Pokémon Image Classification Challenge

Task 2

Team LLMH

Basics of Task 2

- The team
  - GALLO Lorenzo 72719
  - LEKBOURI Lina 72697
  - LICHTNER Marc 72690
  - WERCK Hugo 72692
- The work
  - Best public score: 0.959
  - Best private score: 0.97
  - Leaderboard position in the private leaderboard: 16

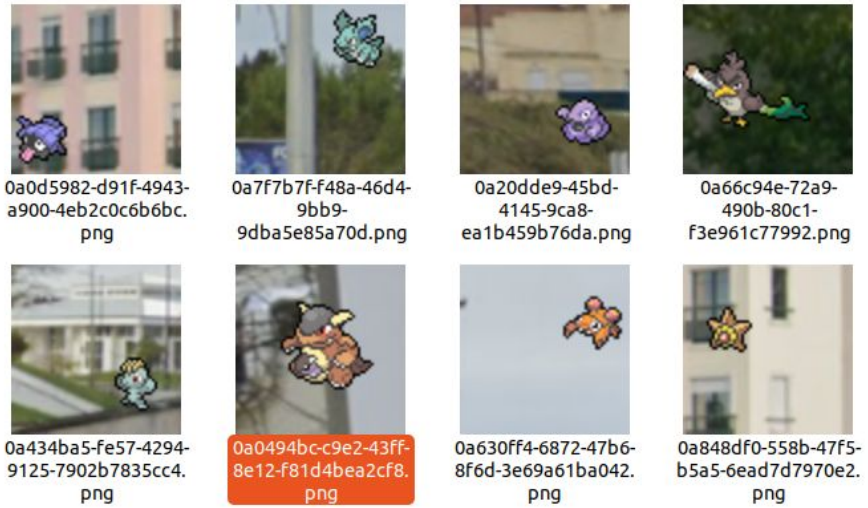
Task 1: Multilayer Perceptron (MLP) Classification

Feedbacks and summaries

Data Exploration: Summary

- **Dataset look:**

Train and Test images folder



train_labels.csv

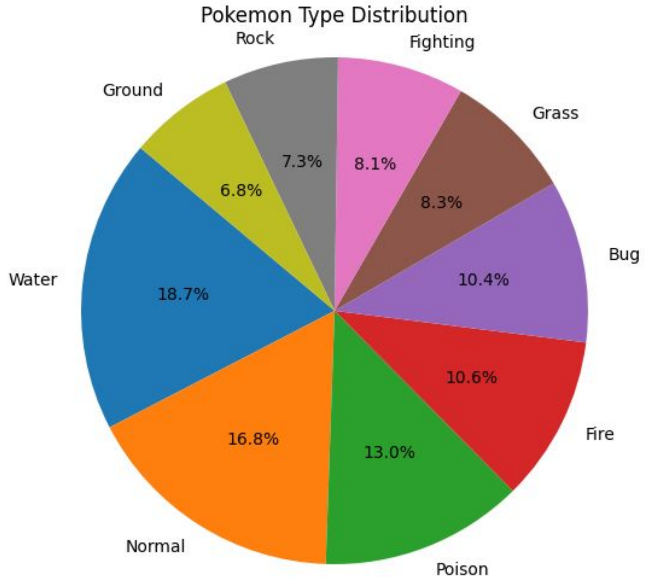
	Id	label
0	6fc9045b-9983-41e2-be2d-8796ecd97412	Normal
1	874716ce-9048-4e8a-b980-5ed9a5c0110e	Poison
2	c3613b20-ead8-48e1-8c8d-2f219d8e19d4	Normal
3	c7264ebc-ba44-460a-9b2b-df23c04783bc	Normal
4	a72045db-8fae-458b-993e-23d2aab1a5c6	Normal

Data Exploration: Summary

- **Global information and class distribution:**

  - 3600 in the Train folder
  - 9 labels
  - 0 duplicates
  - size of each images: 64*64
  - background: nothing to do with the pokémon

class distribution



⇒ **imbalance**

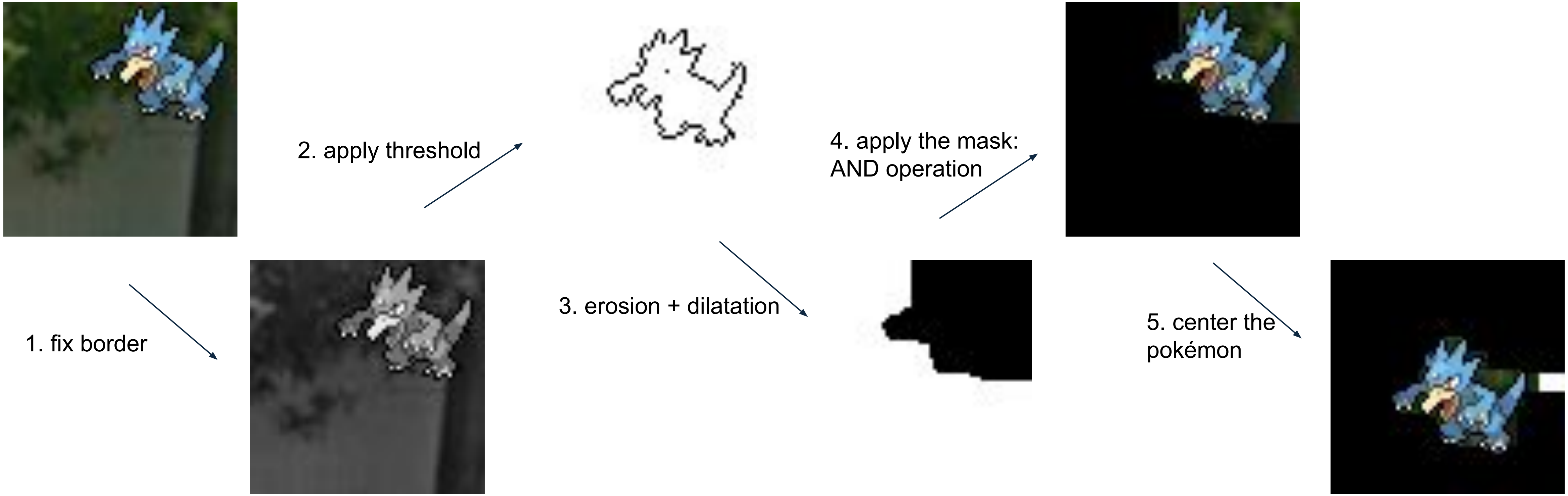
Data Preprocessing: Summary

- **Balancing the data:** undersampling

Data Preprocessing: Summary

- **Removing the background + centering the pokémon**

⇒ to help the model to focus more on the pokémon itself



Model Development

- MLP architecture diagram
- Justification for chosen architecture
- Ablation study

Model Development

- **MLP architecture diagram:**

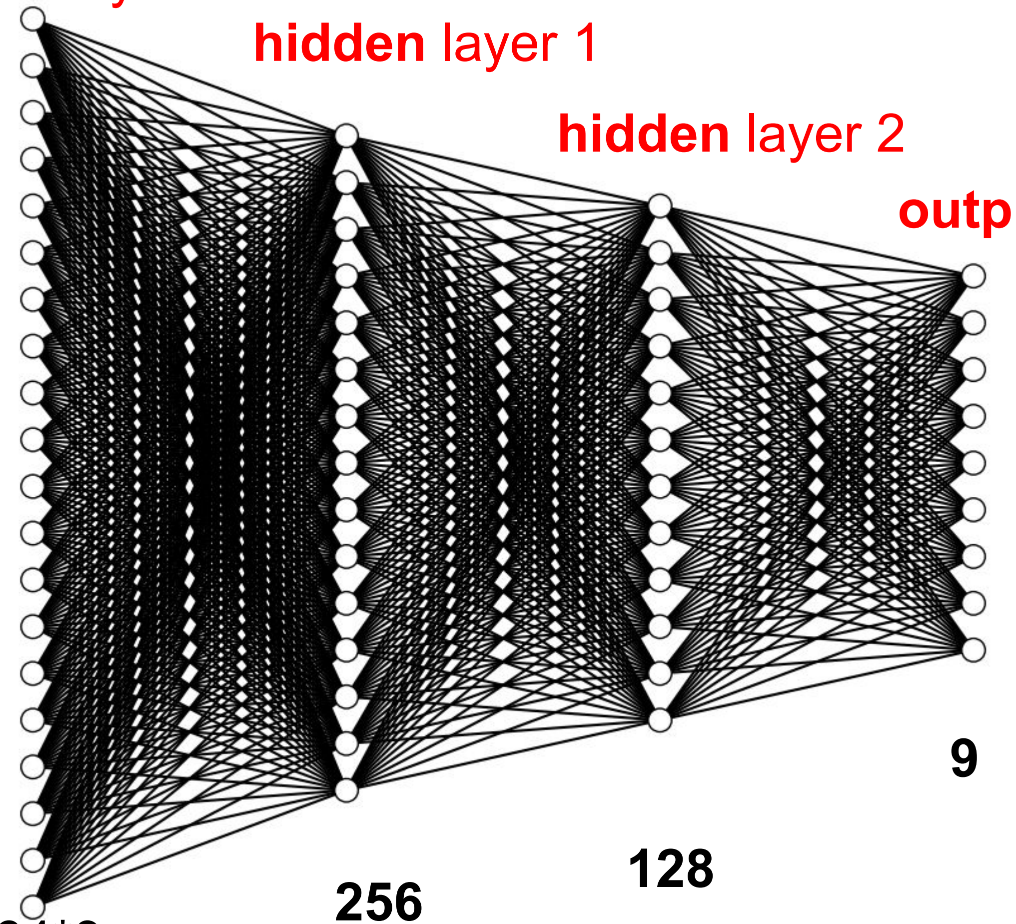
(Given the big numbers of neurons, we can not represent the real numbers on this slide, so we take random numbers of neurons respecting orders of magnitude.)

input layer

hidden layer 1

hidden layer 2

output layer



number of neurons:

64*64*3
= **12288**

256

128

9

Model Development

- **Justification for chosen architecture**

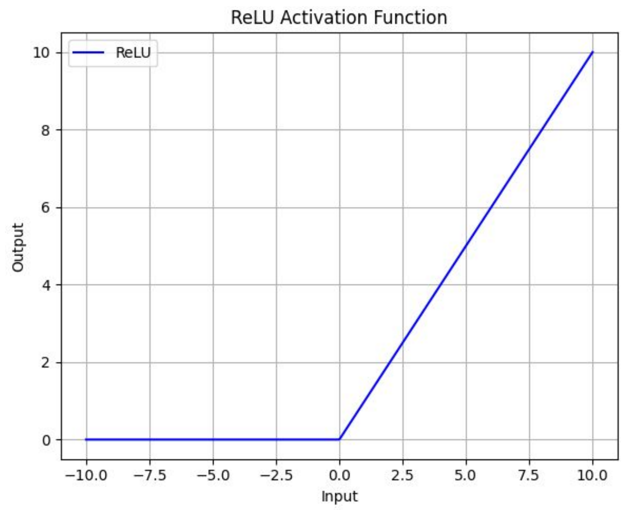
4 layers:

- 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).

The hidden architecture giving the best result after testing.

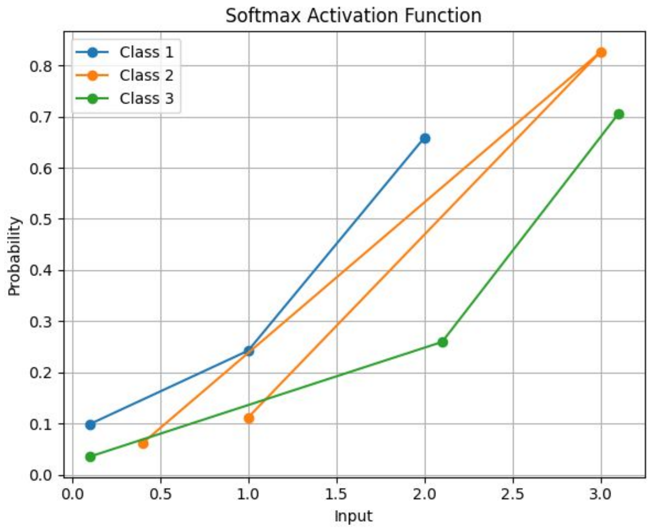
Model Development

- **Justification for activations**



Activation functions:

- **ReLU** (from input to h1 and from h1 to h2): efficient and improve generalization (avoid gradient vanishing).
- **Softmax** (from h2 to output): Converts raw scores into probabilities, making the model interpretable.



Model Development

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

Training Efficiency

- Training time
- GPU usage
- Strategies employed for efficient training.

Training Efficiency

Training time

As we converged rapidly to a maximum. We were able to complete the training within a **couple of minutes**. (local run)

Using the GPU and using early stopping, the training was completed within **24 seconds**.

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

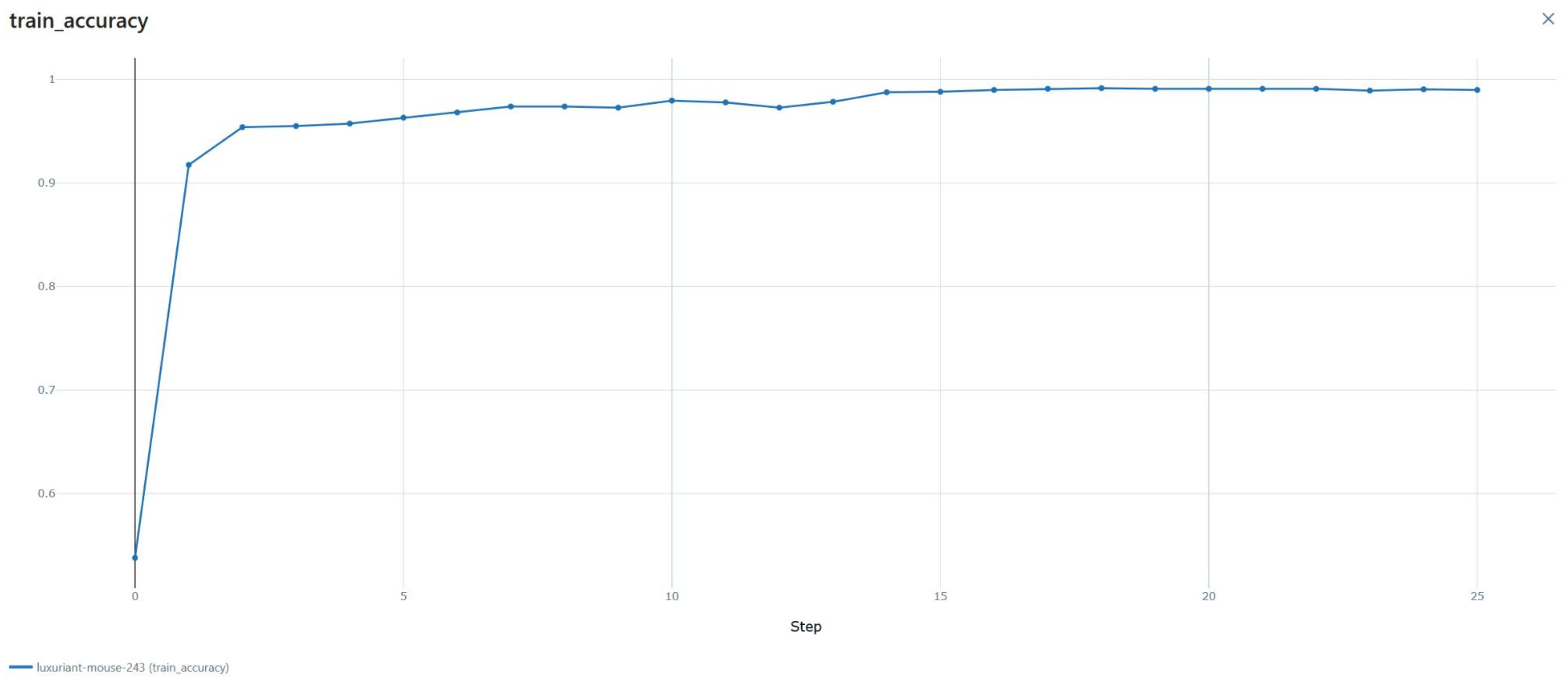
- **Batch size**: In order to select the batch sizes, we followed the “linear scaling rule”. We chose a batch size of 32 and a learning rate of 0.001.

Performance Evaluation

- Justification of the metrics used
- Interpretation of results

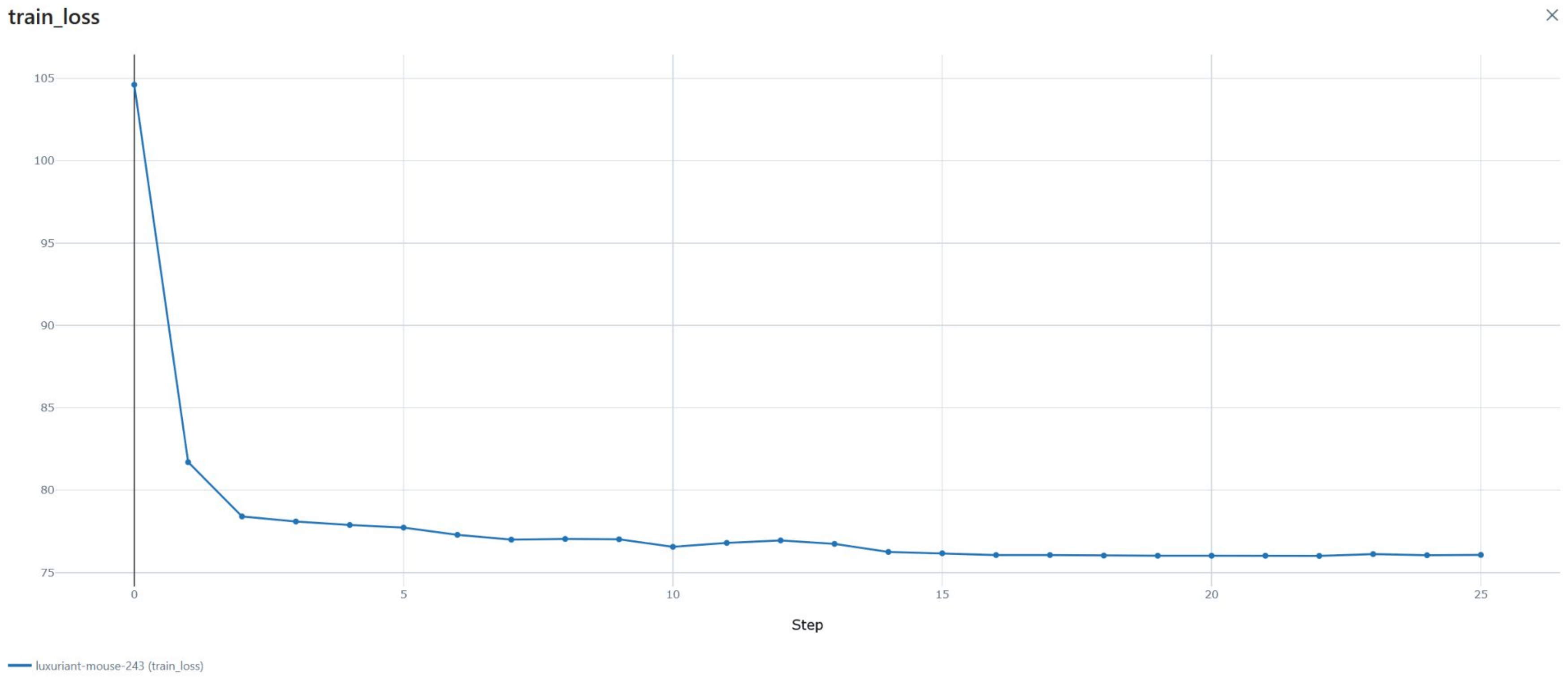
Performance Evaluation

- **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 **95%** (against 85% without centering). This suggests the model is learning very effectively from the training data, images being centered.



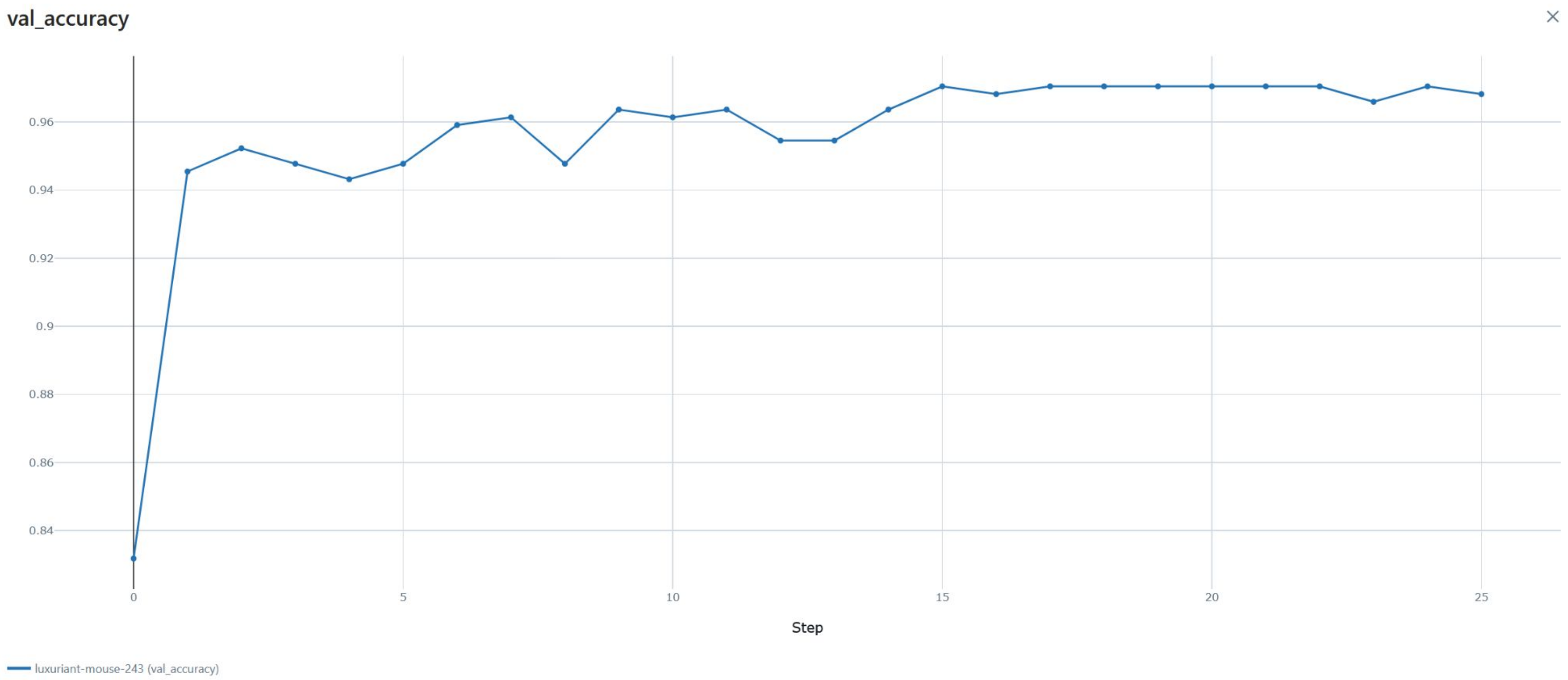
Performance Evaluation

- **Train loss**
  - Represents the error during training, helping to track convergence and detect overfitting.
  - The loss decreases rapidly at **76%** (against 82% without centering), indicating successful learning.



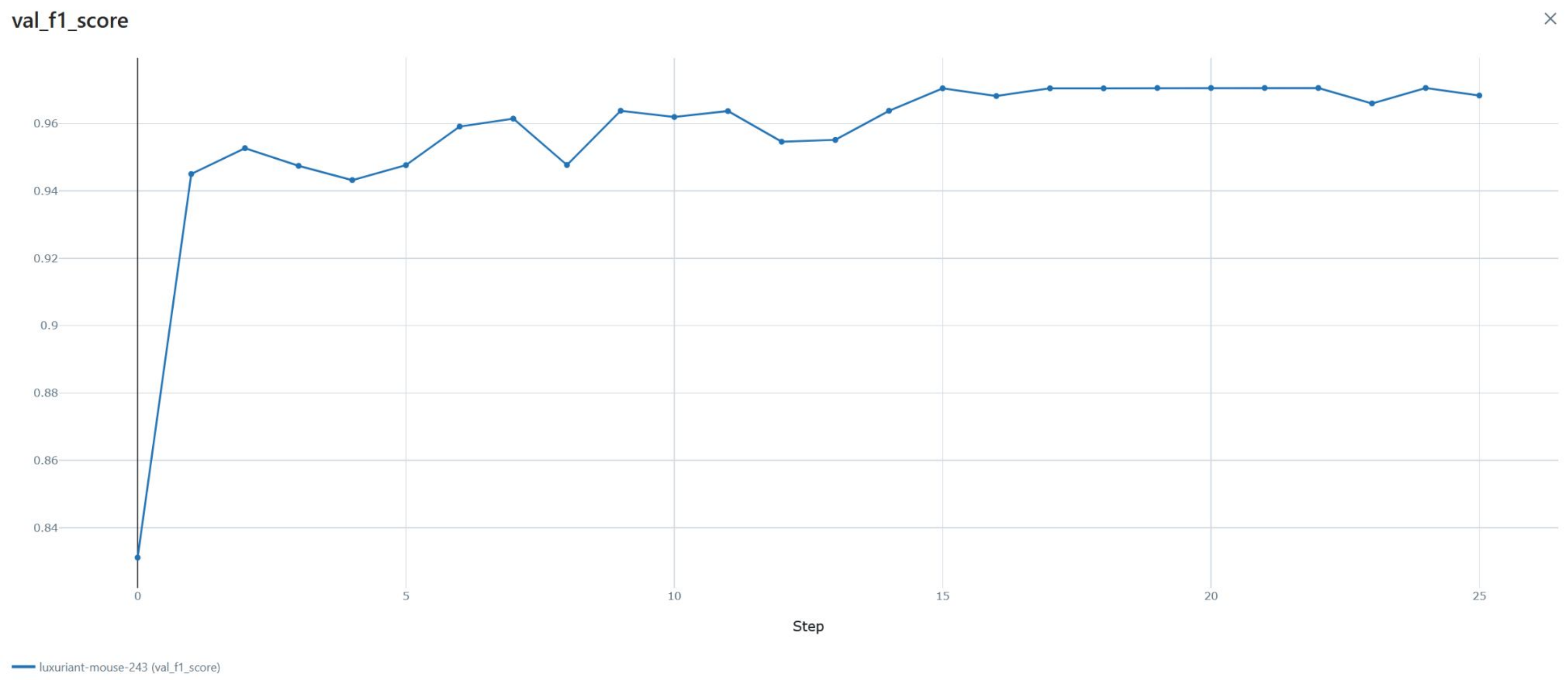
Performance Evaluation

- **Validation accuracy**
  - Evaluates performance on unseen data, reflecting generalization ability.
  - The curve fluctuates around **96%** (against 30% without centering !) which suggests the model generalize well with that improvement.



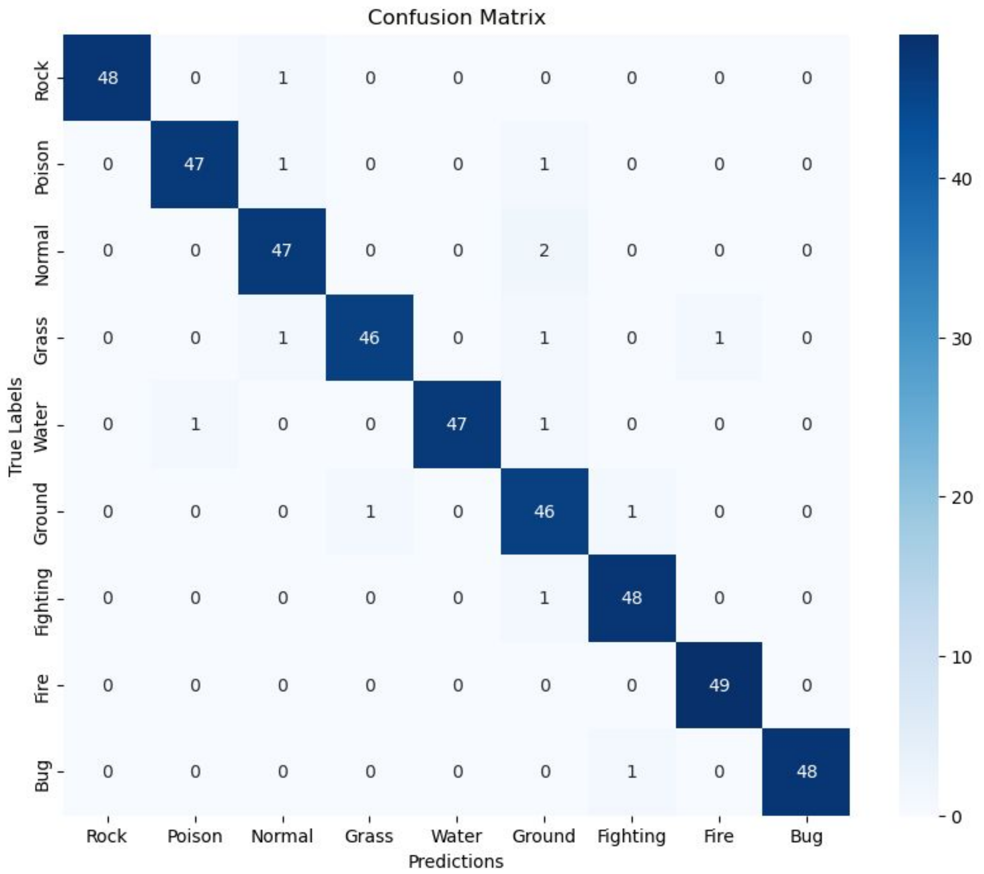
Performance Evaluation

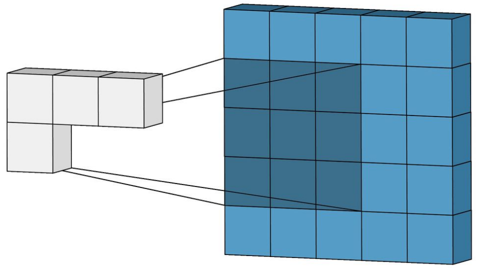
- **Macro F1-score**
  - Balances precision and recall, crucial for handling class imbalances.
  - The high variance suggests unstable predictions across classes.
  - **96%** now with less variance than before, and f1 at 30% without centering.



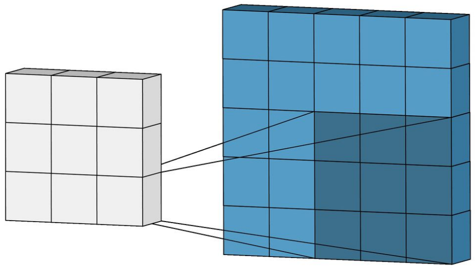
Performance Evaluation

- **Confusion matrix**
  - Provides a detailed evaluation of the model's performance beyond a single accuracy score.
  - Strong performance on every class, only few mistakes.



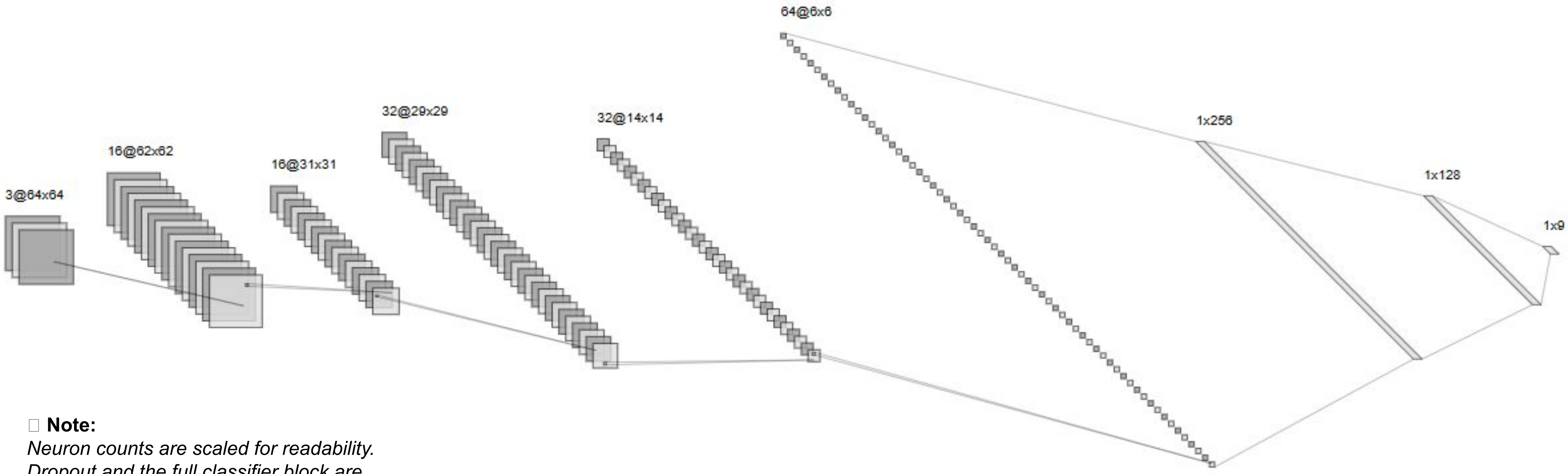


Task 2: Convolutional Neural Network (CNN) Development



CNN Model Development

- **CNN architecture diagram**



 Note:
*Neuron counts are scaled for readability. Dropout and the full classifier block are omitted from the diagram but are present in the **implemented model.***

CNN Model Development

• Justification for chosen architecture

6 layers (feature extraction):

- **Input layer:** 64×64 RGB image →**3 channels**, 64×64 pixels = **3@64×64**
  → the pokémon images are colored (RGB) 64*64 sized images.

- **Conv Layer 1 + ReLU:** 16 filters, 3×3 kernel→ output: **16@62×62**
  → Extracts low-level features such as edges, corners, and textures, using the ReLU.

- **Max Pool 1:** 2×2→ output: **16@31×31**
  →Reduces the spatial dimensions (by half) while preserving the most important features.

- **Conv Layer 2 + ReLU:** 32 filters, 3×3 → output: **32@29×29**
  →Extracts more abstract and complex features such as shapes and textures by learning from the low-level features provided by the first convolutional layer.

- **Max Pool 2:** 2×2 → output: **32@14×14**
  →Further reduces spatial dimensions to retain only the most important features and reduce the risk of overfitting.

- **Conv Layer 3 + ReLU:** 64 filters, 3×3 → output: **64@12×12**
  →Extracts high-level and more abstract features, such as object parts or larger shapes.

- **Max Pool 3:** 2×2→ output: **64@6×6**
  →Final reduction of spatial dimensions to prepare for the classification layers, it ensures that the most critical features are retained while reducing computation for the classifier.

CNN Model Development

•Justification for chosen architecture

Classifier layers (flatten + fully connected):

- **Flatten Layer**: 64×6×6 = 2,304 neurons
  →Converts the feature map from the last convolutional layer ((64 \times 6 \times 6)) into a 1D vector of size (2,304) neurons.
- **Fully connected layer 1**: 2,304 → 512 neurons *(with ReLU and Dropout)*
  →Reduces the dimensionality of the feature vector while preserving important information. (The **ReLU activation** enables the network to learn complex patterns in the data and the **Dropout layer** helps prevent overfitting by randomly deactivating neurons during training, improving generalization.)
- **Fully connected layer 2 (output)**: 512 → 9 neurons *(number of Pokémon types)*
  → Produces the final class scores, one for each Pokémon type, using the softmax activation function.

CNN Model Development

To justify our choice, we tried to use **different architectures** of CNN Model, but none of them give us a better result than the one with the architecture we choose.

CNN Model Development

- Trying out a different architecture:

LeNet architecture:

1. **Input:** images of size 64x64.

2. **Convolutional Layers:**

    - **Conv1:** 6 filters of size 5x5, output 6x28x28, activation **ReLU**.
    - **Pooling1:** Sub-sampling (pooling) 2x2, output 6x14x14.
    - **Conv2:** 16 filters of size 5x5, output 16x10x10, activation **ReLU**.
    - **Pooling2:** Sub-sampling 2x2, output 16x5x5.

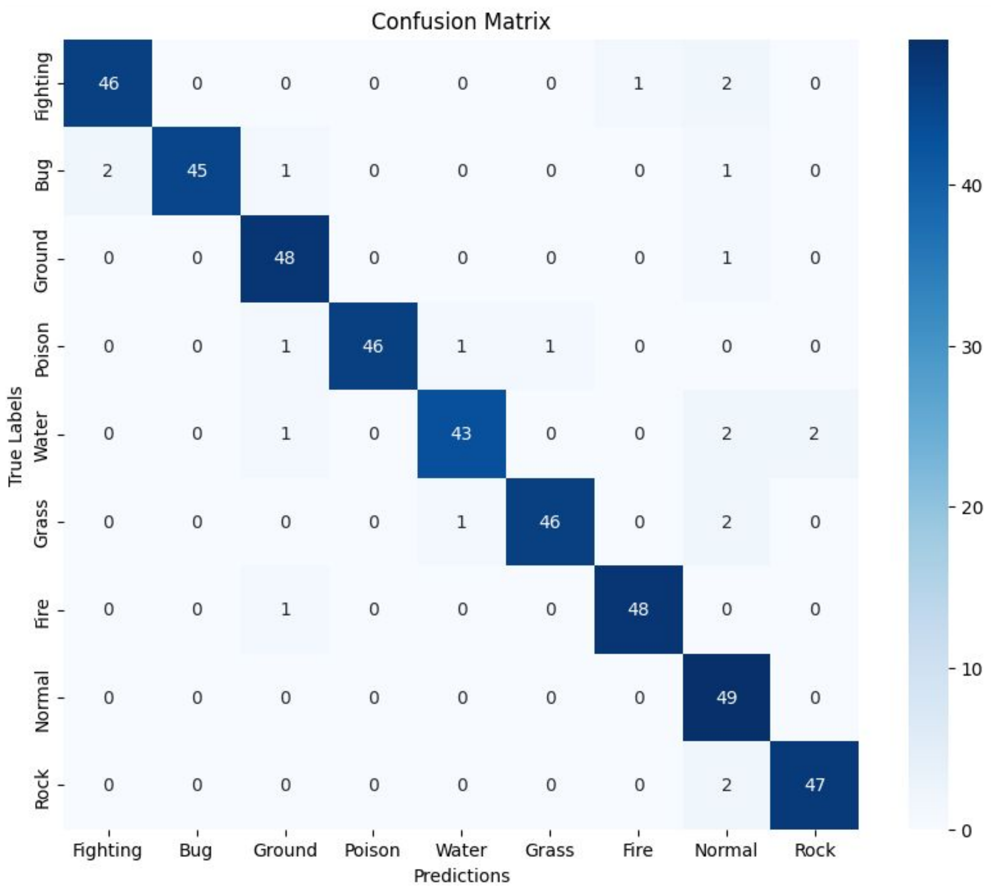
3. **Fully Connected Layers:**

    - **FC1:** 400 (16x5x5) → 120 neurons, activation **ReLU**.
    - **FC2:** 120 → 84 neurons, activation **ReLU**.
    - **FC3 (Output):** 84 → Number of classes

4. **Output:**

    - Class probabilities through **Softmax** activation.

F1-score obtained: 0.9548

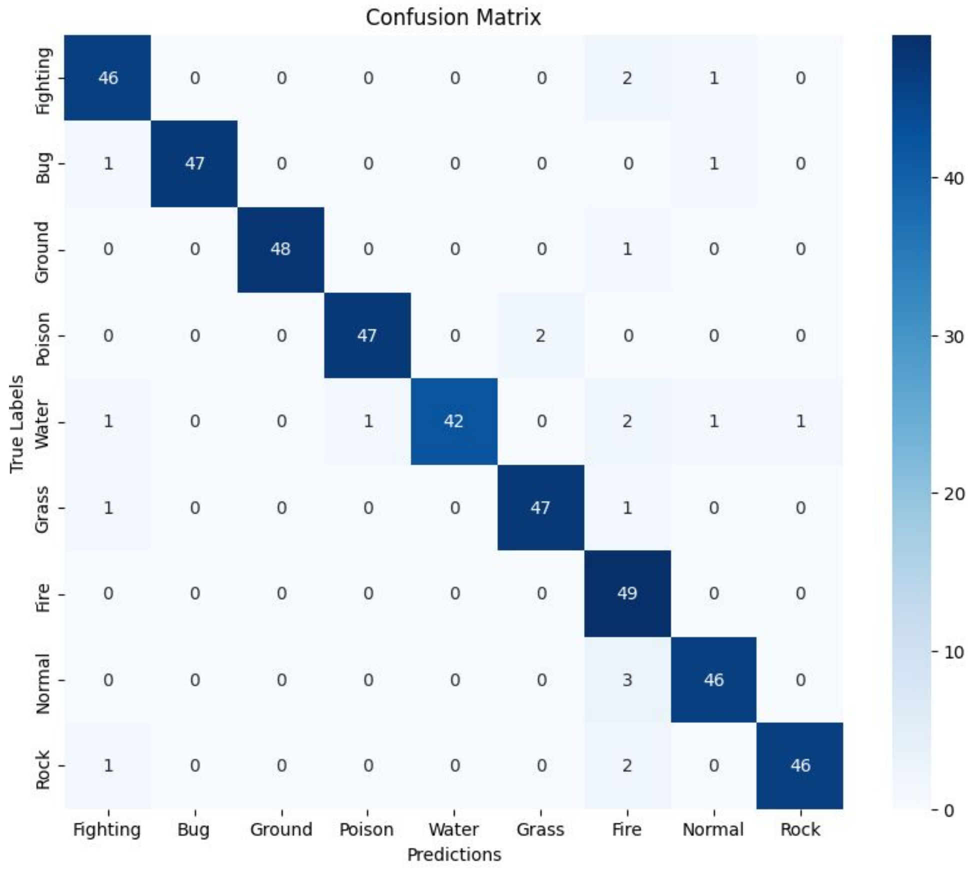


CNN Model Development

- Trying out a different architecture:
  Enhanced LeNet architecture

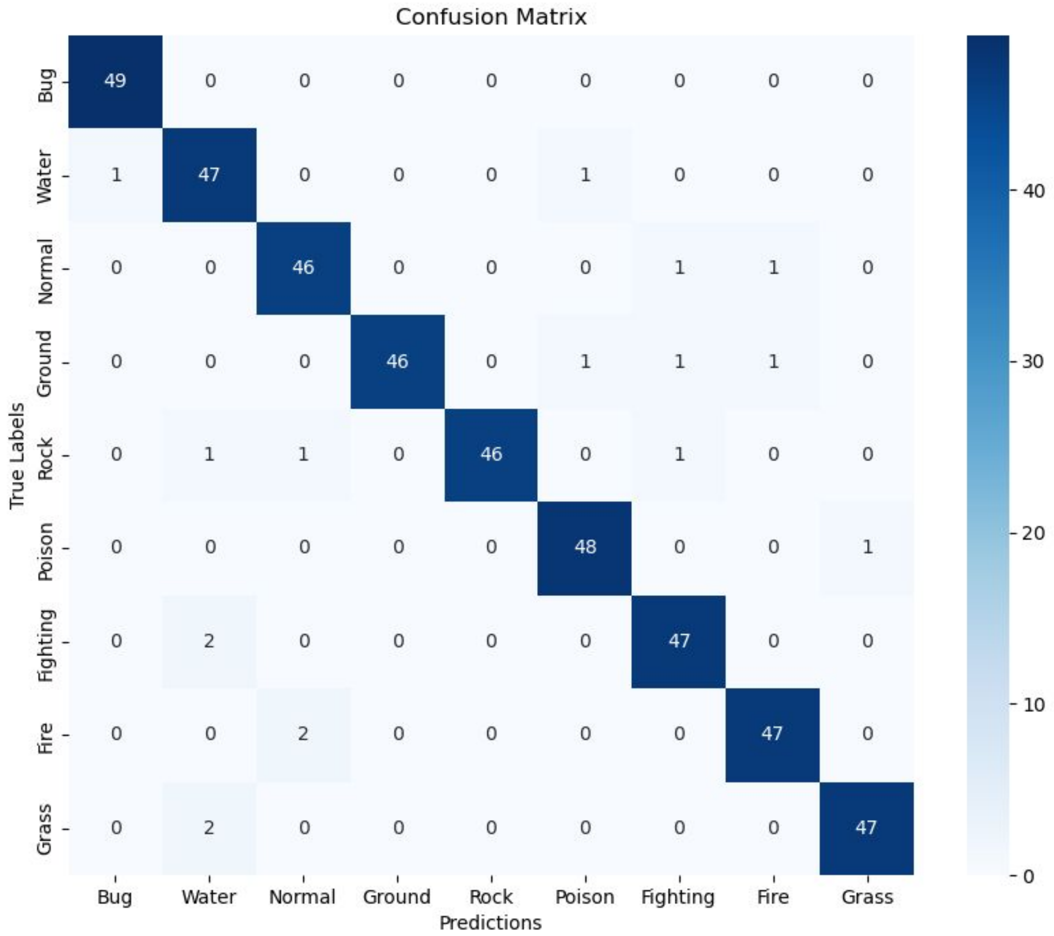
1. **Input:** Images with three color channels (RGB) of size 64 \times 64.
2. **Feature Extraction:**
   - **Conv1:** 3x3 kernel → Output: 16x62x62.
     Followed by **max pooling** → 16x31x31.
   - **Conv2:** 3x3 kernel → Output: 32x29x29.
     Followed by **max pooling** → 32x14x14.
   - **Conv3:** 3x3) kernel → Output: 64x12x12.
     Followed by **max pooling** → 64x6x6).
3. **Classification Head:** Fully connected layers for classification:
   - **Flatten:** Converts the feature map 64x6x6 into a flat vector of size 2304.
   - **FC1:** Fully connected layer with 2304→512 neurons. Activation: **ReLU**.
   - **Dropout:** Dropout with a probability of 0.5 for regularization.
   - **FC2:** Fully connected layer with 512→num_classes.
4. **Output:** The final layer outputs logits for each class, which can be converted to probabilities, using **softmax**.

F1-score obtained: 0.9507

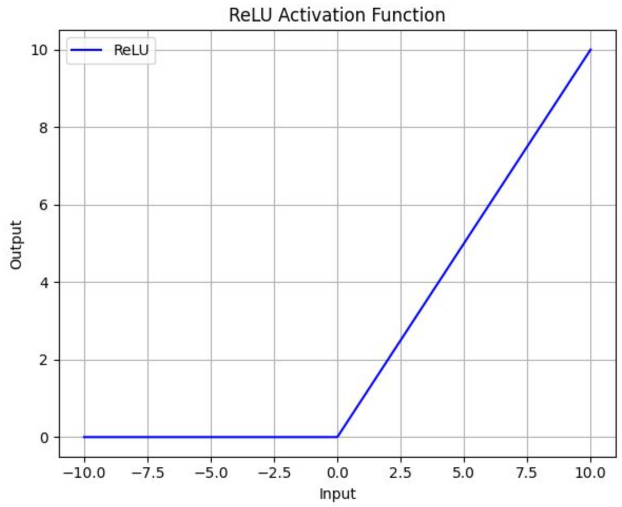


CNN Model Development

Result with the model we choose:



CNN Model Development



• Justification for activations

ReLU (Rectified Linear Unit) is used after each convolutional and fully connected layer because It introduces non-linearity, helping the network to learn complex patterns.

ReLU is:

- Computationally **efficient;**
- Helps prevent **vanishing gradients;**
- Activates only **positive values** → creates sparse representations.

No activation in output layer, because we use the **CrossEntropyLoss** → This function **internally applies Softmax**, which transforms logits into class probabilities, the most important part for **classification**.

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\\text{size}}} y_i \cdot \log \hat{y}_i$$

CNN Model Development

- **Ablation study:**

Key Experiments:

- **Remove Conv Layer 3:** Test the effect of fewer convolutional layers, analyze the importance of deeper convolutional layers in extracting high-level features.
- **Reduce Filters:** Analyze the impact of fewer filters in each layer.
- **Remove Max Pooling:** Replace pooling with strided convolutions.
- **Remove Dropout:** Evaluate the risk of **overfitting**.
- **Skip FC1:** Directly connect Flatten to output layer.
- **Change Activations:** Replace ReLU with LeakyReLU or ELU.

Every component contributes to **generalization and performance**.

The **full model with 3 conv layers, ReLU, dropout, and 2 FC layers** gave the **best results** on validation accuracy and confusion matrix.

Training Efficiency

- Training time
- GPU usage
- Techniques to address class imbalance and overfitting

Training Efficiency

GPU usage:

By using the GPU, it took **2min13** to compute **200** epochs whereas it took **4min46** 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

Training time

As we converged rapidly to a maximum. We were able to complete the training within **2m13**. (local run)

Using the GPU and using early stopping, the training was completed within **19s**.

Performance comparison (CNN vs MLP)

- Side-by-side performance metrics, with tables and plots
- Analysis of improvements and remaining challenges.

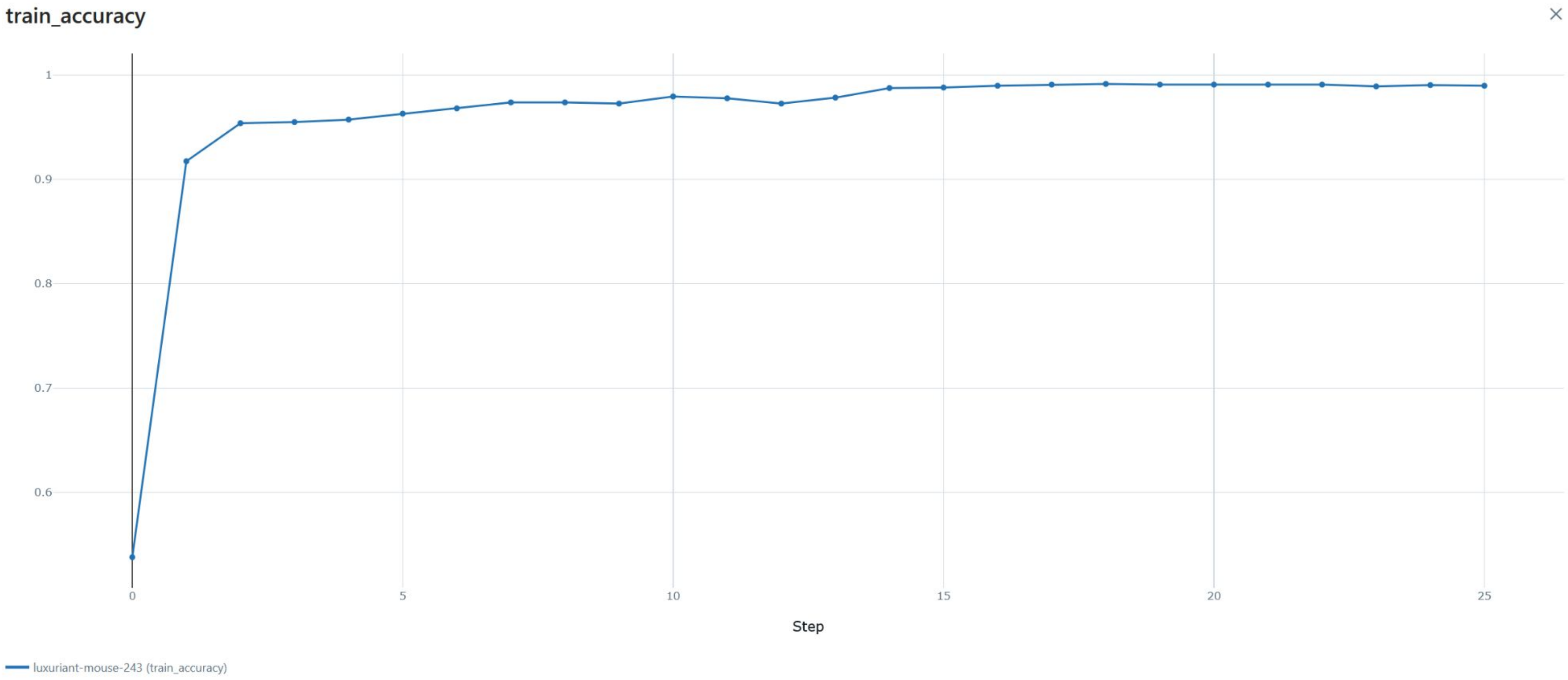
Performance comparison (CNN vs MLP)

Training accuracy :

- The MLP achieves a **high validation accuracy**, demonstrating its ability to generalize well to unseen data when benefiting from effective preprocessing techniques. This indicates that the simplifications introduced during **preprocessing** help the MLP perform competitively on validation datasets.
- The CNN shows a **slightly higher validation** accuracy compared to the MLP. This suggests that the CNN's architecture, designed to capture spatial hierarchies in image data, provides a marginal advantage in generalization, even when preprocessing benefits the MLP significantly.

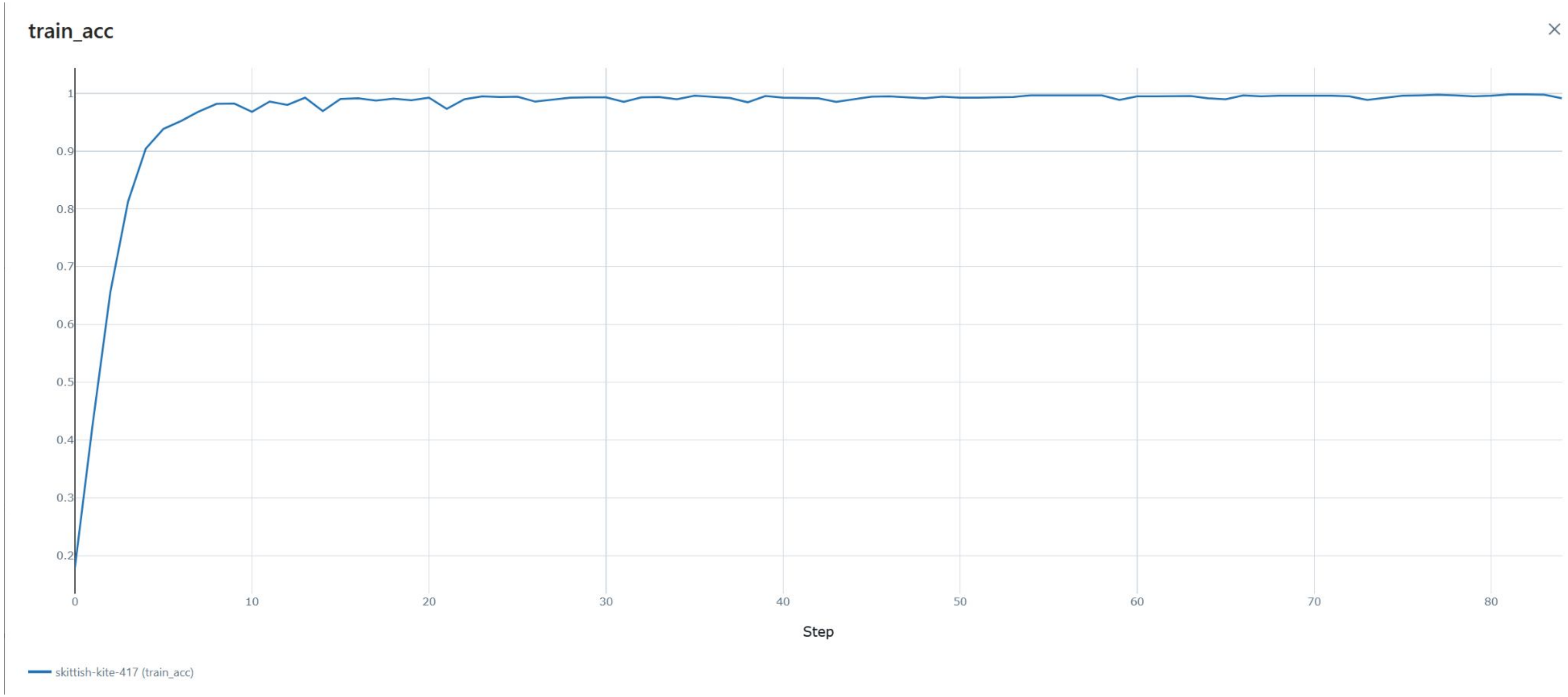
Performance comparison (CNN vs MLP)

Training accuracy of the **MLP**:



Performance comparison (CNN vs MLP)

Training accuracy of the **CNN**:



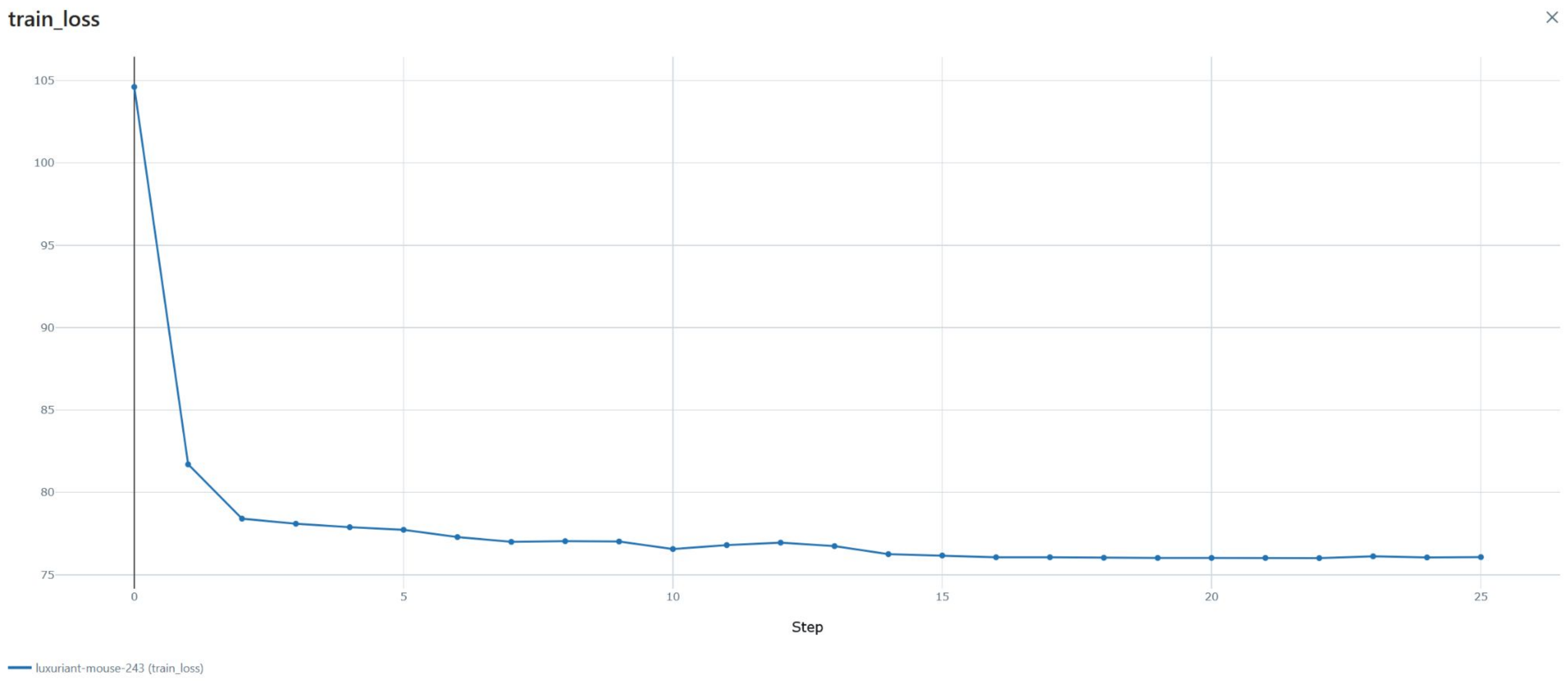
Performance comparison (CNN vs MLP)

Training Loss:

- The MLP starts with a **high training loss** but rapidly decreases to **stabilize** around 76%. This indicates that while the MLP can reduce errors significantly, it may still have difficulty minimizing the loss further, potentially due to its simpler architecture or overfitting tendencies.
- In contrast, the CNN demonstrates a **sharp decrease in training loss**, stabilizing at a much lower value of around 1%. This suggests that the CNN is more effective at minimizing errors on the training data, benefiting from its ability to learn complex patterns and features inherent in image data.

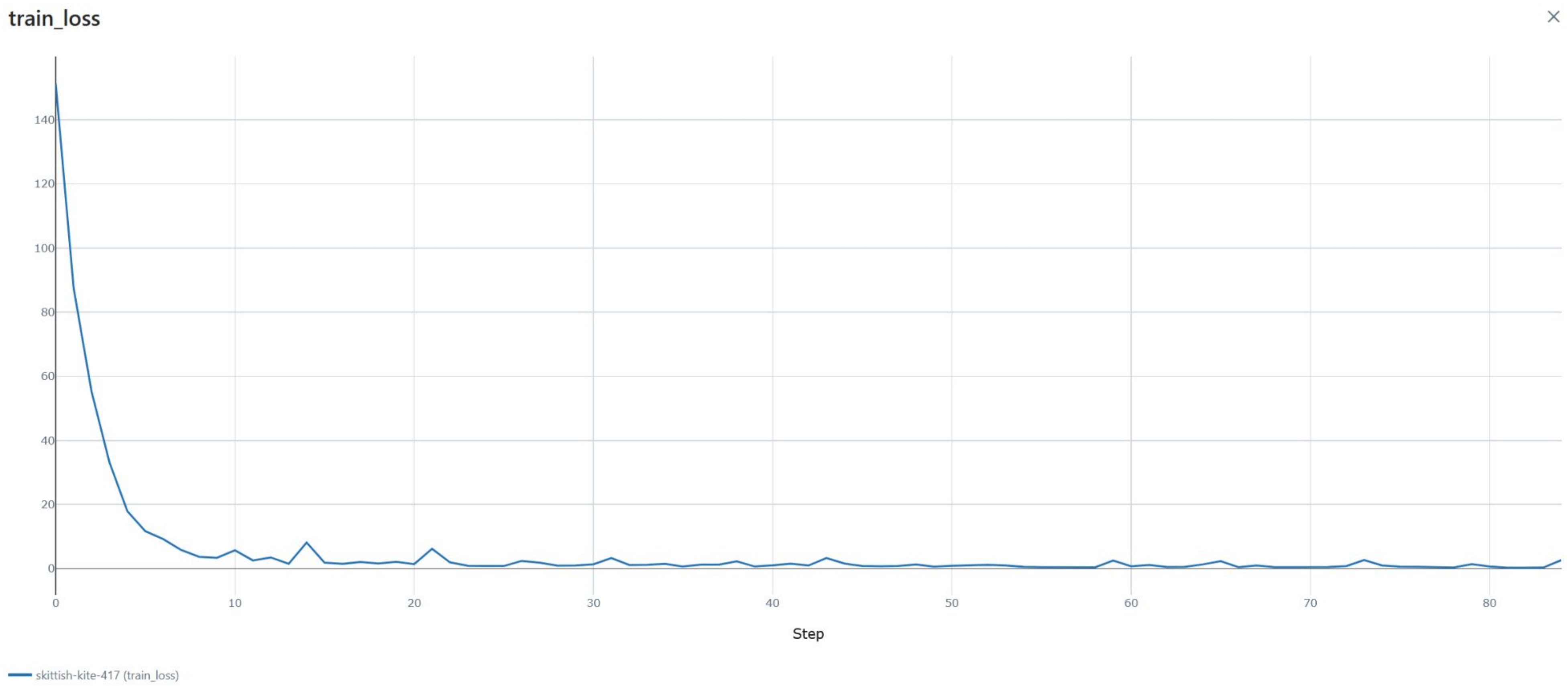
Performance comparison (CNN vs MLP)

Training loss of the **MLP**:



Performance comparison (CNN vs MLP)

Training loss of the **CNN**:



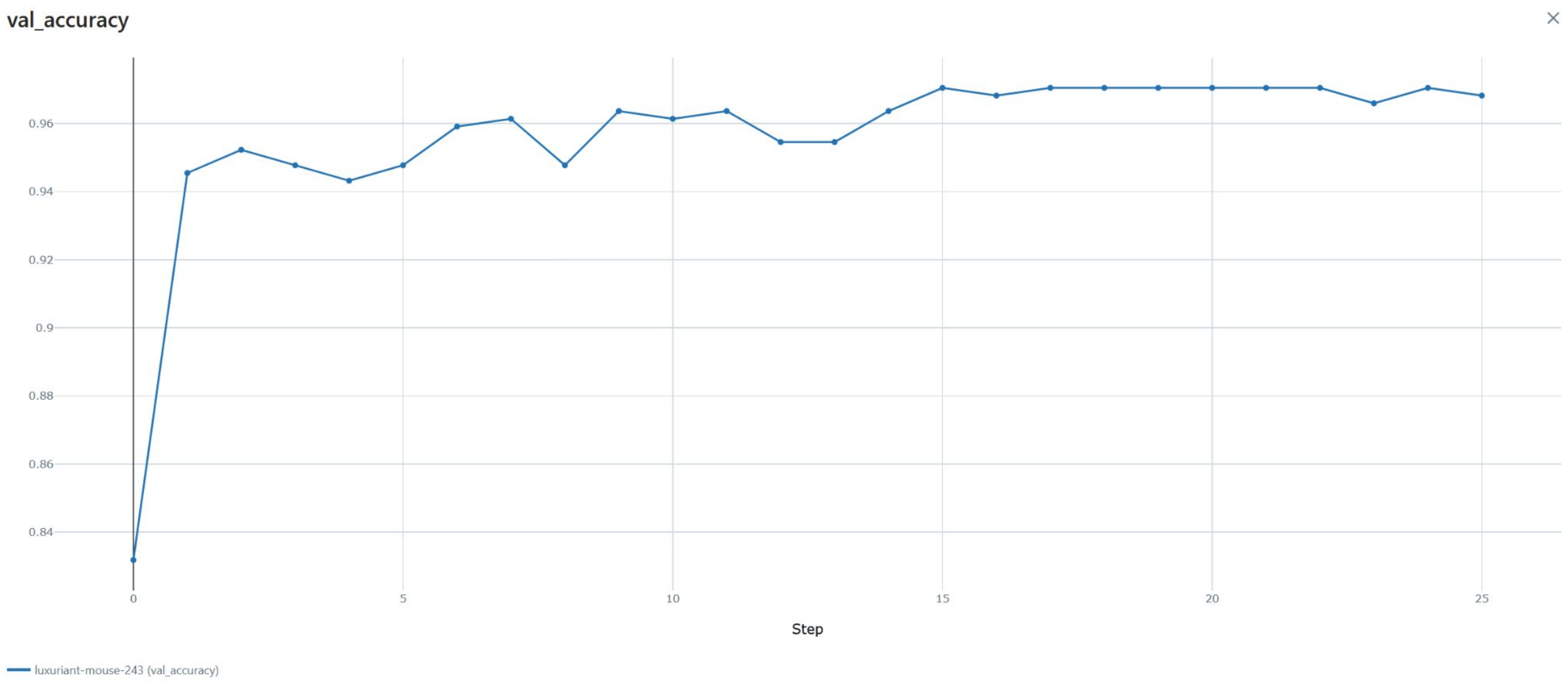
Performance comparison (CNN vs MLP)

Validation Accuracy:

- The MLP achieves a **high validation accuracy**, demonstrating its ability to generalize well to unseen data when aided by effective **preprocessing** techniques. The rapid stabilization suggests that the simplified input data allows the MLP to maintain **consistent performance** across both training and validation datasets.
- The CNN exhibits a **slightly higher validation accuracy** compared to the MLP, indicating its superior capability to generalize. This reflects the CNN's **strength** in **capturing intricate patterns and spatial hierarchies** in image data, even when preprocessing is beneficial to both models.

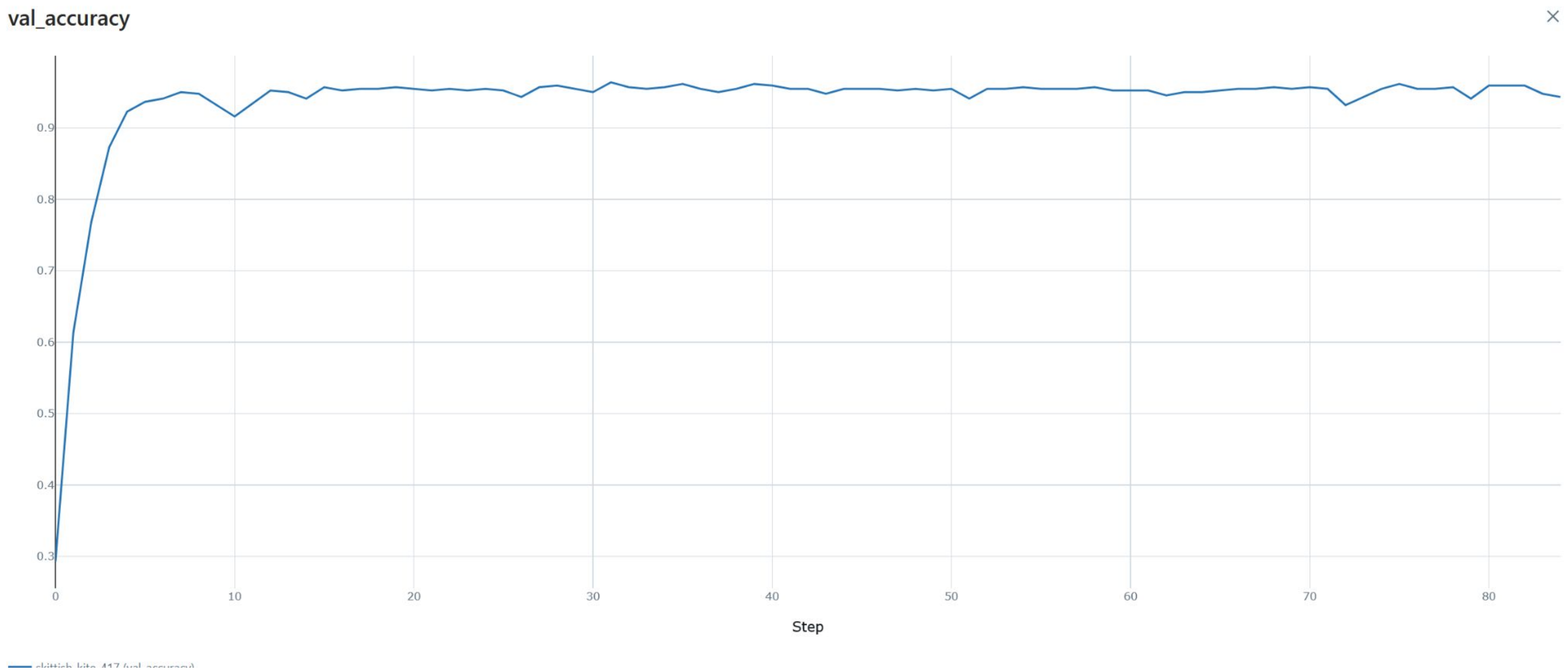
Performance comparison (CNN vs MLP)

Validation accuracy of the **MLP**:



Performance comparison (CNN vs MLP)

Validation accuracy of the **CNN**:



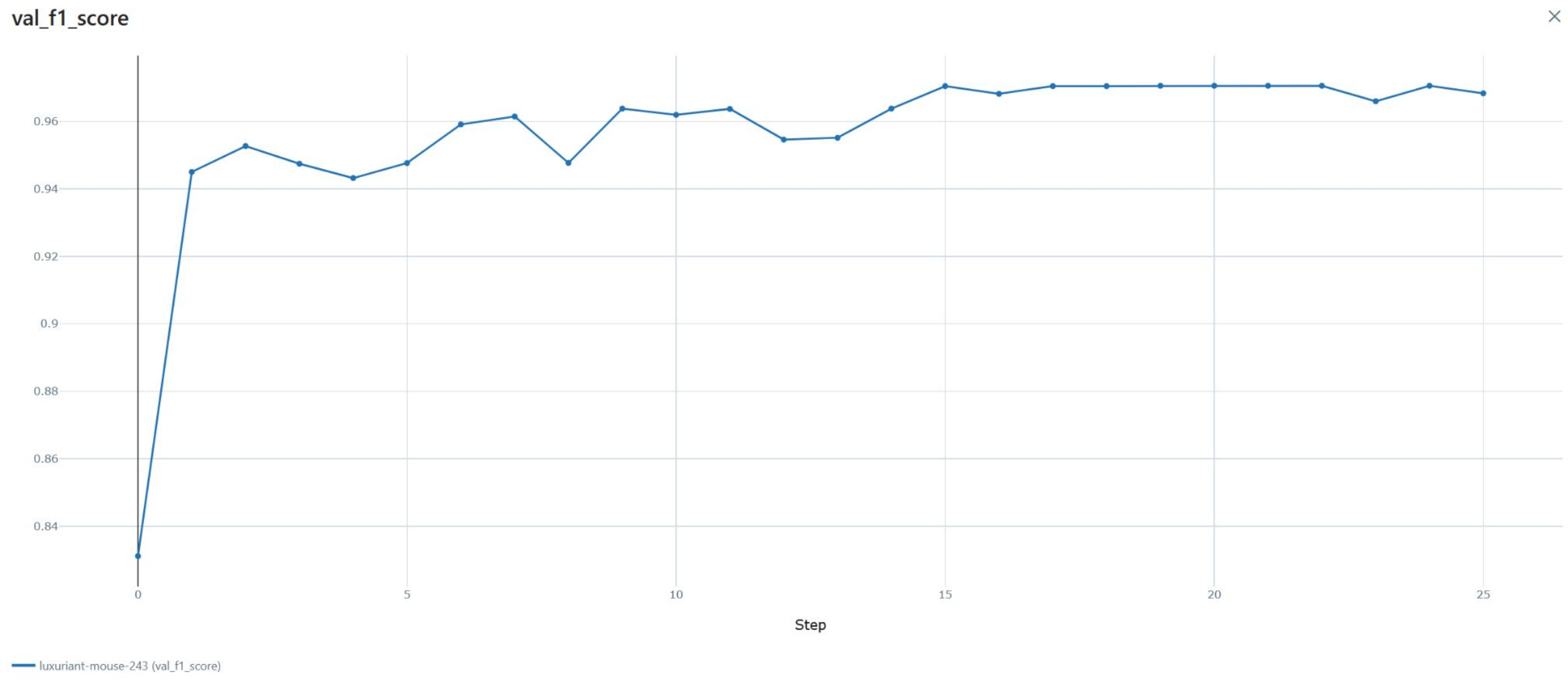
Performance comparison (CNN vs MLP)

F1 Score:

- The MLP achieves a **high F1 score**, indicating a good balance between precision and recall on the validation set. This suggests that the MLP, with **effective preprocessing**, can effectively classify instances, maintaining a strong performance in terms of both false positives and false negatives.
- The CNN also demonstrates a **high F1 score**, slightly outperforming the MLP. This reflects the CNN's ability to capture complex patterns in the data, leading to better precision and recall. The CNN's architecture allows it to **generalize** well and **maintain high performance metrics** across different evaluation criteria.

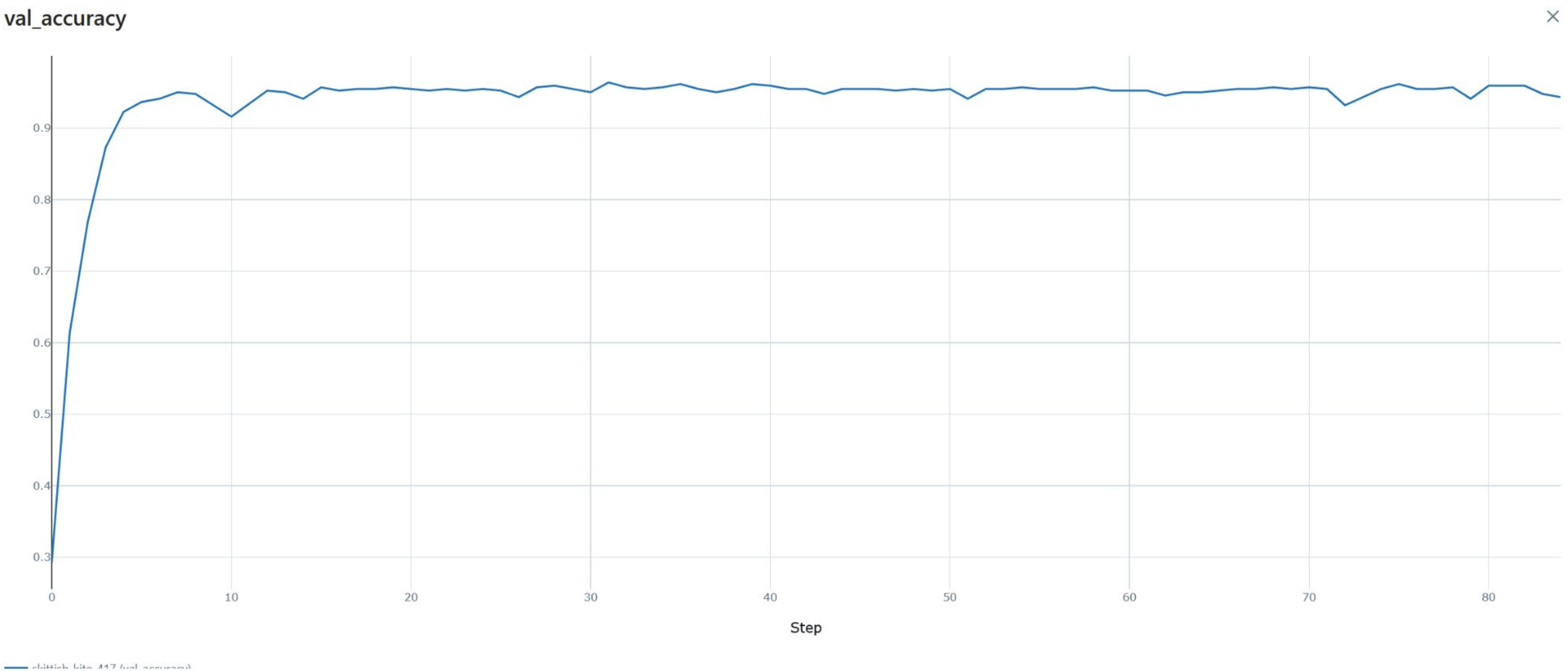
Performance comparison (CNN vs MLP)

F1-Score of the **MLP**:



Performance comparison (CNN vs MLP)

F1-Score of the **CNN**:



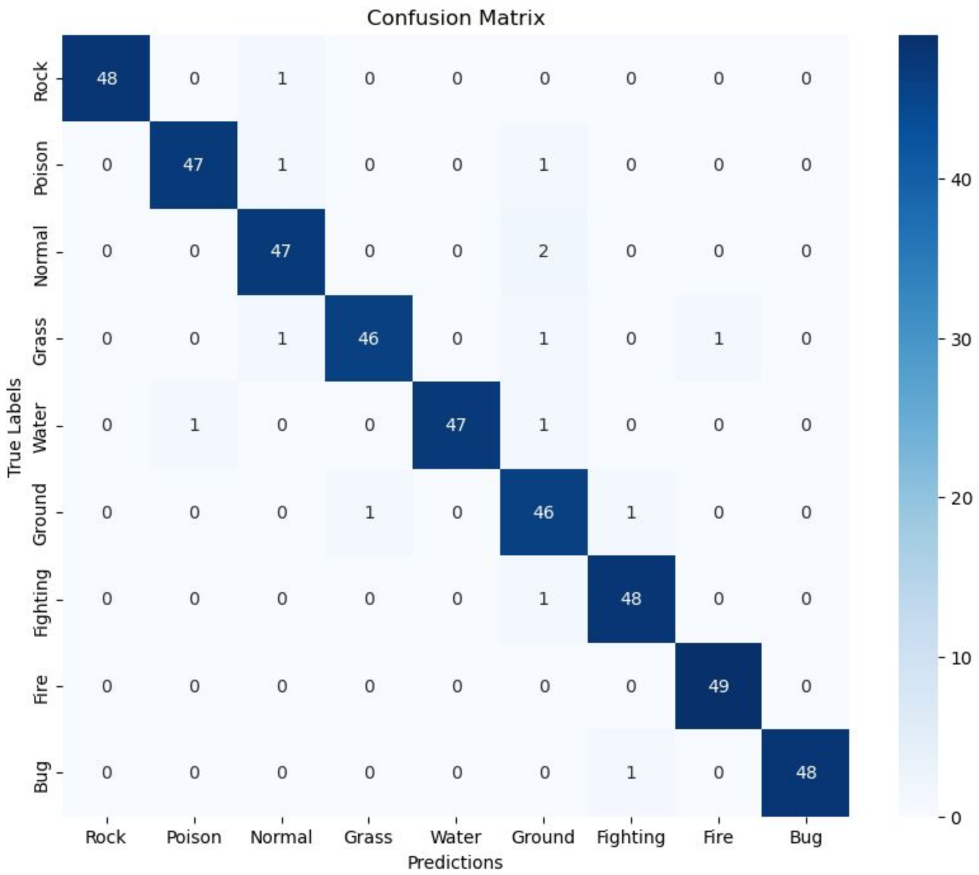
Performance comparison (CNN vs MLP)

Confusion matrix:

- The CNN's confusion matrix displays a **prominent diagonal**, reflecting **accurate predictions** across most classes. The MLP demonstrates slightly better performance in terms of correct classifications, as evidenced by the higher sum of the diagonal in its confusion matrix (427 vs 423).
- Despite the CNN's architectural advantages in capturing spatial hierarchies, the MLP's simpler architecture, combined with effective preprocessing, allows it to achieve better overall classification accuracy in this case.
- Both models face challenges in distinguishing between similar classes, but the MLP appears to handle these distinctions more effectively in this dataset.

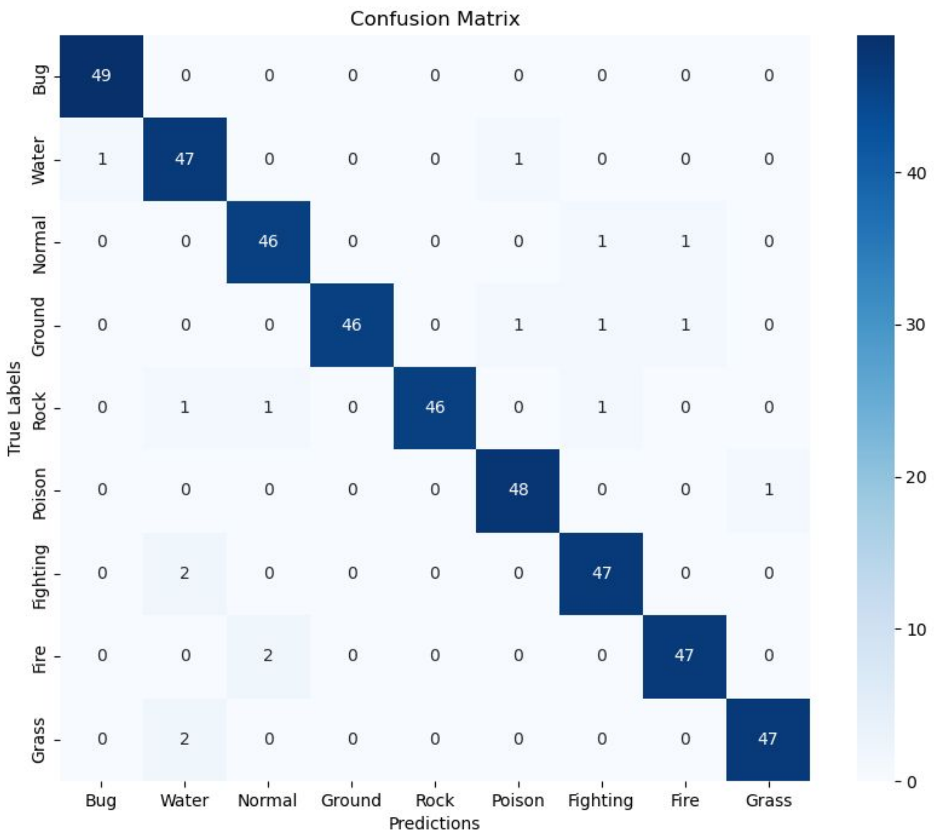
Performance comparison (CNN vs MLP)

Confusion matrix of the **MLP**:



Performance comparison (CNN vs MLP)

Confusion matrix of the **CNN**:



Performance comparison (CNN vs MLP)

Analysis of improvements

Model Performance:

- Both the CNN and MLP models have shown **significant improvements** in performance metrics such as accuracy, F1 score, and reduced training loss. The CNN, in particular, demonstrates **superior performance** in capturing complex patterns, leading to better generalization and fewer misclassifications.

Preprocessing Impact:

- **Effective preprocessing techniques** have played a crucial role in enhancing model performance. By simplifying the input data, the models can focus on learning relevant features, leading to improved accuracy and stability in predictions.

Generalization:

- The models, especially the CNN, exhibit **strong generalization capabilities**, as evidenced by high validation accuracy and F1 scores. This indicates that the models are not only memorizing the training data but are also learning to make accurate predictions on unseen data.

Performance comparison (CNN vs MLP)

Remaining challenges

- **Class Distinction:**
  - Despite improvements, both models still face **challenges** in distinguishing between similar classes, as seen in the confusion matrices. Further refinement in feature extraction and model architecture may be necessary to address these misclassifications.

- **Computational Efficiency:**
  - While the CNN performs well, it typically requires more computational resources compared to the MLP. **Optimizing the CNN for efficiency** without sacrificing performance remains a challenge.

- **Data Quality and Quantity:**
  - The quality and quantity of training data significantly impact model performance. Ensuring a diverse and representative dataset is essential for further improving model accuracy and robustness.

- **Model Interpretability:**
  - As models become more complex, interpreting their decisions becomes challenging. Developing techniques to explain model predictions will be crucial for building trust and ensuring the models are used effectively in real-world applications.

Lessons learned and insights gained

→ Since our **preprocessing** on images with the MLP were already very optimized, it is **difficult** for us to improve our score of 0.95 with the CNN.

→ We use **data augmentation and normalization** with the CNN, that just improve a little bit the result.

→However, **without the image preprocessing**, we understand that CNN results are **way better and way faster** than MLP results on image classification.

What went wrong

We had **difficulties** trying to find a way to improve our score. Indeed, since our image **preprocessing** developed during the MLP task was very **efficient**, we didn't figure it out to improve the training of the CNN.

We thought about using **data augmentation**. But the **crop** transformation, as being a random operation and sometimes pokémons being small, totally **erases** pokémons. So it affects performance.

What went great

We find a good way to reuse **our pipeline** made during the MLP task, so it was easy to understand, implement and use our CNN Model.

Since we did not want to reach **overfitting**, we've stopped trying to find a way to optimise our model.