Image analysis and pattern recognition: Chocolate recognition project

EPFL, Mai 2025

Group 10

Rayan Bouchalouf	311491
Vincent Ellenrieder	329051
Tomas Garate Anderegg	313984

Table of Contents

1 Model : Justification of final design choices

The goal of this project was to develop a pattern recognition algorithm to correctly classify types of chocolates found in an image set as well as count each type's occurrences. Two challenge categories were proposed by the course's teaching assistants, namely a classic approach comprising of image analysis techniques for feature extraction followed by a machine learning algorithm for classification, as well as a deep learning solution. We chose to participate in the **deep learning challenge**. A central rule of this approach was that our model could not exceed a total of 12 million parameters.

Two sets were given be the TAs, one train set comprising of 90 labeled images and a test set with 180 images, for which contrariwise no .csv-type labels were given. In each set, images comprised of different backgrounds on which the chocolates were arranged. Furthermore, some images featured foreign objects, which did not count as chocolates and which our model had to correctly ignore during training and, eventually, inference.

The best F-1 score we achieved on the test set was 95.34%.

1.1 Overall structure of the model

Our pipeline consists of a convolutional neural network that extracts 512 features of each image, followed by a fully connected neural network for multi-class classification and counting, as shown in figure 1.
The original images are of very high resolution (6000 $\times$ 4000 pixels). To decrease computation time and memory/RAM constraints on our limited hardware, the images provided as input to the model were reduced to 1024 $\times$ 1536 pixels.

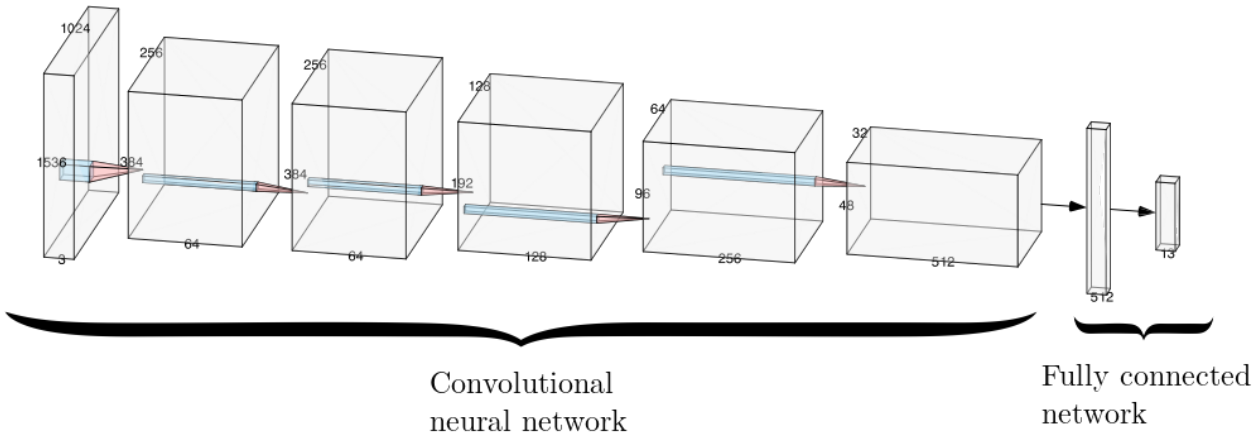

Figure 1: Overview of the model architecture. In the CNN, the input corresponds to an image, then each box corresponds to the blocks of the ResNet18 architecture to extract 512 features. The fully connected neural network is a simple 1 layer feedforward network.

1.2 ResNet18 architecture

1.2.1 Description of the architecture

After many cycles of trial and error, the final model we converged to is the well known ResNet18 model [1]. Figure 1 features the dimensions of each layer in the CNN, while Figure 2 describes the implemented ResNet18 architecture :

1. A first layer consists of a 2-stride convolution followed by a batch norm, 2-stride MaxPool and a ReLU activation function. This initial layer decreases the size of the original image by 4, due to both strides of 2 in the convolution and the MaxPool. During the convolution, 64 filters are applied.

   The batch norm placed after the convolution normalizes this output so that it has zero mean and unit variance. This eases backpropagation during training and ultimately speeds up convergence in training.

   This primary convolutional layer detects basic visual features, like linear boundaries, textures (smooth vs. rough structures), color and intensity differences (e.g. white vs. brown vs. dark).

2. Then, a single block of two successive 1-stride convolutions takes place. This allows refinement of existing features without shrinking the feature map too quickly, nor adding any additional feature dimensions [2].

3. The next part comprises 3 layers of 2 convolutional residual network blocks. The first block implements two convolutions, where only the initial convolution reduces the spatial dimension. At the end of all three layers, the size of the original image is reduced by an additional factor of $2^3 = 8$ and the number of filters (features) is doubled during each convolution. Thus, the final feature map prior to the global average pooling will have a width 32 times smaller than that of the original image, and a depth of 512 channels.

   In the middle layers, as more features are abstracted, the model begins to identify more complex shapes, like chocolate contours, surface characteristics (e.g. glossy, matte, decorative patterns).

   The deeper layers synthesize the lower-level features to form precise segmentation, such as specific details that activate for specific chocolate types as well as an ability to form a contextual understanding (background, little shadows and even foreign objects). These layers are thus responsible for the final feature extractions that enable correct classification.

4. Finally, a global average pooling layer reduces the spatial dimension of the feature map to a single feature vector of size 512×1.

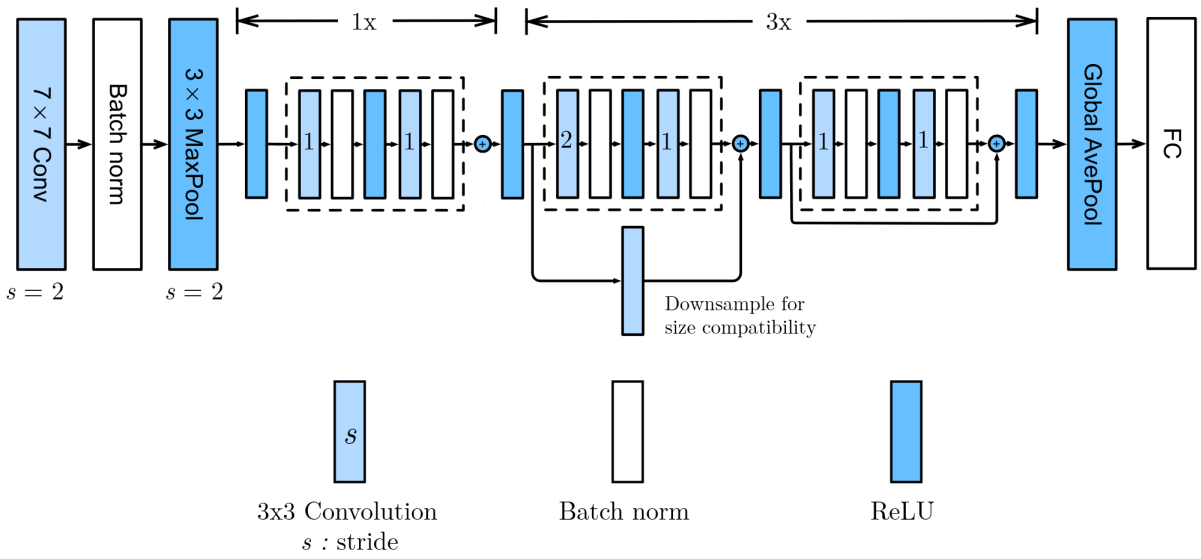

Figure 2: Complete architecture of the ResNet18 CNN model model followed by a global average pooling and a fully connected neural network. Figure adapted from [3].

While ResNet18 provided us with some good results, we wondered if the width and the depth of the CNN was adapted for our purpose or not.

1.2.2 Depth of the CNN

As described earlier, the number of output features is 512. While we feared this many amount of features would lead to a problem involving the curse of dimensionality - and thus the need of an unreasonably large dataset [4] - the classifier yielded good results, which comforted us in that choice. The intuitive reasoning behind these positive results is that this number of features allows to describe very fine patterns in the image, which leads to good classification.

1.2.3 Width of the CNN

The last feature map of the feature extractor has dimensions 32×48, which is the result of a dimension reduction factor of 32. This means that a pixel in our feature map will represent an area in the original image of 32×32 pixels. Knowing that the size of the input image is of size 1024×1536, we found that a chocolate in such an image had a size of approximately 150×150 pixels, meaning that a chocolate in the original image will spread on 5×5 pixels in the feature map. This is a reasonably good size, as one pixel in the feature map will thus be mapped to small features inside chocolates.

1.2.4 Skip connections in ResNets

A problem we encountered early on in our initial models was the vanishing gradient. As we used a relatively deep model, the gradient could near zero after a few layers of backpropagation. ResNet architecture solves this problem using skip connections, which is depicted in figure 3. By adding the input of the first convolutional layer to the output of the second convolution (prior to the ReLU activation), this allows to keep the gradient from converging to zero. Indeed, essentially,

$$\frac{\partial H}{\partial \mathbf{x}}(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}}(F(\mathbf{x}) + \mathbf{x}) = \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}) + 1$$

where the identity addition prevents the gradient to vanish, even if $\frac{\partial F}{\partial \mathbf{x}}(\mathbf{x})$ does.

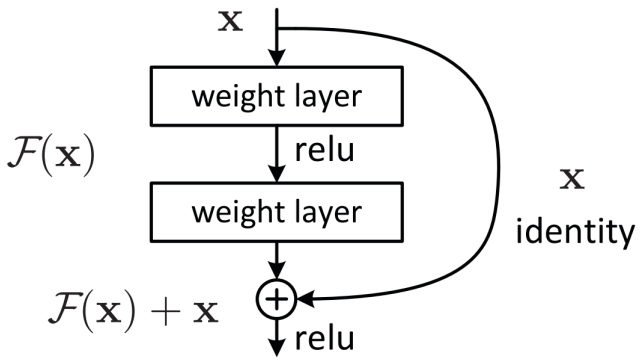

Figure 3: Schematic of a residual connection block used in ResNets architectures [1].

1.3 Number of parameters of the model

The number of parameters of our model reaches 11,183,181. The detailed distribution of parameters is described in the image below.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 171, 256]	9,408
BatchNorm2d-2	[-1, 64, 171, 256]	128
ReLU-3	[-1, 64, 171, 256]	0
MaxPool2d-4	[-1, 64, 86, 128]	0
Conv2d-5	[-1, 64, 86, 128]	36,864
BatchNorm2d-6	[-1, 64, 86, 128]	128
ReLU-7	[-1, 64, 86, 128]	0
Conv2d-8	[-1, 64, 86, 128]	36,864
BatchNorm2d-9	[-1, 64, 86, 128]	128
ReLU-10	[-1, 64, 86, 128]	0
Block-11	[-1, 64, 86, 128]	0
Conv2d-12	[-1, 64, 86, 128]	36,864
BatchNorm2d-13	[-1, 64, 86, 128]	128
ReLU-14	[-1, 64, 86, 128]	0
Conv2d-15	[-1, 64, 86, 128]	36,864
BatchNorm2d-16	[-1, 64, 86, 128]	128
ReLU-17	[-1, 64, 86, 128]	0
Block-18	[-1, 64, 86, 128]	0
Conv2d-19	[-1, 128, 43, 64]	73,728
BatchNorm2d-20	[-1, 128, 43, 64]	256
ReLU-21	[-1, 128, 43, 64]	0
Conv2d-22	[-1, 128, 43, 64]	147,456
BatchNorm2d-23	[-1, 128, 43, 64]	256
Conv2d-24	[-1, 128, 43, 64]	8,192
BatchNorm2d-25	[-1, 128, 43, 64]	256
ReLU-26	[-1, 128, 43, 64]	0
Block-27	[-1, 128, 43, 64]	0
Conv2d-28	[-1, 128, 43, 64]	147,456
BatchNorm2d-29	[-1, 128, 43, 64]	256
ReLU-30	[-1, 128, 43, 64]	0
Conv2d-31	[-1, 128, 43, 64]	147,456
BatchNorm2d-32	[-1, 128, 43, 64]	256
ReLU-33	[-1, 128, 43, 64]	0
Block-34	[-1, 128, 43, 64]	0
Conv2d-35	[-1, 256, 22, 32]	294,912
BatchNorm2d-36	[-1, 256, 22, 32]	512
ReLU-37	[-1, 256, 22, 32]	0
Conv2d-38	[-1, 256, 22, 32]	589,824
BatchNorm2d-39	[-1, 256, 22, 32]	512
Conv2d-40	[-1, 256, 22, 32]	32,768
BatchNorm2d-41	[-1, 256, 22, 32]	512
ReLU-42	[-1, 256, 22, 32]	0
Block-43	[-1, 256, 22, 32]	0
Conv2d-44	[-1, 256, 22, 32]	589,824
BatchNorm2d-45	[-1, 256, 22, 32]	512
ReLU-46	[-1, 256, 22, 32]	0
Conv2d-47	[-1, 256, 22, 32]	589,824
BatchNorm2d-48	[-1, 256, 22, 32]	512
ReLU-49	[-1, 256, 22, 32]	0
Block-50	[-1, 256, 22, 32]	0
Conv2d-51	[-1, 512, 11, 16]	1,179,648
BatchNorm2d-52	[-1, 512, 11, 16]	1,024
ReLU-53	[-1, 512, 11, 16]	0
Conv2d-54	[-1, 512, 11, 16]	2,359,296
BatchNorm2d-55	[-1, 512, 11, 16]	1,024
Conv2d-56	[-1, 512, 11, 16]	131,072
BatchNorm2d-57	[-1, 512, 11, 16]	1,024
ReLU-58	[-1, 512, 11, 16]	0
Block-59	[-1, 512, 11, 16]	0
Conv2d-60	[-1, 512, 11, 16]	2,359,296
BatchNorm2d-61	[-1, 512, 11, 16]	1,024
ReLU-62	[-1, 512, 11, 16]	0
Conv2d-63	[-1, 512, 11, 16]	2,359,296
BatchNorm2d-64	[-1, 512, 11, 16]	1,024
ReLU-65	[-1, 512, 11, 16]	0
Block-66	[-1, 512, 11, 16]	0
AdaptiveAvgPool2d-67	[-1, 512, 1, 1]	0
Flatten-68	[-1, 512]	0
Linear-69	[-1, 13]	6,669

Total params: 11,183,181
Trainable params: 11,183,181
Non-trainable params: 0

Figure 4: Distribution of the parameters of the model among each layer (left and right halves).

2 Training : Technical description

2.1 Training/Validation Strategy

At the beginning of the project, our dataset was used solely for training without any explicit validation split. However, as the model development progressed, it became clear that separating the data into training and validation sets was essential for meaningful evaluation and for tuning our model's performance. The validation set played a crucial role in fine-tuning hyperparameters and avoiding overfitting. We paid special attention to ensure a uniform distribution of backgrounds, foreign objects and chocolate types between the training and validation sets to avoid bias. We experimented with different splitting strategies until we found a configuration that worked well. Initially, 72 images were allocated for training and 18 for validation. This approach allowed us to better evaluate how well the model could generalize to unseen data rather than just memorizing the training set. Eventually, we expanded the dataset while maintaining a similar ratio, ensuring a more robust and intelligent division of the data.

2.2 Data augmentation

To improve the model's robustness and generalization, we applied several data augmentation techniques and generated over 1000 new images for our training set.

- *RandomRotation* introduces orientation variability, ensuring the model is not biased toward any specific angle. *RandomRotation* transformation was used to make the model invariant to the horizontal position of chocolates, ensuring it does not overfit to specific object placements in the training images. We chose a maximum rotation angle of 7°.
- *ColorJitter* modifies brightness, contrast, and saturation, simulating different lighting conditions and enhancing the model's ability to generalize to real world scenarios.
- *Resize* ensures consistency in input size, which is required by the network architecture.
- Finally, we applied *normalization* using the mean and variance computed from our dataset, allowing us to scale the pixel values appropriately such that we the final set had zero mean and unity variance. This step ensures that the model receives inputs with consistent statistical properties, which facilitates more stables and efficient training. This step was made after data augmentation, when loading the data using PyTorch's Dataloader prior to training.

2.3 Loss Function

We chose the Mean Squared Error (MSE) as loss function because our objective was not to perform single-label classification but rather to predict the number of chocolate of each class present in the image. Unlike traditional classification tasks where the model outputs a single class or a probability distribution over classes, our task required the model to output a count for each class, which represents a regression problem [5].
MSE is suited for such regression task because it penalized the difference between the predicted and true counts. Additionally, since multiple chocolates of the same class can appear in the same image, a standard cross-entropy loss would not be appropriate, as it is designed for mutually exclusive class predictions.
Using MSE allowed the model to learn not only to identify the classes but also to estimate their quantities accurately, making it a more natural fit for our problem formulation.
We considered different loss functions such as L1 norm, Smooth L1 and MSE for our counting and classification task. While L1 is more robust to outliers, it is less sensitive to small errors. Smooth L1 is a compromise but is often used in object detection. Since large errors were rare and precision was important, MSE better penalized bigger mistakes, helping the model to learn ore effectively [6][7]. Overall, MSE provided stable and efficient training for our problem.

2.4 Optimizer

We selected the Adam optimizer due to its ability to adaptively adjust learning rates for each parameter, leading to faster and more stable convergence compared to standard optimizer like SGD. While SGD can perform well, it requires often careful manual tuning of the learning rate and momentum. Adam's adaptive nature makes it more robust to such hyperparameter settings.

2.5 Hyperparameters

- Learning rate : We set the learning rate $\eta = 10^{-3}$, based on preliminary experiments to balance the convergence time and quality.

- Batch size : The batch size $B = 4$ was chosen as a compromise between training speed and memory constraints, ensuring efficient gradient estimation without overwhelming hardware resources.
- Number of epochs : the number of epochs was fixed to 1000, however, as we used early stopping to automatically determine the optimal epoch, we simply had to set a number of epoch large enough to allow the early stopping to trigger. This mechanism stops the training once the validation loss stops improving, thus preventing overfitting and saving computational resources. In our best performing model, early stopping led to the parameters being found at epoch **145**.

2.6 Additional Tools

To prevent overfitting and save computational resources, we implemented an early stopping mechanism during training. This technique monitors the validation loss and stops the training process when the loss ceases to improve for a predefined number of epochs. Additionally, we introduce the weight decay ($=10^{-4}$) in the Adam optimizer as a form of L2 regularization. This penalizes large weight values during training which helps prevent the model from becoming too complex and overfitting the training data. Both early stopping and weight decay contributed to better generalization performance and more efficient training.

3 Inference: Quantitative and Qualitative Analysis

3.1 Inference Pipeline for Chocolate Counting

The inference process for chocolate counting involves several sequential stages, designed to ensure consistency with the training phase and produce accurate predictions. The pipeline is described as follows:

3.1.1 Model loading

The trained model is loaded from the saved checkpoint file `best_model.pth`. The underlying architecture, a ResNet18, is re-initialized, and its parameters are restored using the saved weights. This step ensures that the model during inference exactly replicates the one used during training.

3.1.2 Test data preparation

Test images are loaded using a custom PyTorch `Dataset` and `DataLoader`. Each image undergoes preprocessing steps that include resizing and normalization. The normalization is performed using the same mean and standard deviation values as those applied during the training phase, ensuring consistency in the input data distribution.

3.1.3 Forward pass and output extraction

For each batch of test images:

- The images are passed through the trained model.
- The model outputs a vector of real-valued logits for each image, where each logit corresponds to the predicted count of a specific chocolate class.

3.1.4 Post-processing of model outputs

Since the model was trained to perform regression, the raw outputs are continuous real values (correspond to the fully connected output logits). These are post-processed as follows:

- **Clipping:** All negative values are set to zero using the function `np.clip(outputs, 0, None)`, as negative chocolate counts are physically meaningless.
- **Rounding:** The clipped values are rounded to the nearest integers using `np.round(...)` to produce discrete count estimates.
- **Type Conversion:** The rounded outputs are converted to integer type to conform to the expected output format.

3.1.5 Result formatting and saving

For each test image:

- The predicted counts for all 13 chocolate classes are collected.
- The results are organized into a `pandas.DataFrame`, where each row corresponds to an image and each column corresponds to a chocolate class.
- The final DataFrame is saved as a CSV file named `submission.csv`, with the first column containing the image ID and the subsequent columns containing the predicted counts for each chocolate type.

3.2 Kaggle results and major milestones

We conducted multiple submissions on Kaggle throughout the project, our group had one of the highest amount of submissions with more than 40 in total, indicating our initial struggles but willingness to improve. These can be categorized into three key milestones, each representing progressive improvements in model architecture, training methodology, and evaluation practices:

1. **Initial baseline model – score:** ≈ 0.30

   - Architecture: AlexNet.
   - Loss Function: Cross-Entropy Loss.
   - Optimizer: Adam (no regularization).
   - No validation set.
   - Minimal training epochs, no data augmentation.

   This baseline submission primarily served to validate the data pipeline and verify the implementation of evaluation metrics. The limited performance highlighted the necessity for enhanced regularization, more complex architectures, an other optimizer and better training strategies.

2. **Intermediate model with data augmentation – Score:** ≈ 0.68

   - Architecture: ResNet18.
   - Loss Function: Mean Squared Error (MSE).
   - Optimizer: Adam with L2 regularization.
   - Validation set selected randomly.
   - Extensive but unoptimized data augmentation ($\sim$ 6000 samples).

   At this stage, a deeper architecture was adopted, and initial efforts were made to tune hyperparameters and apply regularization. The significant performance gain was largely driven by the introduction of data augmentation and of a better model, which improved generalization. This submission surpassed the median of the Kaggle leaderboard.

3. **Final refined model – Score:** ≈ 0.95

   - Architecture: ResNet18.
   - Loss Function: Mean Squared Error (MSE).
   - Optimizer: Adam with L2 regularization.
   - Early stopping with minimal required epochs.
   - Manually selected training (72 images) and validation (18 images) sets.
   - Moderately tuned data augmentation ($\sim$ 1152 samples).
   - GPU acceleration leveraged, achieving $\sim \times 50$ reduction in training time.

   This final submission integrated comprehensive refinements across all stages of development: from architecture selection and hyperparameter tuning to dataset curation and training optimization. The use of early stopping mitigated overfitting, and the application of GPU acceleration significantly reduced computational time, enabling more rapid experimentation. These combined efforts led to our highest-scoring submission and represented the peak of our model development lifecycle.

3.3 Qualitative analysis

3.3.1 F1 scores on training and validation sets

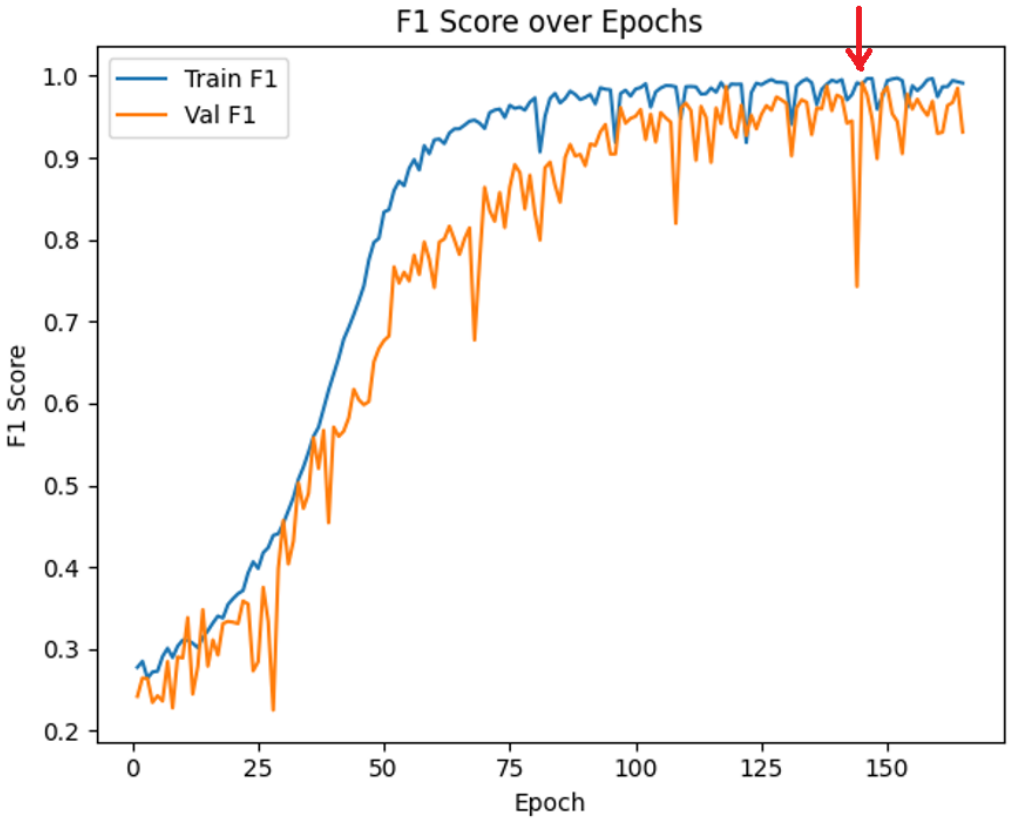

Figure 5: F1-score per epoch evolution. The red arrow indicates the best epoch chosen after early stopping.

A steadily increasing training F1-score accompanied by a stagnant or decreasing validation F1-score suggests potential overfitting. This behavior indicates that the model is increasingly tailoring its predictions to the training data at the expense of its generalization capability on unseen data. In our case and as indicated previously, we got the best validation F1-score after 145 epochs, then early stopping triggered at the epoch 165 since no improvement was achieved.

3.3.2 Loss function evolution

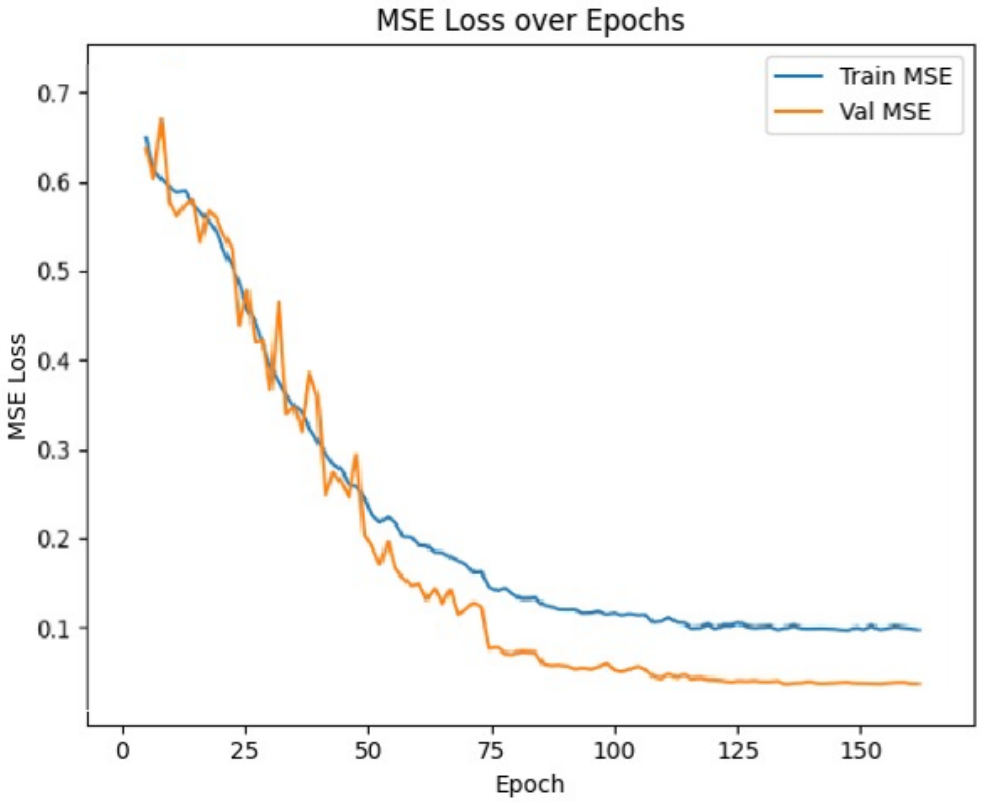

Figure 6: Training and validation loss per epoch.

A noticeable divergence between the training and validation loss curves after a certain number of epochs is a strong indicator of overfitting. While the training loss continues to decrease, suggesting improved fit to the training set, the validation loss either plateaus or increases, implying a degradation in performance on unseen data.

3.3.3 Per-class scores (multi-label setting)

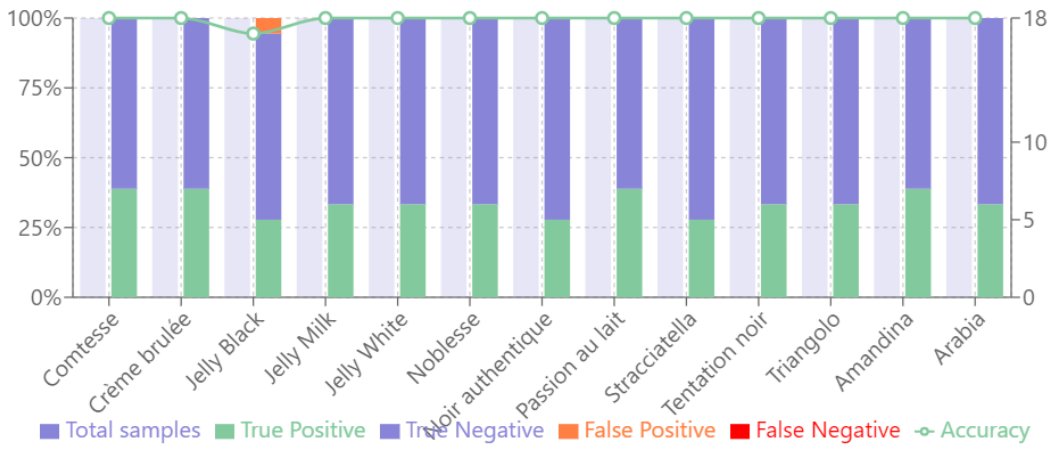

Figure 7: Score histogram (TP, TN, FP, FN) and accuracy, one per chocolate class (multi-label setting).

In a multi-label classification scenario, a separate confusion matrix is computed for each class. Each matrix presents True Positives (TP), False Positives (FP), False Negatives (FN), and True Negatives (TN), helping assess class-wise performance. Poor scores may indicate label imbalance or visual similarity between certain chocolates.

Key Observations:

- Perfect precision (1.00) for all items except Jelly Black (which has one false positive)
- Perfect recall (1.00) for all items (no false negatives) Accuracy ranges from 94.4% to 100% across all items
- Jelly Black is the only item with classification errors, probably because it resembles Jelly Milk

4 Conclusion

Model Optimization and Generalization. An alternative, lightweight variant of ResNet18 was also evaluated, in which the number of channels in each convolutional layer was reduced by half. This modification significantly reduced the model's complexity to approximately 2.8 million parameters, compared to the original 11 million. Despite the reduced capacity, this compact model achieved a higher validation performance, with a submission score of approximately 0.97%. We attribute this improvement to a reduction in overfitting, which allowed the model to generalize more effectively to unseen data. Although, this version was submitted on Kaggle, the report had already been finalized and the model weights were unfortunately lost during subsequent training runs.

Limitations and Future Improvements. An alternative data augmentation strategy was explored, namely CutMix. This implementation was unfortunately suboptimal, probably due to the fact that we only pasted chocolates on the backgrounds present in the training set, and did not paste any other foreign objects. Additionally, the training and validation set were not well implemented (no uniform distribution of chocolate types, backgrounds and no foreign object presence). This led the model to be easily fooled by non-chocolate type objects during testing. For future work, a more rigorous integration of CutMix should be pursued—ensuring that the augmented samples preserve distribution integrity of the original training set. We expect this to enhance the robustness of the model by improving generalization under complex visual conditions. Additionally, an improvement for the detection of the Jelly Milk and Jelly Black classes would be to adopt a progressive training strategy. This would involve initially training the model on only these two classes to boost its sensitivity to their specific visual features and subsequently fine-tune the model on the full set of classes. Such a phased approach could help the network build a stronger representation for the underperforming classes before generalizing to the complete classification task.

References

[1] K. He et al. (2015), *Deep Residual Learning for Image Recognition*

[2] CS231n Deep Learning for Computer Vision, *Convolutional Neural Networks (CNNs / ConvNets)*, (Online) https://cs231n.github.io/convolutional-networks/

[3] A. Zhang et al. (2023), *Dive into deep learning*, (Online) https://d2l.ai/chapter_convolutional-modern/resnet.html

[4] Medium, *The curse of dimensionality*, (online) https://medium.com/vlgiitr/the-curse-of-dimensionality-15f950e519d2

[5] Neptune.ai, *Comparison of Different Loss Functions in PyTorch*, https://neptune.ai/blog/pytorch-loss-functions#Mean-Absolute-Error, accessed May 2025.

[6] GeeksforGeeks, *PyTorch Loss Functions*, https://www.geeksforgeeks.org/pytorch-loss-functions/, accessed May 2025.

[7] PyTorch Forums, *Multi-label regression with target tensor as normalized class counts*, https://discuss.pytorch.org/t/multi-label-regression-with-target-tensor-as-a-normalized-class-counts-c94054, accessed May 2025.