning in martial arts has to do with the gathering of face and body gesture data to classify MMA moves and types of punches. Although the data in gesture classification models is used solely within that system, there is always a possibility that the images can be harvested by outside parties and sold to ill-intentioned clients online.

2.4 Voice Recognition Features in Machine Learning

Our decision to implement a shout recognition aspect in our game led us to consult previous research we had done on feature extraction for audio and voice recognition.

When it comes to extracting features from audio for audio analysis, there are three main characteristics that are of interest to us: time period, frequency, and amplitude. Each of these contain their own unique features that are used to classify and compare audio samples.

The time period of a sound is defined in seconds as the time taken to complete one cycle of vibrations. Frequency, sometimes known as pitch, indicates the number of sound vibrations per second and is measured in Hertz. In visual representations, frequency is often plotted against the time period. Amplitude refers to the "loudness" or sound intensity of a sample, measured in decibels. Amplitude is often expressed visually using a waveform, which illustrates its change over time.

Important audio features that fall under the umbrella of the time domain are the Amplitude Envelope, the Root-Mean-Square Energy, and the Zero-Crossing Rate. The Amplitude Envelope splits a waveform diagram to find the beginning and end of sounds, and hence their duration. The Root-Mean-Square Energy is the average energy value of the signal. The Zero-Crossing Rate is a measure of how many times the signal wave crosses the x-axis per frame, detecting the presence and/or absence of speech and sound.

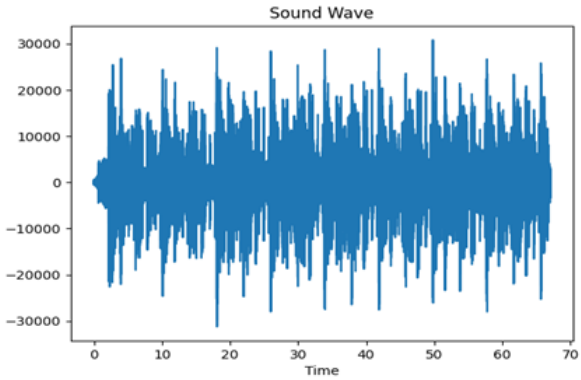

Figure 2: Waveform Diagram

Some features of the frequency domain include averages, the Band Energy Ratio, and the Signal-to-Noise Ratio. These averages usually refer to the mean and median value of the audio frequency. The Band Energy Ratio is a determination of relations between higher and lower frequency bands. The Signal-to-Noise Ratio defines a comparison between the signal strength of the desired sound and background noise.

Time-frequency domain features known as Mel-Frequency Cepstral Coefficients (MFCC) are arguably the most useful and significant in audio extraction and voice recognition. MFCCs are used to capture the unique characteristics of audio signals. They represent the short-term power spectrum of a sound and are based on the Mel Scale and Mel Spectrograms.

The process of extracting these features from audio clips more often than not consists of converting the clips to graphs and procuring visual representations of each feature. Classification of audio clips in machine learning involves comparing graphs of these features found in the input sound samples against those in the training data. This method is effective as computers are much better at deciphering the similarities and differences between graphs than interpreting raw audio files, hence allowing us to build classification algorithms.

3 DEVELOPMENT PROCESS

3.1 Overview of the Initial Idea

We wanted to build a system that not only took in the acceleration of a punch but wanted to include some other factors to make it more interesting. Initially we decided to combine 3 outputs to generate a final score: punch acceleration, shout volume, and anger in facial expression. An accelerometer would be placed in the player's punching hand, while another is positioned near the face to act as a microphone, and a laptop camera is analysing the player's facial expression. All of these values would be manipulated to generate scores for each individual aspect, and an overall score, presented with feedback in the form of text, ratings, and images. As our system is a Punch-A-Thon game, we intended to keep our feedback funny and lighthearted, in no way a serious assessment of punching ability.

3.2 Choice of Algorithms

3.2.1 Regression:

We decided to utilise a regression algorithm, particularly a neural network, for the punch measuring system. There were a number of continuous input values (x, y, z acceleration, x, y, z rotation initially) that we wanted to use to calculate one output. This output would be a punch score in the range 0 to 1, which would later be multiplied by the maximum acceleration and combined with the other outputs. The choice of a neural network over polynomial regression was due to how both algorithms responded to outliers in the training data. Punches can vary immensely in direction, timing, speed, and rotation, to name only a few factors. Therefore it was crucial that we choose a model which could consider such variances in the output, rather than ignore them completely, which is what we found a polynomial regression algorithm did. In the end, a neural network with 6 nodes (for the 6 inputs) per layer and 1 hidden layer was sufficient in recognising a straight punch and awarding an appropriate score.

3.2.2 Classification:

The algorithm we chose to use for recognising shouts was classification using a neural network because it allowed us to detect whether the player has shouted, or not. Since everyone shouts differently, using a machine learning model was the most efficient and effective way to differentiate whether someone shouts or just says a word loudly. We ended up using the certainty value of the machine learning model to score the player on how close their shout was to the sample shouts we recorded.

3.2.3 Dynamic Time Warping:

We opted to use a Dynamic Time Warping algorithm for face gesture recognition. We wanted to have a system that reliably detects the changes in a person's face as their expression changes in a short frame of time. The output algorithm provides a score based on the user's gesture (Anger: 1.5, Neutral: 1, Happy: 0.5) that would be added to the shout certainty score and multiplied into the final score and achieve the final output. As everyone has a different face shape, Dynamic Time Warping allowed us to tweak the certainty threshold of a person's facial expression which was important for differentiating between Angry and Neutral expressions on account of their similarity. We also needed to set a timed recording period for the expression in order to get an average expression for the duration of the punch so you couldn't change your facial expression last second or have it change if you stop doing the expression too early. These criteria led us to choose Dynamic Time Warping, using 8 inputs that we thought would be best for classifying each facial expression and were suitable for detecting the correct expression and thus providing the related score multiplier.

3.3 Training Process and Training Data

3.3.1 Acceleration using Regression:

Training data for the punch acceleration regression model was recorded using Wekinator. We took samples from various people with differing punch styles and strength levels to create a dynamic and accurate definition of a good punch, which in this case was as fast and straight in the x direction as possible. To account for poor quality punches and non-punch movements, we also recorded the IMU values when it was stationary, being held still in the player's hand, and being moved in the y or z direction. We trained the Wekinator programme to output numbers on a scale of 0 to 1, where 1 was allocated to perfectly straight punches with a high x acceleration, 0 was for non-punch movements, and all other punch attempts fell somewhere in between. Initially, we trained the model using 80 samples from many different people, but found that there was too large of a deviation in the samples that we had deemed a "perfect punch". The outputs were quite random and unreliable, so we reduced the amount of training data to 40 samples. This immediately made a significant improvement to our model, with the output matching our perception of the punch score.

We'd like to note, following valuable peer feedback, that it is possible to incorporate punch acceleration into our game without the use of machine learning. If we were to compute x acceleration mathematically, either with the distance formula or the magnitude of x acceleration as a vector in relation to the other 2 directions, we would receive values that could be mapped to outputs manually, for example in Python code or MaxMSP. This process is just as probable and accurate as a machine learning model and both methods would generate the same outcome. However we deduced that it was less time-consuming and less prone to error to train a regression model, so personal preference led us to choose that approach.

3.3.2 Detecting shouts using Classification:

We used Edge Impulse to record data and train a machine learning model that we later deployed to run off of an Arduino Nano 33 BLE Sense. This process had many challenges that we had to overcome, our greatest challenge being obtaining useful sample data of shouts to train the machine learning model. We did not find a suitable dataset for this project online, which meant that we needed to record our own samples. We sampled shouts from members of our team as well as from other people who agreed to provide samples. We used the shout samples to train a classification model to recognise shouts, however we also ended up using the audio samples of "other" and "unknown" from previous labs to erase background noise and improve the accuracy of the model.

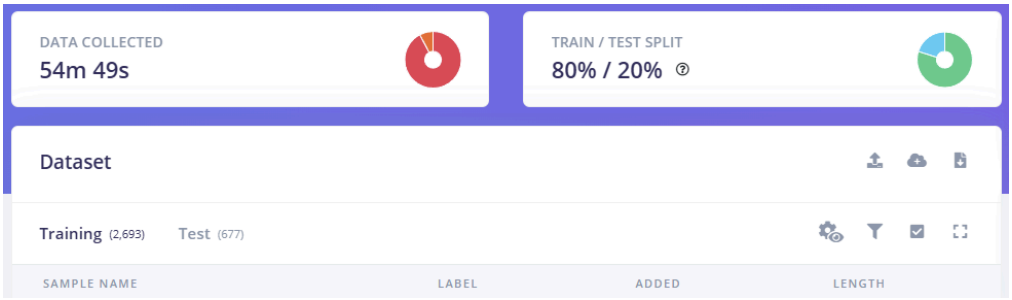

Figure 3: Edge Impulse data acquisition

After splitting our data to have an 80% training and 20% test split, we trained and retrained the model using different settings until we were satisfied with the model's accuracy. The final version used 150 training cycles of a neural network and had a 91.6% accuracy in correctly identifying shouts. We believe that this result could be greatly improved with a larger and more varied sample size, however due to our limited resources this was not possible. To test the accuracy of our model, we deployed it onto the Arduino microcontroller, ran the provided Edge Impulse sample code and had multiple different people shout. Unfortunately, each new version of the model took over 20 minutes to compile and deploy before we could test it. This greatly impacted the rate at which we could test different versions of the model. We did not use the built in "Live Classification" section of Edge Impulse projects to test each version of the model as it was very slow in testing large amounts of different live samples.

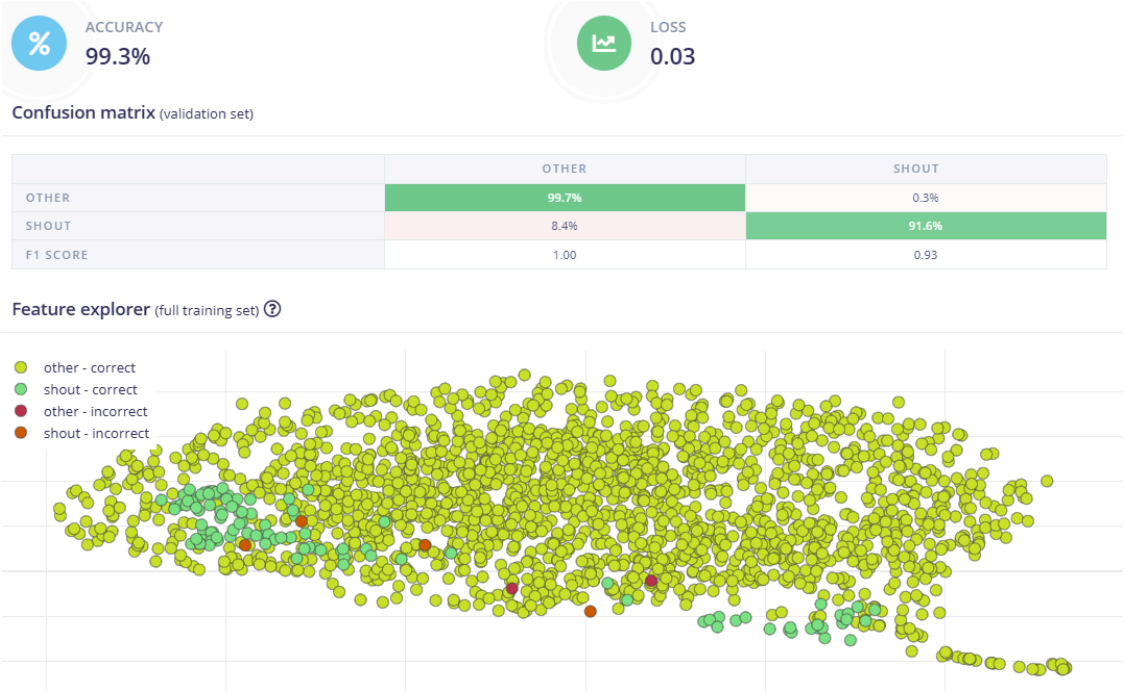

Figure 4: Final model training performance

We decided not to use the "continuous" version of the sample code as we found it struggled to identify shouts that occurred between two sampling windows. We required the system to reliably detect a single shout, therefore we decided to use the timed version of the sample code that we modified by replacing the loop segment to also be able to communicate with MaxMSP and provide a visual LED countdown to the sampling frame and feedback.

3.3.3 Face gesture using Dynamic Time Warping

The best way we found to train our Dynamic Time Warping model was using Wekinator. Initially we tried using the raw data points that FaceOSC provided but we found that lead to inaccuracy in the model due to the variance of the location and scale of the face within the FaceOSC window. We decided to put the data through MaxMSP instead, as Wekinator is unable to take multiple input messages. This allowed us to get more useful data from the other data sets that FaceOSC provided, which helped with the raw data problems and also the overall accuracy of the model. We trained the Wekinator program on what we thought would be the most fitting expressions for each of our 3 outputs. To take account of the different sizes and shapes of a face, we gathered training data

from many different people. The most difficult part of training our model came from Wekinator struggling to differentiate between Angry and Neutral, but in the end we managed to train the model to be able to distinguish them to a suitable standard.

Upon reflection, if we were going to create a system that could better detect facial expressions we would try and incorporate a better application for face tracking, as whilst using FaceOSC the tracker would occasionally become distorted or lose track of the users face and cause the output of the model to deviate. Additionally, we would try to include a larger data set for more accurate results, however after searching the internet we were unable to find a resource that would provide us with the data we needed. Though if we were to include an online data set, we believe it wouldn't be a fair demonstration of our knowledge in training and applying our model.

3.4 Feature Extraction

3.4.1 Acceleration:

The choice of features to extract for the punch acceleration came down largely to trial and error. We experimented with a number of different inputs and input combinations before deciding which was most accurate and most useful for our project. We started off measuring only acceleration in the x direction for the punches, but found that this was too simple to implement machine learning and so wouldn't be a solid representation of our knowledge. We tried to incorporate gyroscope values along with 3-directional acceleration, but found that training this model for accuracy of punch score proved very difficult. There were too many variations in input to map to one output, for example a punch with some rotation could still be considered by us to be a quality punch, but in trying to mitigate loose, swinging punches, the system deemed it poor. For this reason, these features were not appropriate for our game. In the end, what worked best was simply measuring 3-directional acceleration and training the model to award higher points to punches with high x acceleration, and little to no y or z acceleration.

We extracted the acceleration features by editing the code for the continuous accelerometer in Arduino's LSM9DS1 library so that the serial monitor received the value of acceleration preceded by the letter of its direction, as seen below.

```
39
40  if (IMU.accelerationAvailable()) {
41    IMU.readAcceleration(x, y, z);
42
43    Serial.print('x');
44    Serial.print(x);
45    Serial.print('\t');
46    Serial.print('y');
47    Serial.print(y);
48    Serial.print('\t');
49    Serial.print('z');
50    Serial.print(z);
51    Serial.println('\t');
52
```

Figure 5: Arduino accelerometer serial output code

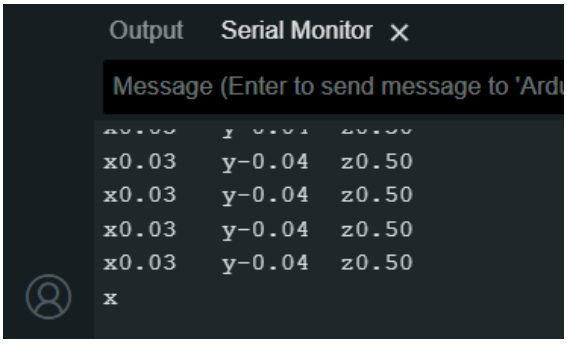

Figure 6: Serial monitor output for accelerometer

These continuous figures were then sent into our MaxMSP patch, where they were routed by preceding letter and then fed to Wekinator as inputs.

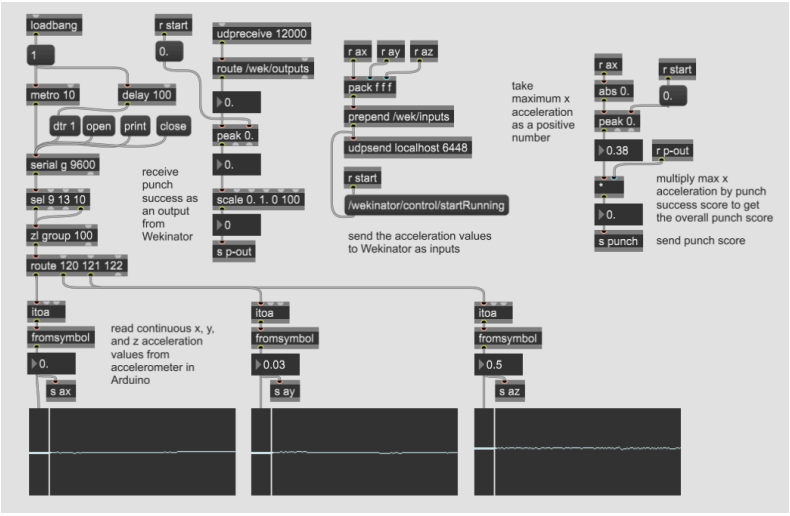

Figure 7: MaxMSP manipulation of accelerometer data

3.4.2 Audio:

To extract useful features from audio samples for training the machine learning model, we used the Audio MFCC (Mel Frequency Cepstral Coefficients) processing block in Edge Impulse as it is more suited to extracting features of human voice compared to Audio MFE or Spectrogram. We left most of the MFCC parameters unchanged and adjusted the window size and frame length of the classification model by trial and error to get more accurate results. In the final version we set the window size to 560 ms with an increase of 100 ms and a frame length of 20 ms because most shouts are single syllable and short.

3.4.3 Face:

For our first method of feature extraction from FaceOSC, we attempted to centre it on a point using the raw data points, but we discovered that this was too inaccurate and would require more than 100 calculations. For our second method, we adopted a different strategy, instead sending the data to MaxMSP to extract the most suitable data to send to Wekinator. Receiving data from the gestures area of FaceOSC greatly improved the model, as it no longer required an x, y, z position, but rather the distance between multiple nodes and also scaled as the user got closer or further from the camera. The data was sent to Wekinator and trained on 3 outputs. The gestures we extracted from Face OSC and the output values can be seen down below.

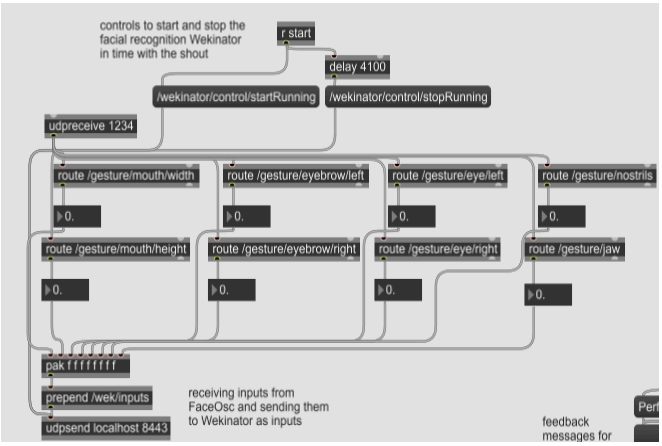

Figure 8: FaceOSC features extracted in MaxMSP

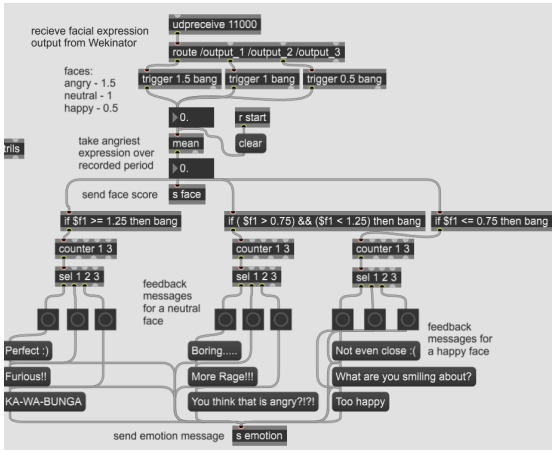

Figure 9: Results from Wekinator

3.5 Development of the Output

We programmed the Arduino to output a visual countdown for the microphone, by replacing the loop section of the Edge Impulse sample code with our own. After this change the Adruino counted the player down from three until their shout window. We did this using the built-in LEDs on the microcontroller. Controlling the red, green and blue LEDs allowed us to display a colour coded countdown. The Arduino blinks once white to confirm that it's receiving serial, then begins a countdown of three yellow blinks followed by a magenta-coloured recording phase. After the audio is classified, the Arduino blinks green if it is over 70% certain that the audio was a shout and blinks red if not.

```
78  void loop() {
79    if (Serial.available()) { // if serial recieved
80      //BLINK WHITE to begin
81      digitalWrite(LEDR, LOW);
82      digitalWrite(LEDG, LOW);
83      digitalWrite(LEDB, LOW);
84      delay(100);
85      digitalWrite(LEDR, HIGH);
86      digitalWrite(LEDG, HIGH);
87      digitalWrite(LEDB, HIGH);
88      delay(500);
89      int maxMSP = Serial.read();
90      if(maxMSP == 1){ // if serial corresponding to START
91      //BLINK YELLOW
92        ei_printf("\nStarting in 3 seconds...\t\n");
93        ei_printf("3...\t");
94        digitalWrite(LEDR, LOW);
95        digitalWrite(LEDG, LOW);
96        delay(500);
97        digitalWrite(LEDR, HIGH);
```

Figure 10: Code to control LED feedback

We decided to use MaxMSP to bring all of our outputs together and calculate a final score. All three of the separate machine learning systems' outputs were fed into one MaxMSP patch, where they were modified and combined into a final score. This score is calculated by adding together the shout prediction output and the number of the facial expression classification, then multiplying that by the multiplication of the punch certainty score and the maximum recorded x acceleration. We tried a few different combinations of scores before we found one that we felt best represented the given punches, shouts, and faces. At first, we used only the punch certainty score as the punch output, but found that this gave the player no indication of the strength of their punch. Multiplying by the maximum acceleration felt like a much more accurate depiction. We also changed the way we dealt with the facial scores. Initially, we had it taking the maximum expression, which we found didn't give us the results we needed as it would more than likely give Angry the entire time. We therefore changed it to the mean value of the expression outputs. It allowed us to have a better score distribution and made the gameplay experience less structured.

The final score is then displayed on a scale from 0-999 and put into four levels based on how well you did.

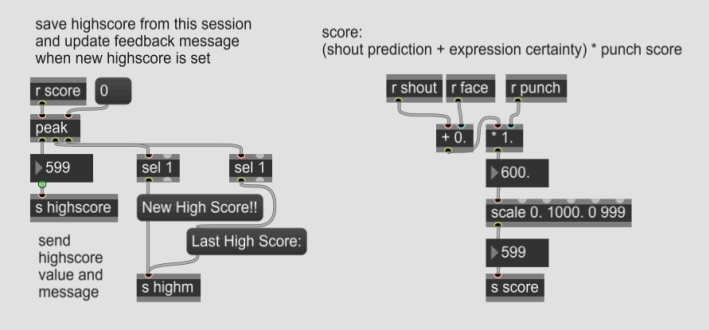

Figure 11: Calculations for score and highscore message

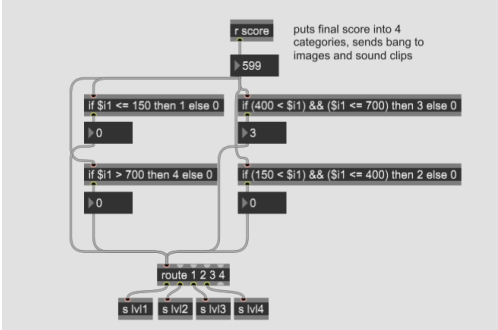

Figure 12: Dividing the score into levels

Level 1 is for the weakest punches, which score between 0 and 150. Any score between 151 and 400 gives level 2 as the output. Level 3 scores go from 401 to 700, with all higher scores being considered level 4.

In our first draft, we planned to only have one image per level and display only the image and overall score.

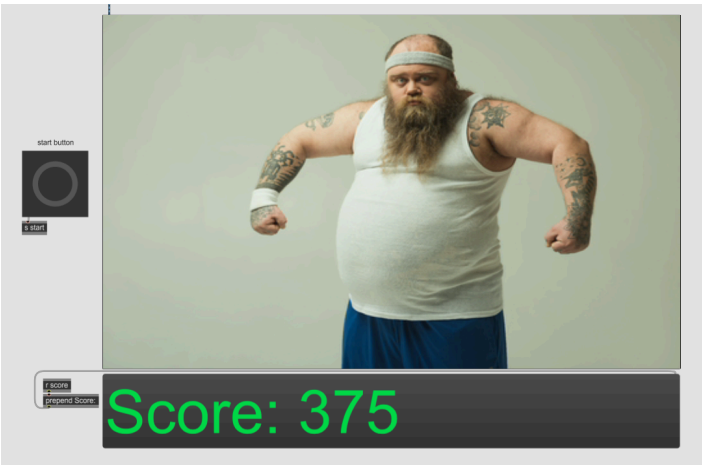

Figure 13: First draft of MaxMSP output

However we later decided to make it more interesting by adding a selection of images per level, where one would be chosen randomly when the level was triggered. We chose the images that we believed produced the best representation of each of the levels. To further enhance the enjoyment factor of our game, we decided to include an audio output. We chose 4 funny sounds, one for each level, that would play alongside the image output. In terms of user feedback, we thought it would also be useful to display the individual breakdown of each graded aspect, for example a percentage for your punch score and shout score, and text corresponding to the facial expression. So now, each level has an audio, visual, and text output.

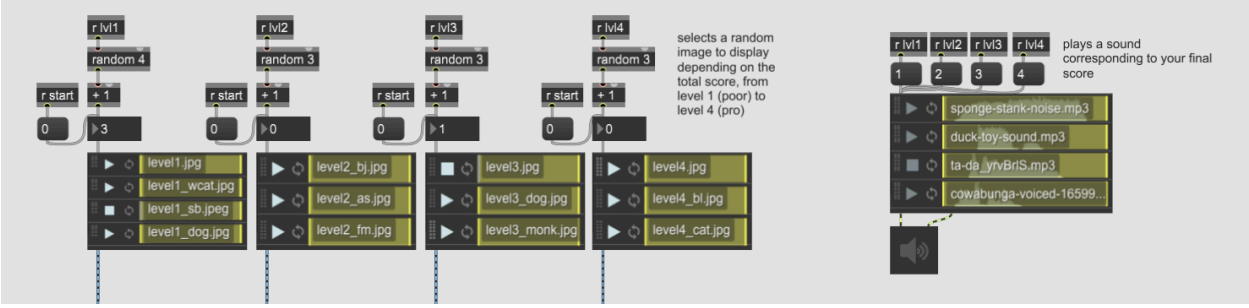

Figure 14: Audio and image selection per level in MaxMSP

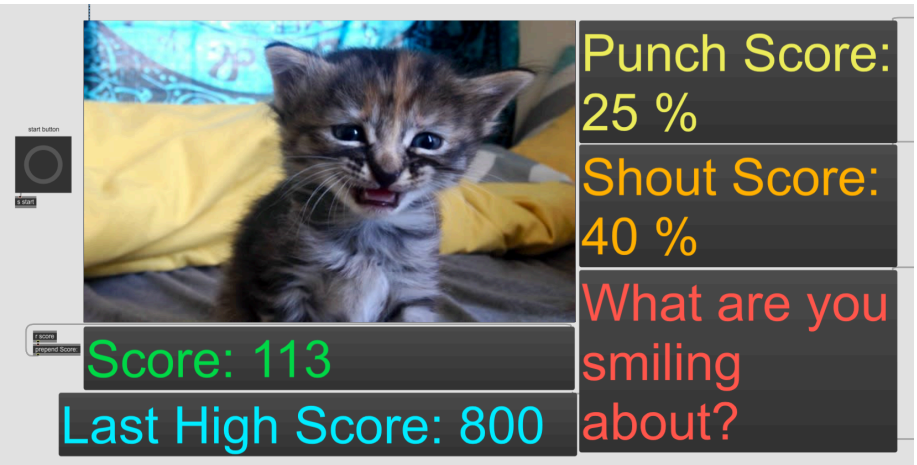

Figure 15: Wekinator final output

4 CONCLUSION AND FUTURE WORK

4.1 What We Learned

We learned how to apply our knowledge of machine learning to create an entertaining interactive system with possible future applications. This project has also taught us to work as a team and split workload effectively. We learned that creating an accurate voice recognising machine learning model is very difficult for short, single syllable keywords such as shouts, as in our project. However, we found that substantially decreasing the window size for sampling did help with recognising a shorter keyword.

We have learned how to process and interpret incoming data from applications and how to manipulate it to better suit our model's development. We learned the effect of having an appropriate amount of training samples on the model's accuracy. We found that having less, but more accurate, data points was more effective in defining each gesture. For FaceOSC, switching from raw points to gestures had less data but made the overall model more accurate.

For the punches specifically, we discovered that having too many samples where the same output was assigned to too many varying inputs created an unreliable system with erratic, unpredictable behaviour. As mentioned before, reducing sample size and making the training data more specific solved this issue.

4.2 What Worked and What Didn't

During the creation of the voice recognising machine learning model, we quickly discovered that our initial idea of using continuous sampling will not work. During continuous sampling, samples are taken one after the other. This means that if the keyword happens to fall between two sampling windows, it will not be registered. This is especially prominent with very short keywords, such as shouts in our case. Instead of the continuous approach, we decided to try a timed approach. By modifying the sample code given by Edge Impulse, we were able to create a countdown followed by a single sampling window. This ensured that the shout would always fall inside of a sampling window as long as the player shouts on time.

We encountered an issue with the limitations of the built-in Arduino accelerometer. When testing the punch aspect of our game, we noticed that all good punches couldn't surpass a maximum acceleration of $4m/s^2$, even when there was a visible difference in their acceleration. After looking through Arduino Library documentation, editing the source files, and experimenting with similar libraries, we couldn't extend this threshold past $4m/s^2$. Instead we decided to work around this by combining the punch certainty score and maximum acceleration to give a better representation of the player's punch.

In the beginning we wanted to run the game completely off of one computer, with just its built-in camera and two IMUs connected via USB. However when we tried to install FaceOSC on the main laptop, we got a recurring error no matter how many times we uninstalled and reinstalled the program. Our solution was to send the FaceOSC inputs from the other laptop to the main computer using its IP address and this worked perfectly.

4.3 Future Work

To improve the accuracy of the voice recognition, a far larger dataset could be collected, as our sample size was relatively small. That would improve the accuracy of our shout recognising model greatly. With the current amount of sample data the system is quite unreliable and only works with a few specific shouts. Increasing the recording window of the voice recognition would make the game much more playable but would require an upgrade in equipment. Facial expression recognition also requires a lot more training data. As more data with a variety of different people with different face shapes and skin tones would allow more people to use the game and make the expression recognition much more efficient.

In future, we could upgrade from the Arduino IMU to a higher quality device capable of measuring acceleration higher than $4m/s^2$. This could be connected via Bluetooth rather than USB cable to increase freedom and range for the punch. Fitting the IMU into a boxing glove or Taekwondo glove allows for greater comfort and makes the game more realistic overall.

If we were to move the output from MaxMSP to a different program, we could create a more appealing output design, possibly with more detailed feedback and generally a cleaner, more professional interface. We could look into other programming languages or IDEs to achieve this in the further development of our Punch-A-Thon game.

5 REFERENCES

5.1 Sources

[1] Mauro Frota. 2021. BHOUT. The first bag with a brain. https://www.bhout.com/

[2] Gaurav Sharma and Atul Kumar Uttam. 2021. Multilayer Neural Network Model for Mixed martial arts Winner Prediction. 5th International Conference on Information Systems and Computer Networks (ISCON), Mathura, India. https://ieeexplore.ieee.org/abstract/document/9702452

[3] Global Techtricks. 2023. Deep Strike - Jabbr.ai | Boxing AI Judge. Youtube. https://youtu.be/qk7Iw6cy3dA?si=iCGwZtg_Rp35qR3Y

[4] A.J. McClurg. 2007. In the face of danger: Facial recognition and the limits of privacy law. Harvard Law Review. https://cyber.harvard.edu/iif/sites/iif/images/Facial_recognition_privacy_law.pdf

[5] Mika Westerlund. 2019. The Emergence of Deepfake Technology: A Review. Technology innovation Management Review. https://timreview.ca/sites/default/files/article_PDF/TIMReview_November2019%20-%20D%20-%20Final.pdf

[6] Figure 2. AltexSoft. 2022. Audio Analysis With Machine Learning: Building AI-Fueled Sound Detection App. AltexSoft Software R&D Engineering. https://www.altexsoft.com/blog/audio-analysis/

[7] Alberto Casas Ortiz and Olga C. Santos. 2020. Capturing, Modelling, Analysing and providing Feedback in Martial Arts with Artificial Intelligence to support Psychomotor Learning Activities. Master's Final Project of the University Master's Degree in Advanced Artificial Intelligence. Higher Technical School of Computer Engineering, Universidad Nacional de Educación a Distancia (UNED). http://e-spacio.uned.es/fez/eserv/bibliuned:master-ETSInformatica-IAA-Acasas/Casas_Ortiz_Alberto_TFM.pdf

[8] ADPowers001. 2014. Punching bag to a measurable game. Reddit thread. https://www.reddit.com/r/arduino/comments/20py9c/punching_bag_in_to_a_measurable_game/

[9] Unknown. 2016. How do some people score high on arcade boxing machines even though I can punch harder? What is the real trick to it? Quora thread. https://www.quora.com/How-do-some-people-score-high-on-arcade-boxing-machines-eventhough-I-can-punch-harder-What-is-the-real-trick-to-it

[10] Ilshat Khasanshin. 2021. Application of an Artificial Neural Network to Automate the Measurement of Kinematic Characteristics of Punches in Boxing. Department of Data Analysis and Machine Learning, Financial University under the Government of the Russian Federation, Russia. https://www.mdpi.com/978222

[11] Figure 1. Matthew T. O. Worsey, Hugo G. Espinosa, Jonathan B. Shepherd, and David V. Thiel. 2020. An Evaluation of Wearable Inertial Sensor Configuration and Supervised Machine Learning Models for Automatic Punch Classification in Boxing. Griffith School of Engineering and Built Environment and Griffith University Sports Technology (GUST), Griffith University, Australia. https://www.mdpi.com/889064

[12] Andrey Labintsev, Ilshat Khasanshin, Dmitri Balashov, Mikhail Bocharov, and Konstantin Bublikov. 2021. Recognition Punches in Karate Using Acceleration Sensors and Convolution Neural Networks. Department of Data Analysis and Machine

Learning, Financial University under the Government of the Russian Federation, Russia. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9559989&tag=1

[13] Ilshat Khasanshin and Aleksey Osipov. 2021. Using an artificial neural network to develop an optimal model of straight punch in boxing and training in punch techniques based on this model and real-time feedback. PLOS ONE Journals. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8629283/

5.2 Software

[1] Arduino Library. https://docs.arduino.cc/
[2] Arduino LSM9DS1 Library. https://github.com/arduino-libraries/Arduino_LSM9DS1
[3] Wekinator. http://www.wekinator.org/
[4] Edge Impulse. https://edgeimpulse.com/
[5] Cycling74 Max 8 (MaxMSP). https://cycling74.com/products/max
[6] FaceOSC. https://github.com/kylemcdonald/ofxFaceTracker/releases