INFO8010: Classification of Metal Scrapes using Deep Learning Recognition

Romain Lambermont,[1] **Arthur Louis**,[2] and **Adrien Saulas**[3]

[1] *romain.lambermont@student.uliege.be (s190931)*
[2] *alouis@student.uliege.be (s191230)*
[3] *adrien.saulas@student.uliege.be (s184481)*

I. INTRODUCTION

In recent years, the issue of waste management has become a significant global concern due to its environmental and socioeconomic implications. As populations continue to grow and urbanize, the volume of waste generated has reached alarming levels, straining existing waste management systems. Traditional methods of waste detection and classification often rely on manual labor, leading to inefficiencies, errors, and increased costs.

However, with the rapid advancements in artificial intelligence and computer vision, there is a promising opportunity to revolutionize waste management practices. Deep learning algorithms have shown remarkable capabilities in image recognition tasks. By leveraging these algorithms, waste detection can be automated, leading to more accurate, efficient, and cost-effective waste management processes.

The primary objective of this project was to help the the Pick-it team by using the data collected by the GeMMe group at University of Liège and try to predict the class of metal scrapes with the biggest accuracy using 3 main features, a 3 dimensional scan, a visible and near-infrared (VNIR) picture and an X-Ray. The samples are separated between 7 classes. To achieve this objective, we will delve into the fundamental principles of deep learning, including convolutional neural networks (CNN's), which have demonstrated exceptional performance in image analysis tasks, and will also try improving our predictions by using the residual network implementation. We will also explore various data preprocessing techniques and training strategies that can enhance the model's ability to detect and classify waste accurately.

Additionally, we will discuss the challenges and limitations associated with waste detection using deep learning algorithms and propose potential solutions to overcome them.

II. RELATED WORK

A lot of literature was found when looking for waste classification using Deep Learning and Hyperspectral features. The following scientific papers captured our attention when starting our project :

- "Recyclable waste image recognition based on deep learning"[1]: In their implementation they based their model, called the CTR, on ResNet18 and they built an additional block , the self-monitoring module (SMM) that "can continuously adjust the importance of waste image feature information, so that the model can better approach the global optimal solution and accelerate the network convergence".

  The structure of their network and SMM block can be seen in Figures 1 and 2 that are taken from [1].

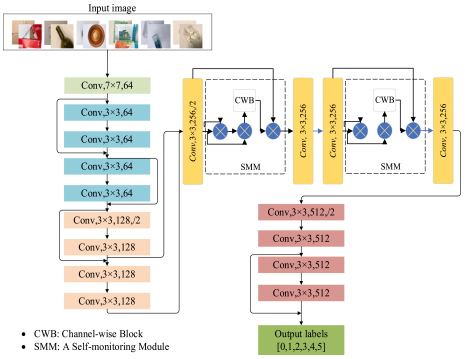

FIG. 1: CTR Structure

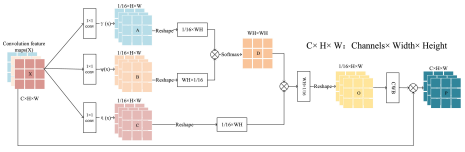

FIG. 2: SMM Structure

- "Identification of wheat kernels by fusion of RGB, SWIR, and VNIR samples"[2]: This article gave interesting points of view for management of images in RGB and VNIR. Indeed, they developed and utilized four image fusion methods that greatly facilitate the learning of the neural network:
  - Brovey transform (BT)-based fusion, which aims to merge the spatial information behind

the RGB information with the intensity of VNIR

- Intensity-hue-saturation (IHS)-based fusion, which combines the high spatial resolution band with a captured multi-spectral–low spatial resolution image
- Discrete Cosine Transformation (DCT)-based fusion, which merge by maximising the variance of certain pixels.
- Common Vector Approach-based fusion, however, utilizes VNIR, RGB, and SWIR images, making it irrelevant for our project.

From these ideas of fusion, we were inspired by the first one to create a transformer multiply, allowing us to establish connections between the different channels of our dataset. For example we simply multiplied or normalized 3D channels with each VNIR. This allows the neural network to establish a connection between the brightness of VNIRs and the 3D image.

Finally, in this research, the utilized architecture is VGG16, which is a renowned algorithm in image classification. We have tried these architecture as you can see it later in the Methods section.

- "The Effectiveness of Data Augmentation in Image Classification using Deep Learning"[3]: This paper gave us a lot of details concerning the different data augmentation techniques that could be useful in the case of image classification in deep learning. The main takeaway of this paper was that applying data augmentations techniques correctly to our data could only lead to a better accuracy on the test set as can be seen in Figure 3.

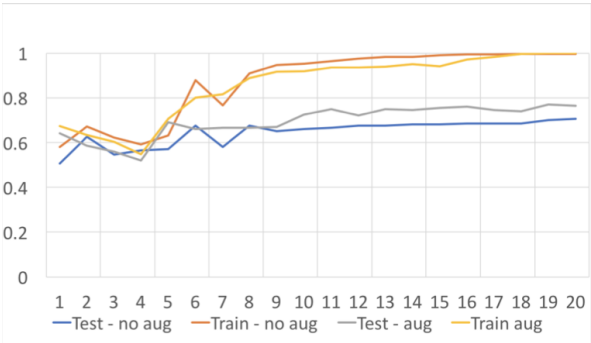

FIG. 3: Accuracies reported by [3] for both train and test sets with and without data augmentation

- "Hyperspectral Image Classification With Deep Learning Models"[4]: This scientific paper showed us the superiority of the recurrent 2D and 3D CNN, having an accuracy above 99%, when dealing with Hyperspectral images such as the ones we have in our own dataset.

III. METHODS

A. Data

1. Data Visualization

The initial dataset was provided by the Pick-it team. It consists of 3163 metallic pieces captured by 11 sensors (one layer representing the metallic piece in 3D, 8 layers representing the VNIR and 2 layers for the X-Ray). It had to be reshaped, for instance, to encode the various labels as numerical values since PyTorch only accepts numeric inputs.

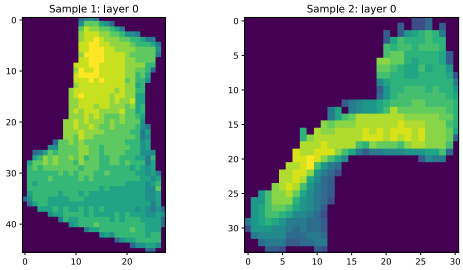

FIG. 4: 3D layer for two samples in our dataset

As depicted in the figure 4, the different pieces vary in size, and the same applies to the images. This could potentially pose a problem in our subsequent process.

2. Dataset Class and Dataloader

Once we understood the different layers of the dataset correctly, we decided to create our own `Dataset` class called `PickItDataset` that would load it in our specified way with three different structures:

- `data`: The feature cube, containing the different values for 3D scan, VNIR images and X-Ray.
- `class`: The target variable
- `masks`: Delimiting the piece in 2 dimensions.

Transforming the dataset into a `PyTorch` tensor allowed us to apply the transforms needed for the data augmentation we tackled in section III A 3.

3. Data Augmentation

Data augmentation plays a crucial role in improving the performance and generalization capabilities of deep learning models for hyperspectral image classification. In this study, we employed several data augmentation techniques that we tuned from the basic version from `PyTorch` to fit our dataset.

These tuned versions of the transformations were the following:

- `Rescale`: We decided to rescale each of the layers to squares of 256 pixels as our layers were not all of the same size.
- `RandomHorizontal` and `RandomVerticalFlip`: As our pieces could come on the conveyor belt in any possible orientation, we decided to apply a random flip to the layers, making the network more robust. A probability of 50% was chosen for those flips to occur when applying the transformations.
- `Normalize`: This normalization step was only applied to the `data` structure, because normalizing the mask would not make any sense. This normalization would lead to the network learning the patterns between features more consistently.

The result of applying these transformations can be seen in Figure 5.

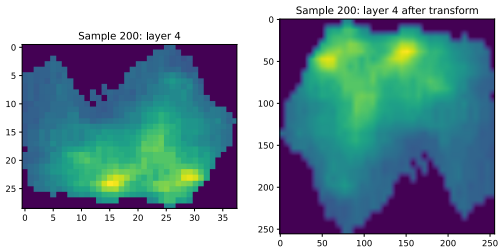

FIG. 5: 3D layer of a sample before and after applying transformations

We also decided to apply one final idea for the data augmentation. We applied the idea of multiplying the layers from this previously cited publication [2]. We created a function `multiply_layers` that multiplies every pair of layers in the `data` structure, creating 26 new layers for our `data`.

Of course, we used the flipping transformations only for the training set, as these transformations are only useful during the training phase.

4. *Training Testing Splitting*

To split our dataset into a training and testing set, we used a simple strategy of 90% for training and 10% for testing.

The only tuning we did was to force the percentage of each class to be roughly the same between the test set and the training set as our dataset is not well distributed between all the classes as showed in the table I.

TABLE I: Distributions of the samples between classes

	Class	Probability
zinc	903	0.285488
aluminium	901	0.284856
copper	557	0.176099
brass	448	0.141638
nickel	181	0.057224
pcb	102	0.032248
painted	41	0.012962
rubber	30	0.009485

The final distribution between the training and testing set can be seen in Figure 6.

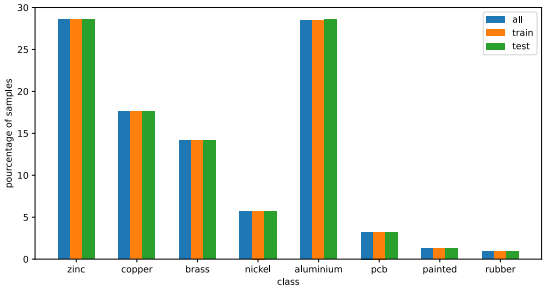

FIG. 6: Distribution of the classes in all the dataset and across the training and testing set

B. Architecture

1. *Simple CNN*

At the beginning of the project, before delving into more elaborate architectures, we first looked into a simple `CNN` architecture. This initial architecture was composed of `Conv2D` layers and `MaxPooling`. We then used three Linear neural networks to obtain the desired number of outputs corresponding to the number of classes in our dataset. With this simple CNN and the data augmentation techniques explained earlier, we already achieved some pretty decent results.

2. *Residual Network*

We quickly turned our attention to the most well-known architecture in the field of image classification: `ResNet` writes in the paper Deep Residual Learning for Image Recognition [5].

To understand how it works, we initially attempted to write this architecture ourselves based on several sources. `ResNet` operates on the principle of residual learning.

Instead of directly learning to represent the complete transformation of the input data, each layer learns to represent the difference between the desired transformation and the current input by adding skip connections as we can see in Figure 7. This method primarily helps overcome the vanishing gradient problem. We have selected `ResNet50` among its peers as it appeared to be a commendable compromise between performance and efficiency, enabling us to conduct an ample number of tests without incurring excessive compilation time.

Since `ResNet50` was already pretrained in `torchvision`, we directly imported it with its pretrained weights. To adapt it to our problem, we modified the input of the first convolutional network from 3 to 11 (as 11 is the number of channels in our dataset). Then, we needed to specify the desired number of outputs by modifying the outputs of the last Linear neural network to 8, our number of classes.

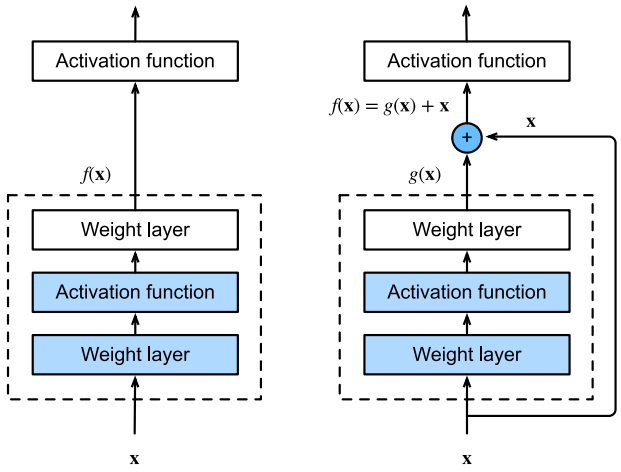

FIG. 7: Skip connections used in `ResNet`

C. Visual Geometry Group 16

Since the `VGG16` architectures seems to be a useful architecture in the paper :"Identification of wheat kernels by fusion of RGB, SWIR, and VNIR samples" [2] we decided to give it a try. `VGG16` (from the paper Very Deep Convolutional Networks for Large-Scale Image Recognition [6]) is a renowned architecture in the field of deep learning for image classification. It consists of sixteen layers, including convolutional and pooling layers. The convolutional layers filter the image's features, while the pooling layers reduce its spatial dimensions. These convolutional and pooling layers are stacked multiple times to increase the network's depth.

Following the convolutional and pooling layers, VGG16 incorporates three fully connected layers, followed by a `ReLU` activation function. The last fully connected layer is connected to an output layer that utilizes the `softmax` activation function for classification.

What sets `VGG16` apart is its depth and the use of 3x3 convolutions with a `stride` of 1. This approach enables the learning of rich and discriminating representations while limiting the number of model parameters.

Therefore, we imported this architecture from `torchvision` along with its pretrained weights. Similar to the `ResNet` architecture, we needed to tune the first convolutional layer to match our number of channels and the last linear layer to match our number of classes.

A representation of the `VGG16` network can be seen in Figure 8.

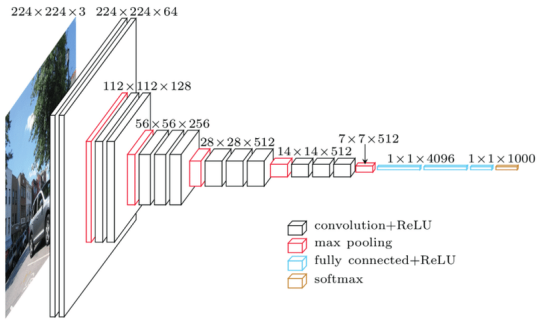

FIG. 8: `VGG16` Structure

D. Training Parameters

During our various attempts, we experimented with two well-known optimizers: `SGDoptimizer` and `AdamOptimizer`. Throughout these trials, we observed that `SGDoptimizer` tended to converge much faster than `AdamOptimizer`, resulting in a 10% higher accuracy for the `SGDoptimizer`. However, in the long run, it is expected that `AdamOptimizer` would converge better than `SGDoptimizer`.

From this, we concluded that we did not spend enough time training the model to allow `AdamOptimizer` to achieve a lower loss than `SGDoptimizer`. Therefore, we have decided to continue using `SGDoptimizer` in our further work.

Regarding hyperparameters, we determined that with a constant `momentum` (=0.9), we achieved the best results with a `learning rate` of 0.0003.

E. Loss Function

To train our models, we used two different metrics :

- Cross Entropy Loss

- A weighted Cross Entropy Loss

The second metric is really useful when looking at our dataset. Indeed, our dataset is not well distributed between the classes as there are a lot of samples for variables such as `nickel`, `pcb`, `painted`, `rubber`. Penalizing the network more when predicting the wrong class for classes with less samples is thus a good idea to make it learn in more details the patterns for these classes.

IV. RESULTS

In this section we reported the performance of our models both qualitatively and quantitatively.

A. Qualitative Analysis

We deployed our best model (discussed in section IV B) and used it to compute predict the class of a certain waste in the Jupyter Notebook `deployment.ipynb`. We obtained the following results (a correct prediction in Figure 9 and a wrong one in Figure 10) in a pretty reasonable amount of time (around 0.2 seconds) which could make the sorting process really fast on the conveyor belt at the GeMME.

FIG. 9: Example of a good prediction

FIG. 10: Example of a wrong prediction

B. Quantitative Analysis

When looking in the results table in Appendix A, we can clearly see that our best model is the `VGG16` with an overall accuracy of 93.63%. Regarding the other models, the CNN already performs well for the metallic waste, when using the augmented dataset and the weighted cross entropy, but it still does not perform well on the non-metallic wastes. Indeed, it seems to not capture well the properties of these matters.

The `ResNet` took everything a step further and directly understood the more complex patterns to detect the non-metallic wastes. Applying the model to the augmented dataset and the 37 features created by the multiplication of all the layer pairs, we achieved even better results, but still lower than the `VGG16` model using the 37 features.

When delving into the loss plots located in Appendix B, we can see that all the models are improving along the iterations in the training dataset and as predicted an expected from the previously discussed results, the model that reaches the lowest lost is the `VGG16`.

Lastly, by taking further attention in the correlation matrix located in Appendix C, we can once again see that all the models get better step by step understanding more and more the patterns to detect the non-metallic wastes as the scores for metallic ones are always good.

V. DISCUSSION

In Table II, we can observe that the provided results are satisfactory; however, there are certain discussions to be raised regarding these outcomes.

Firstly, since we are dealing with the sorting of metallic waste, it is expected that the results for non-metallic waste such as paint, PCB, and rubber are significantly lower. This is primarily due to their much lower presence in the dataset.

Furthermore, the dataset provided by the Pick-it team could be larger and consequently yield better results.

Additionally, when comparing the models, we immediately noticed that applying cross-entropy loss with modified weights allowed us to improve the sorting of less frequent waste categories in our dataset but tended to decrease accuracy more frequent one such as copper, for example.

Regarding the `CNN` architecture, it is noteworthy that we are achieving fairly satisfactory results for

certain types of waste. Hence, one may speculate that within a specific context, separating Zinc, Copper, Brass, and aluminum from other waste categories using a simple `CNN`, such as the one we have proposed, could be acceptable, albeit less efficient compared to more sophisticated architectures.

Moreover, we observe that the results for `VGG16` are better than those for `ResNet`. However, we know that `ResNet` is capable of addressing the vanishing gradient problem, and it should logically be more efficient after intensive training. Therefore, we can conclude that the vanishing gradient problem may not have truly manifested in our `VGG16` model yet, indicating that we could have trained both models more intensively. Unfortunately, due to time constraints, we were unable to pursue this idea.

Lastly, we would have liked to conduct several additional tests with the `VGG16` model. However, this model took approximately 50% more time to train compared to `ResNet`. Since we discovered this model later in the process, we did not have the opportunity to conduct as many tests as we did with `ResNet`.

[1] Qiang Zhang, Xujuan Zhang, Xiaojun Mu, Zhihe Wang, Ran Tian, Xiangwen Wang, and Xueyan Liu. Recyclable waste image recognition based on deep learning. *Resources, Conservation and Recycling*, 171:105636, 2021.

[2] Kemal Özkan, Şahin Işık, and Büşra Topsakal Yavuz. Identification of wheat kernels by fusion of rgb, swir, and vnir samples. *Journal of the Science of Food and Agriculture*, 99(11):4977–4984, 2019.

[3] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621, 2017.

[4] Xiaofei Yang, Yunming Ye, Xutao Li, Raymond Y. K. Lau, Xiaofeng Zhang, and Xiaohui Huang. Hyperspectral image classification with deep learning models. *IEEE Transactions on Geoscience and Remote Sensing*, 56(9):5408–5423, 2018.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

Appendix A: Results

TABLE II: Results obtained by the different models using the three different types of datasets and the two different loss functions

Model	Dataset	Loss	Accuracy	Zinc	Copper	Brass	Nickel	Aluminum	PCB	Painted	Rubber
`CNN`	Original	CE	70.70%	99%	87%	0%	0%	94%	0%	0%	0%
	Augmented		79.61%	99%	76%	73%	0%	97%	0%	0%	0%
	Original	CE + Frequency Weights	78.66%	96%	78%	80%	0%	92%	0%	0%	0%
	Augmented		84.39%	98%	95%	84%	0%	98%	0%	0%	0%
`ResNet50`	Original	CE	88.85%	98%	96%	70%	44%	97%	80%	75%	33%
	Augmented		89.80%	98%	98%	75%	50%	98%	80%	0%	67%
	37 Features		89.17%	97%	98%	80%	56%	91%	80%	75%	33%
	Original	CE + Frequency Weights	89.17%	99%	87%	75%	61%	94%	90%	75%	67%
	Augmented		91.08%	96%	87%	80%	78%	99%	80%	75%	100%
	37 Features		91.40%	98%	95%	80%	72%	96%	80%	75%	67%
`VGG16`	37 Features	CE + Frequency Weights	93.63%	99%	96%	84%	78%	97%	90%	75%	67%

Appendix B: Plots of the Loss along iterations

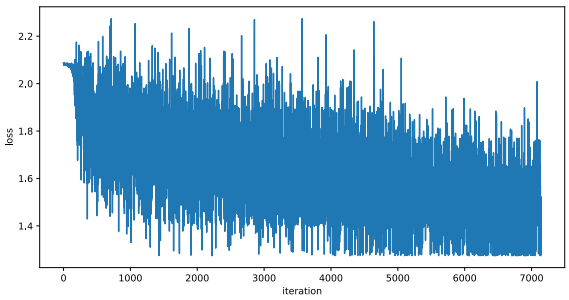

FIG. 11: Evolution of the loss for the basic `CNN` model

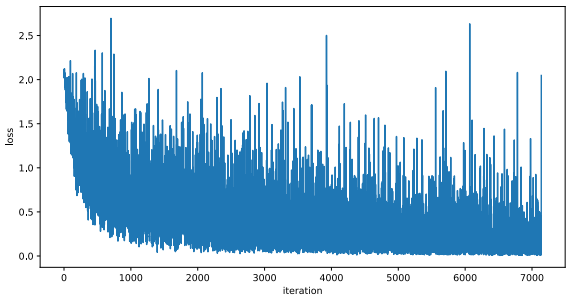

FIG. 12: Evolution of the loss for the `ResNet` with basic dataset and Cross Entropy Loss

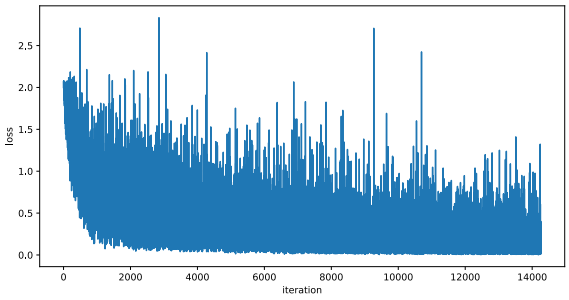

FIG. 13: Evolution of the loss for the `ResNet` with augmented dataset and basic Cross Entropy Loss

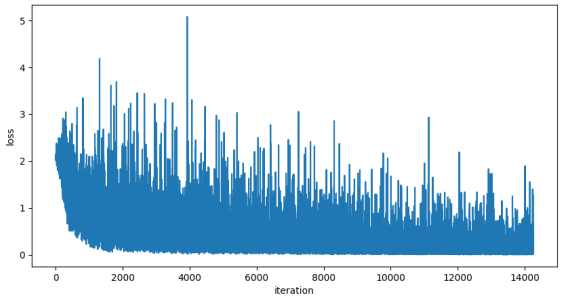

FIG. 14: Evolution of the loss for the `ResNet` with augmented 37 features dataset and weighted Cross Entropy Loss

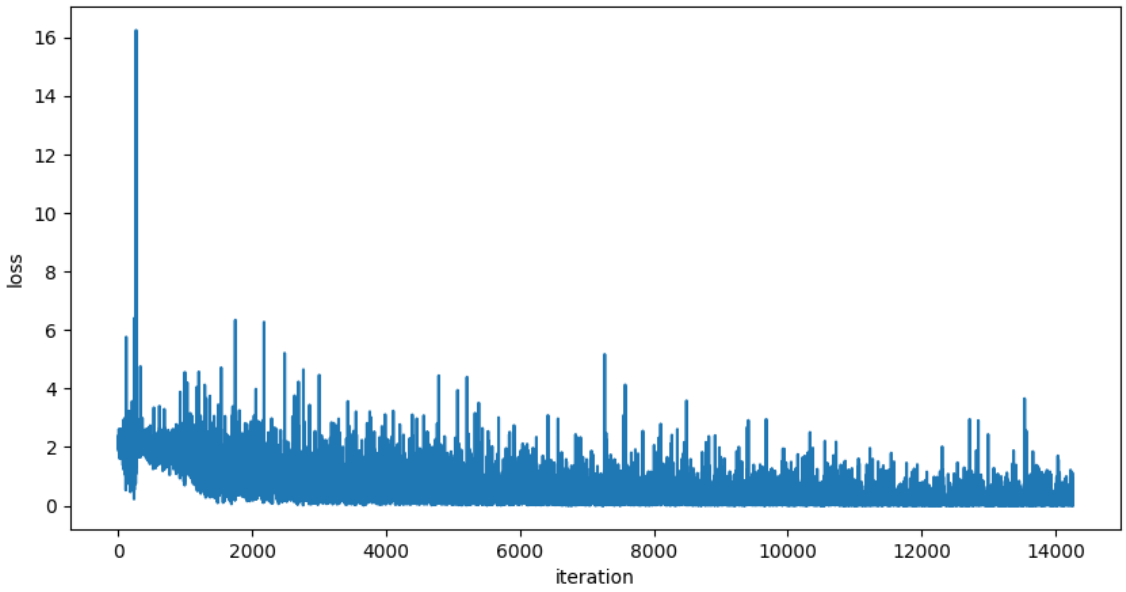

FIG. 15: Evolution of the loss for the `VGG16` with 37 features augmented dataset ans weighted Cross Entropy Loss

Appendix C: Correlation Matrices

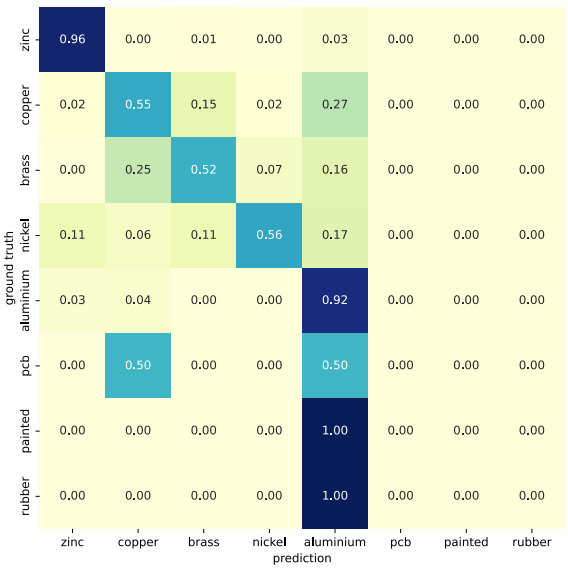

FIG. 16: Correlation Matrix for the basic `CNN` model

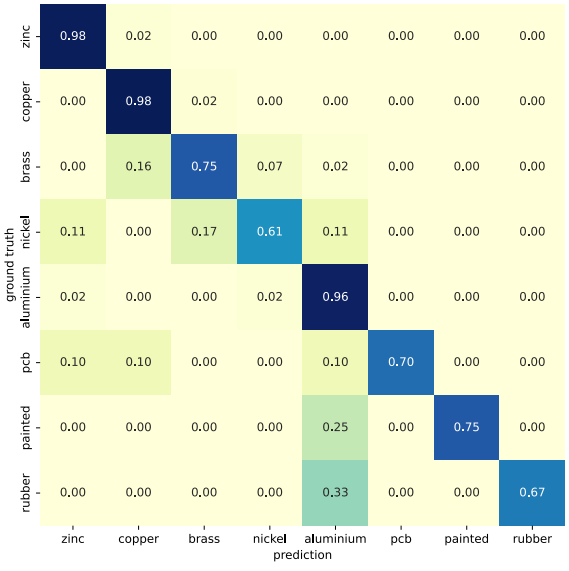

FIG. 17: Correlation Matrix for the `ResNet` with basic dataset and Cross Entropy Loss

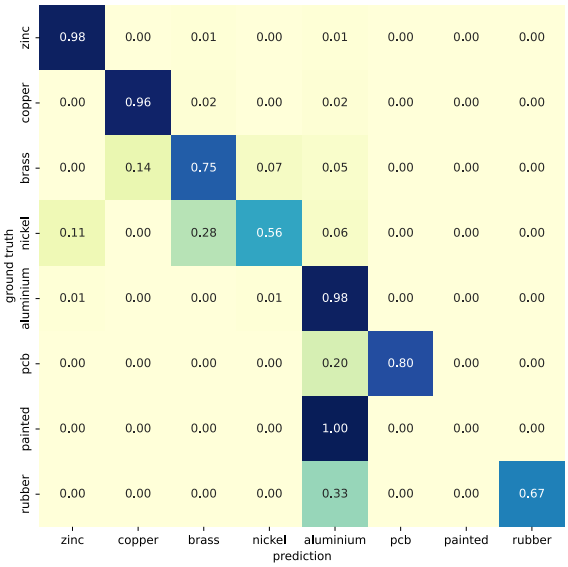

FIG. 18: Correlation Matrix for the `ResNet` with augmented dataset and basic Cross Entropy Loss

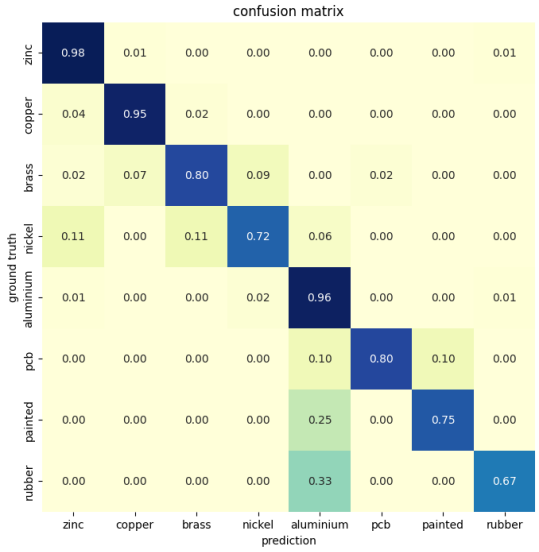

FIG. 19: Correlation Matrix for the `ResNet` with the 37 features augmented dataset and weighted Cross Entropy

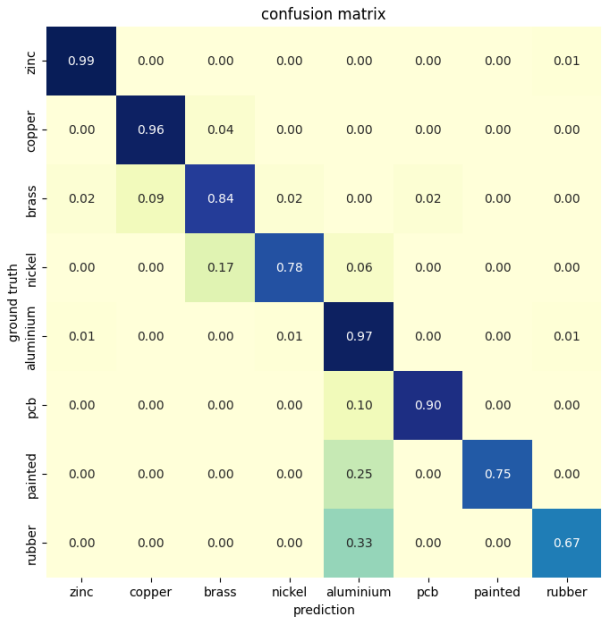

FIG. 20: Correlation Matrix for the `VGG16` with the 37 features augmented dataset and weighted Cross Entropy