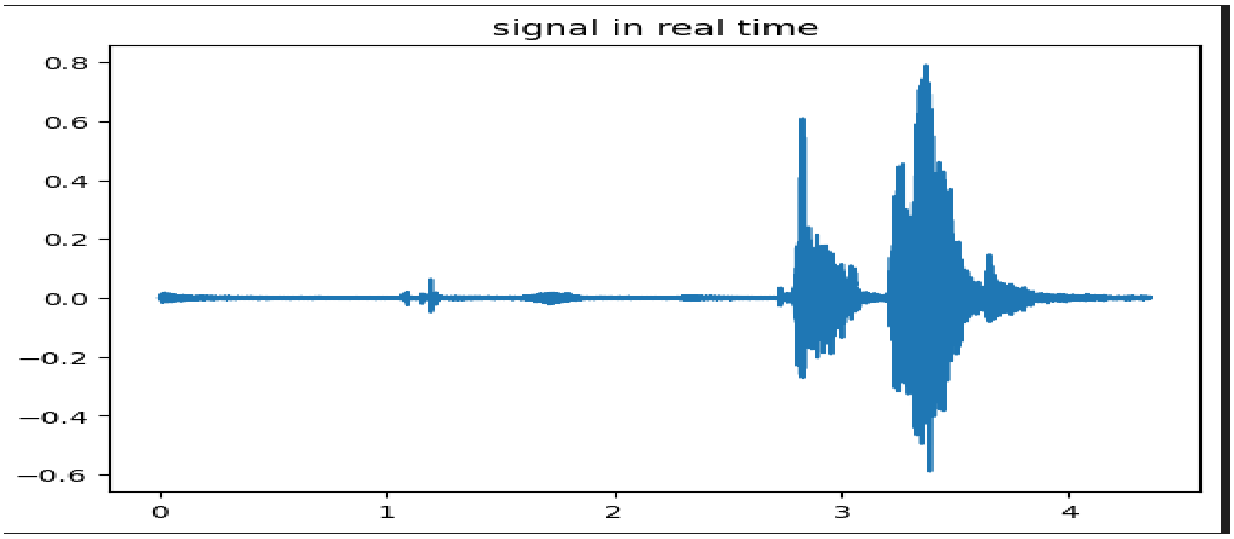
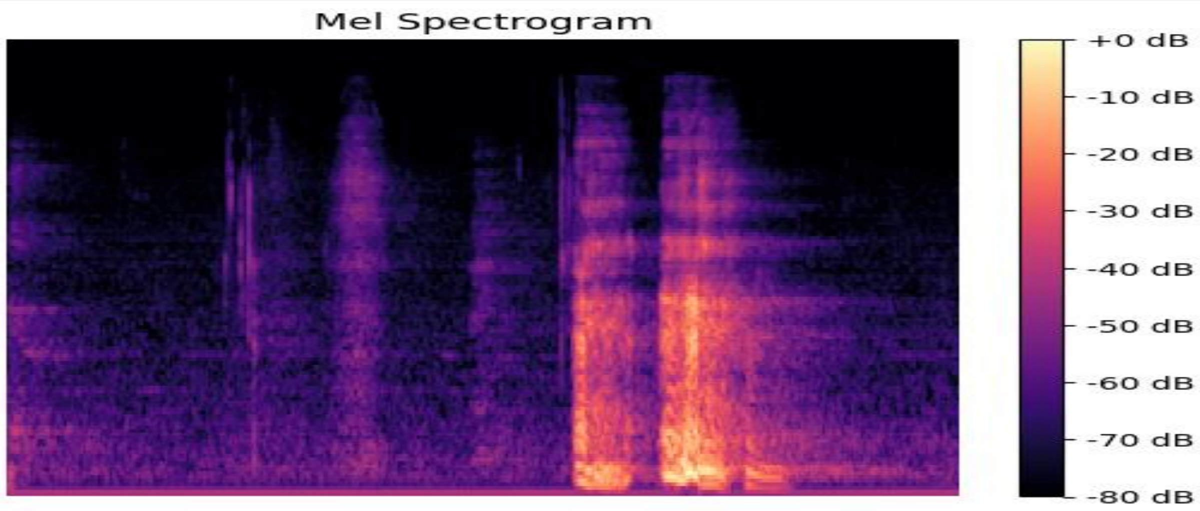
BLG 561E DEEP LEARNİNG (CRN:23953) HOMEWORK-1

Nazif ÇELİK 090200712

Question-1 The Turkish Academy Voice Challenge

For this assignment, I used machine learning to distinguish the voices of prestigious Turkish academics and researchers. The audio files for this challenge include Aziz Sancar, Biykem Bozkurt, Cahit Arf, Canan Dağdeviren, and Koray Kavukçuoğlu. Each audio file is approximately 5 seconds long. I created my own neural network to classify audio files. I used mel-spectrograms to prepare your introduction.

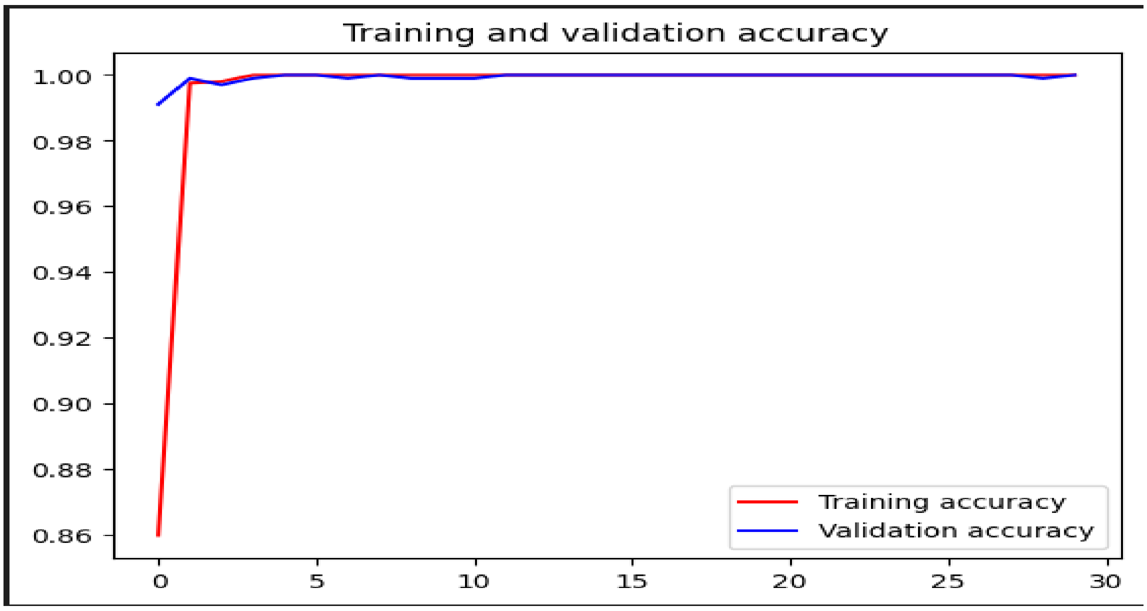




This project classifies sound signals from different environmental classes in the Turkish Academy Voice Challenge – 2025 dataset. the above photo summarizes the model steps:

1. The model read all the signals of different classes and assign a label number to each class.
2. The Mel Spectrogram are extracted from the time domain.
3. Full Convolutional Neural Network(CNN) is defined and used to classify 5 different classes of Turkish Academy Voice Challenge – 2025 dataset.

The performance results of the model are as follows:



Question-2 The Mine Game

For a computer game where the main character tries to reach the end of a given map, I made an AI-powered player agent that decides the next action by taking screenshots of the board. The main goal is to reach the end point by avoiding the mines in some of these grids. If our character is close to a mined grid, a face with the expression "shocked" appears in the top right.

I used pyautogui to simulate these actions.

Here, I followed the following strategies:

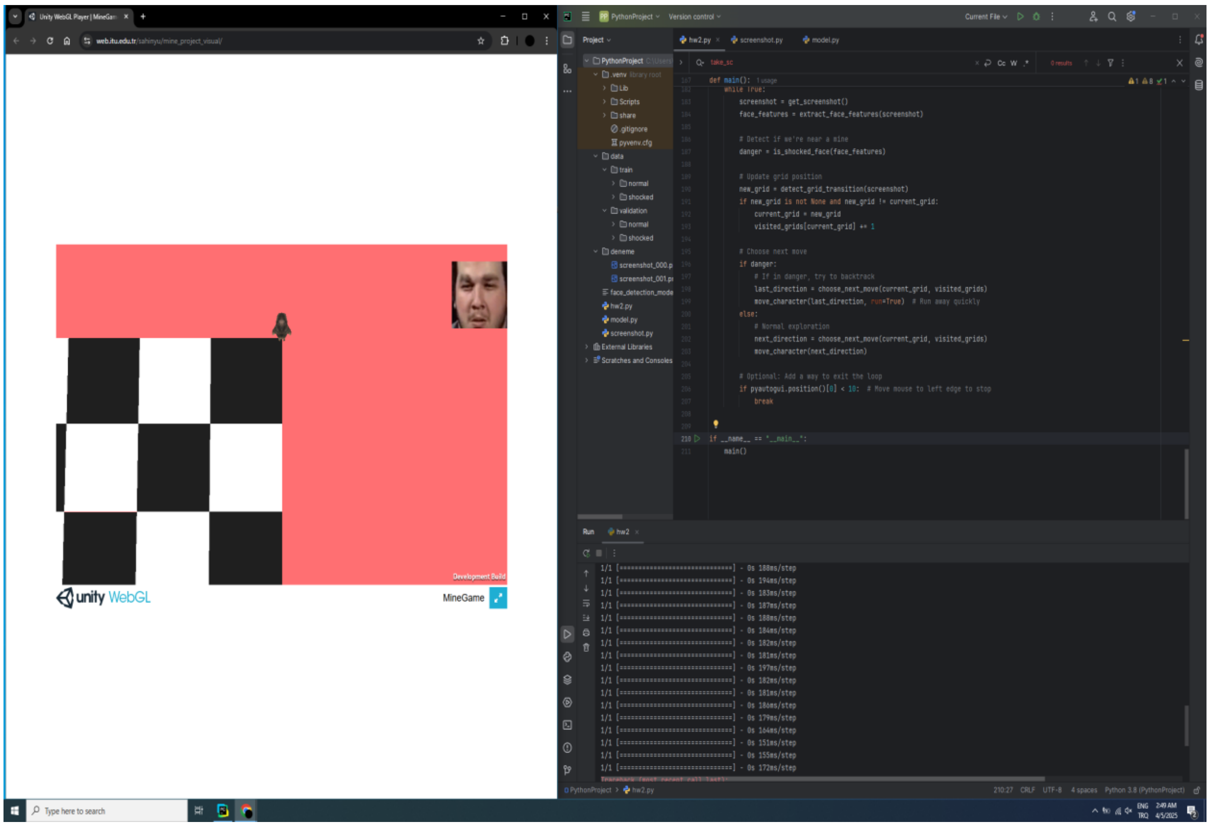
• I used features from a pre-trained convolutional neural network to detect the difference between the "shocked" face and the "normal" face. So that it tries to avoid the mined grids.

• I kept a list of the visited grids so that the character does not tend to revisit some grids over and over again. Therefore, I used corner and edge detection to detect if the character has entered another grid.

• I made sure that the character reaches the final grids safely.

I used the textbfpyautogui library to simulate mouse and keyboard interactions in Python. I also took screenshots of the game.

Since I didn't know exactly where the finish line was, I chose the top right point and in this way I successfully reached the finish line.



```
def main():  1 usage
    while True:
        screenshot = get_screenshot()
        face_features = extract_face_features(screenshot)

        # Detect if we're near a mine
        danger = is_shocked_face(face_features)

        # Update grid position
        new_grid = detect_grid_transition(screenshot)
        if new_grid is not None and new_grid != current_grid:
            current_grid = new_grid
            visited_grids[current_grid] += 1

        # Choose next move
        if danger:
            # If in danger, try to backtrack
            last_direction = choose_next_move(current_grid, visited_grids)
            move_character(last_direction, run=True)  # Run away quickly
        else:
            # Normal exploration
            next_direction = choose_next_move(current_grid, visited_grids)
            move_character(next_direction)

        # Optional: Add a way to exit the loop
        if pyautogui.position()[0] < 10:  # Move mouse to left edge to stop
            break


if __name__ == "__main__":
    main()
```

```
1/1 [==============================] - 0s 188ms/step
1/1 [==============================] - 0s 194ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 187ms/step
1/1 [==============================] - 0s 188ms/step
1/1 [==============================] - 0s 184ms/step
1/1 [==============================] - 0s 182ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 197ms/step
1/1 [==============================] - 0s 182ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 164ms/step
1/1 [==============================] - 0s 151ms/step
1/1 [==============================] - 0s 155ms/step
1/1 [==============================] - 0s 172ms/step
Traceback (most recent call last):
```

Question-3 Evolution

I made several significant improvements to the neural network architectures to enhance the learning capacity and robustness of the soft robot controllers.

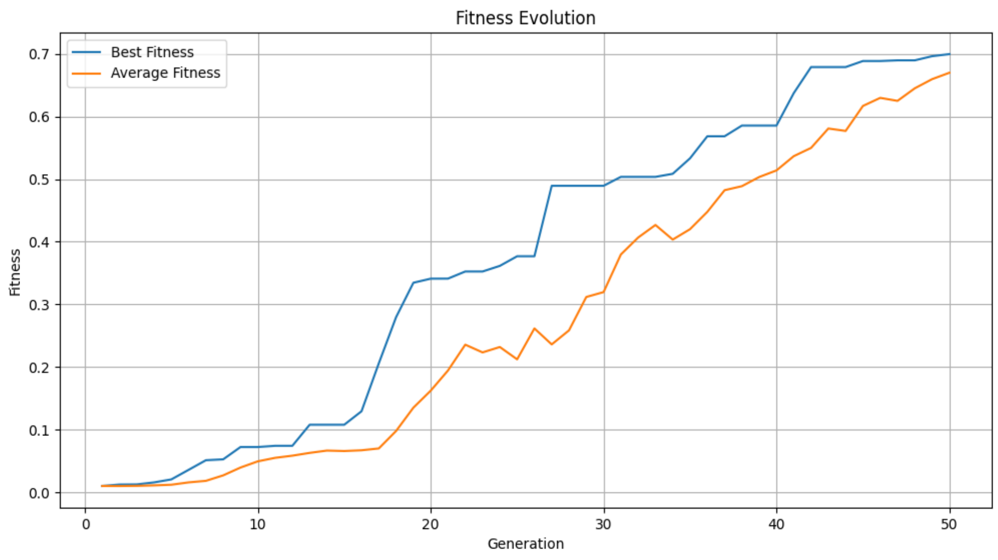
Changes I made to the network structure:

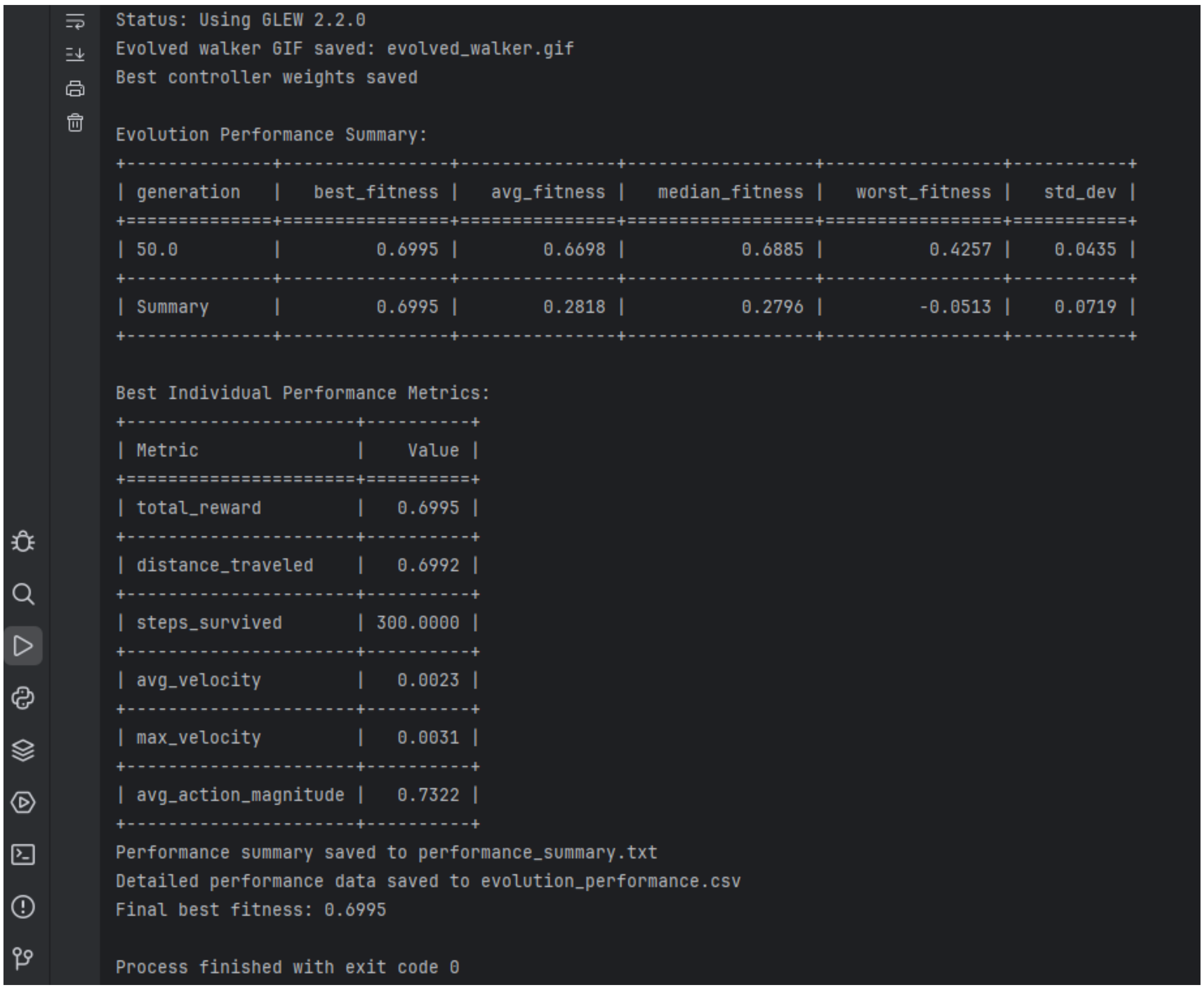
- *Deeper architecture*: Increased from a shallow 32-unit network to a deeper network with layer sizes 64→128→64→output
- *Better activation functions*: Replaced standard ReLU with LeakyReLU (0.1 slope) to prevent "dying ReLU" problem
- *Regularization*: Added dropout layers (rates of 0.2, 0.2, and 0.1) between layers to reduce overfitting and improve generalization
- *Increased capacity*: The network now has significantly more parameters, allowing it to learn more complex control strategies

These improvements should lead to:

1. *Better learning dynamics*: LeakyReLU and GELU avoid the "dead neuron" problem of standard ReLU
2. *Improved generalization*: Dropout and batch normalization help the network generalize to new morphologies
3. *Feature diversity*: The dual-branch architecture allows the network to learn different types of features
4. *Higher expressivity*: The deeper and wider architecture can represent more complex control policies

The evolution stage of the robot is as in the table below





```
Status: Using GLEW 2.2.0
Evolved walker GIF saved: evolved_walker.gif
Best controller weights saved

Evolution Performance Summary:
+--------------+----------------+--------------+------------------+-----------------+-----------+
| generation   |   best_fitness |  avg_fitness |   median_fitness |   worst_fitness |   std_dev |
+==============+================+==============+==================+=================+===========+
| 50.0         |         0.6995 |       0.6698 |           0.6885 |          0.4257 |    0.0435 |
+--------------+----------------+--------------+------------------+-----------------+-----------+
| Summary      |         0.6995 |       0.2818 |           0.2796 |         -0.0513 |    0.0719 |
+--------------+----------------+--------------+------------------+-----------------+-----------+

Best Individual Performance Metrics:
+---------------------+----------+
| Metric              |    Value |
+=====================+==========+
| total_reward        |   0.6995 |
+---------------------+----------+
| distance_traveled   |   0.6992 |
+---------------------+----------+
| steps_survived      | 300.0000 |
+---------------------+----------+
| avg_velocity        |   0.0023 |
+---------------------+----------+
| max_velocity        |   0.0031 |
+---------------------+----------+
| avg_action_magnitude |   0.7322 |
+---------------------+----------+
Performance summary saved to performance_summary.txt
Detailed performance data saved to evolution_performance.csv
Final best fitness: 0.6995

Process finished with exit code 0
```