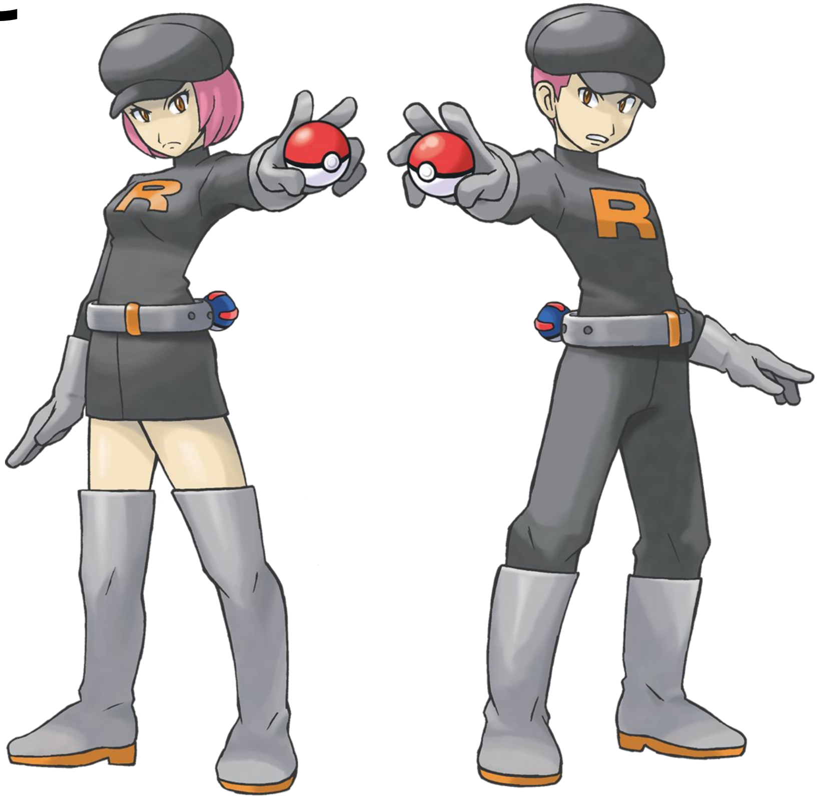
Pokémon Image Classification Challenge

Task 1

Team LLMH



Basics

- The team
  - GALLO Lorenzo 72719
  - LEKBOURI Lina 72697
  - LICHTNER Marc 72690
  - WERCK Hugo 72692
- The work
  - Best public score: 0.93
  - Best private score: 0.93
  - Leaderboard position in the private leaderboard: 2

Task 1: Multilayer Perceptron (MLP) Classification

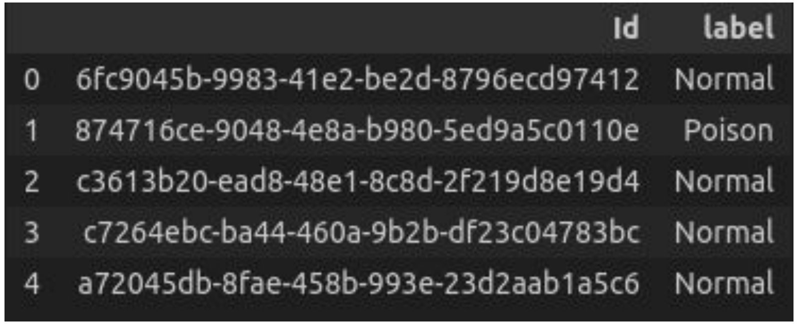
Data Exploration

- Key insights and visualizations (e.g., class distribution, sample images)
- Quantitative analysis of class imbalance
- Dataset characteristics

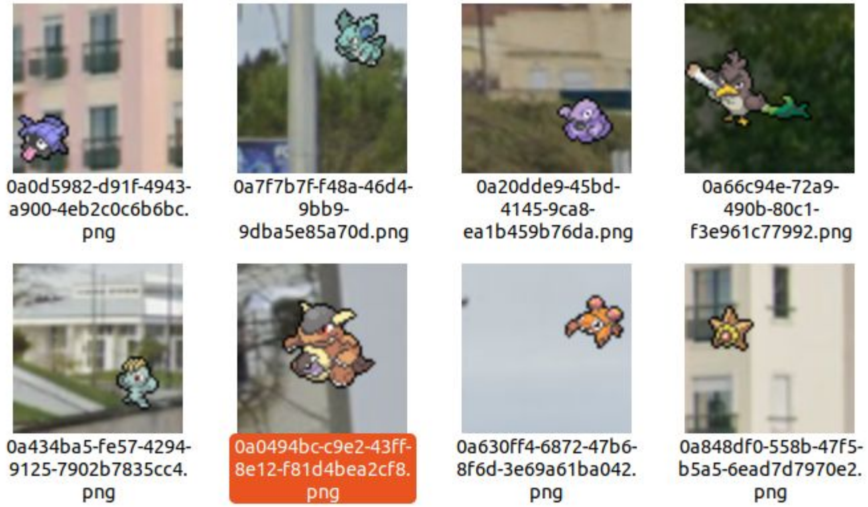
Data Exploration

What can we notice?

- train_labels.csv : Id + labels
- labels + ".png" = images files



	Id	label
0	6fc9045b-9983-41e2-be2d-8796ecd97412	Normal
1	874716ce-9048-4e8a-b980-5ed9a5c0110e	Poison
2	c3613b20-ead8-48e1-8c8d-2f219d8e19d4	Normal
3	c7264ebc-ba44-460a-9b2b-df23c04783bc	Normal
4	a72045db-8fae-458b-993e-23d2aab1a5c6	Normal

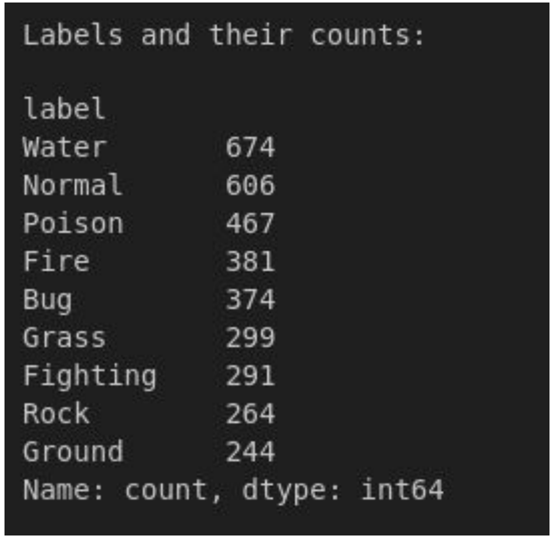


0a0d5982-d91f-4943-a900-4eb2c0c6b6bc.png
0a7f7b7f-f48a-46d4-9bb9-9dba5e85a70d.png
0a20dde9-45bd-4145-9ca8-ea1b459b76da.png
0a66c94e-72a9-490b-80c1-f3e961c77992.png
0a434ba5-fe57-4294-9125-7902b7835cc4.png
0a0494bc-c9e2-43ff-8e12-f81d4bea2cf8.png
0a630ff4-6872-47b6-8f6d-3e69a61ba042.png
0a848df0-558b-47f5-b5a5-6ead7d7970e2.png

Data Exploration

Global descriptive statistics:

- number of images: 3600
- number of labels: 9
- duplicate ones: 0
- labels and their counts:



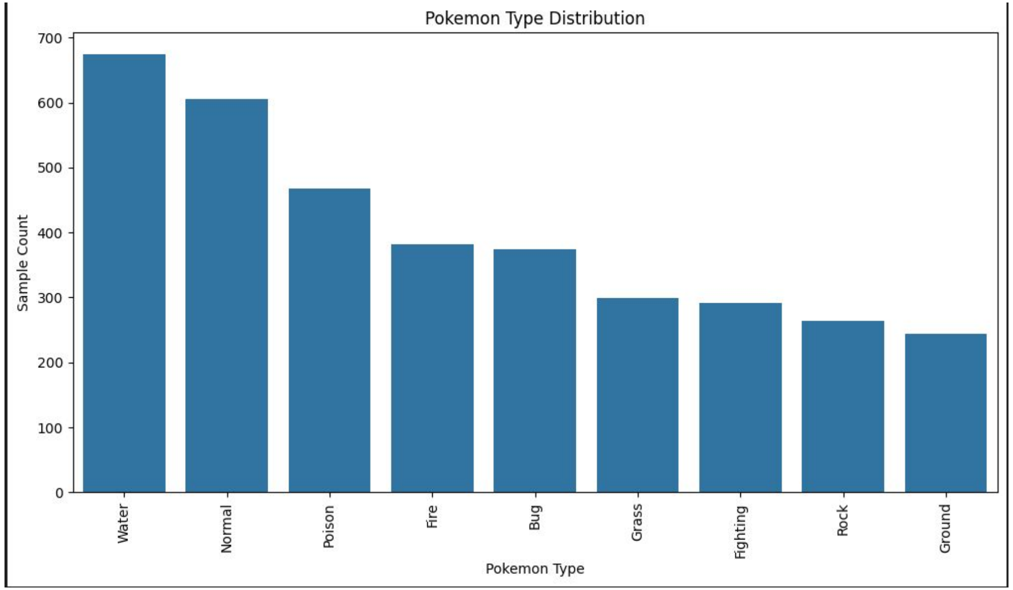
```
Labels and their counts:

label
Water       674
Normal      606
Poison      467
Fire        381
Bug         374
Grass       299
Fighting    291
Rock        264
Ground      244
Name: count, dtype: int64
```

Data Exploration

- Class distribution visualization

  an histogram?

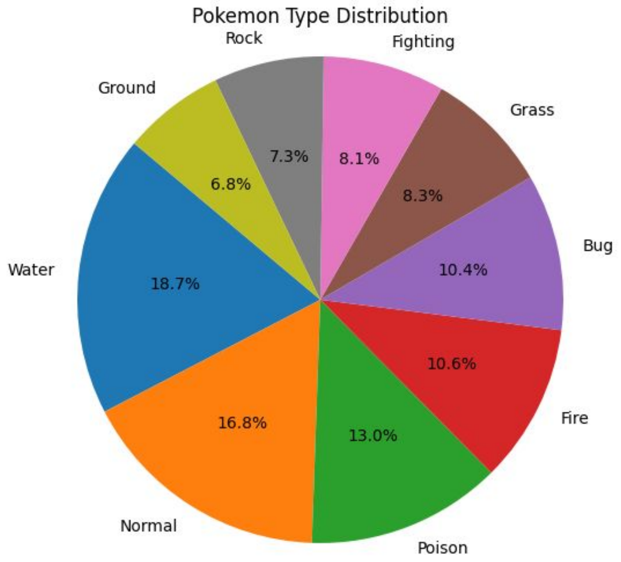


Data Exploration

- Class distribution visualization

  →a pie chart with percentages

  Best way to see any **imbalance**



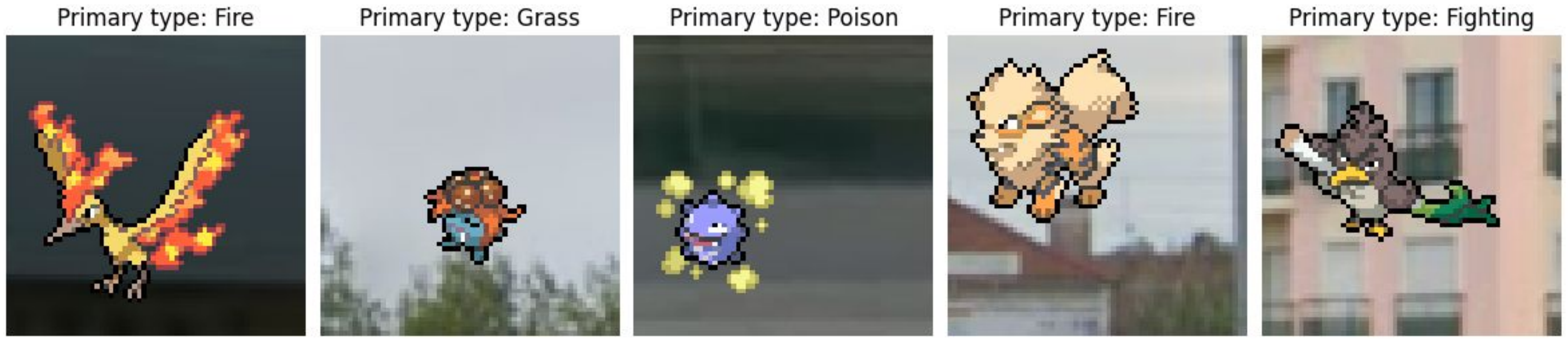
Data Exploration

Images analysis

- size: **64*64 pixels**
- background? It has nothing to do with the pokémon

Data Exploration

- Overview of some primary types



Data Exploration

- How do we balance data ?

  Possible methods:

  - undersampling: Decrease the number of instances in the majority class.
  - oversampling: Increase the number of instances in the minority class.

→ Choice : **undersampling**, by using the value of the less frequent class.

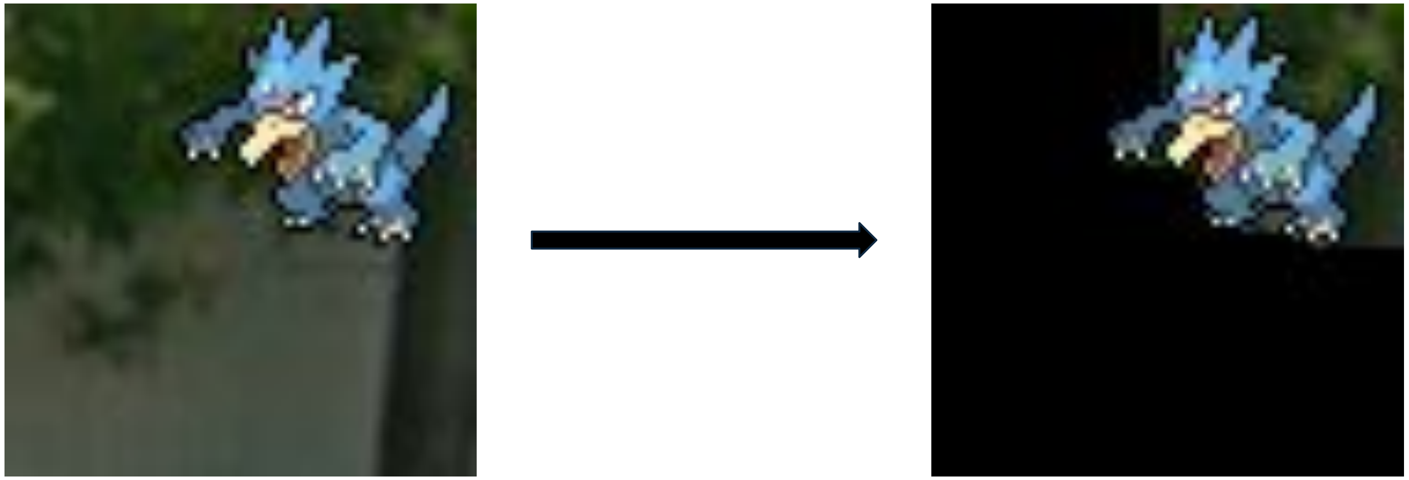
Disclaimer

It is the day before the submission of the first task and we achieved to center the Pokemon on the picture. As a result, the accuracy of the model exploded and went from **around 0.3** on validation test to **0.93.**

The result you will see below are the result before the centering.

Data Exploration

- Data preprocessing : removing the background

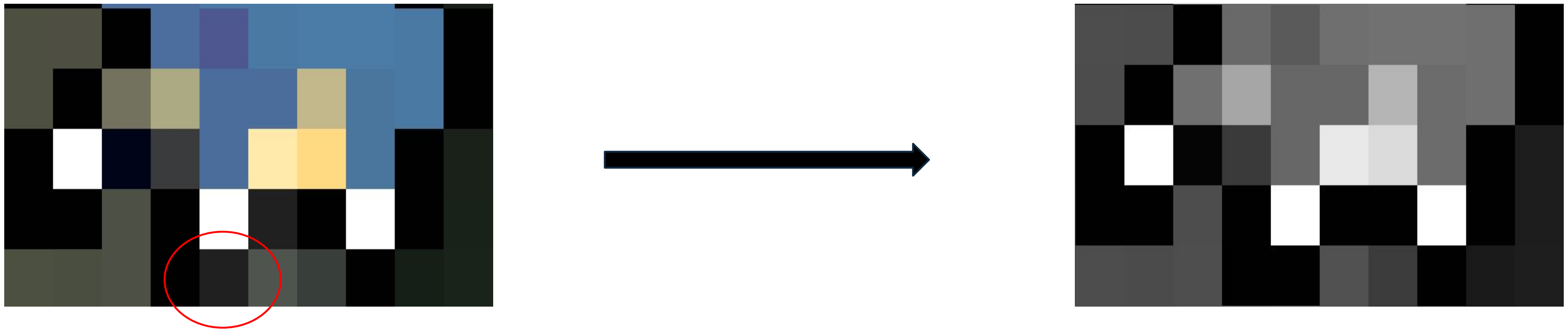


Remove the background

- Few examples

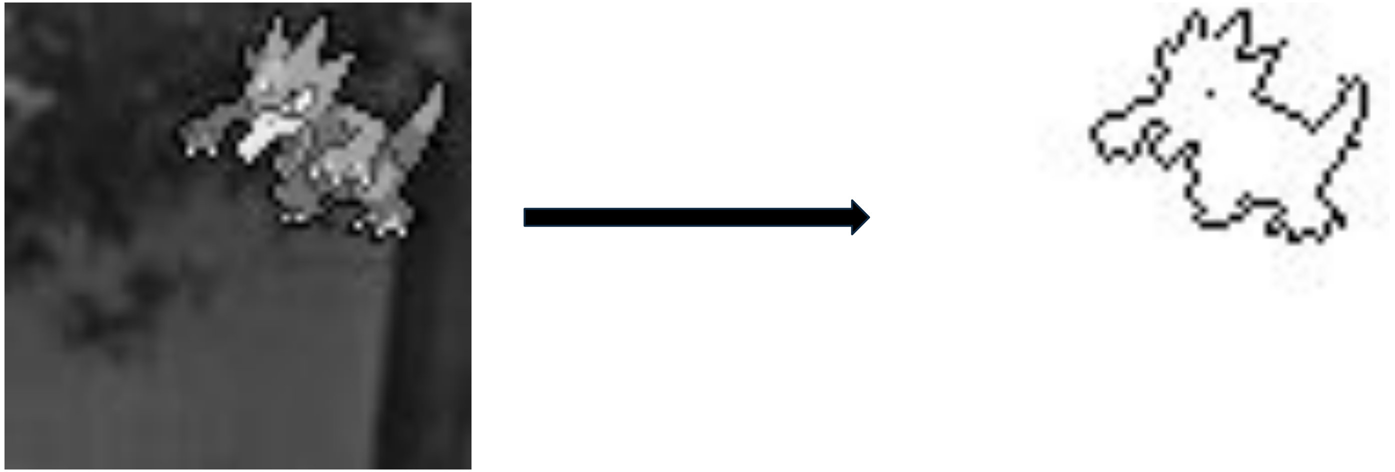
Remove the background

First step: fix_border: Make every pixels of the border the exact same value



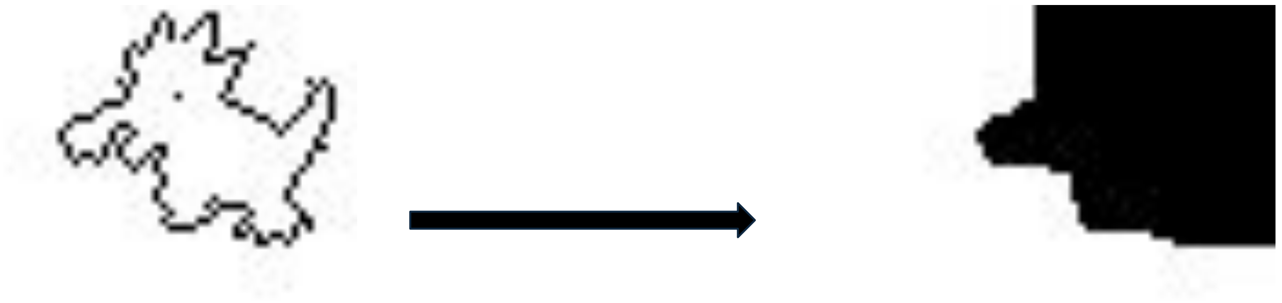
Remove the background

Second step: apply_threshold: Make the border black and everything else white



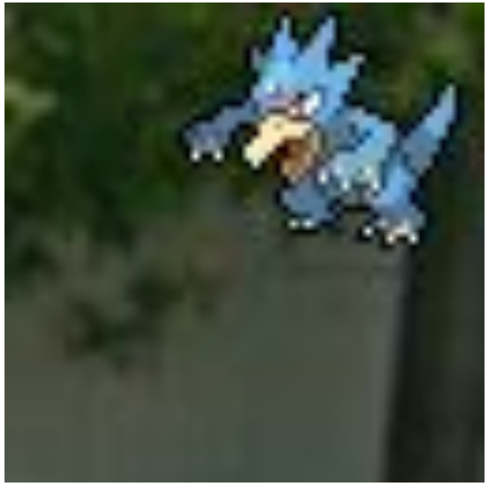
Remove the background

Third step: erode_and_dilate: Fill the inside of the border using the closing operation which creates the mask as a result



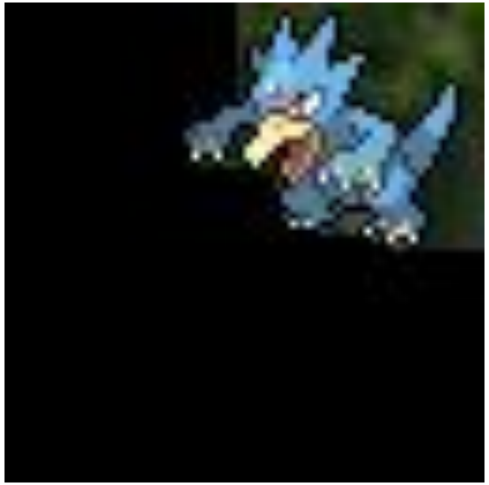
Remove the background

Last step: apply the mask on the original image



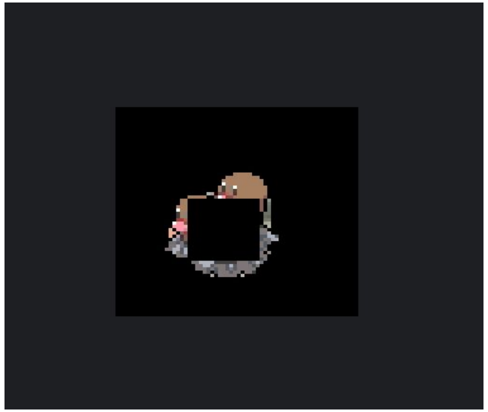
AND

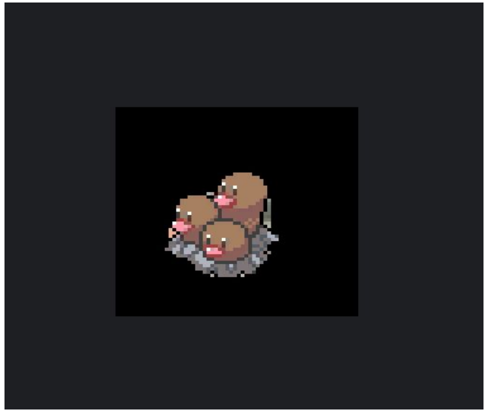
=



Remove the background

Remark: This filter keep parts of the background because of the number of time we iterate using the closing algorithm. We had to make sure that data from the pokemon don’t get remove


9 iterations


10 iterations

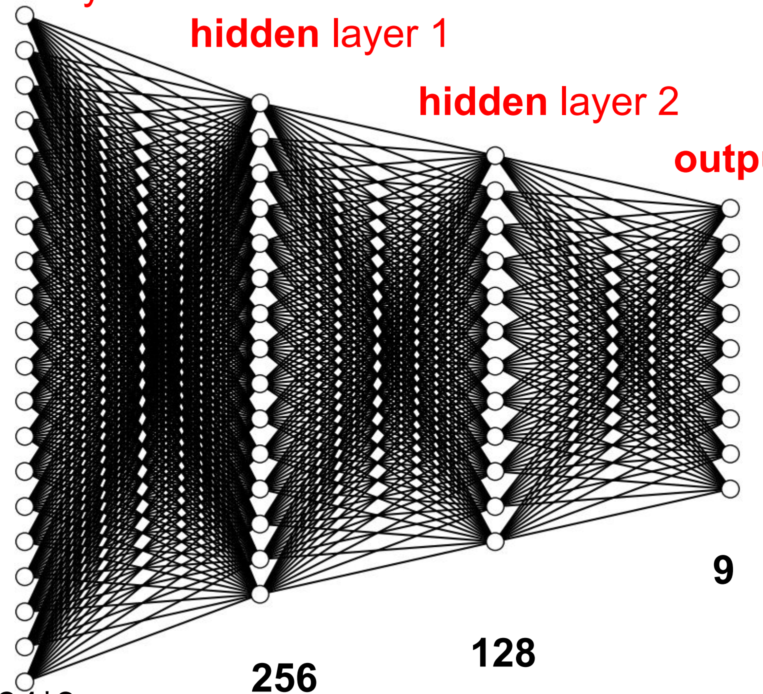
Model Development

- MLP architecture diagram
- Justification for chosen architecture (number of layers, neurons, activations)
- Ablation study

Model Development

- MLP architecture diagram:

(Given the big numbers of neurons, we can not represent the real numbers on this slide, so we take random numbers of neurons respecting orders of magnitude.)



number of neurons: 64*64*3 = **12288** **256** **128** **9**

Model Development

- Justification for chosen architecture

4 layers:

- input layer: **64*64*3 = 12 288 neurons** (number of pixels in each image by the RGB channel)
- hidden layer 1: 256 neurons
- hidden layer 2: 128 neurons
- output layer: **9 neurons** (number of primary types of pokémon)

Model Development

- Justification for chosen architecture: trying other architectures.

  ⇒ justification for the **hidden layers**.

  To justify our architecture, we trained our model with **other architectures**, printed the **result on the validation set,** and concluded that the **best results** were given by our chosen one.

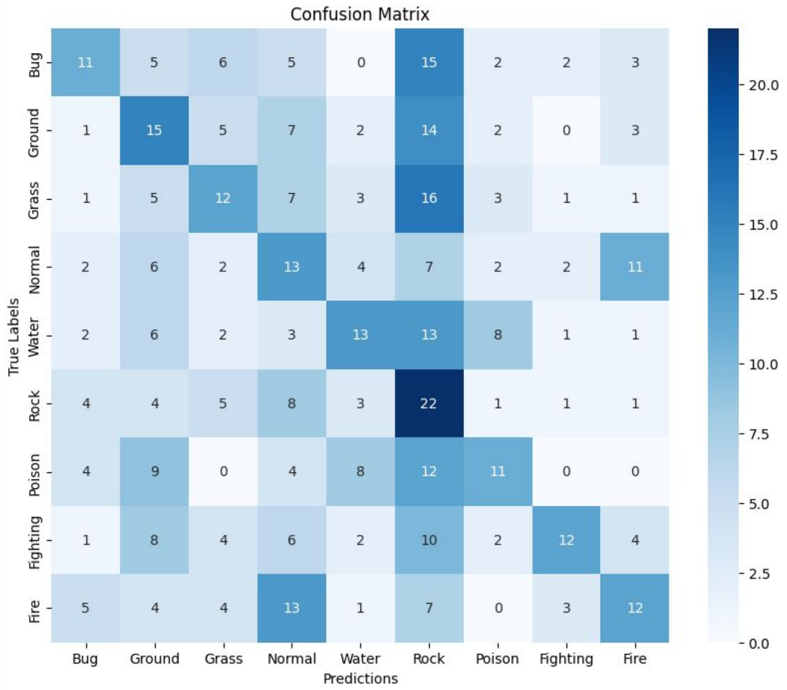
Model Development

- Justification of the architecture of **hidden layers**.

<u>Configuration 1:</u>

- hidden layer 1: **512**
- hidden layer 2: **256**

→ a class is more predicted than the other.



True Labels \ Predictions	Bug	Ground	Grass	Normal	Water	Rock	Poison	Fighting	Fire
Bug	11	5	6	5	0	15	2	2	3
Ground	1	15	5	7	2	14	2	0	3
Grass	1	5	12	7	3	16	3	1	1
Normal	2	6	2	13	4	7	2	2	11
Water	2	6	2	3	13	13	8	1	1
Rock	4	4	5	8	3	22	1	1	1
Poison	4	9	0	4	8	12	11	0	0
Fighting	1	8	4	6	2	10	2	12	4
Fire	5	4	4	13	1	7	0	3	12

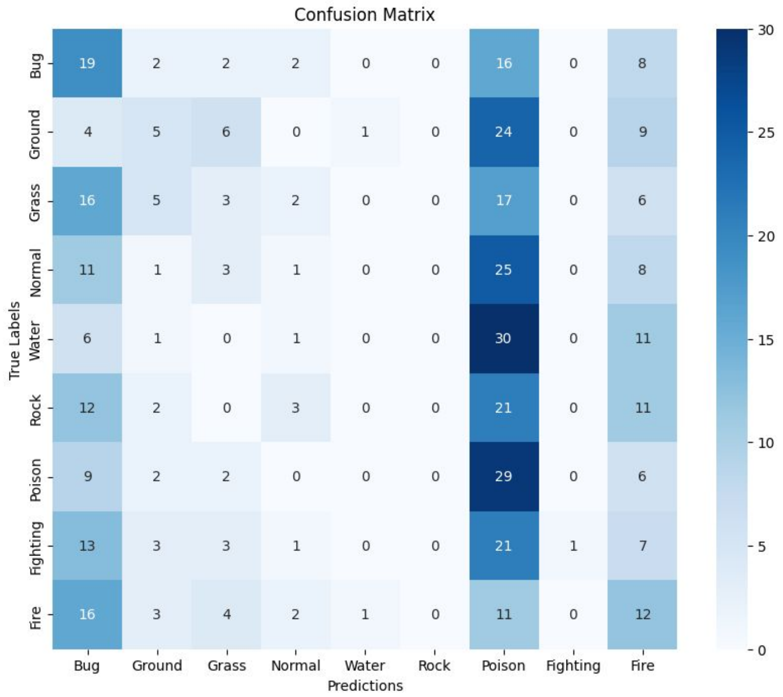
Model Development

- Justification of the architecture of **hidden layers**.

<u>Configuration 2:</u>

- hidden layer 1: **1024**
- hidden layer 2: **512**
- hidden layer 3: **256**

→ **2 classes are way more predicted than the other.**



True Labels \ Predictions	Bug	Ground	Grass	Normal	Water	Rock	Poison	Fighting	Fire
Bug	19	2	2	2	0	0	16	0	8
Ground	4	5	6	0	1	0	24	0	9
Grass	16	5	3	2	0	0	17	0	6
Normal	11	1	3	1	0	0	25	0	8
Water	6	1	0	1	0	0	30	0	11
Rock	12	2	0	3	0	0	21	0	11
Poison	9	2	2	0	0	0	29	0	6
Fighting	13	3	3	1	0	0	21	1	7
Fire	16	3	4	2	1	0	11	0	12

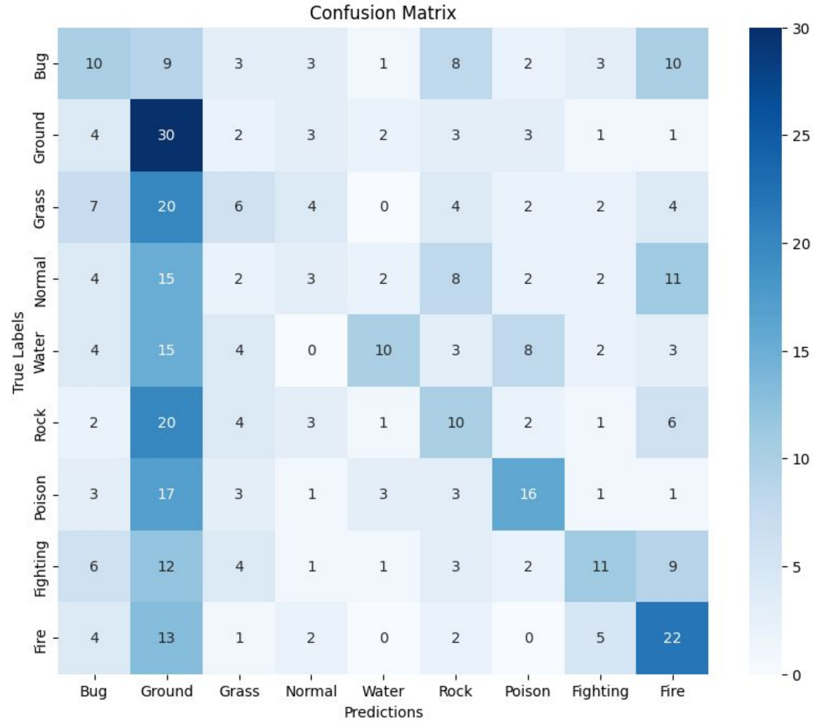
Model Development

- Justification of the architecture of **hidden layers**.

<u>Configuration 3:</u>

- hidden layer 1: **1024**
- hidden layer 2: **512**

→a class is more predicted than the other.



	Bug	Ground	Grass	Normal	Water	Rock	Poison	Fighting	Fire
Bug	10	9	3	3	1	8	2	3	10
Ground	4	30	2	3	2	3	3	1	1
Grass	7	20	6	4	0	4	2	2	4
Normal	4	15	2	3	2	8	2	2	11
Water	4	15	4	0	10	3	8	2	3
Rock	2	20	4	3	1	10	2	1	6
Poison	3	17	3	1	3	3	16	1	1
Fighting	6	12	4	1	1	3	2	11	9
Fire	4	13	1	2	0	2	0	5	22

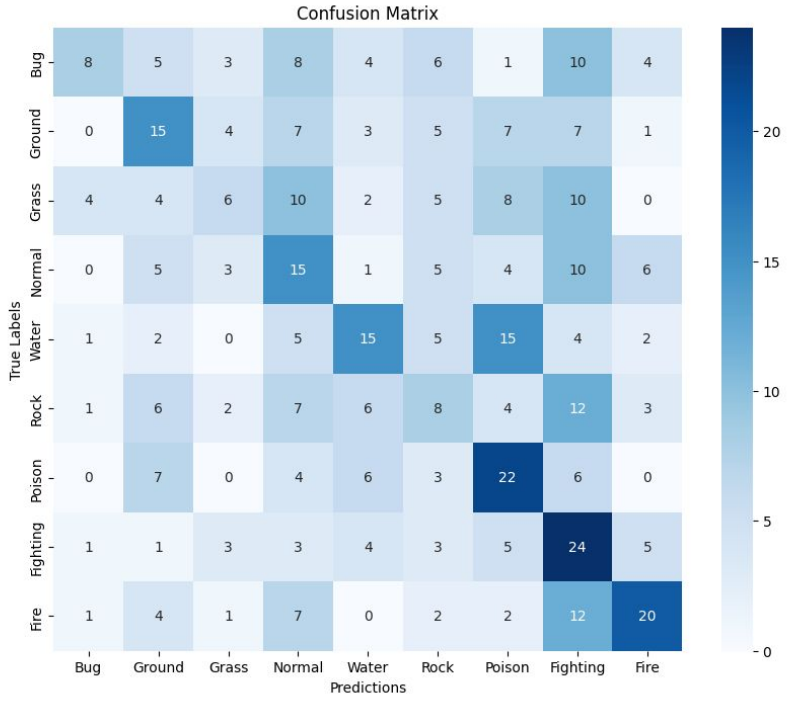
Model Development

- Justification of the architecture of **hidden layers**.

<u>Configuration 4:</u>

- hidden layer 1: **256**
- hidden layer 2: **128**
- hidden layer 3: **64**

→ less good on the test data.



True Labels \ Predictions	Bug	Ground	Grass	Normal	Water	Rock	Poison	Fighting	Fire
Bug	8	5	3	8	4	6	1	10	4
Ground	0	15	4	7	3	5	7	7	1
Grass	4	4	6	10	2	5	8	10	0
Normal	0	5	3	15	1	5	4	10	6
Water	1	2	0	5	15	5	15	4	2
Rock	1	6	2	7	6	8	4	12	3
Poison	0	7	0	4	6	3	22	6	0
Fighting	1	1	3	3	4	3	5	24	5
Fire	1	4	1	7	0	2	2	12	20

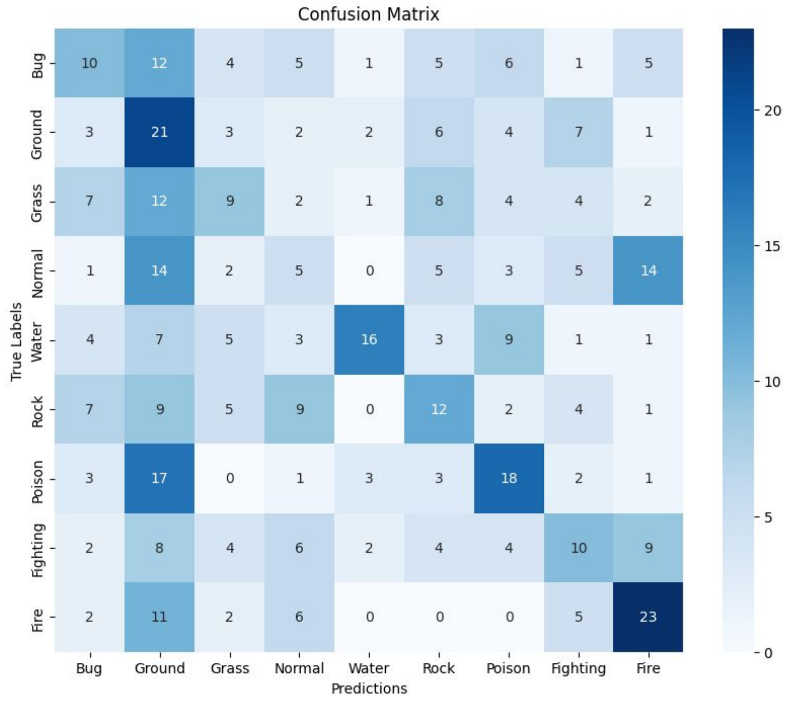
Model Development

- Justification of the architecture of **hidden layers**.

<u>Final configuration:</u>

- hidden layer 1: **256**
- hidden layer 2: **128**

→ the one we choose, because it is also better on the test data.



	Bug	Ground	Grass	Normal	Water	Rock	Poison	Fighting	Fire
Bug	10	12	4	5	1	5	6	1	5
Ground	3	21	3	2	2	6	4	7	1
Grass	7	12	9	2	1	8	4	4	2
Normal	1	14	2	5	0	5	3	5	14
Water	4	7	5	3	16	3	9	1	1
Rock	7	9	5	9	0	12	2	4	1
Poison	3	17	0	1	3	3	18	2	1
Fighting	2	8	4	6	2	4	4	10	9
Fire	2	11	2	6	0	0	0	5	23

Model Development

- That is why we choose this hidden architecture.

4 layers:

- input layer: **64*64*3 neurons** (number of pixels in each image by the RGB channel
- hidden layer 1: **256** neurons
- hidden layer 2: **128** neurons
- output layer: **9 neurons** (number of primary types of pokémon)

Model Development

- Justification for activations

Activation functions:

- **ReLU** (from input to h1 and from h1 to h2): $\mathrm{ReLU}(x) = \max(0, x)$
- **Softmax** (from h2 to output): $\mathrm{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$

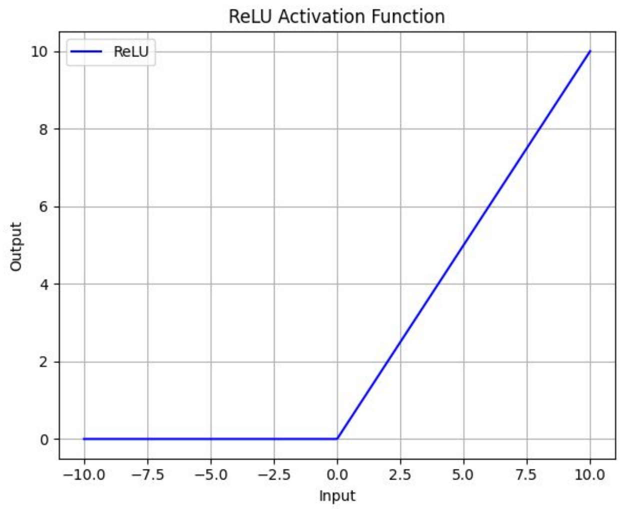
Model Development

- Justification for activations
  - **ReLU activation function, why?**

    - Efficient and computationally cheap;

    - Avoids vanishing gradient issues common in other activations;

    - Improve generalization.

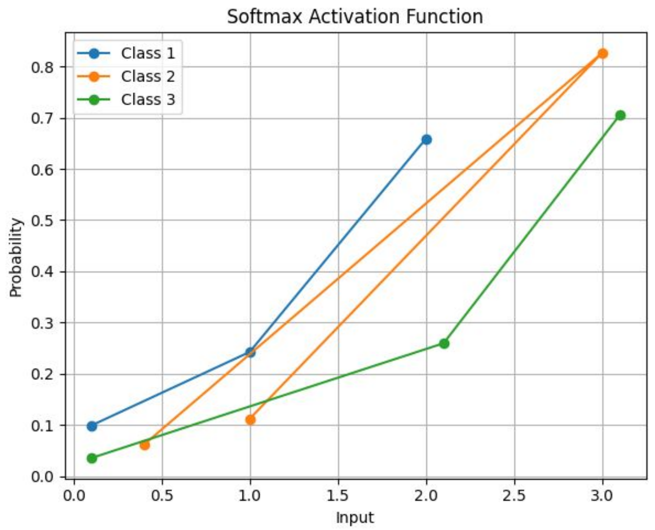


Model Development

- Justification for activations
  - **Softmax activation function, why?**

    - Converts raw scores into probabilities, making the model interpretable;

    - Ensures sum of outputs equals 1.



Model Development

- Ablation study: understanding the impact of every component in the performance

Experiments Conducted:

- **removing ReLU** → Model struggles to learn complex patterns, lower accuracy;
- **replacing ReLU with Sigmoid** → Slower convergence, possible vanishing gradient issues;
- **removing Softmax** → Outputs unbounded raw scores instead of interpretable probabilities;
- **reducing hidden layers** →Model underfits, struggles with feature extraction.

Model Development

- Ablation study: conclusions on the experiments conducted

  - ReLU significantly improves **training efficiency and performance**;
  - Softmax is essential for **classification tasks**;
  - Deeper architectures capture **more complex representations**.

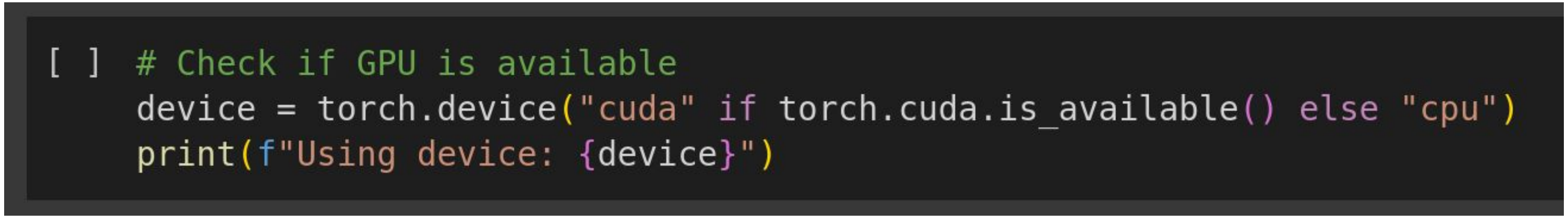
Training Efficiency

- GPU usage
- Strategies employed for efficient training (early stopping, batch sizes, etc.)
- Training time

Training Efficiency

GPU usage:

By using the GPU, it took 1min59 to compute 200 epochs whereas it took 4min54 with the CPU. This is what we used to take advantage of the GPU if available:



```
[ ]  # Check if GPU is available
     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
     print(f"Using device: {device}")
```

Training Efficiency

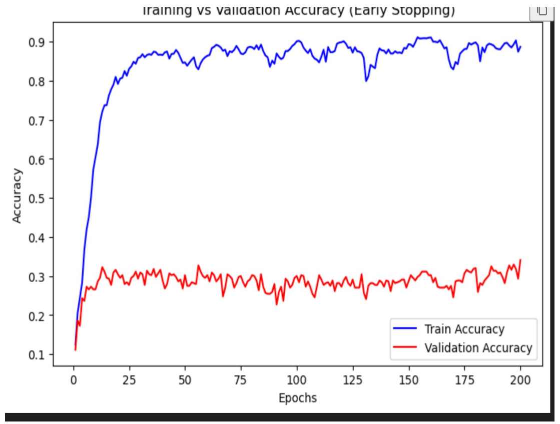
- Strategies employed for efficient training (early stopping, batch sizes, etc.)

  - **Early stopping**: Avoid useless computations by stopping the training when the model don’t improve anymore

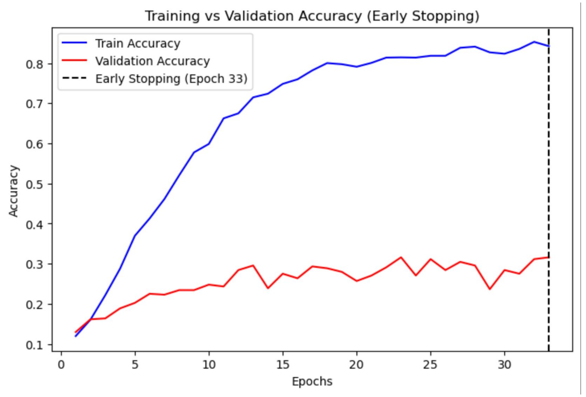
  - **Batch size**: In order to select the batch sizes, we followed the “linear scaling rule”. We chose a batch size of 32 and a learning rate of 0.001.

Early stopping

We used early stopping because the model was quickly converging to a maximum around 25 epochs and won’t improve it’s accuracy on the validation dataset anymore



Without early stopping accuracy of 0.3087



With early stopping accuracy of 0.2977

Training Efficiency

Training time

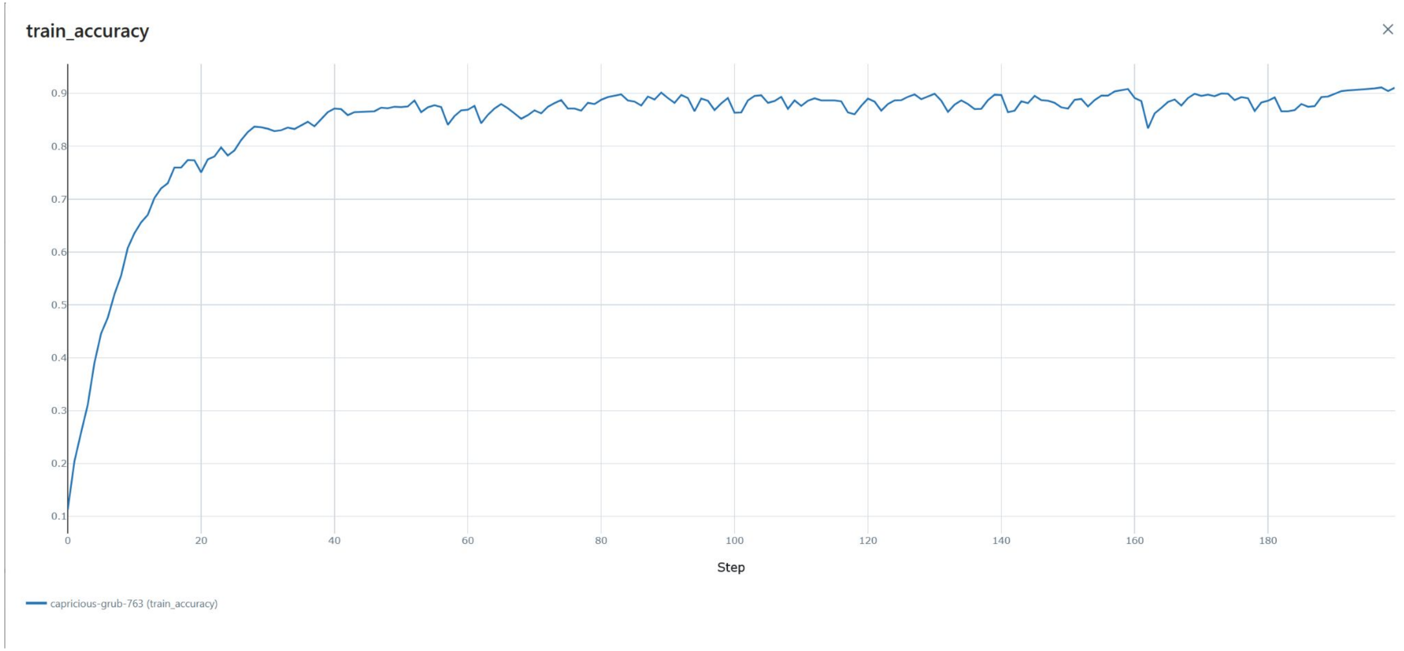
By using early stopping and the gpu, the training took **24 seconds.**

Performance Evaluation

- Justification of the metrics used (macro F1 score, accuracy, confusion matrix)
- Interpretation of results

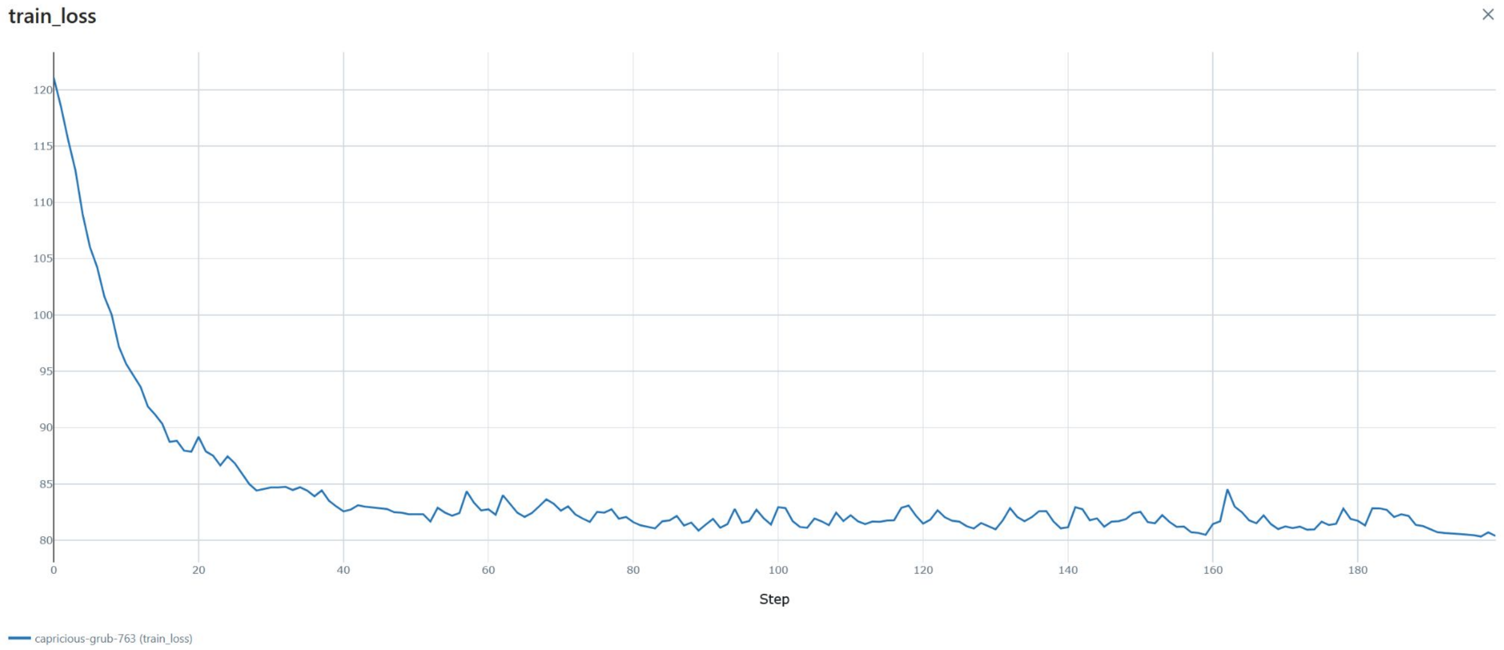
Performance Evaluation

- Train accuracy
  - Measures the percentage of correctly classified training samples, indicating how well the model fits the training data.
  - The curve shows an increasing trend, reaching over 80%. This suggests the model is learning effectively from the training data.



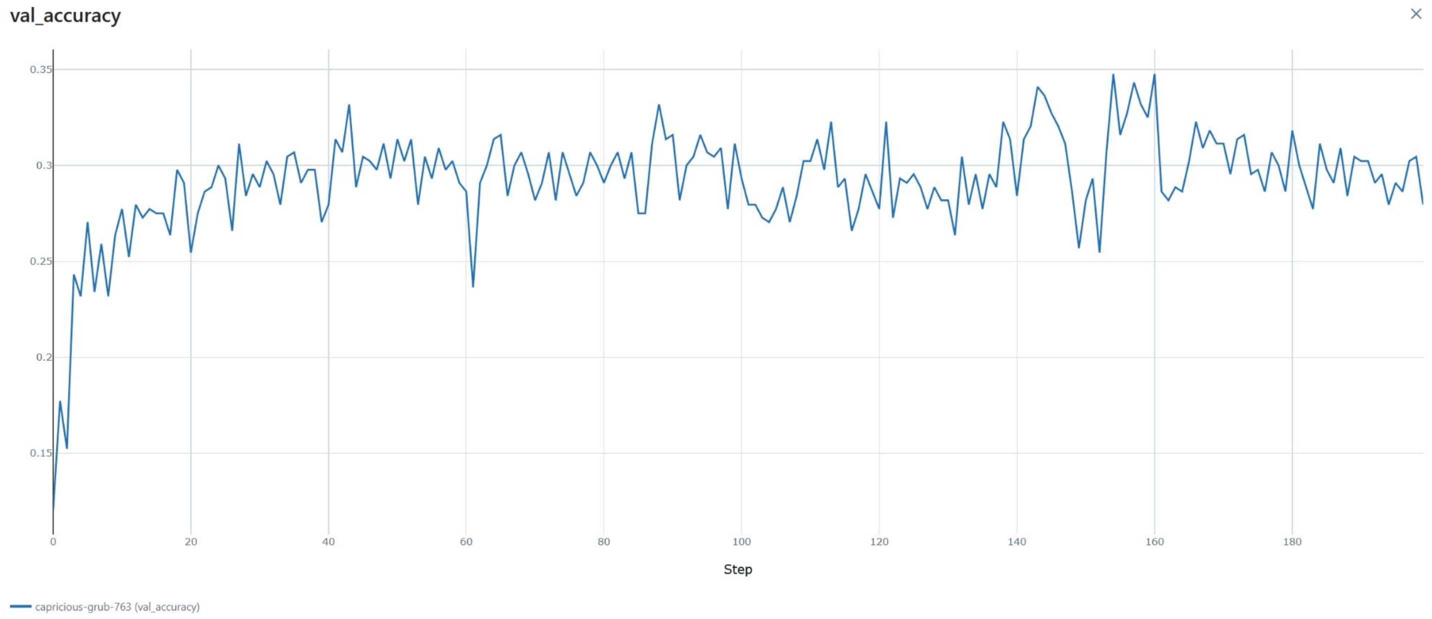
Performance Evaluation

- Train loss
    - Represents the error during training, helping to track convergence and detect overfitting.
    - The loss decreases rapidly, indicating successful learning.



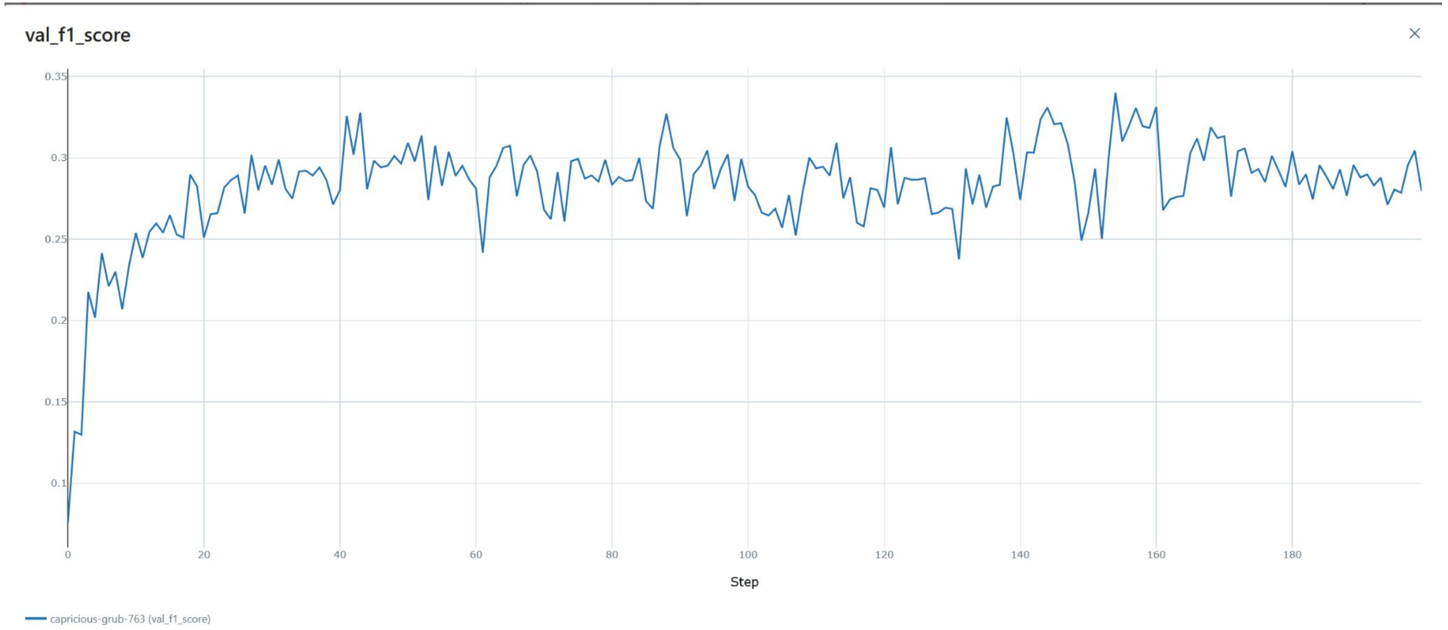
Performance Evaluation

- Validation accuracy
  - Evaluates performance on unseen data, reflecting generalization ability.
  - The curve fluctuates significantly around 30%, which suggests the model struggles to generalize.



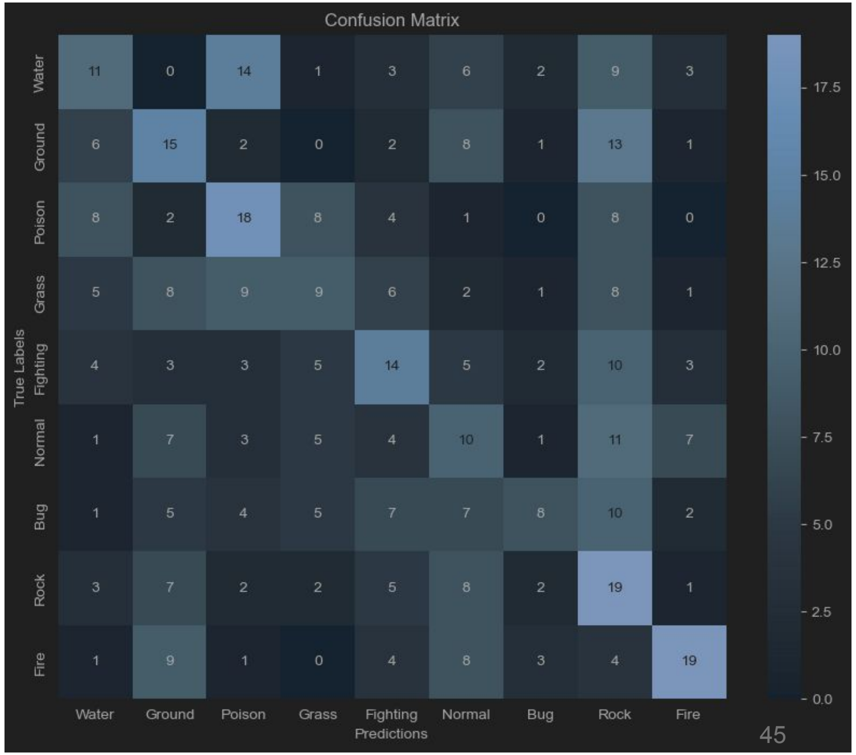
Performance Evaluation

- Macro F1 score
  - Balances precision and recall, crucial for handling class imbalances.
  - The high variance suggests unstable predictions across classes.



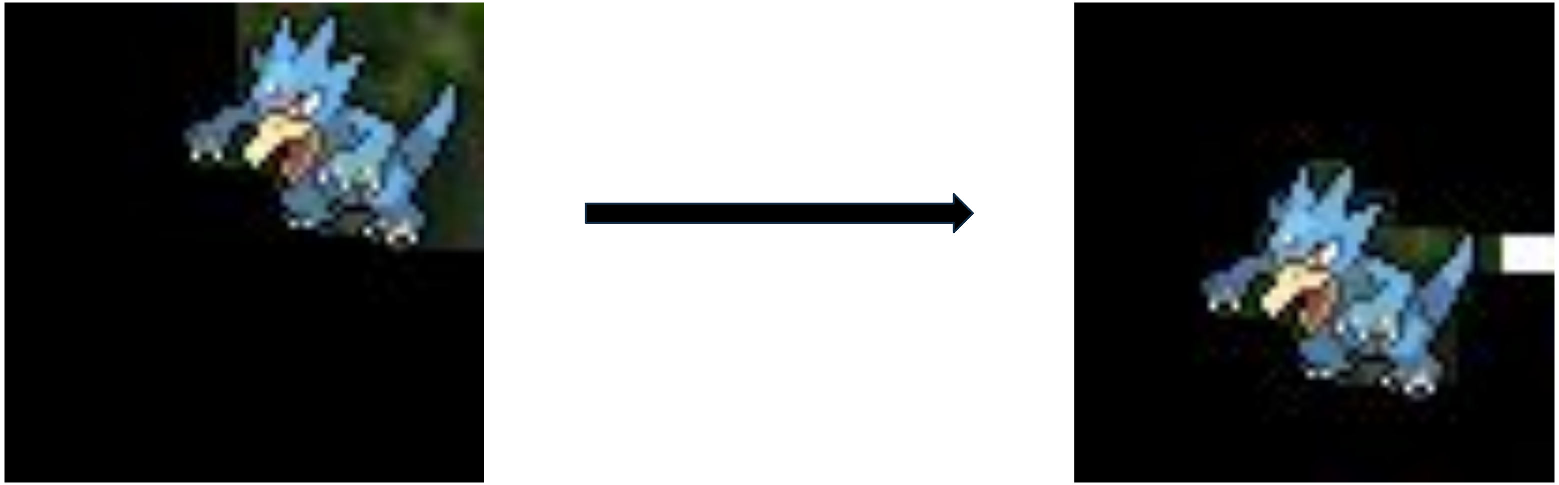
Performance Evaluation

- Confusion matrix
  - provides a detailed evaluation of the model's performance beyond a single accuracy score.
  - Strong performance on Poison (18), Fighting (14), Rock (19), and Fire (19), indicating the model learns distinct features for these types.
  - Frequent misclassifications:
    - Water → Poison (14x) and Ground → Rock (13x) suggest visual similarities.
    - Grass → Poison and Fighting ↔ Bug/Rock indicate shared features.
    - Normal is scattered, showing weak representation.
  - Improvements: Better feature extraction, class balancing, and data augmentation could enhance performance.



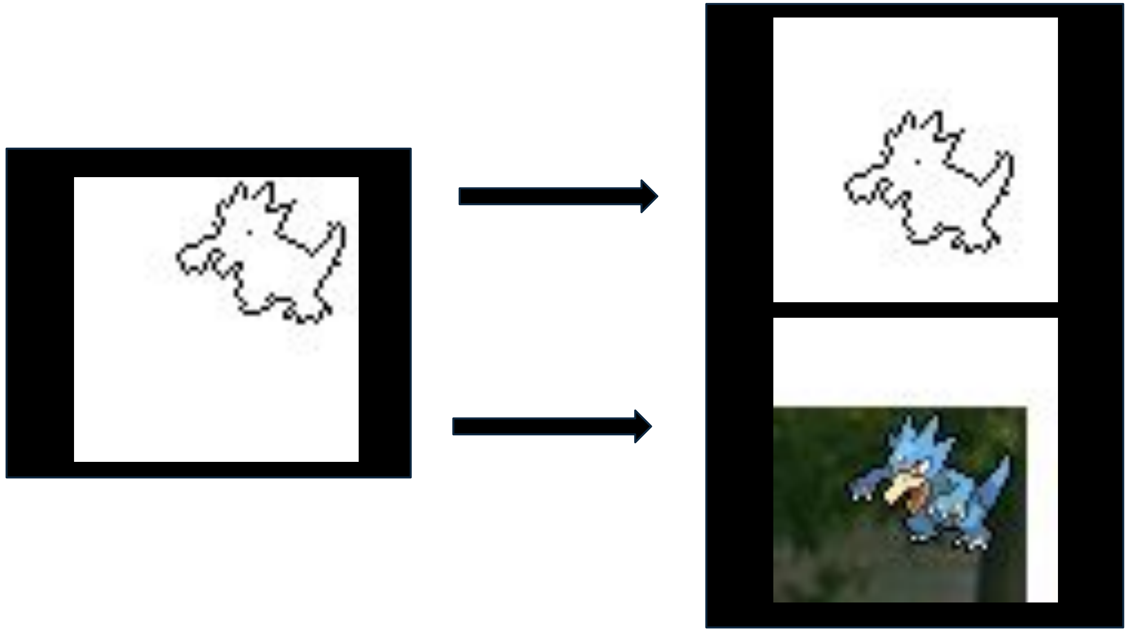
Breakthrough: centering the Pokemons

We achieve to **center** the Pokemon in addition to removing the background.



Centering the Pokemon

We used the thresholded image to locate and get the dimension of the Pokemon easily. Basically we “draw” a rectangle around the Pokemon.With this we know exactly what part we wanted to move on the picture. Once we have this, the process is the same to remove the background

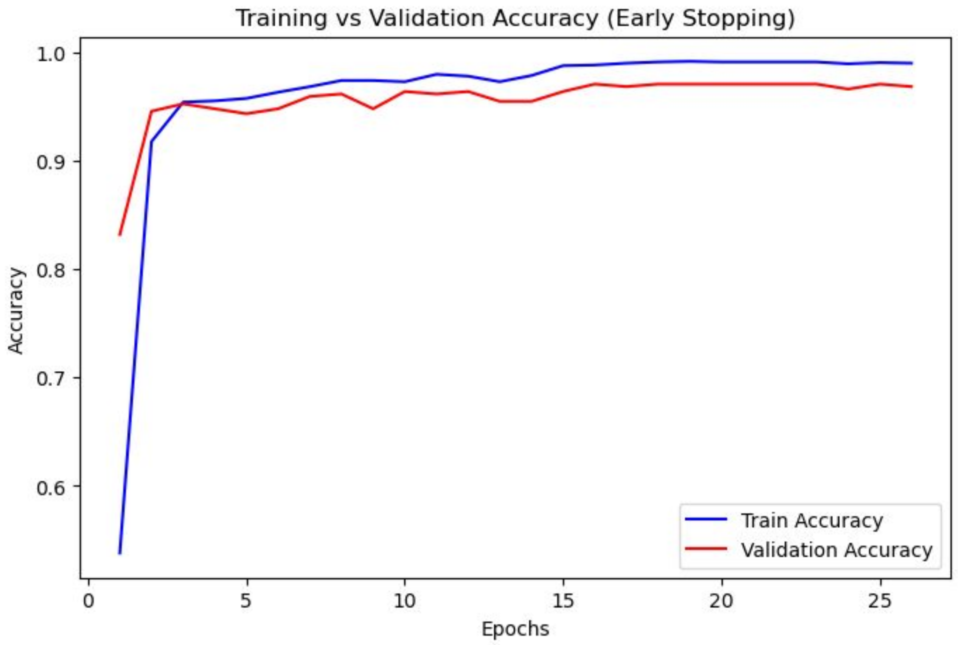


Training efficiency

As the validation accuracy

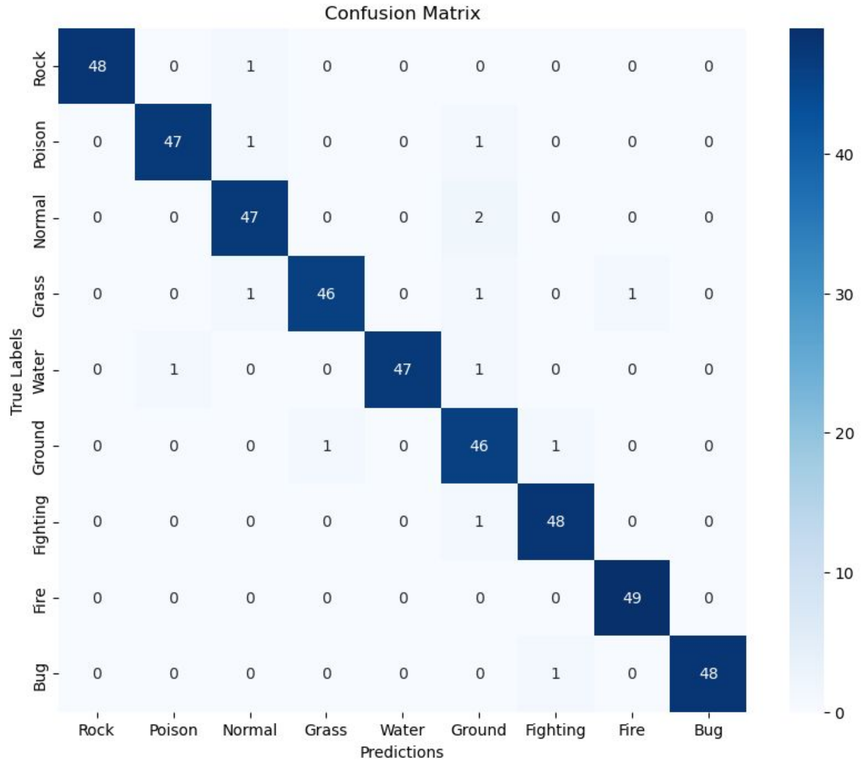
converged even more aggressively, the training stopped even earlier than before.

The training took only 26 epochs which represent 18sec using gpu and early stopping



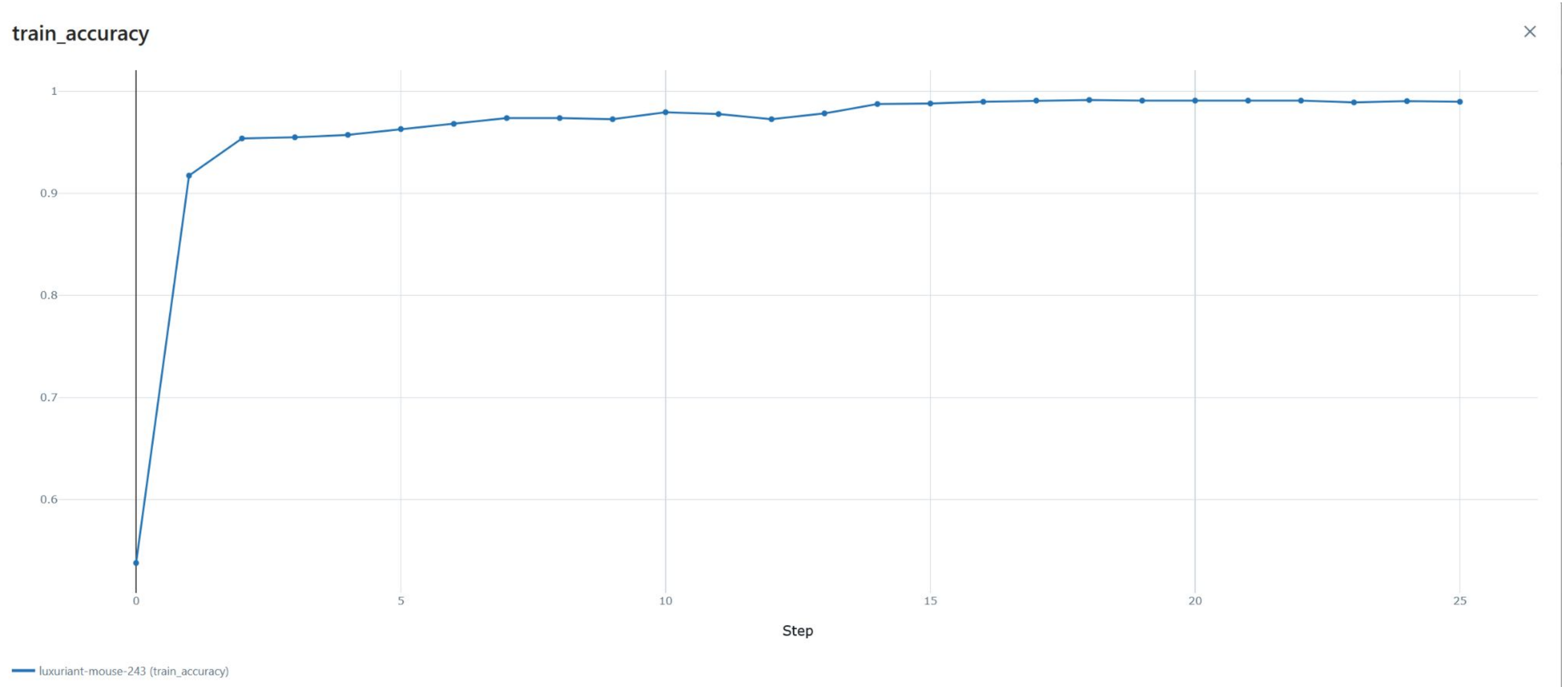
Performance evaluation

The new **confusion matrix**:

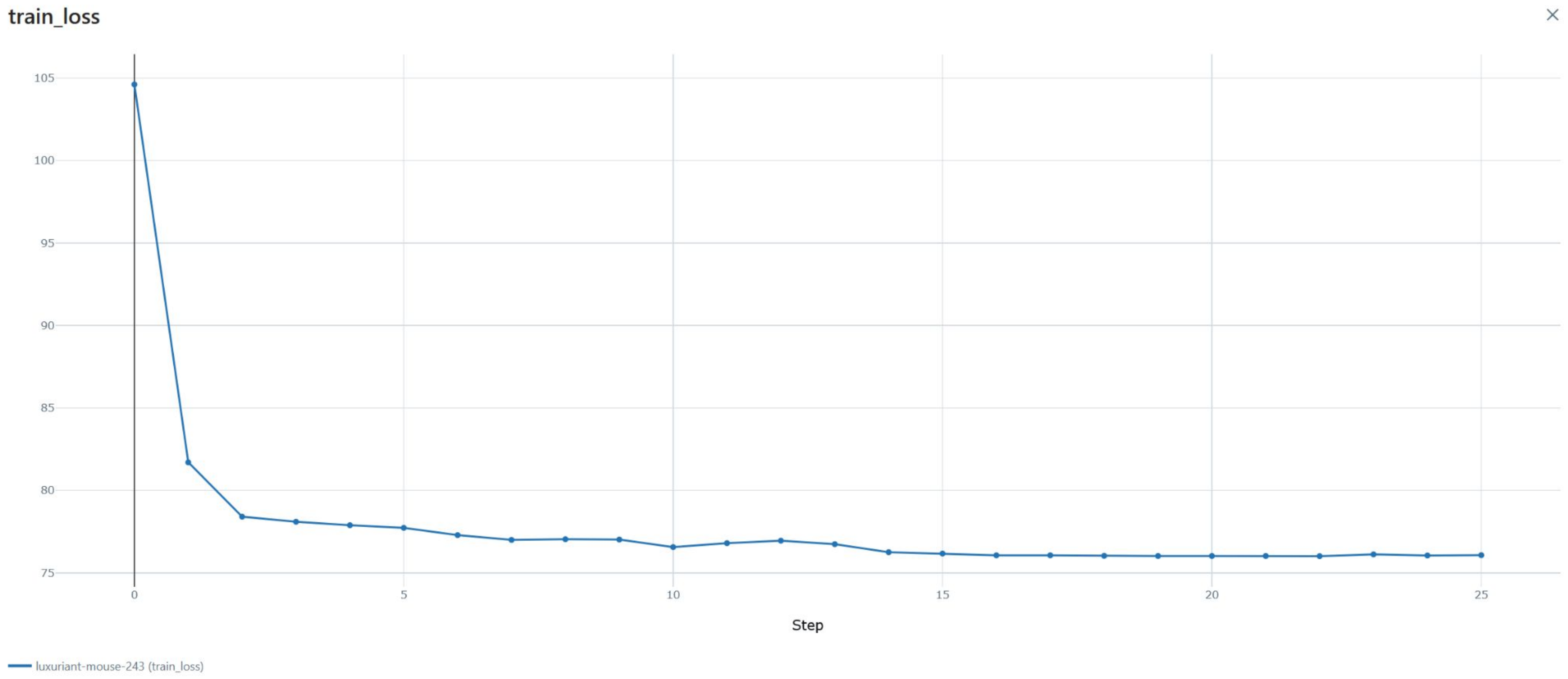


True Labels \ Predictions	Rock	Poison	Normal	Grass	Water	Ground	Fighting	Fire	Bug
Rock	48	0	1	0	0	0	0	0	0
Poison	0	47	1	0	0	1	0	0	0
Normal	0	0	47	0	0	2	0	0	0
Grass	0	0	1	46	0	1	0	1	0
Water	0	1	0	0	47	1	0	0	0
Ground	0	0	0	1	0	46	1	0	0
Fighting	0	0	0	0	0	1	48	0	0
Fire	0	0	0	0	0	0	0	49	0
Bug	0	0	0	0	0	0	1	0	48

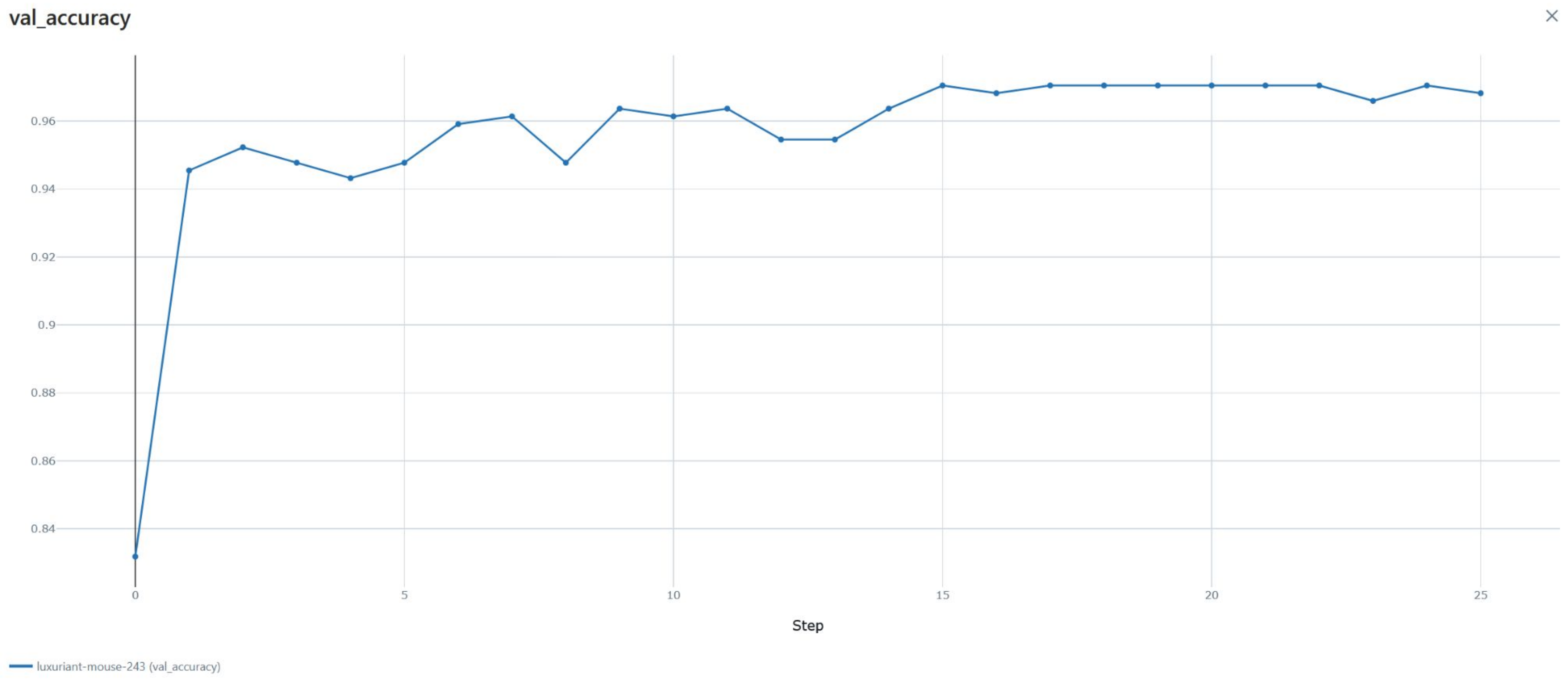
Performance Evaluation



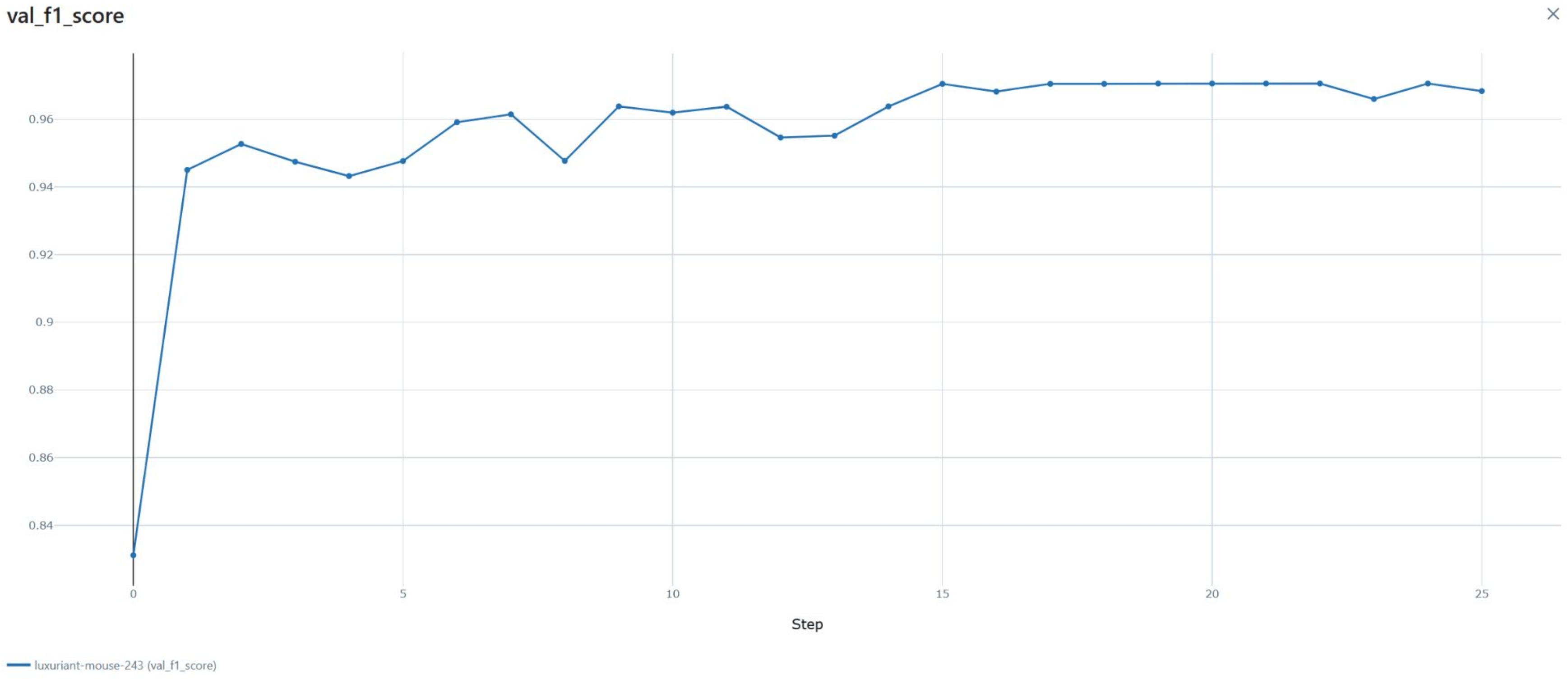
Performance Evaluation



Performance Evaluation



Performance Evaluation



Lessons learned and insights gained

- Even though by centering the Pokemon, the MLP was actually pretty accurate, we can still do better using a **CNN** model.
- **Preprocessing** correctly the data is the key to get **good results**.

What went wrong

- Installing MLFlow locally didn't work, so we had to do a free trial on Databricks.
- It takes a long time for MLFlow to work in all environments.

What went great

- Data exploration;
- Teamwork;
- Comprehension of how MLP work;
- Image preprocessing.