Pokebot: An Automated Gameboy Advanced Emulator Bot for Leveling Up Your Pokémon

Akib Sadmanee
ICS 637: Deep Learning with Neural Networks

Introduction:

For nearly two decades, the world of Pokémon has captivated players' imaginations, with fans of all ages immersing themselves in the series' exciting gameplay and nuanced plot. Levelling up your team of pocket monsters is one of the most crucial components of Pokémon, with each level attained making your Pokémon stronger, quicker, and more capable of facing the challenges ahead. For example, as a casual player, if you pick Squirtle as a starter, you will have to depend on your luck a bit if you want to win against Misty. Bulbasaur makes the whole game easy mode. You get the upper hand in all the gym battles up to the fire gym. By then you are able to make a strong team to go against the gym leader. If you choose Charmander as your starter, because it has the coolest final evolved form, you will have a hard time at the starter gyms. Both the starter gyms are strong against fire-type Pokémons. 1 solution is to over-level your Pokémon so the type of weakness doesn't matter anymore. This is a tedious process that requires hours of mindless grinding. I present Pokebot to help with this process. Pokebot is a strong application developed to automate the levelling-up process in Pokémon games for Gameboy Advance (GBA) emulators. The bot analyzes in-game images and makes judgments depending on the current state of the game using convolutional neural networks (CNN) [1]. Players may substantially minimize the time and effort necessary to level up their Pokémon by employing Pokebot, giving them more time to explore the rich and vivid world of Pokémon.

The dataset

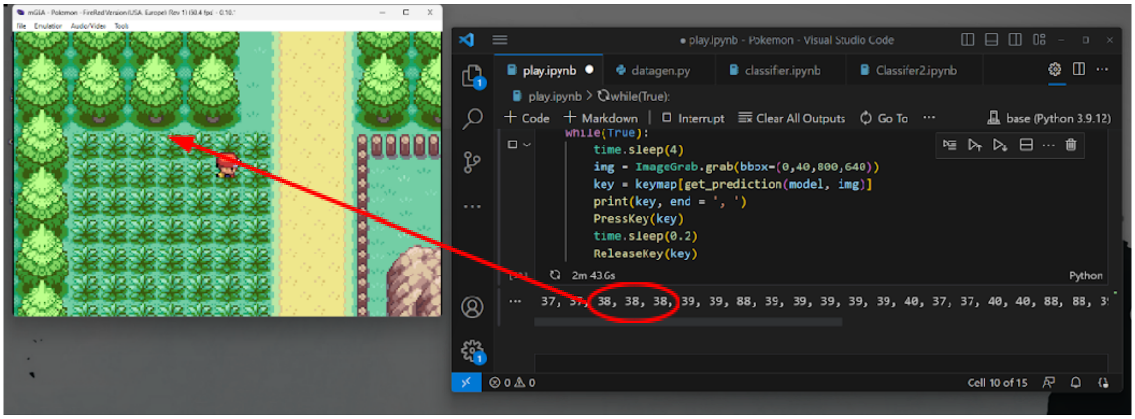
I searched online for publicly available standard datasets for the purpose of training a Pokémon bot. The closest data I found was "Twitch Plays Pokémon" data which is available as a video stream. The data is chaotic and requires preprocessing worth being a course project on its own. Hence, I created my own dataset. To create my custom dataset, I used the ImageGrab module of the Pillow package. I specified a certain corner of my window with pixel values and took a snippet of that area every time I pressed one of the defined keys: Down, Up, Left, Right, and X. The arrow keys are responsible for movement and X is the action button [2, 4]. I Generated 2053 images from playing the game for over 8 hours. The data was imbalanced and to make it balanced, I had to trim off some data from a few classes resulting in a dataset of 1575 images. I split the data into 1500 training (300 in each class for 5 classes) data and 75 (15 in each class for

5 classes) validation data resulting in a 95% - 5% Train-validation split. My test data is generated with real-time gameplay, which probably generates unseen data (most game states might be seen in training).

Models

For this project, I am using a CNN model with 2 different architectures. CNN models tend to extract deep features from images through different convolution layers. The first model I used is a vanilla CNN model. I chose this model as I have distinguishable visual features in the game states which should be easy to identify. However, one downside to this model is that each action is independent and is not connected to the previous action. Therefore, there is no sense of “next goal” in this model, which means the model does not know what it should do next.

The Model has a bad sense of direction but a good sense of action. It gets stuck in places as marked in Figure 1. But with some human intervention, it comes back on track. Although It detects when to press “X” (Action button) really well.



```
while(True):
    time.sleep(4)
    img = ImageGrab.grab(bbox=(0,40,800,640))
    key = keymap[get_prediction(model, img)]
    print(key, end = ', ')
    PressKey(key)
    time.sleep(0.2)
    ReleaseKey(key)
```

```
37, 37, 38, 38, 38, 39, 39, 88, 39, 39, 39, 39, 39, 40, 37, 37, 40, 40, 88, 88, 3
```

Figure 1: Here 38 means UP, and the model gets stuck in edge areas, trying to go up or left. However, with some human intervention, the model comes back to making good predictions. 37 is Down, 39 is left, 40 means right, and 88 means X. X is normally pressed to fight when a wild Pokemon is found.

The other architecture uses two image inputs [5]. Two images are passed into CNN layers and then concatenated to be considered towards a single output which is the class of the images. The challenges faced in this model were that the data I had already created was already randomized and there wasn’t enough time to conduct the whole data-gathering process again. That’s why the model doesn’t perform as expected, however, I’ve tested the method and the model architecture on the MSVD dataset and it works just above average. I believe fixing the data will fix this issue and we will have a good Pokémon levelling-up model. However, I still ran the model to see how it performs. The model is highly biased towards “UP” among the movement buttons and occasionally performs an action (Press “X”). The main issue I faced with this model which was not solved is that the model expects 2 image inputs, but during inference, I only get 1 image at each timeframe. I am using the same image twice which might be causing some anomaly in the model.

During the project, I randomly tried different hyperparameters to choose one that performs the best. The performance evaluation is qualitative as it comes down to how the game feels with the bot. The submitted hyperparameters (in code) were found to be the best performing among the values I tried. With these best hyperparameters, I got 0.92 AUC on the vanilla CNN on the action button and 0.718 AUC on average for the movement buttons.

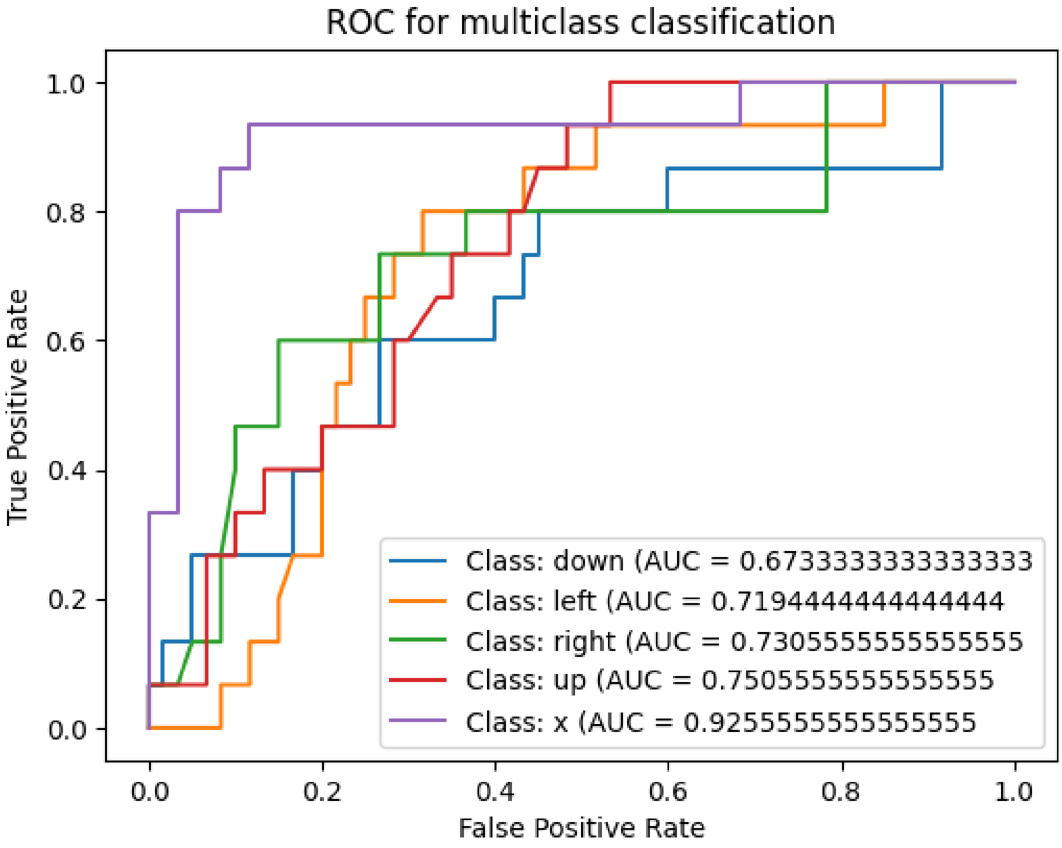


Figure 2: Vanilla CNN AUC for multiple classes1

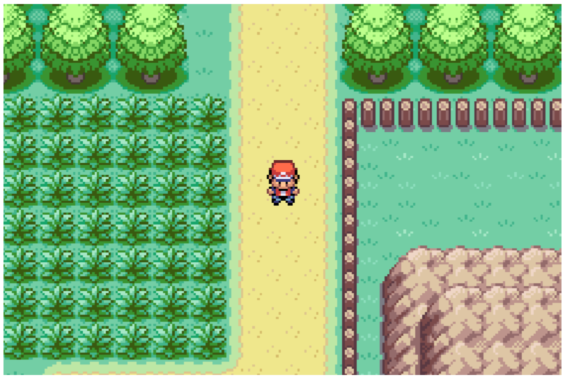


Figure 3: Multi-way classification possibility of one image

Despite having a respectable AUC, the model doesn't perform that well in real-time gameplay. My observation of this phenomenon is that when in a similar state as Figure 3, we can have 3 possible directions to choose from. And the model becomes confused for that.

When in the grass, we can go anywhere we like. Therefore, the grass images like Figure 4 can be found in all the classes. These data-related issues are causing the model to get confused and predict anomalies.

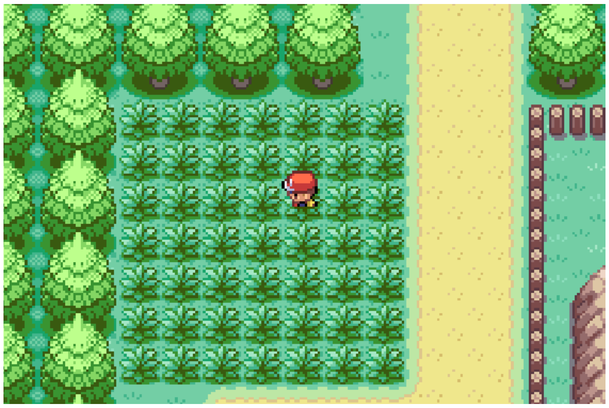


Figure 4: Multi-way classification possibility of one image

Learned Techniques

As an NLP enthusiast from my sophomore year in undergrad, I have never needed or done any image-related project. Therefore, in spite of having my own research projects, I took this project as a challenge and learned a lot in return.

Taking continuous inputs

This part was straightforward. I needed to define a box in the screen and use the ImageGrab module to take screenshots of that area. All I needed was to run my GBA emulator in that area.

In-game key control

This technique was a hard one. Normal pyinput package has functions like press and release. They work perfectly in my notebook, but my games don't detect the key press. To identify and solve that issue I had to spend about a week. I solved that by using the ctypes package and using the hex codes for the keys [3].

Image preprocessing

As my first time doing a computer vision project, I had a hard time with the dimensions of the images and how they change across the convolution layers. I came to an understanding that applying meaningful and proper filters on images works better than unnecessarily increasing the number of filters. I used RandomInvert which is a transformation in the PyTorch deep learning library that randomly inverts the colors of an image by subtracting the pixel values from the maximum value of the data type (255 for uint8) and ColorJitter which is a technique used in computer vision to apply random color transformations to an image (brightness, contrast, saturation, and hue) during training.

Missed Opportunities

Though I learned some techniques, for time restrictions, I couldn’t implement a few ideas I had. The first idea was to consider the problem as a time series data, format it as a sequence and apply the LSTM+CNN model to that. I believe that would solve the problem of 1 frame not having any impact on the following frame. The other basic idea that I missed before and just noticed today is that I can use a pre-trained model and finetune that on my Pokémon data. I tried it for some hours but didn’t have the time to completely debug it. I believe this would significantly boost my model’s accuracy if it could be successfully completed.

Conclusion

Despite some initial challenges with the Pokebot project, I was able to achieve a respectable AUC on the test data. Although, with careful observation, I noticed that some anomalies were being injected into the model due to the randomness in the data - having similar images that classify into multiple classes (Figures 3 and 4). To address this issue, I implemented a powerful CNN architecture that takes in two images and outputs their class. Though the initial goal was to pass two sequential images into that model to get a sense of “intention” - which way the character is actually moving, after building the model for several hyperparameters and testing it for several hours I noticed the randomness in the data which is, hypothetically, causing the anomaly in the data. Another issue I faced was that during the inference I have access to only 1 frame at a given time. I’m not sure how I could implement the same intention of using 2-image CNN in the inference. Provided I had more standard and consistent data, with the same effort, I could end up with a more useful model.

Reference

1. Wu, Jianxin. "Introduction to convolutional neural networks." National Key Lab for Novel Software Technology. Nanjing University. China 5.23 (2017): 495.
2. Sentdex. "Python Plays: Grand Theft Auto V" YouTube, uploaded by Sentdex, 10 Apr 2017, www.youtube.com/watch?v=ks4MPfMq8aQ&list=PLQVvvaa0QuDeETZEOy4VdocT7TOjfSA8a
3. Snomunism. "Creating A python script to auto level my pokemon." YouTube, uploaded by Snomunism, 28 Mar. 2021, www.youtube.com/watch?v=zWUuQ3-zoYw.
4. Rempton Games. “How I Taught an AI to Play Pokemon Emerald.” YouTube, uploaded by Rempton Games, 22 Jun 2020, www.youtube.com/watch?v=9YyQJIuN7n0.
5. Simonyan, Karen, and Andrew Zisserman. "Two-stream convolutional networks for action recognition in videos." Advances in neural information processing systems 27 (2014).