ISS Project (Graduate Certificate in Pattern Recognition Systems)

Magic Fight

Peng Shaoze (A0261840J)
Tian Yuyang (A0261848U)
Xu Xindi (A0261833E)
Zhang Xingyu (A0261781B)

National University of Singapore | Institute of Systems Science

https://github.com/zxypro1/MagicFight

Executive Summary

At present, traditional games have experienced a period of vigorous development, and the market contains tens of thousands of games for players to choose according to their own preferences, such as action games, role-playing games, simulation games, real-time strategy games, casual puzzle games Class games, etc., games have also formed their own architecture.

At the same time, with the continuous development of somatosensory technology, gamers have higher expectations for the game presentation form of certain types of games such as action games and role-playing games. They generally expect traditional games to be presented in combination with emerging technologies, such as transplanting traditional on-screen games into real life through AR enhancement, VR technology, and somatosensory enhancement technology.

Therefore, the development from traditional games to somatosensory games has become an important branch in the game development process. At present, somatosensory games are mostly based on the changes of the player's body movements for game operations. We take into account the pain points of current players and users. In order to meet expectations, we chose to create a game called Magic fight. By collecting and identifying the player's voice and gestures, we can determine the player's spell and spell casting gesture, breaking through the previous operation of simply inputting with the handle buttons. Way, magic fight is a new type of video game that is performed (operated) through speech recognition, body movement changes, etc., and is finally presented on the computer screen.

In order to realize this game, we have adopted a rich technology stack, including a mantra speech data collection website based on js, html and css, and embedded development of machine learning based on Arduino Nano 33 board, which involves C++, EloquentML,

Tensorflowlite, and finally we also developed games based on Unity, using technologies such as Unity BLE, C#, etc.

CONTENTS

1. Problem Statement 4
1.1. Our Proposal 4
1.2. Project Objectives 4
2. Tools and Techniques 6
2.1. Data Collection 6
2.2. Embedded Development 6
2.3. Machine Learning 7
2.4. Game Development 7
3. Model Design 7
3.1. Speech Recognition Model 8
3.1.1. CNN model 8
3.1.2. MLP Model 10
3.2. Gesture Recognition Model 12
3.2.1. CNN Model 12
3.2.2. MLP Model 14
3.3. TensorFlow Lite 15
3.4. Actual Performance 16
4. Board Embedded Development 16
4.1. Speech Recognition 17
4.2. Action Recognition 18
4.3. Bluetooth Connection 18
5. Unity Game Development 19
5.1. Game Planning 19
5.2. Art Design 20
5.3. Program Development 21
6. Conclusion 22
6.1. Summary of Achievements 22
6.2. Future Improvements 22

1. Problem Statement

1.1. Our Proposal

In 2021, the size of China's mobile game market will be about 307.8 billion yuan, a year-on-year increase of 9.8%. Judging from the overall performance of mobile game companies, the performance of the leading companies is more prominent, and the performance of the middle and tail players is quite different. Under the background of stricter policy supervision and industry standardization in the game industry, it is expected that the mobile game industry will maintain a growth rate of more than 10% in 2022. The future growth opportunities of the industry will come from the imagination space brought by new technologies such as 5G, cloud games, and metaverse, as well as the combination of new formats and mobile games. This is also an important driving factor for the continuous innovation and transformation of the mobile game industry.

The initial input method of electronic games came from computer keyboards, and later professional electronic game consoles were derived to carry out electronic game content in the form of gamepads or consoles. Later, with the advancement of technology and the need to enhance players' gaming experience, related game companies developed specialized and specialized game input devices and large-scale arcade game projects. For example, the games we played on the well-known FC (Nintendo red and white machine) at first, such as hunting ducks and flying saucers, which required the support of game light guns, became the original prototype of somatosensory games. Including the well-known VR war police in arcades, games that require players to hold a light gun with arms, waist and other limbs can be understood as the basis of somatosensory games.

The development from traditional games to somatosensory games has become an important branch in the process of game development. At present, somatosensory games are more based on the changes of the player's body movements to perform game operations. We integrate magic fight into the game With the element of sound, users can coordinate the changes of body movements through voice spells, and jointly operate the game characters to attack, defend and do other actions.

1.2. Project Objectives

Games will gradually penetrate into more diverse user life scenarios, and the barriers and boundaries between different scenarios will also be broken. Our goal is to integrate intelligent hardware devices with game scenarios to provide users with an all-round interactive gaming experience.

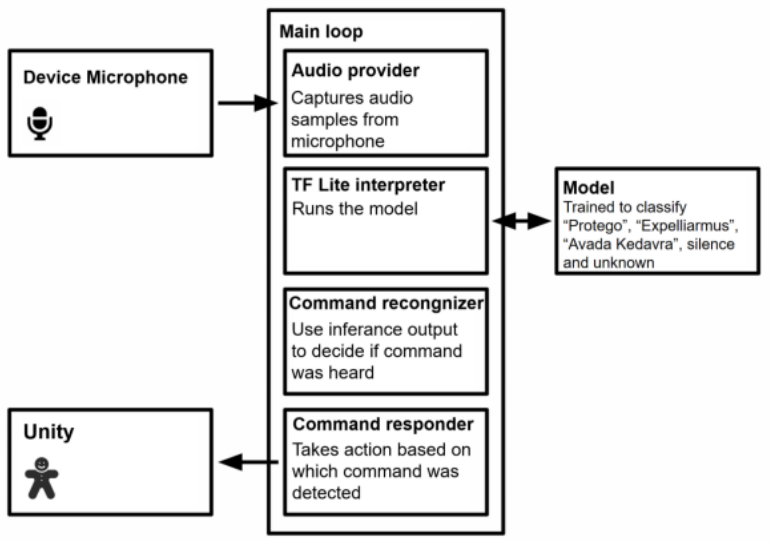

Figure 1.1: Speech recognition system architecture

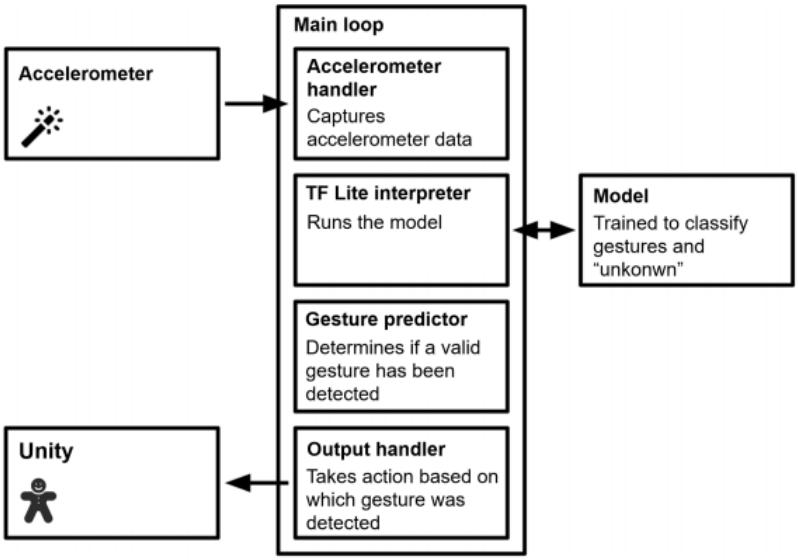

Figure 1.2: Action recognition system architecture

2. Tools and Techniques

2.1. Data Collection

The data set used for training the model is collected in two ways. The first part is the collection of voice incantations. We completed the collection of voice incantations through our own voice collection website (https://magicfight.de.r.appspot.com). , in which the website uses JS, HTML, and CSS technologies. The second part is the collection of spell casting gestures. We convert the spell casting gestures into two-dimensional coordinate data through Arduino Nano 33 BLE. In this part, we completed the embedded development of the board through C++.

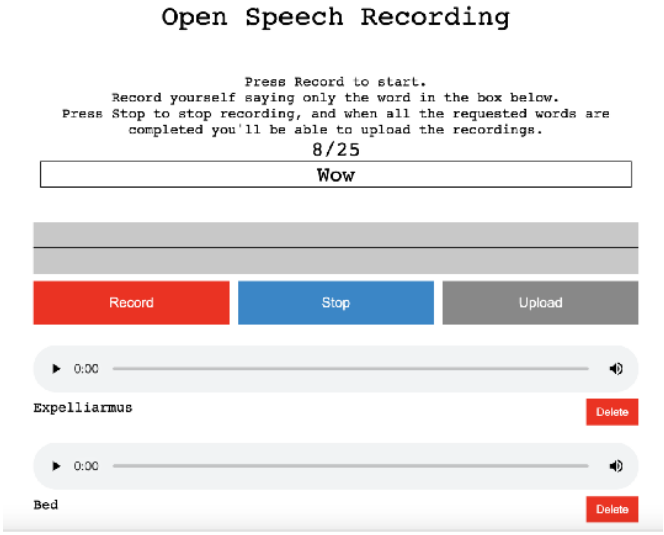

Figure 2.1: Voice data collection website

2.2. Embedded Development

We mainly carried out embedded development for Arduino Nano 33 BLE, including Bluetooth transmission module, machine learning embedded development, technology stack including C++, EloquentML, Tensorflowlite, we realized the recognition of voice spells and casting gestures, the specific machine learning model will be in Section 2.3 expands, and at the same time we complete the Bluetooth-based communication between Arduino Nano 33 BLE and PC, and transmit the data of the recognized voice spells and casting gestures to Unity through the Bluetooth module.

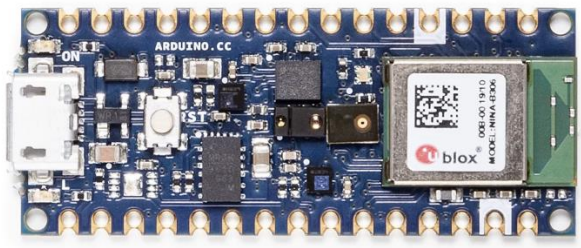

Figure 2.2: Nano 33 BLE Sense

2.3. Machine Learning

We have completed mantra speech recognition and spell casting gesture recognition based on Tensorflow and tensorflow lite. After model training, the recognition accuracy rate remains above 90%. Initially, our speech model was only implemented by a convolutional layer, a fully connected layer, and a softmax layer. And the accuracy of the model for the test set is as high as 98%, but the final accuracy of the voice data collected by Arduino Nano 33 is low, so we optimized the voice model.

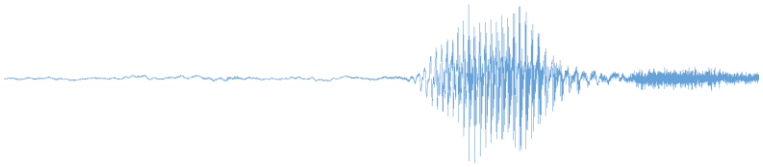

Figure 2.3: Wave of an audio recording

2.4. Game Development

Finally, we completed the game development part based on Unity2D, mainly using Unity BLE to realize the Bluetooth communication with Nano 33 BLE, and complete the game script development based on C#.

Figure 2.4: Game interface

3. Model Design

In this chapter, we will process the acquired signal data in different ways and try to train our recognition module with different models. In addition, the real effect of the trained model is compared in actual use, and the recognition model with the best effect is selected as our recognition module.

Our model is mainly divided into speech recognition model and gesture recognition model. Each model uses signals obtained from Arduino Nano 33 BLE to make judgements. But because it is deployed on the board, we also need to transform the model with Tensorflow Lite so that it can be used on the board.

3.1. Speech Recognition Model

When we train speech recognition models, we use two different models: CNN and MLP, based on how the data is processed. The following details the implementation of the two models and the performance comparison.

3.1.1. CNN model

- Data preprocessing

The voice data we use is made by our own voice collection website (https://magicfight.de.r.appspot.com). With the help of the website, we collected 3 different spell data("Avada_Kedavra", "expelliarmus","protego"), each containing 200 samples. Each sample is a 2 seconds long wav format file captured by the computer's microphone. Besides the spell we want to use, we also recorded lots of noise as negative samples which are labeled as 'Unknown' in order to reduce the impact of noise on our model recognition.

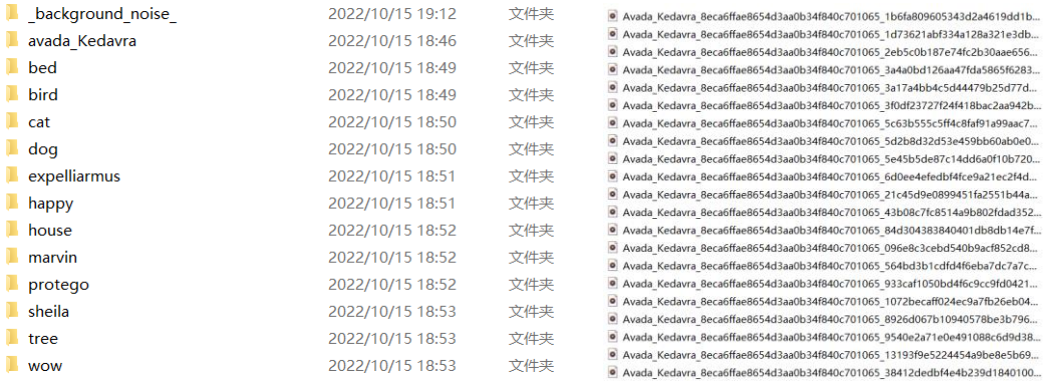

Figure 3.1: Voice collection

In this experiment, we wanted to use CNN to identify our voice data.Because CNN has a strong feature extraction and classification ability for images, we made a key transformation of the voice data we obtained. We converted our original timing signals into picture information, which are called spectrograms.

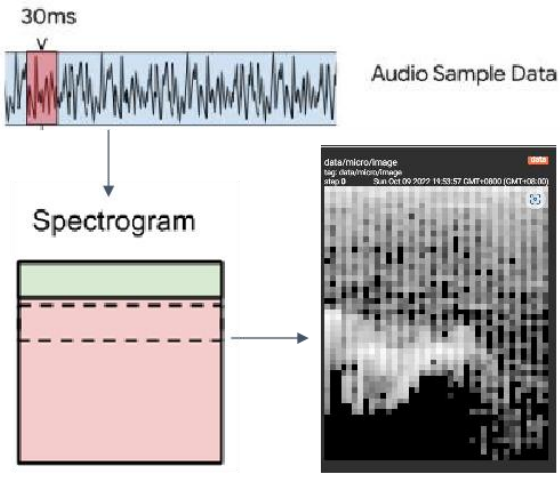

Figure 3.2: Voice data transformation

As the graph shown above, we use a 30ms window that's moved forward 20 ms each time between iterations, to cover the full two seconds of audio input data. So there will be 99 windows in total, and each window contains 30ms of audio data. In order to build spectrograms, we also need to generate a Fourier transform for a given time slice. We use FFT algorithm to extract frequencies from 30ms of audio data, so the 30ms time slice is converted to a vector which contains frequency information of the audio data. In this way, we got a 2D array of values that's 40 elements wide and 99 rows high, which repesents a spectrogram. Next, we will use spectrograms to train our CNN model.

- Model structure

After trying several different convolutional neural network structures, we choose the model below. Because we only need to recognize three types of speech data, and the speech data has a good differentiation after being converted into spectrograms, so we don't need a very complicated model. In addition, although the complex network is more conducive to feature extraction, it will also cause the overfitting of the model. More importantly, because the board space is limited, we should make the model as small as possible in order to achieve other functions.

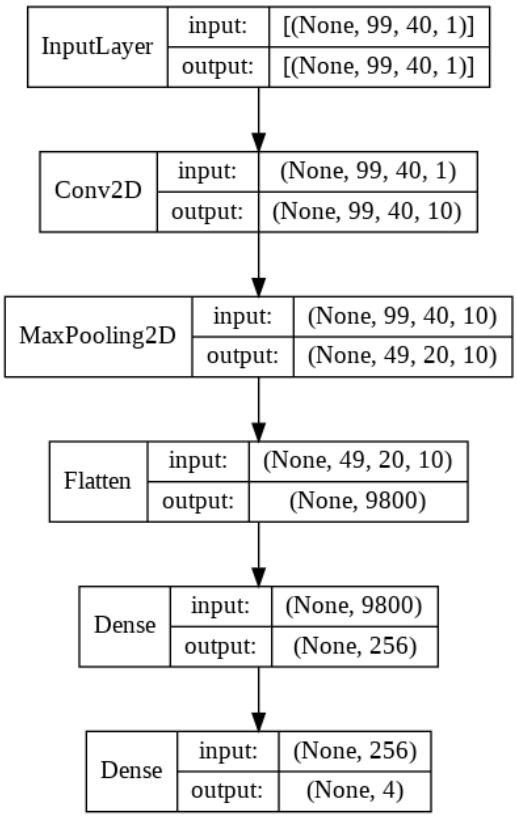

Figure 3.3: Model structure

- Model performance

As is shown below, we use TensorBoard to record the accuracy and loss during training the model. The left graph shows accuracy and the right graph shows loss. The blue line represents training data and the red line represents validation data. Finally, we got an

accuracy of 98% on the train set and 96% on the test set, which is pretty good.

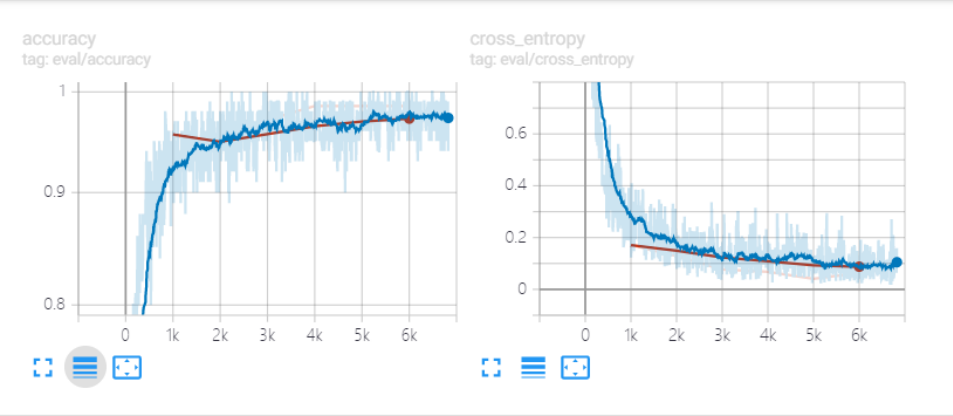

Figure 3.4: CNN model performance

3.1.2. MLP Model

- Data preprocessing

Besides CNN model, we try to use MLP for speech recognition. Unlike before, this time we directly use the board to sample the sound signal, instead of using the website to record the audio and convert to the spectrum.

Since Arduino Nano 33 BLE itself has a sound sensor, we can set different sampling intervals and sampling duration to obtain the required training data.When training the MLP model, two types of sampling data were used for comparison. For the first type of data, we used a sampling interval of 20ms and sampled 64 times as the sound data of a spell. But for the second type of data, we used a sampling interval of 10ms and sampled 128 times as the sound data of a spell. The two sampling methods retain different data, but the length of time for each spell is the same.

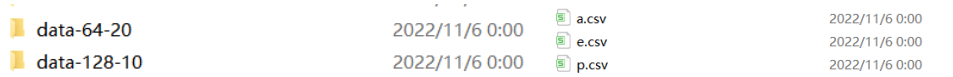

Figure 3.5: Data collection

- Model structure

After trying several different network structures, we choose the model below. In addition, we used two types of sampling data for training different models, and the final result proved that the data with 64 sampling intervals had higher accuracy than that with 128 sampling intervals. Finally, we decided to use a sampling interval of 20ms and sampled 64 times as the sound data of a spell. So the input shape of the model should be 1x64. At first, we only used two Dense layers to train the model, but later we added two layers of Dropout structure to inhibit overfitting.

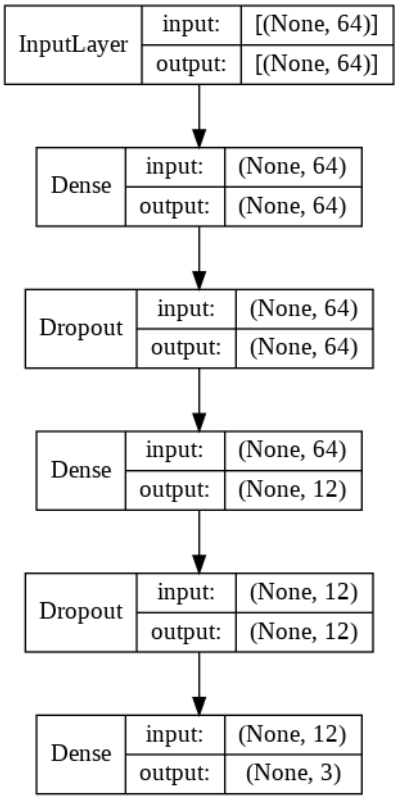

Figure 3.6: Model structure

- Model performance

As is shown below, our MLP model has a strong recognition effect for the three types of spell, but in terms of accuracy, there is still some gap compared with the previous CNN. However, these are only the results of model training, the specific use effect needs to be tested on the board. If the time cost and space cost are taken into account, the MLP model is still better than the CNN model.

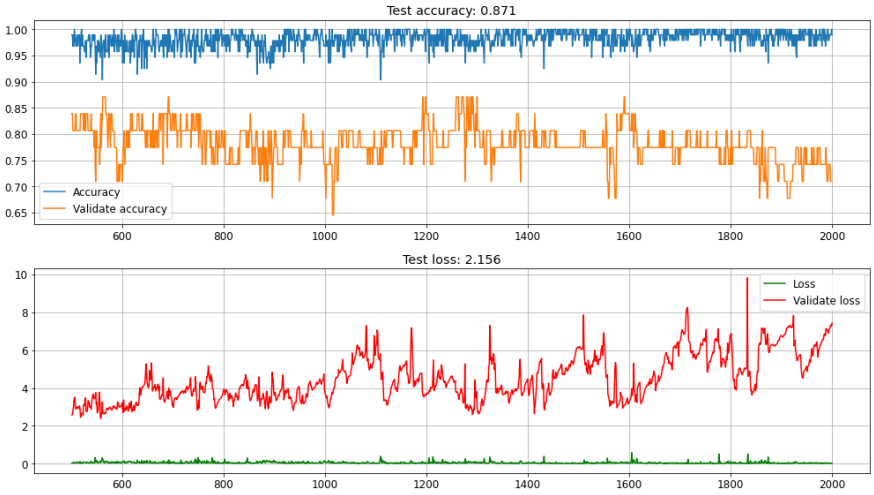

Figure 3.7: MLP model performance

3.2. Gesture Recognition Model

In the gesture recognition module, we also tried two different recognition models with CNN and MLP. Similar to the processing of sound data, we use different models for different data to test the recognition effect and finally determine the model to use.

3.2.1. CNN Model

- Data preprocessing

Because we want to use CNN model for image feature extraction and recognition, we need to convert the data collected by the sensor into image data in advance. First, we use the orientation sensor of the board to get the coordinate information of the board, which contains the board's three-dimensional coordinate information in space. Then we recorded the coordinates of each action. In this way, we can project the trajectory of each action in space onto a two-dimensional plane as an image. As is shown below, we can use images to represent different gesture data. Then, we can use image data to train our CNN model.

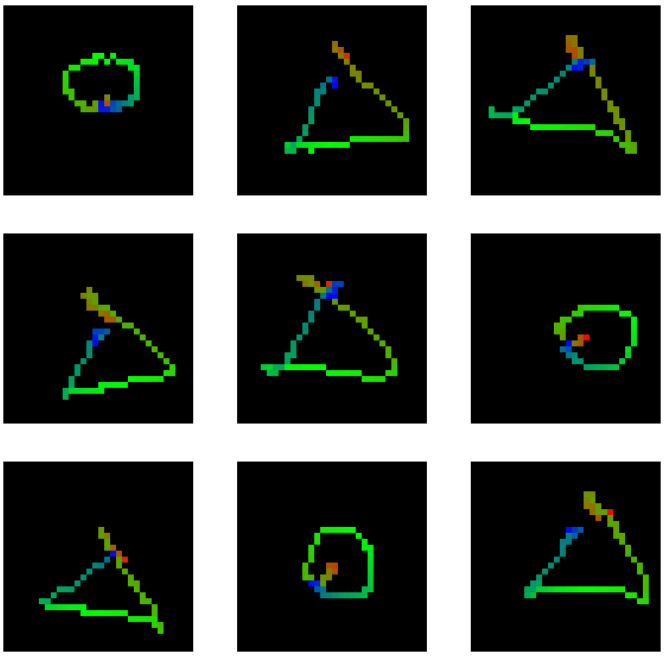

Figure 3.8: Image data

- Model structure

In the training process of the model, we constantly adjusted the model structure and finally selected the model as shown in the figure below, which has a very high recognition accuracy for our action data.

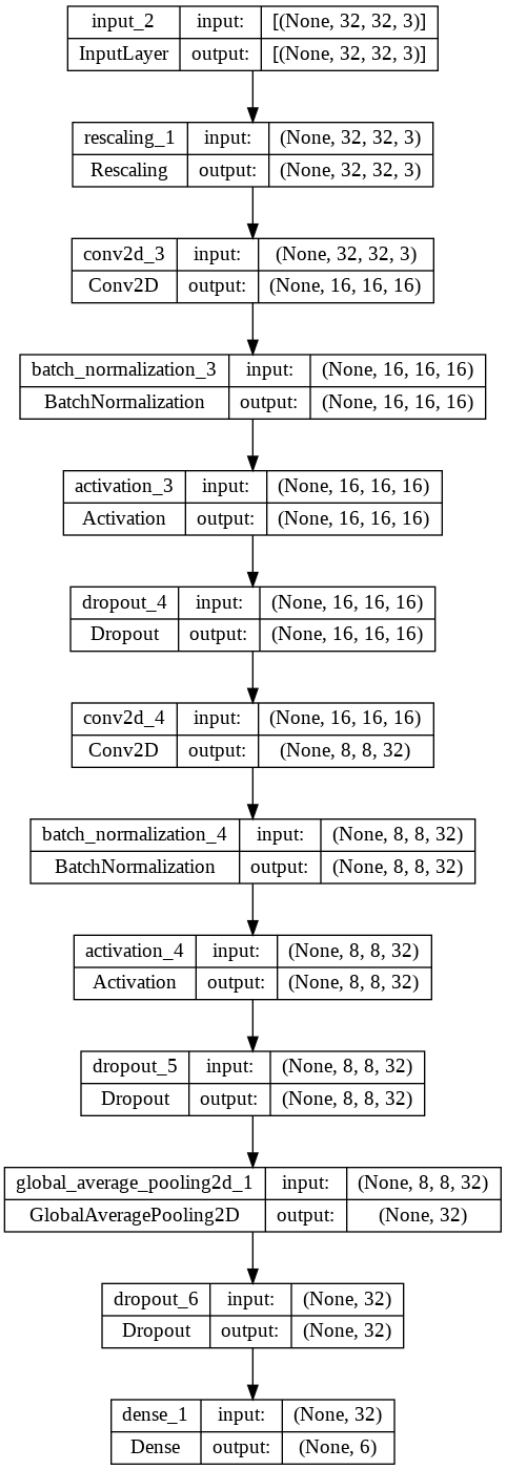

Figure 3.9: Model structure

- Model performance

According to accuracy and loss shown in the following two graphs, we can see that our model has a very high recognition accuracy. And the model can get very good recognition accuracy at the early stage of training. However, the fluctuation on the test set may be due to the small sample selection of the test set, resulting in relatively obvious fluctuation. In general, it is a good solution to transform the motion recognition

problem into an image classification problem.

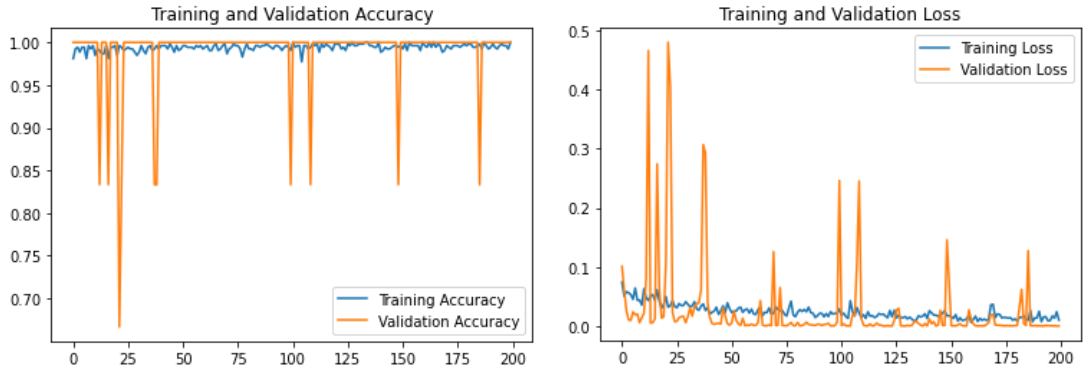

Figure 3.10: CNN model performance

3.2.2. MLP Model

- Data preprocessing

In this part, instead of converting signals into images, we directly use the retained signal information as training data. Since we need to train signal data directly, we want to use as many different signals as possible to train the network, so as to improve the accuracy of network recognition and reduce the interference of non-target actions.

```
// read the acceleration and gyroscope data
IMU.readAcceleration(ax, ay, az);
IMU.readGyroscope(gx, gy, gz);
```

Figure 3.11: Data collection

For the above consideration, we finally adopted motion sensors and coordinate sensors to obtain signal information. Each sensor returns a three-dimensional message. In this way, we'll have six different signals to represent a gesture, as is shown below.

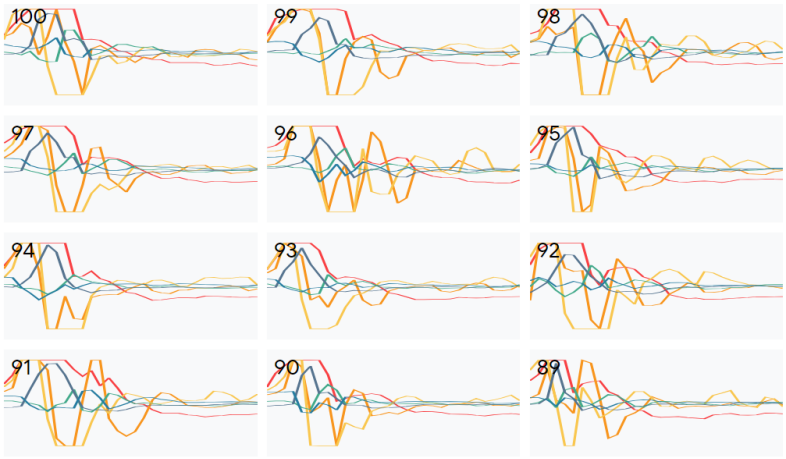

Figure 3.12: Signal respresentation

- Model structure

We spliced six sets of signals into a one-dimensional signal as input to the model. Through three simple Dense layers of training, we finally get the classification results. Because the input data was very simple and we wanted to improve the recognition speed as much as possible, we finally chose the following network model.

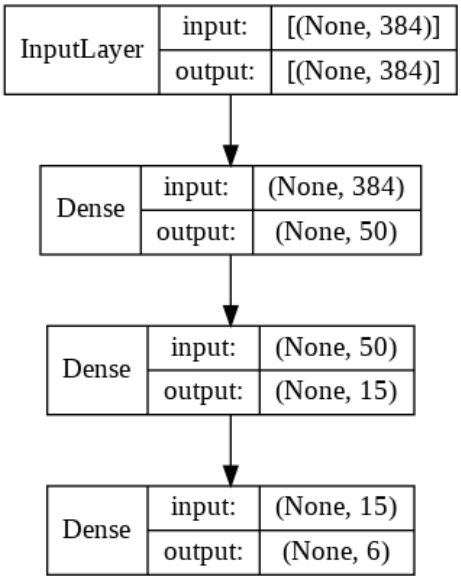

Figure 3.13: Model structure

- Model performance

According to the training process recorded in following figures, it can be seen that our MLP model has excellent results in the recognition of several different kinds of gesture signals, which is even better than the previous CNN model.

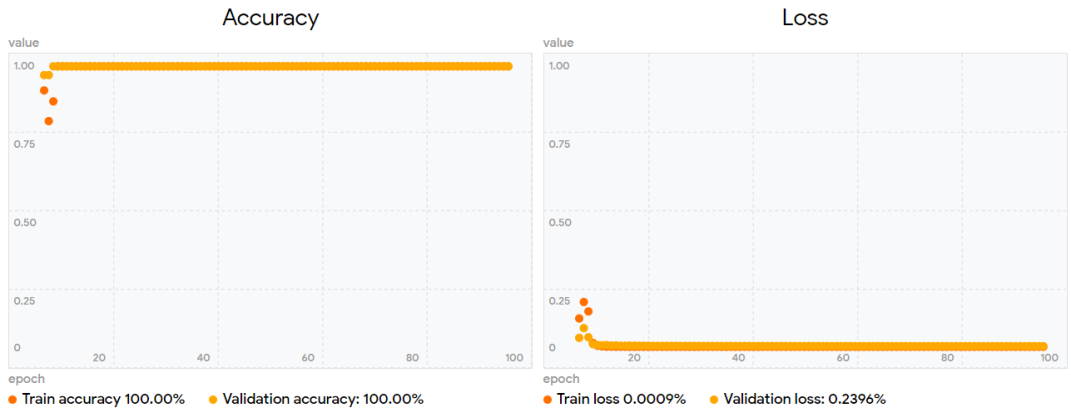

Figure 3.13: Model performance

3.3. TensorFlow Lite

The above models are all models trained by TensorFlow. Although they have a good

recognition effect, they cannot be directly deployed on our board. Since we'll be running our models on tiny microcontrollers, we need a different interpreter that's designed for our use case. Fortunately, TensorFlow provides an interpreter and accompanying tools to run models on small, low powered devices. This set of tools is called TensorFlow Lite.

Before TensorFlow Lite can run a model, it first must be converted into the TensorFlow Lite format and then saved to disk as a file. We do this using a tool named the TensorFlow Lite Converter. The converter can also apply special optimizations aimed at reducing the size of the model and helping it run faster, often without sacrificing performance.

3.4. Actual Performance

After the above models are deployed on the board through TensorFlow Lite, we will examine the effects of the models in real use one by one and make a choice.

- Speech Recognition Model

Finally, we chose MLP model as our speech recognition model. Although there are some gaps in the training results of the CNN and MLP models mentioned above, we find that the recognition accuracy of the two models is very high in actual use, and there is no obvious difference. In addition, MLP model has faster recognition speed and occupies smaller board memory than CNN model. Therefore, after comprehensive consideration, we finally choose MLP model as our speech recognition model.

- Gesture Recognition Model

As for the model of gesture recognition, to our surprise, the performance of CNN model in practice is extremely poor. We speculate that because the model converts signals into two-dimensional images for recognition, it requires very high precision of users' actions. Any change in the motion amplitude will affect the drawn image and thus affect the judgment of the model. On the contrary, gesture recognition using MLP model is not only faster, but also highly accurate. So in our final recognition system, we abandoned the CNN model and adopted the MLP model to recognize the gestures we used to release the spell.

4. Board Embedded Development

Our Magic Fight project is based on the Nano 33 BLE Sense board. The Nano BLE Sense is a fantastic, versatile board that can be used for environmental & gesture sensing, machine learning and connectivity projects. The structure of this board is shown below.

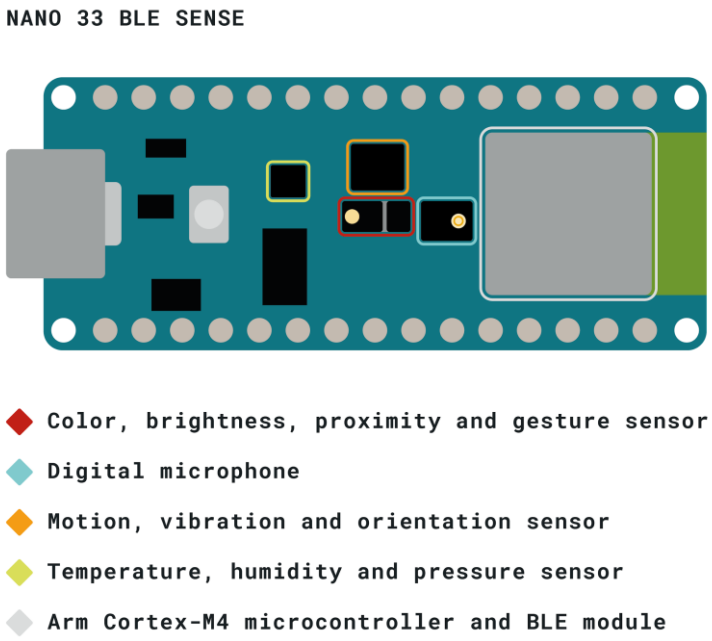

Figure 4.1: Structure of Nano 33 BLE Sense

In our project, two embedded sensors are mainly used:

- Microphone: to capture and analyze sound in real time. In our project, the microphone is used to record and then make the audio dataset at first. After training models, this sensor would be used to capture voice commands and recognize which command was detected.
- Gesture sensor: to sense and analyze movements in real time. In our project, the gesture sensor is used to sense the direction and track of the board moves, and then detect gestures based on the model trained.

4.1. Speech Recognition

In speech recognition part, we mainly use the microphone sensor on the board for two things:

- Record audio samples for dataset making
- Capture audio for voice command recognition

Originally, we built a recording website, which could covert audio formats automatically. However, after data processing and model training, we found that the model performance is not as good as imaged, which means the microphone sensor could not analyze voice command accurately. We guess that this result was caused by different environment of recording audios between training data and real sound received by the microphone on the board. Therefore, we tried to change to use the microphone on the board to record audio samples for dataset making and training, which could ensure that the environment of recording is always similar.

We designed three incantations as project samples: Expelliarmus, Avada Kedavra, and Protego. Each incantation has its own model, which is used to recognize which voice command was detected. In coding, we defined a function, which is used to process audio received by microphone in real game scene. In this function, the audio received is sent to the trained TensorFlow model, and then obtain the prediction result, which means predict which label the audio belongs to. Finally, the predicted incantation would be shown on the screen and transmitted to Unity.

4.2. Action Recognition

In addition to voice, we hope to control game characters with actions, which means we can swing the stick tied to the board to make gestures we want, and then the screen could show which gesture we made just now.

In this part, we set X, Y, Z axes to record the position every millisecond, thus getting the track of the board, that is, the gesture we made.

We mainly designed two actions, supposing the board is already tied to the stick and the player is holding the stick in his hand:

- Extend your arm forward so that the board is perpendicular to your chest. This action is needed after shouting the incantation. Only when the system recognize the speech command and action command at the same time could show the magic effect on the screen.
- Extend your arm laterally, making the board goes from perpendicular to the ground to parallel to the ground. This action is used to block the attacking spell from the opponent.

In section 3, we have trained a successful action recognition model. In real game scene, after players making actions, that is, after the board sensing the gesture player made, the track data would be sent to the trained model above. Then, the model would predict which label this gesture belongs to. Finally, the predicted action would be transmitted to Unity and shown on the performance of game characters.

4.3. Bluetooth Connection

The purpose of our project is to control game characters with voice and gestures, and then display the performance on the screen. Our game scene is designed with Unity, so, the board needs to connect to Unity. Fortunately, the Nano 33 BLE Sense provides a Bluetooth module, which could make the board connects to other devices. Therefore, the board could realize information interaction with Unity.

There are three ways data can be exchanged between two connected devices: reading, writing, or notifying. Reading occurs when a peripheral device asks the central device for specific information, think about a smartphone asking a smartwatch for the physical activity information, this is an example of reading. Writing occurs when a peripheral

device writes specific information in the central device, think about a smartphone changing the password of a smartwatch, this is an example of writing. Notifying occurs when a central device offers information to the peripheral device using a notification, think about a smartwatch notifying a smartphone its battery is low and needs to be recharged.

In this project, the main way of data exchange is reading. After the Unity project is started and the board is powered on, the two devices firstly connect to each other via their respective Bluetooth. Then, after recognizing which voice command or which gesture was captured, the identified content could be transmitted to Unity via Bluetooth. So that the game characters could do corresponding actions according to the commands received from the board.

5. Unity Game Development

5.1. Game Planning

- **Set the game worldview：** First of all, we determined the world view and background of Magic fight. The world view of Magic fight is based on the original Harry Potter books. After in-depth interpretation of the original work, the spells, spell casting actions, and magic in the magical world of Harry Potter are restored. We believe that even if players are ordinary people who have not seen Harry Potter, after seeing our game works, they will be infected by the culture behind them. I can't help but want to learn about it, and Magic fight has become an important way for these players to understand Harry Potter.

The rich activity content also gives the game a very high playability. Since we decided to develop Magic fight, we firmly believe that each version of our plan can always bring different surprises to players, such as from the initial spell casting, to the subsequent spell + gesture casting, and the silent spell we are developing. spellcasting. This also makes the player's exploration more meaningful, and unexpected gains will give players a great sense of satisfaction. We will keep changing, so that players always feel interested.

- **Set the game plot and character：** Our game plot is relatively simple at present, that is, two magicians who broke into the magical world of Harry Potter will start a duel of a spell feast here. Characters are prepared to continuously expand into new characters based on the content of the original Harry Potter books.

- **Game props and numerical settings**: At present, the game props only include the staff in the hands of the magician, and the staff can be controlled according to the player's spell + gesture. The game character can perform three types of actions, namely normal attack, block, and special attack. The overall value is set so that the player has 1 health, and both normal and special attacks will deal 1 damage. Normal attacks can be blocked and when the normal attack accumulates five times, a special attack will be accumulated. It should be noted that using a block will reduce the rage value

accumulated by a normal attack. Special attacks cannot be blocked and can only be evaded by dodging. At the same time, we are also building a more comprehensive game numerical system to optimize the player's combat experience.

5.2. Art Design

- Draw the original drawings of game characters. We designed the game characters based on the original drawings of characters in the Unity Store.
- Make the original painting into a character model. Based on the official human body model provided by Unity3D, we adjusted the model's hierarchical structure and bones to make it more in line with the magician's posture and movement.
- Add various actions based on the character model. Based on some standard human animations officially provided by Unity3D, we drag and drop them into the Animator window for use.
- Make other models in the game, such as: buildings, landscapes, props, etc. The overall game background is redesigned and laid out based on the open source models in the Unity Store.
- Use the game scene editor and model to create scenes in the game, create several cube objects, and form floors, walls and obstacles through scaling and transformation.
- Use the script editor of the game to edit various character actions and scene actions. Character actions and scene actions are designed based on the models officially provided by Unity2D.
- Use the game special effect editor to edit the special effects of the game and the magic effects of the battle.

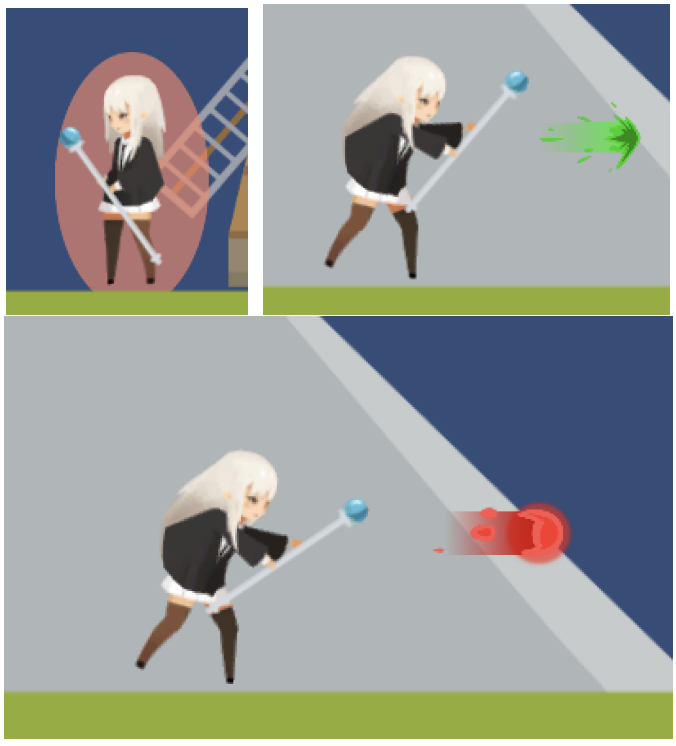

Figure 5.1: Character action display

5.3. Program Development

We complete game script development through C#.

- **Create player characters**

Unity uses a full-component system to build GameObjects. That is to say, all GameObjects are made up of components, which give behaviors and properties to the GameObject. Here are a few of Unity's built-in components:

Transforme: Every GameObject has this component. It holds the GameObject's position, angle and scale.

Box Collider: A cubic collider used to detect collisions.

Mesh Filter: Mesh data for displaying 2D models.

- **Script the player movement**

Once the player character is created, it's time to create the script that receives keyboard input and moves the player.

The PlayerMovement class is the default class that Unity generates in new scripts. It extends the MonoBehaviour base class so that scripts can run in the game, as well as some special methods to respond to certain events. Unity calls several methods in a specific order when running a script. The most common methods include:

Start(): This method is called when the script is updated for the first time.

Update(): When the game is running and the script is available, this method is called every frame refresh.

OnDestroy(): Called before the GameObject this script is attached to is destroyed.

OnCollisionEnter(): Called when a collider or rigid body attached to this script comes into contact with another collider or rigid body.

- **Script magic attacks**

When the player casts a spell, a Projectile object is fired. To achieve this, we need to create a prefab. Unlike objects once created in the scene, prefabs are created on demand based on game logic.Next, we need to instantiate a spell prefab and then get its Projectile component so we can initialize it.

The coordinates of this point should be (x,y,z). X and Z are the coordinates of the current position of the game character projected onto the floor. This transition is very important because the spell has to fly parallel to the floor - otherwise if you could cast it downwards, only a rookie would hit the spell to the ground.Calculates the direction from the player character to pointAboveFloor and create a ray, describe its trajectory with a starting point and direction, and tell the Unity physics engine to ignore collisions between the player and the spell collider. Otherwise, when the two collide, the

OnCollisionEnter() method in the Projectile script will be called.Finally, we set the flight path of the spell.

6. Conclusion

6.1. Summary of Achievements

Our team has spent a difficult but rewarding time together in the magic fight project, and each member has learned and grown a lot during the project.

In this project, the most critical link is the construction and training of the model. We reproduced the neural network that works well in the blog for speech recognition, but the final effect is not satisfactory, so we carried out on this basis. Refactoring and optimization of the model to make it more in line with the application scenarios of our project. At the same time, the data set is also a very important factor. We have realized the collection of speech data and gesture data through our own collection methods, so that the training data perfectly conforms to our original idea.

For us, building a game system is also a very serious challenge, because from the data collection website, to the hardware embedded development, as well as the subsequent model training and game interface development, we have developed it ourselves, which is also due to Yu team members continue to expand their technology stacks through self-study. For the specific technologies used, please refer to Chapter 2. At the same time, through this project, we have also exercised the team's ability to cooperate well, which can help us in the future. Faster integration into a real working development environment.

In general, we also encountered many challenges in the process of project practice. Through the joint efforts of team members, we successfully overcame them and completed the MVP version of magic fight.

6.2. Future Improvements

Despite a successful implementation for a minimum viable product, there are areas for future improvements.

The improvement of the game can be carried out in three aspects. One is to add more game operations to enrich the player's game experience. At present, our game only realizes the voice recognition and gesture recognition of basic spells, and we are developing silent spells. On this basis, we will also combine other elements of combat games, such as enchantments, runes, talents, etc., to enhance attribute values in more dimensions, and build a more complex game value system while balancing the abilities of game characters. The second aspect is that we will also optimize the current game interface to provide players with more abundant battle scenes, and plan to develop an open world game mode in the later stage.Finally, in terms of somatosensory, we also plan to incorporate more interactive methods to enhance the differentiation from

traditional somatosensory games.

We have already started the next step of design: adding silent spells to create more game modes!

We have completed the model training of six silent spells, and the test set also showed good results. Next, we will design a new battle mechanism, that is, silent battles, which do not require spells and only need to control the magic wand to cast spells through gestures.

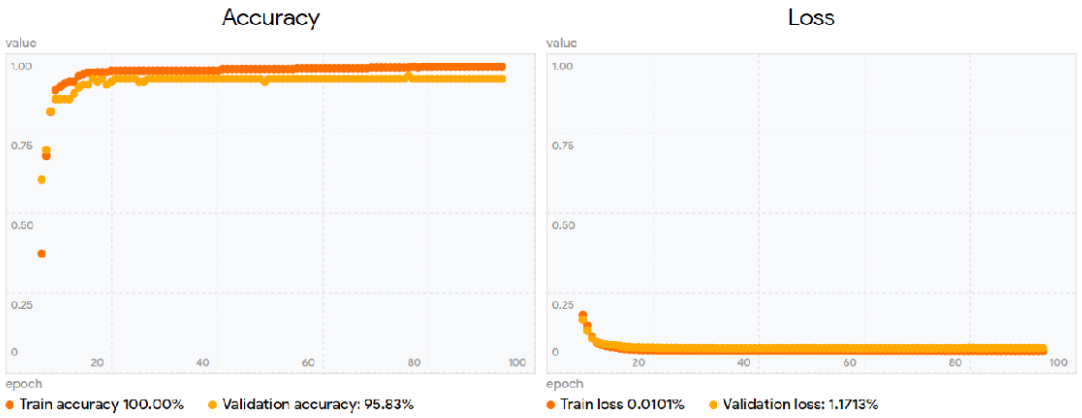

Figure 6.1: Model performance