# Training Mode Start!
# Training a Deep Learning Model for an AI-Controlled NPC


DSI-SG-41

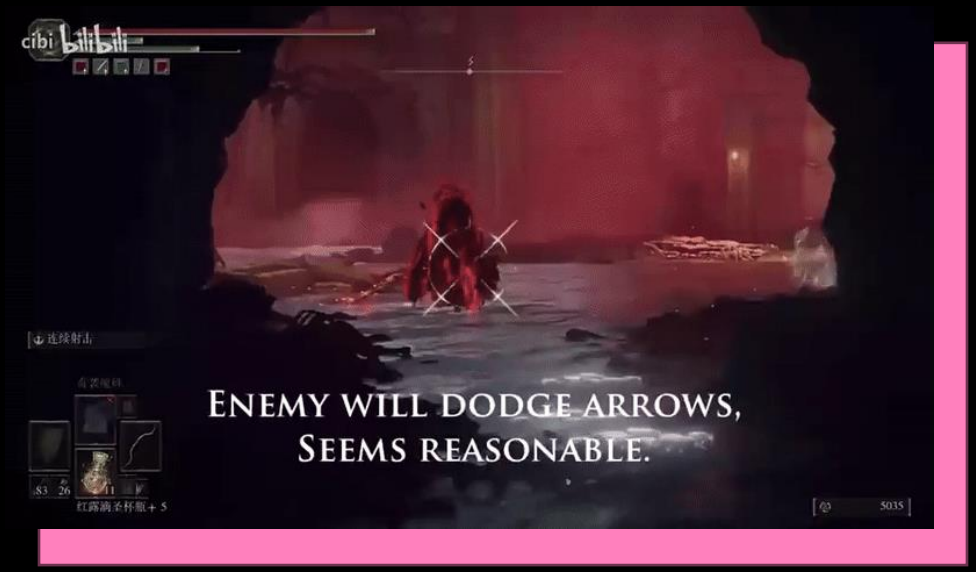
Muhd Faaiz Khan

# Enemy NPCs in Video Games

Enemies in video games are designed to challenge and engage the player, an optimal gaming experience is where a player regards enemies as plausible threats that can feasibly be overcome.

Enemy NPCs that are either too overwhelming or too easy will break the immersion for the player. Thus, there is a push for game developers to design ‘believable’ behavior in enemy NPCs.

**GOAL:** Using machine learning methods to train a non-player entity to behave in a believable and engaging way.



# Game to be trained on

For this  project, I will be training the model on Pokémon: Close Combat, a 2-D fighting game (like Street Fighter) made by Pokémon fans (free to download here!)

# Model objectives

We will train a deep learning model to play through the first three stages of Close Combat's arcade mode, using the default character Blaziken.

The model will be trained on human inputs to react to on-screen information.

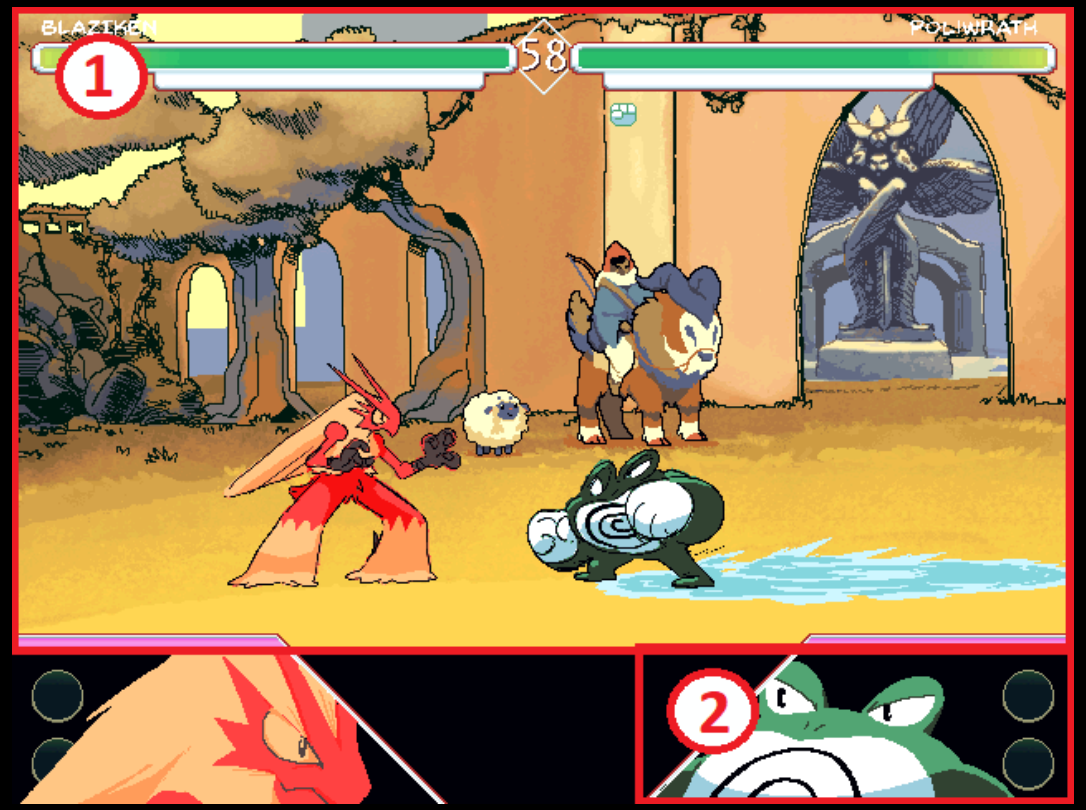
Three separate models will be trained, one for each opponent.

# Data collection

Our data collection function uses the OpenCV Python library to capture these 2 images from an open game window.

- Image 1 (Battle scene)
- Image 2 (Enemy portrait)

When an enemy portrait is detected (using OpenCV's template matching), the function starts saving the images to a folder. The corresponding input performed by the player during the screenshot is recorded using the keyboard library, and saved as a list of 7 booleans.



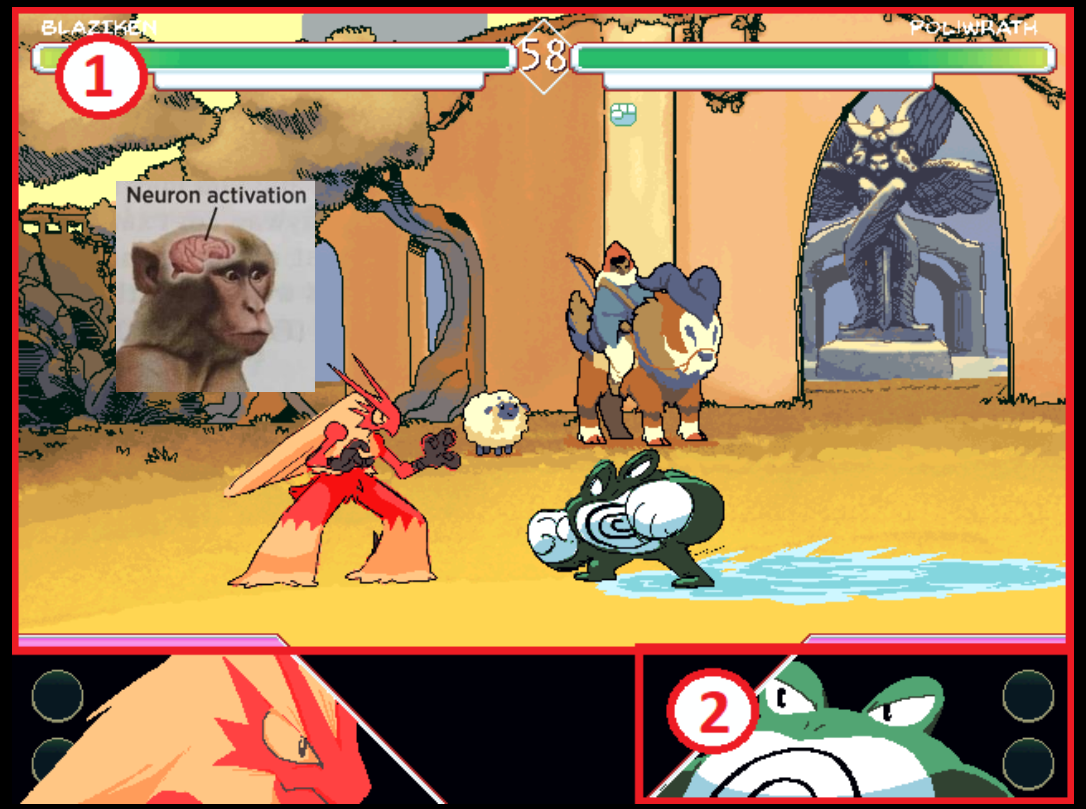

[1, 0, 0, 1, 0, 1, 0]

# Bot script

Once the model is complete, it is deployed using a Python script. This script uses the same OpenCV classes to detect the battle scene and enemy portrait.

When a portrait is detected, read the screen to perform predictions using the model associated with the detected portrait.

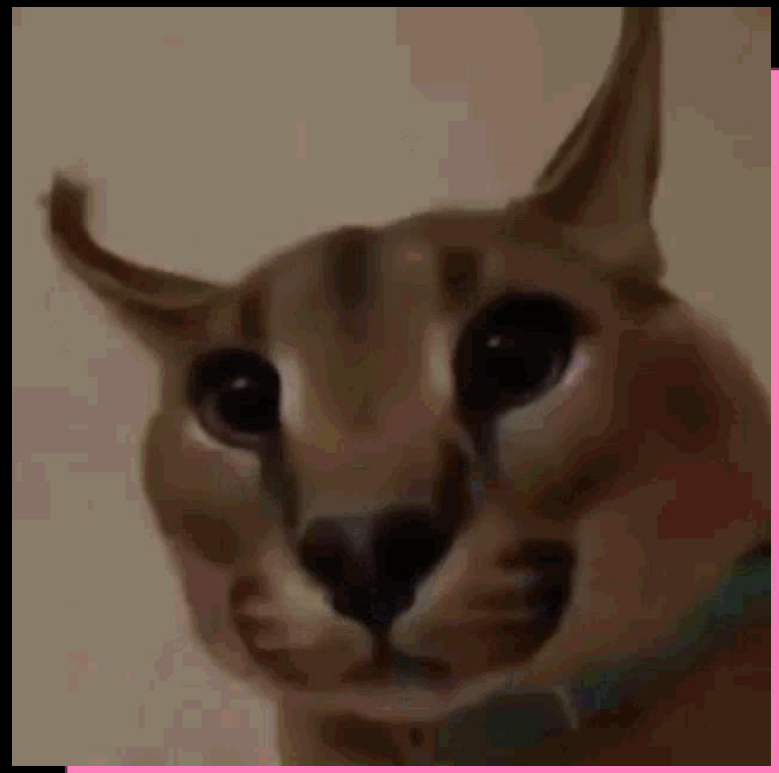
Use the prediction output to perform corresponding key presses using the PyKey library.



# Model used: Convolutional Neural Network

CNNs are a type of supervised machine learning algorithm. The input (usually an image) is initially processed as a numerical array. The array is passed through convolutional layers, where multiple filters are applied to identify spatial/temporal patterns within the image.  After passing through these convolutional layers, the output is flattened into 1-D and passed through dense layers, finally producing the output.

Often used for image classification and object detection, but  have also been used to train models to play board games (such as Go).



# CNNs for fighting games

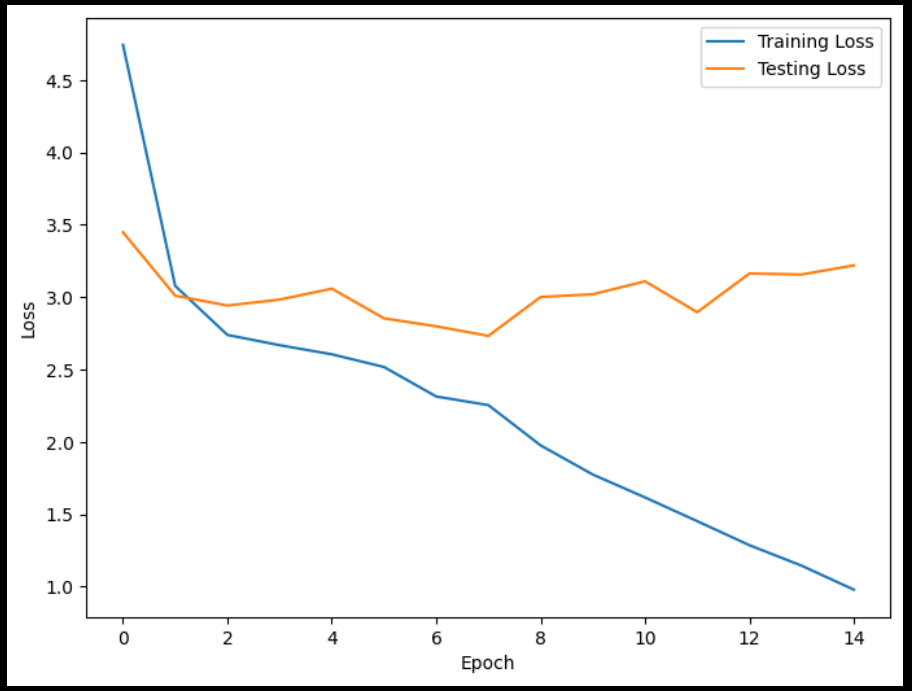
In fighting games, a player's actions are largely dependent on the current state of the battle scene; the player is expected to make quick, split second decisions based on immediate and current information.

# CNNs for fighting games

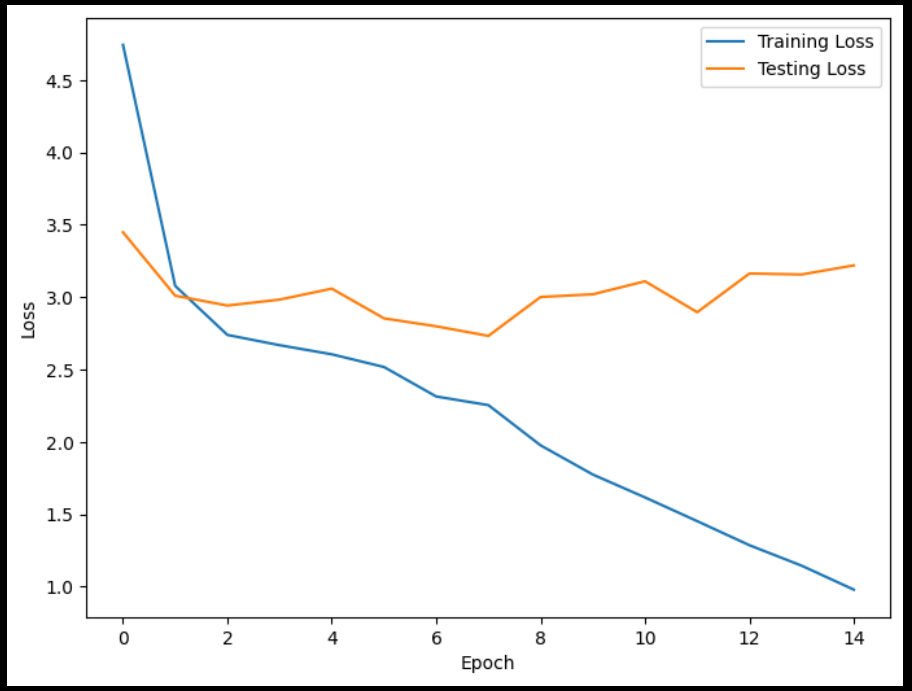
A CNN is thus an appropriate model for a fighting game, as the model will be able to make these split-second decisions based on immediate on-screen information, using trained data.

# Initial Model Performance

| vs Poliwrath | |
|---|---|
| Training loss | 0.9787 |
| Validation loss | 3.07 |



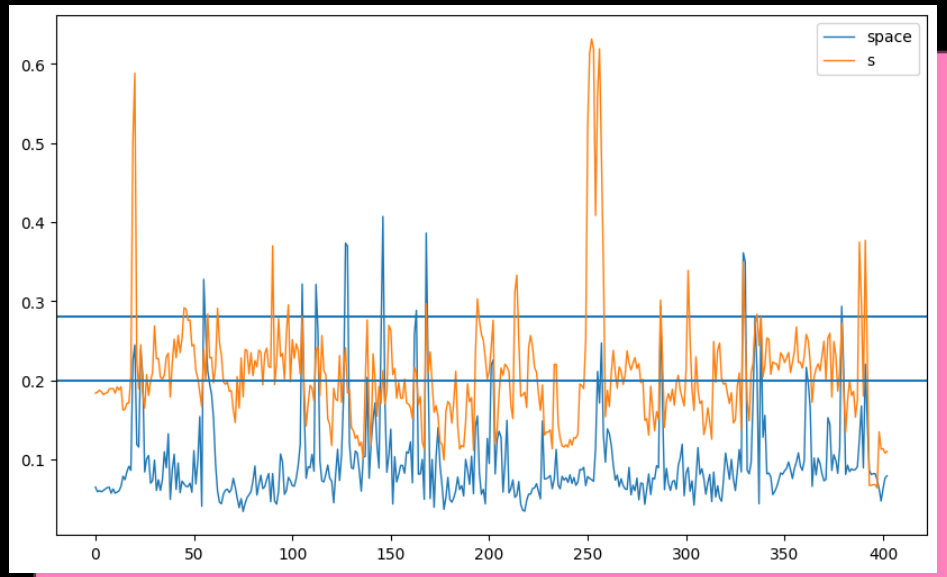
# Initial Model Performance

# Bot Refinement: Defining Thresholds

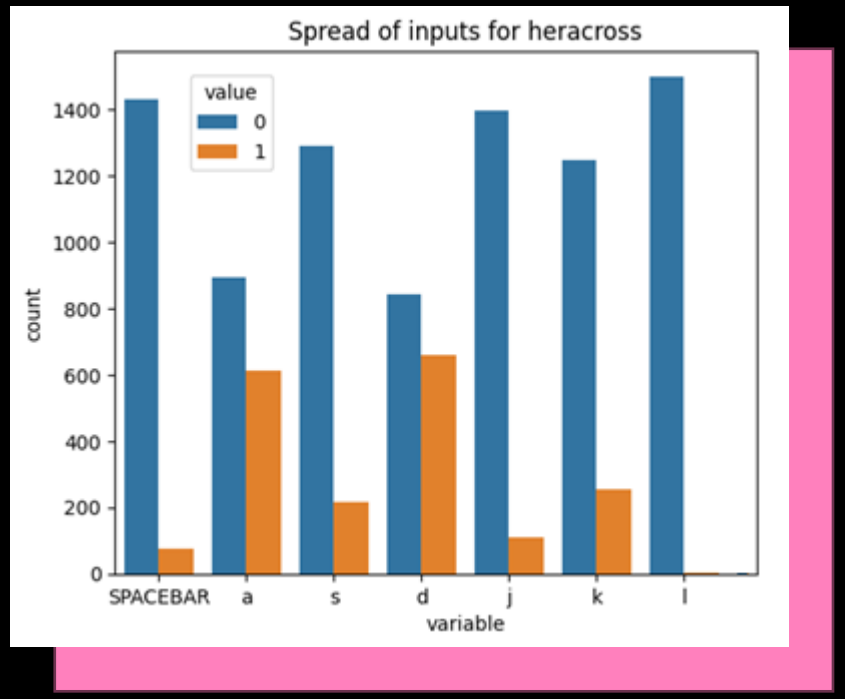
The stochastic nature of deep learning models results in varying prediction values between different models.

To determine the appropriate threshold at which to perform inputs, we first monitor the predictions made by the model over the course of a round, and adapt our thresholds accordingly.



# Data Refinement: Feature Engineering

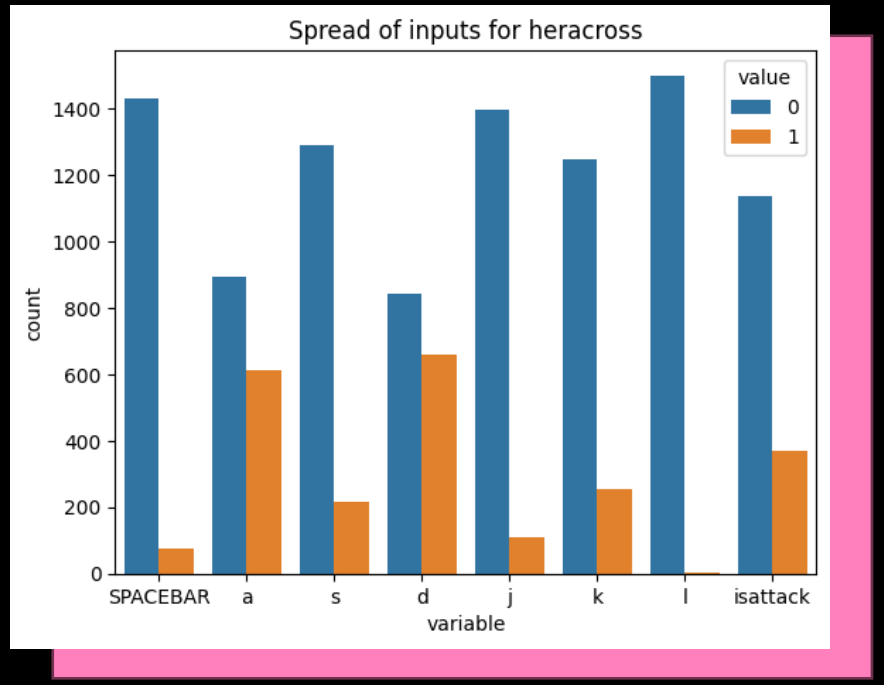
Even after adjusting thresholds,  the model had very low predictive values for the three attack inputs ‘j’, ‘k’ and ‘l’. To improve the frequency of attack detection, we engineer a new feature ‘isattack’, a boolean value that is True for any row where at least 1 attack input is present.



# Data Refinement: Bootstrapping

Certain inputs are underrepresented in the data and are thus poorly predicted by the model.

We bootstrap such inputs to improve the model's confidence in predicting these inputs.



# Data Refinement: Flip it

After multiple model tests, there was a noticeable difference in behaviour for the bot character that was dependent on its position on the screen.

We apply the concept of image distortion on our data, by introducing rows where the existing images are horizontally. This allows the model to predict the same actions regardless of its screen position.

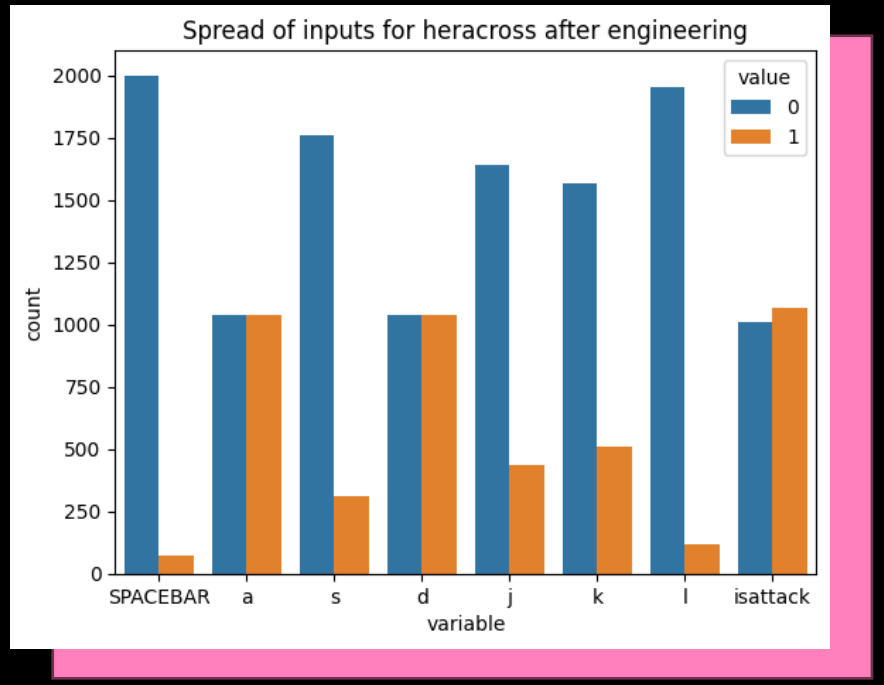
[1, 1, 0, 0, 0, 0, 0]

[1, 0, 1, 0, 0, 0, 0]

# Data Refinement: Putting it together

In addition to bootstrapping and flipping image data, we also implement these data engineering steps:

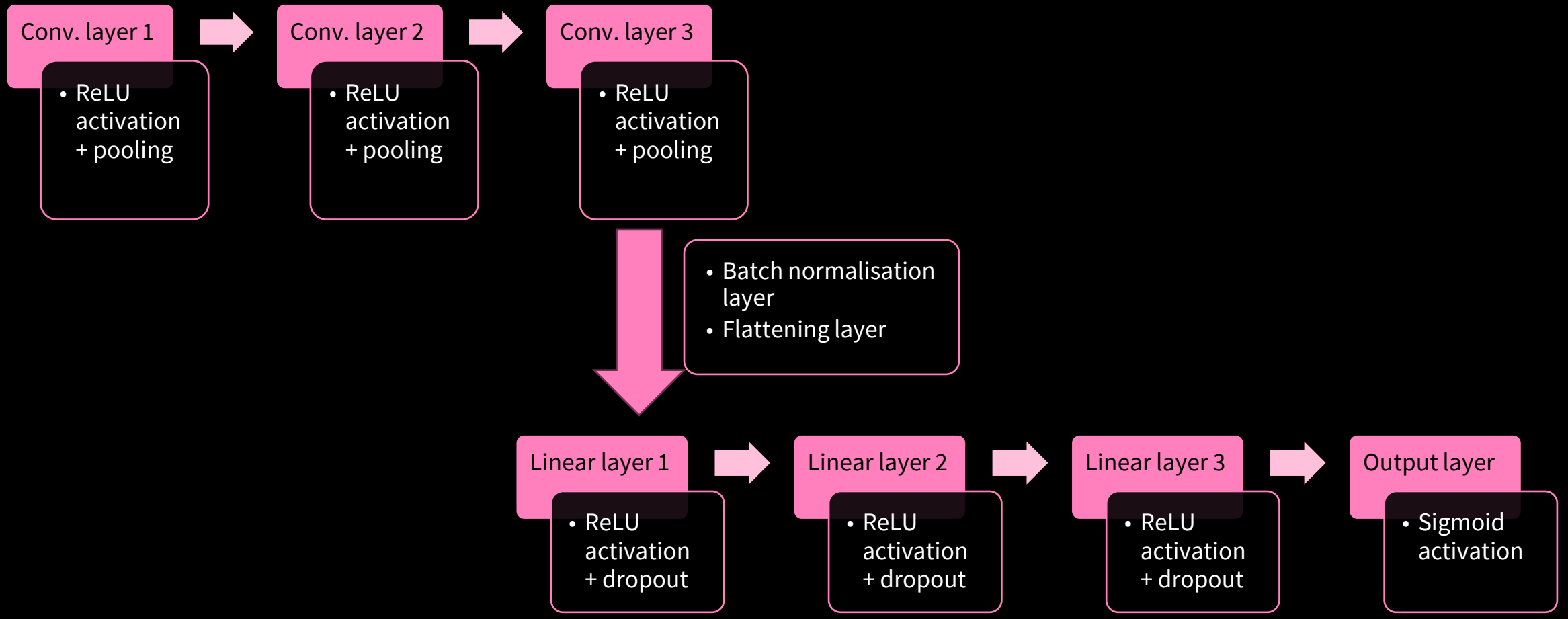
- Randomly dropping a proportion of rows with no attack
- Dropping rows with conflicting inputs
- Dropping rows with no inputs



# Model Refinements

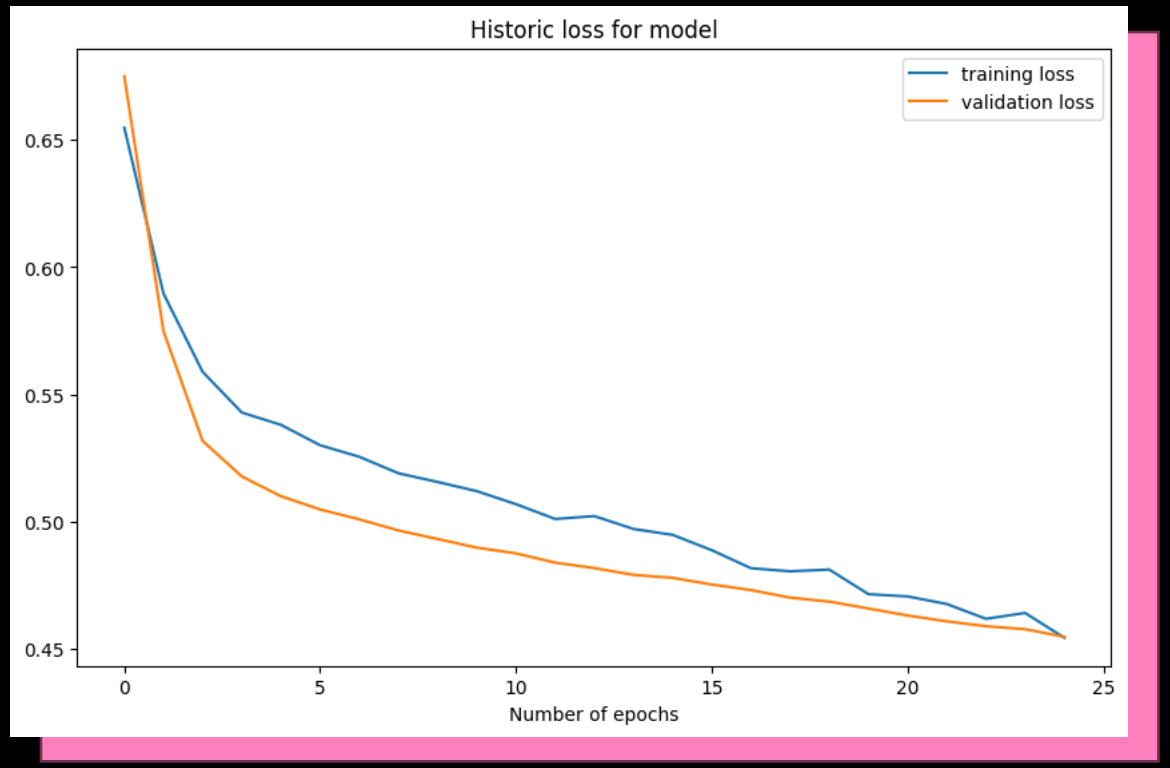
Reducing the learning rate to 1e-6 allowed losses to intercept at higher epochs instead of diverging.

Addition of dropout layer: For each epoch, drop out a randomised proportion of input elements at each linear layer during the forward pass. This is a popular technique for model  regularisation.

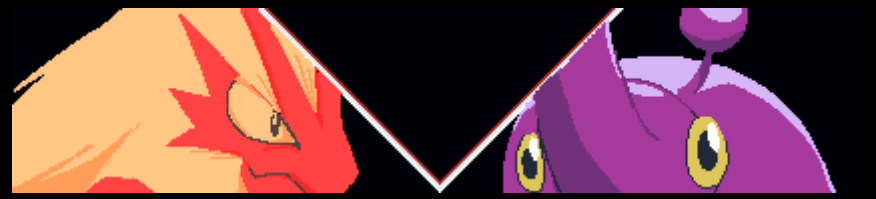

Conv. layer 1
• ReLU activation + pooling
Conv. layer 2
• ReLU activation + pooling
Conv. layer 3
• ReLU activation + pooling
• Batch normalisation layer
• Flattening layer
Linear layer 1
• ReLU activation + dropout
Linear layer 2
• ReLU activation + dropout
Linear layer 3
• ReLU activation + dropout
Output layer
• Sigmoid activation

# Final model performance

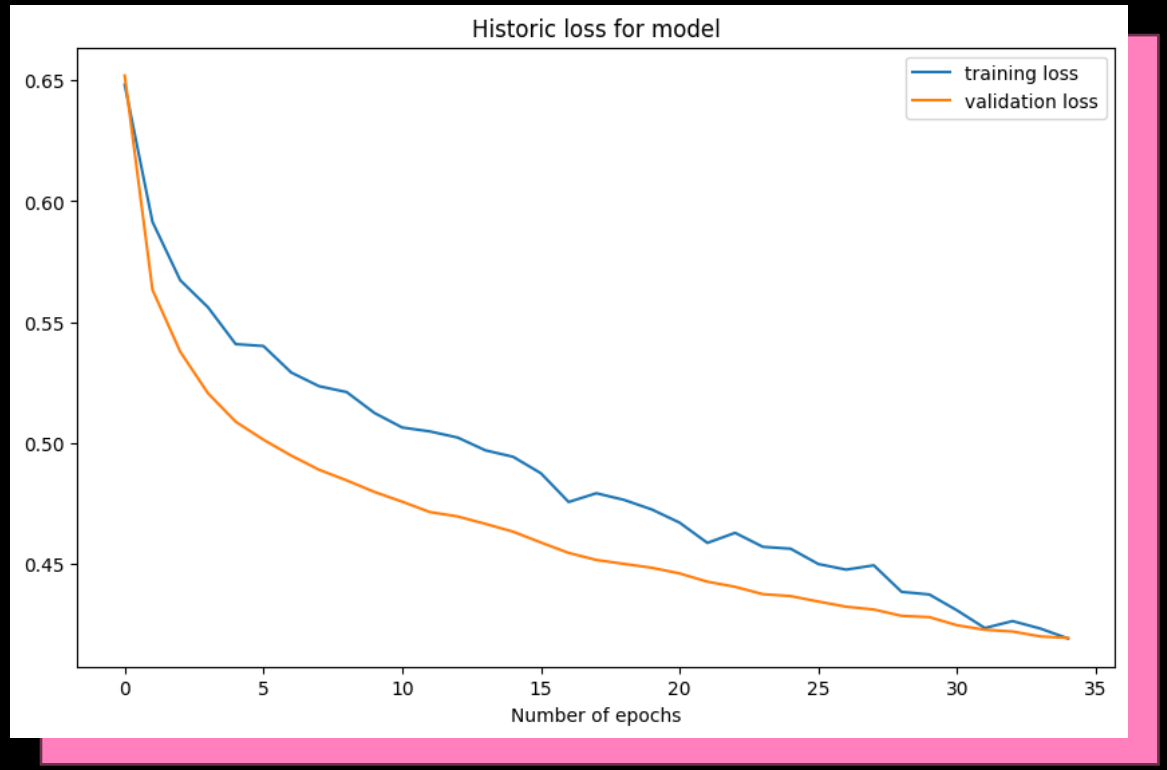
| vs Poliwrath | |
|---|---|
| Training loss | 0.454 |
| Validation loss | 0.455 |
| Training Accuracy | 0.787 |
| Validation Accuracy | 0.783 |



# Final model performance

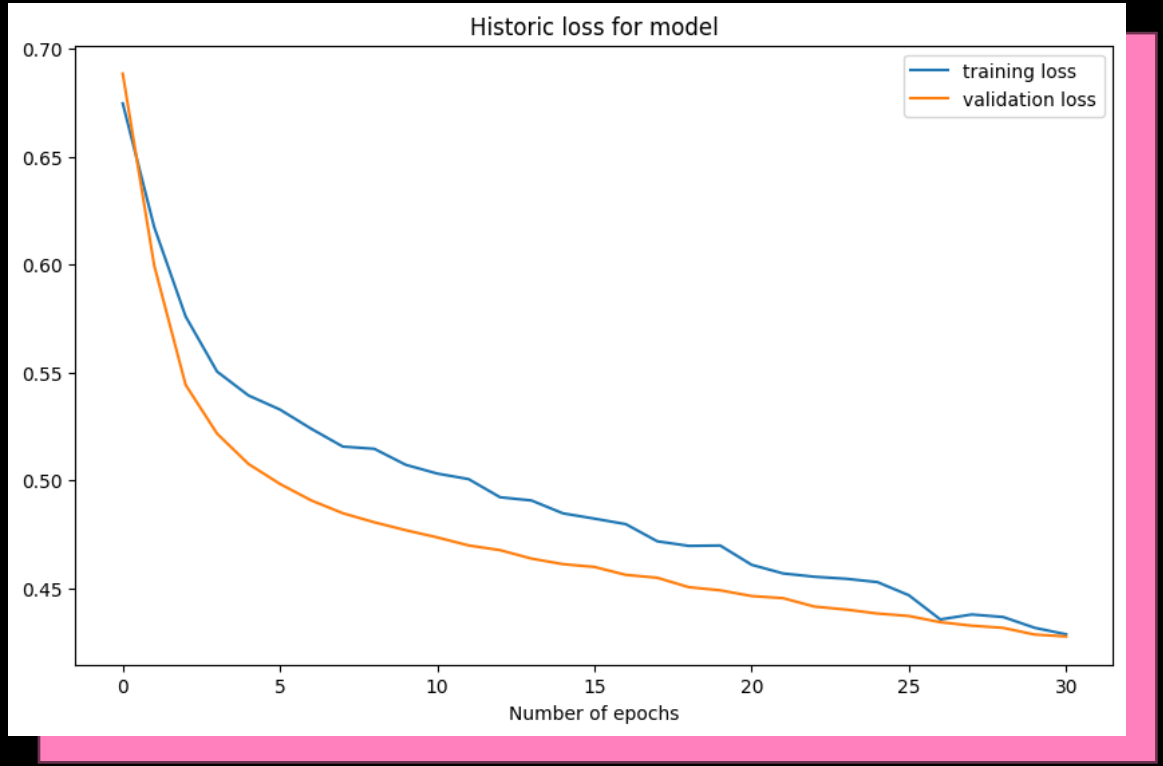


| vs Heracross | |
|---|---|
| Training loss | 0.419 |
| Validation loss | 0.419 |
| Training Accuracy | 0.816 |
| Validation Accuracy | 0.818 |



# Final model performance

| vs Mienshao | |
|---|---|
| Training loss | 0.429 |
| Validation loss | 0.428 |
| Training Accuracy | 0.811 |
| Validation Accuracy | 0.808 |




Demo time!
VS
Blaziken gives you a knowing look.

# Limitations

- Each trained model is huge, almost 900 MB each! It may not be feasible to incorporate these models as they are into a videogame, where multiple different enemy types (requiring different models each) are present.
- Lack of data for specific scenarios leads to the model performing poorly in such scenarios.
- Bot is unable to perform inputs that require a consecutive sequence of inputs, as it has no concept of historical data.

# Future work

- To lower the filesize of the model, we can consider reducing the number of convolutional and linear layers, or reducing the dimensions of the input image. This may have adverse effects on the predictive ability of the model but can be explored as an area for improvement.
- Combining Long Short-Term Memory (LSTM) models with the CNN model would allow the model to incorporate historical data when performing predictions.

# THANK YOU