IAPR: Final project - Chocolate Recognition

EE-451 Image analysis and pattern recognition

Moodle group ID: 42
Kaggle challenge: Deep learning
Kaggle team name (exact): ”Group42”

Mathieu Schertenleib (313318)
Alice D’Agostino (403990)
Jarl Stephansson (402775)

Wednesday 21st May, 2025

1. Introduction

The goal of this project is to develop a model capable of identifying and counting different types of chocolates in images. This task presents several challenges, including variation in appearance, overlapping objects, and limited training data. A patch-level classification pipeline was created using a Wide Residual Network (WRN), which allowed us to extract a large number of training samples by dividing images into patches, and generate object counts from stitching together full predicted segmentation masks. This method proved very effective in leveraging our limited dataset, enabling the model to learn robust representations and deliver accurate predictions at the pixel level, ultimately yielding very satisfactory results on the object counting, and an additional good segmentation mask.

2. Dataset

The dataset consists of a diverse set of high-resolution images containing 13 chocolate types captured with different background conditions. Images may include various arrangements of chocolates, occlusions, and other real-world challenges. The dataset is split into training (90 images) and testing (180 images) sets. The provided training annotations specify the number of chocolates of each type present in the images. We randomly split the 90 training images into two subsets: 80% for training and 20% for validation.

2.1. Patch-Based representation

Due to the large resolution of the images and the limited number of training examples, we downscale all images by a factor of 8×, and then extract fixed-size 32×32 pixel patches from the resulting images (the reasoning behind the usage of patches is explained in section 3.1). This is done using a sliding window (with configurable stride) that extracts local regions. Each patch is associated with a class label, derived from manually created annotations (polygonal masks, manually created with [3]), which corresponds to the most frequent class within the patch area. Examples of patches are

shown in Figure 1.

Figure 1: Examples of patches extracted from one image in the training set.

2.2. Class Balancing

The dataset exhibits strong class imbalance, with many patches dominated by background and certain chocolate types appearing much less frequently. To take into account this unbalancing, we implement weighted random sampling at training time. In particular, we want 50% of training patches to be sampled from background and the remaining 50% to be uniformly distributed among the 13 chocolate classes. This rebalancing ensures that rare chocolate classes are oversampled, while over-represented classes like background are downsampled. The rebalancing happens during the data loading process, without altering the original dataset.

2.3. Data Augmentation

Given the limited size of the dataset, data augmentation plays a key role in improving generalization and diversity of training data, reducing possible overfitting. Each training patch undergoes random transformations like random horizontal flips, random 360° rotation, cropping to 32×32 patch size (after rotation, to eliminate boundary artifacts) and normalization using the mean and standard deviation of the training set. Initially, Gaussian noise was also added to the patches as an additional form of augmentation. However, in this way it was very hard for the model to distinguish between visually similar chocolates such as ”Comtesse” and ”Jelly White”, which differ mainly in subtle texture and pixel-level details, which were compromised by the noise. As a consequence, this type of augmentation was ultimately removed.

3. Model

3.1. Architecture

Classification vs Detection

We explored two approaches to predict the number of chocolates in an image: a YOLO-based object detector that predicts object locations and a ResNet-based patch classification model, which classifies small patches of the image. Our initial plan was to compare both models and choose the better-performing one. However, early results showed that the patch classification model significantly outperformed our tests with YOLO, so we focused solely on that approach.

CNNs vs ResNets

Images are a collection of large amounts of values displayed in a orderly pixel grid. When we train a deep neural network to infer useful features from the relationships between these pixels, we often use convolutional neural networks (CNNs). By learning filters, they preserve the spatial relationships between pixels and generally outperform simple MLPs for images [2]. Deeper layers progressively reduce the spatial dimension of the image, while increasing its number of channels.

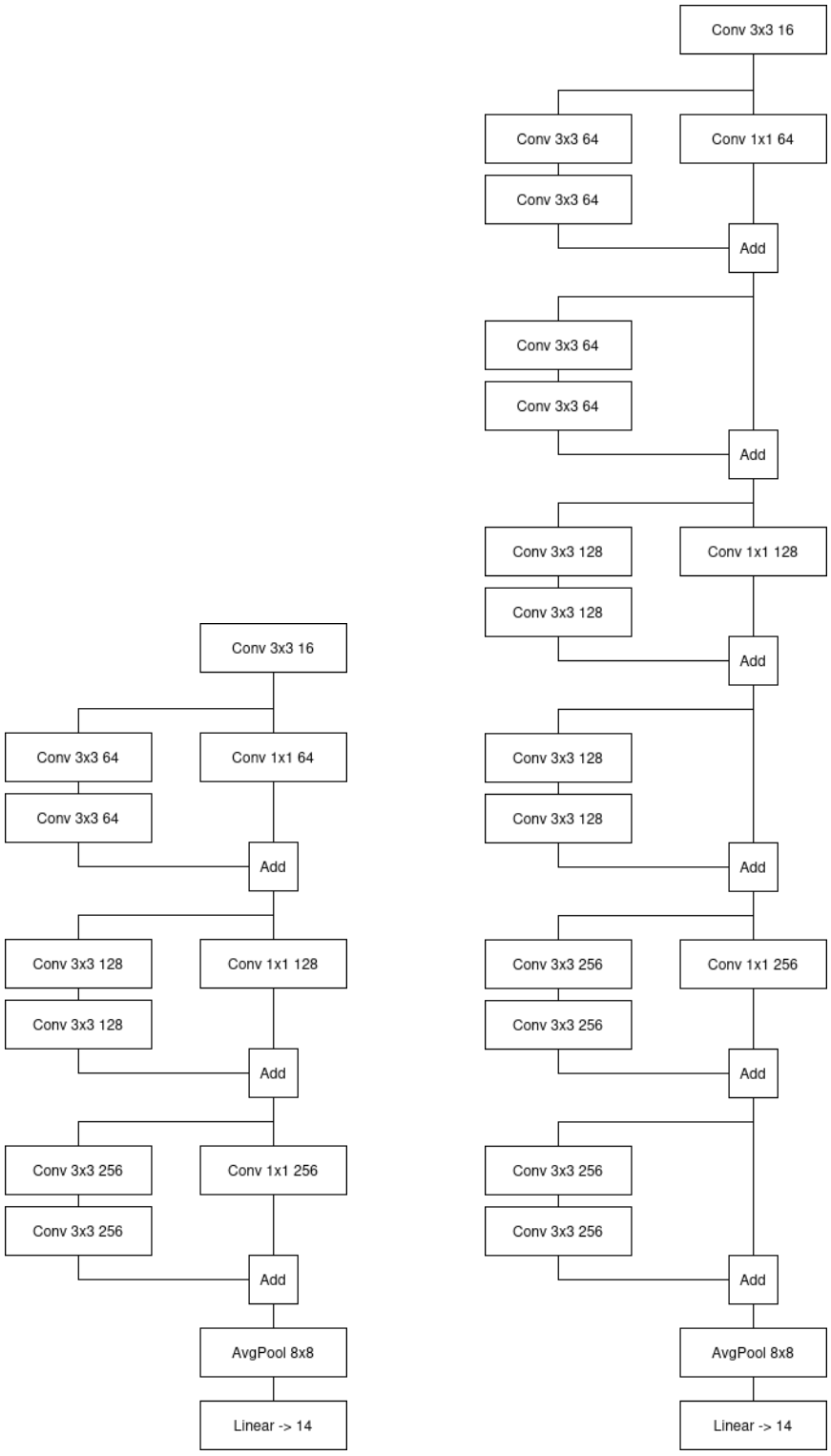
Compared to standard CNNs, ResNets with residual blocks let the network learn changes relative to the input rather than the full transformation. Instead of learning a full mapping from scratch, each block learns a smaller function, and adds it back to the original input using a shortcut connection [2]. This approach improves optimization and performance by enhancing gradient flow, allowing the model to learn more effectively from the data.

Residual blocks allow neural networks to be extremely deep while maintaining strong performance and gradient flow. However, increasing depth comes with drawbacks, it significantly increases training time and leads to diminishing feature reuse, where additional layers contribute less useful information. A solution to this is to expand the width of the network instead of its depth. Expanding the width means increasing the number of feature maps (channels) in each convolutional layer as proposed by [1] compared to the baseline ResNets.

Following this literature, we chose to follow the architecture of a "Wide Residual Network" from [1]. We select rather small versions of WRN based on our limited number of parameters and the relatively easy task of classifying 14 classes. We performed experiments with WRN-10-4, WRN-16-4 and WRN-22-4. We use the same nomenclature as the paper, where the first number is the total number of convolutional layers and the second one is a scaling factor on the number of channels, compared to the basic ResNet. We show in figure 2 the WRN-10-4 and WRN-16-4 architectures.

Our three tested models (all independently) fall way under the 12M parameter limit, with 1'199'838 learnable parameters for WRN-10-4, 2'749'918 for WRN-16-4 and 4'299'998 for WRN-22-4. We want to test different models of increasing complexity to be able to assess whether larger models offer increased classification over the smaller ones, stemming from the observation that our smallest model is still considerably expressive with over 1M parameters.

The goal of this model is to classify small patches into one of 14 classes (the background + all chocolate classes). If we run the model on all of the (overlapping) patches forming the image, we can stitch them together and reconstruct a full image of class predictions per pixel (essentially a segmentation, with its resolution dictated by the stride between patches and the size of each patch). We can then simply post-process this mask into object counts (see section 3.3).



(a) WRN-10-4 model **(b)** WRN-16-4 model

Figure 2: WRN model architecture. Each convolutional block is a set of BatchNorm, ReLU and a 2D convolution layer.

3.2. Methods

The model is trained using the Adam optimizer. To stabilize training in the early stages and improve convergence, a custom learning rate schedule is applied. It consists of a warm-up phase within the first 2.5% of training steps, during which the learning rate increases linearly from 0 to the maximum value, and a decay phase when the learning rate decreases smoothly from the maximum value down to near zero following a cosine curve. The model is trained for 15 epochs with a batch size of 128 patches.

Since the class distribution is highly imbalanced, with the vast majority of the patches being background, we need to use a weighted sampler for the training batches, to include proportionally more of the patches containing chocolate parts. We weight samples as described in section 2.2. For the validation set (and final test set), no re-balancing is done, to keep an unbiased distribution representative of the true dataset distribution.

For each batch a forward pass computes the predictions (logits) and the cross-entropy loss between predictions and true labels is calculated. Training loss is averaged and logged at regular intervals. Validation is performed at the end of each epoch using the original, unbalanced and untransformed validation set. For evaluation, multiple metrics are computed:

- Cross-entropy loss
- Classification error, calculated as the percentage of incorrect predictions: $1 - \frac{\text{correct predictions}}{\text{total samples}}$
- Precision, recall, and F1-score, averaged across all 14 classes

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

3.3. Post-processing of the model outputs

The model output is a pixel-wise classification maps for each test image. The goal is to pass from this pixel-wise classification to a per-class object counts, which represent how many instances of each chocolate type are present in an image.

The pipeline of the process is the following:

1. Pixel-wise class prediction:
   When the model has been run over all the patches forming an image, we can stitch together the softmax class probability predictions (by averaging overlapping predictions). This gives us a 14-channel image representing each class probability. We take the class with maximum probability to get a single class prediction per pixel, which essentially gives us a full segmentation mask, with 0 being the background and 1-13 being each chocolate class.
2. Pixel counting per class:
   The 2D predicted class map is flattened into a 1D array to count how many pixels are predicted for each class. We divide this number by the average area of each chocolate type in the training set, computed based on the contour annotations. This gives us a single fractional ratio for each class per image, representing the number of instances of that class.
3. Deriving object counts from the ratio:
   We could just round that number directly, but it would probably not yield the best estimation, because the predicted mask tends to merge touching similar objects (due to the patch size), and there might be a general offset due to the uncertainty at the edges of objects. To remedy this, the final object count prediction is the following:

   $$n_c = \lfloor \frac{\hat{N}_c}{N_c} + \text{offset} \rfloor$$

   Where $\hat{N}_c$ is the predicted pixel count of class c, N_c is the average ground truth pixel count of class c, and offset is computed based on the validation set, selecting the value that yields the highest F1 on the class count predictions.

4. Saving the results:
   The final per-image predictions are saved to a CSV file in submission format, with each row corresponding to an image and each column to a chocolate class.

4. Results

4.1. Patch classifier

Training

The training loss curve of the patch classifier for the WRN-16-4 model in Figure 3 shows a consistent downward trend as the number of training steps increases, indicating that the model effectively learns from the data and that it is able to extract meaningful features to distinguish between different classes at the patch level.

Moreover, the corresponding decreasing trend in the validation loss curve in Figure 4 suggests that the learned features generalize well to unseen patches. This ensures that the classifier is not simply memorizing training samples, but is capturing discriminative patterns among the chocolate classes.

In addition to the loss curves, Figure 5 shows an upward trend in precision, recall, and F1-score. In particular, the increasing precision suggests that the model is producing fewer false positives over time and the increasing recall indicates improved detection of true positives. These positive trends are summed up in the F1-score.

To compare the performance of the three model configurations tested, we report the patch-level F1-score in Figure 6. The comparison shows that WRN-10-4 under-performs compared to the deeper variants. On the other hand, WRN-16-4 and WRN-22-4 have only marginal difference between them. We select WRN-16-4 as our best model due to its lower computational cost and good results.

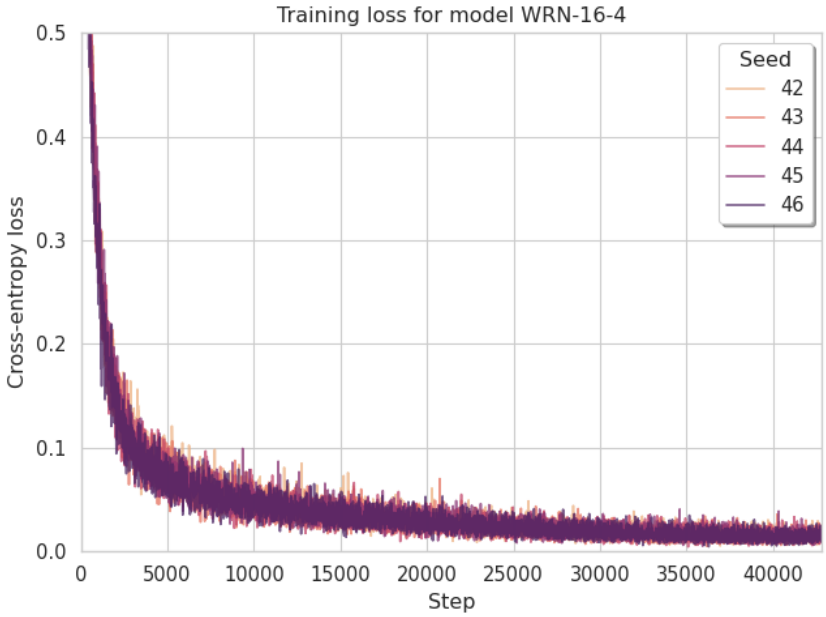


Figure 3: Training loss curve for the WRN-16-4 model. We plot the curve of each of 5 runs, to properly display variance due to the train/validation split and training stochasticity.

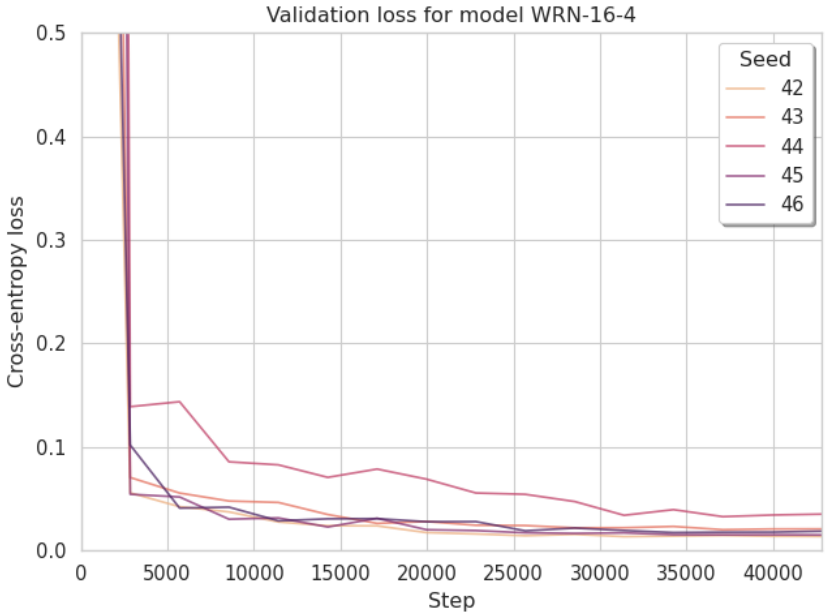


Figure 4: Validation loss curve for the WRN-16-4 model. We plot the curve of each of 5 runs, to properly display variance due to the train/validation split and training stochasticity.

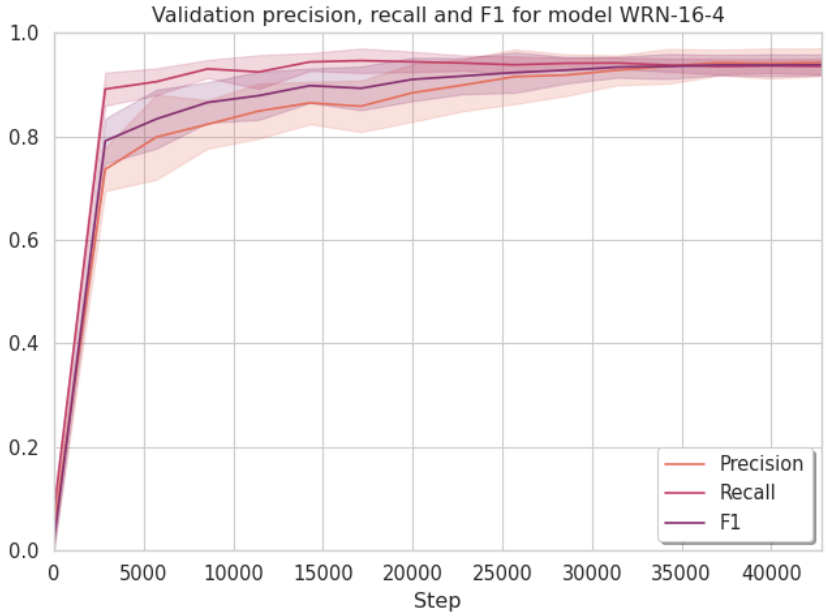


Figure 5: Precision, Recall and F1 curve (patch-level) for the WRN-16-4 model. We plot the average and ± 1 standard deviation confidence interval over 5 seeds.

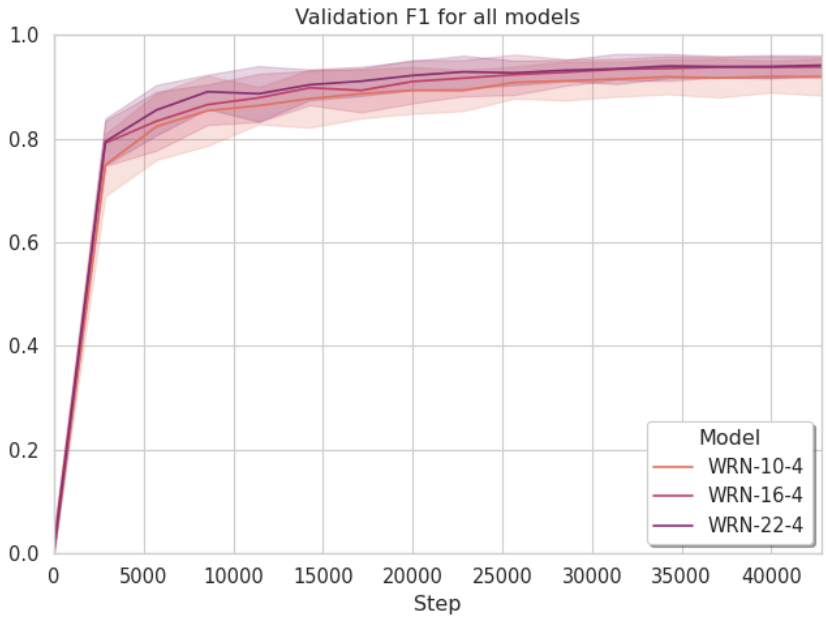


Figure 6: Comparison of patch-level F1 for all three tested models. We notice that WRN-10-4 is noticeably less accurate than the two other models, while WRN-16-4 and WRN-22-4 have only marginal difference between them.

Patch classification

We report the macro F1 (average of F1s computed per class) in table 1. Those are computed on the patch-level predictions, on the validation set. We include the metrics for all three tested models. Aligning with the above results, we notice that WRN-16-4 is noticeably better than WRN-10-4, while WRN-22-4 is only marginally better, but with a slightly worse recall. Due to these results as well as the higher computational cost of WRN-22-4, we selected WRN-16-4 as our final model, selecting the seed that yielded the best results.

Model	Precision	Recall	F1
WRN-10-4	0.907 ± 0.047	0.937 ± 0.017	0.919 ± 0.036
WRN-16-4	0.943 ± 0.027	0.935 ± 0.015	0.938 ± 0.020
WRN-22-4	0.953 ± 0.026	0.932 ± 0.013	0.942 ± 0.019

Table 1: Patch-level classification metrics on the validation set. For each model we report the mean and standard deviation of the metric over 5 independent runs with different seeds.

Pixel classification (segmentation)

On the validation set, we can run the model inference and reconstruct full images from the patches as explained before, and compute the classification metrics now at the pixel-level, on the predicted segmentation mask. We report those results per class in table 2. We notice that the background has excellent recall and very good precision, giving an excellent F1. This is expected due to the large number of background samples, and a good portion of them being of a relatively plain texture. The three Jelly classes have the worse recall of all, with Jelly Black having an average recall as low as 0.595. This gives an F1 around 0.76 for the three Jelly classes, while it is above 0.86 for almost all other classes. One general observation is that precision is very good for all classes, while recall is the real problem. This makes sense on a pixel level, because the inaccuracies come mainly from confusing a chocolate with the background (especially at the boundaries), not so much from differentiating between the chocolate types themselves, as can be seen in section 4.3.

Class	Precision	Recall	F1
Background	0.987 ± 0.001	0.998 ± 0.001	0.993 ± 0.001
Amandina	0.968 ± 0.004	0.833 ± 0.025	0.895 ± 0.015
Arabia	0.974 ± 0.018	0.779 ± 0.017	0.865 ± 0.008
Comtesse	0.973 ± 0.018	0.813 ± 0.018	0.886 ± 0.009
Crème brulée	0.972 ± 0.013	0.863 ± 0.022	0.914 ± 0.008
Jelly Black	0.982 ± 0.020	0.595 ± 0.031	0.741 ± 0.028
Jelly Milk	0.936 ± 0.059	0.651 ± 0.037	0.766 ± 0.020
Jelly White	0.993 ± 0.002	0.646 ± 0.037	0.782 ± 0.027
Noblesse	0.959 ± 0.036	0.782 ± 0.014	0.861 ± 0.018
Noir authentique	0.976 ± 0.013	0.831 ± 0.008	0.897 ± 0.006
Passion au lait	0.947 ± 0.036	0.796 ± 0.038	0.865 ± 0.035
Stracciatella	0.979 ± 0.010	0.792 ± 0.017	0.876 ± 0.007
Tentation noir	0.969 ± 0.025	0.735 ± 0.101	0.831 ± 0.064
Triangolo	0.941 ± 0.058	0.818 ± 0.004	0.874 ± 0.026

Table 2: Precision, Recall and F1 metric for the pixel-level classification of the stitched images per class, computed on the validation set. The mean and standard deviation over 5 random splits are reported.

Optimal offset

As described in section 3.3, we chose to include a parametric offset in our formula for computing the chocolate count from the pixel count. For this, we evaluate our validation set using different offsets from 0 to 1. For each one, we compute the overall object count F1 score as described on Kaggle, and plot them in figure 7. Following this result, we notice that any threshold from 0.75 to 0.95 yields a very good F1, and we select our optimal threshold as 0.85. It is also interesting to see that higher thresholds are generally better, up to 1 which makes all zero-count predictions become one. This indicates that the predicted masks are generally of a higher area than the ground truth, which can be partly explained by the blockiness of the patches and the "merging" of touching regions of the same class.

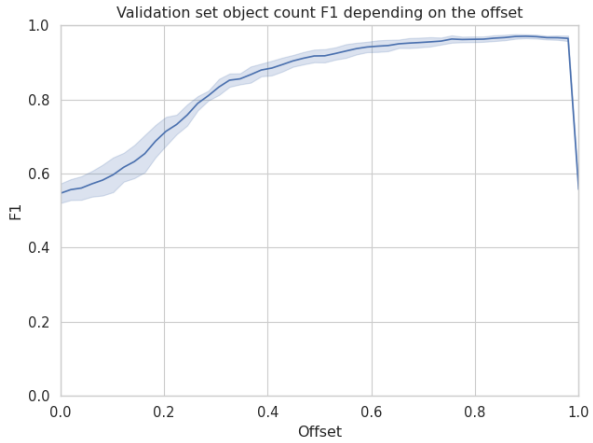


Figure 7: F1 score of the chocolate count predictions on the validation set, depending on the chosen offset described in section 3.3. The average of 5 seeds and its ± 1 standard deviation confidence interval is plotted.

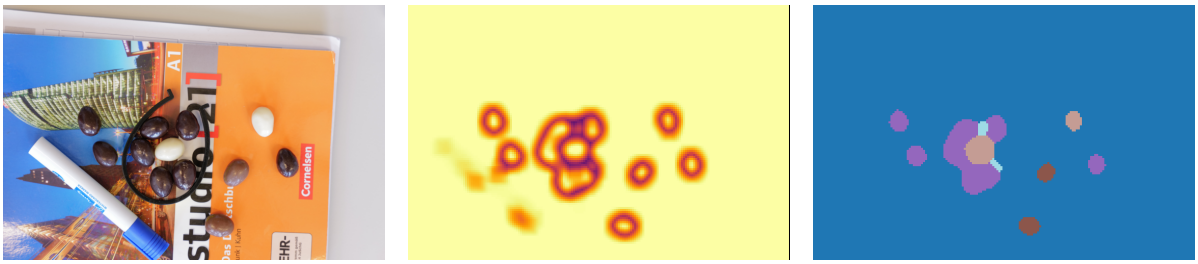
4.2. Final chocolate count predictions

In section 4.1.2 and section 4.1.3, all metrics reported were patch-wise or pixel-wise, which are very useful to assess the quality of the segmented prediction, but less informative about the final chocolate count prediction. This is because even a very bad segmentation mask could yield good count predictions, depending on the amount of post-processing. While our model does give a very good segmentation (section 4.3), it is that final count that forms the leaderboard metric. By running the

best model (WRN-16-4 with the best seed) on the pipeline described in section 3.3, we generate a final CSV file with class count predictions. We obtain an overall image-wise F1-score of 0.96973 on Kaggle submission, much higher than the baseline of 0.86632.

4.3. Qualitative results

To better understand how the model makes its predictions, we visualize the output for a specific example image in figure 8.



(a) Original image **(b)** Max class probability **(c)** Predicted class labels

Figure 8: Model outputs for image L1010018, showing the input, maximum probabilities, and class predictions.

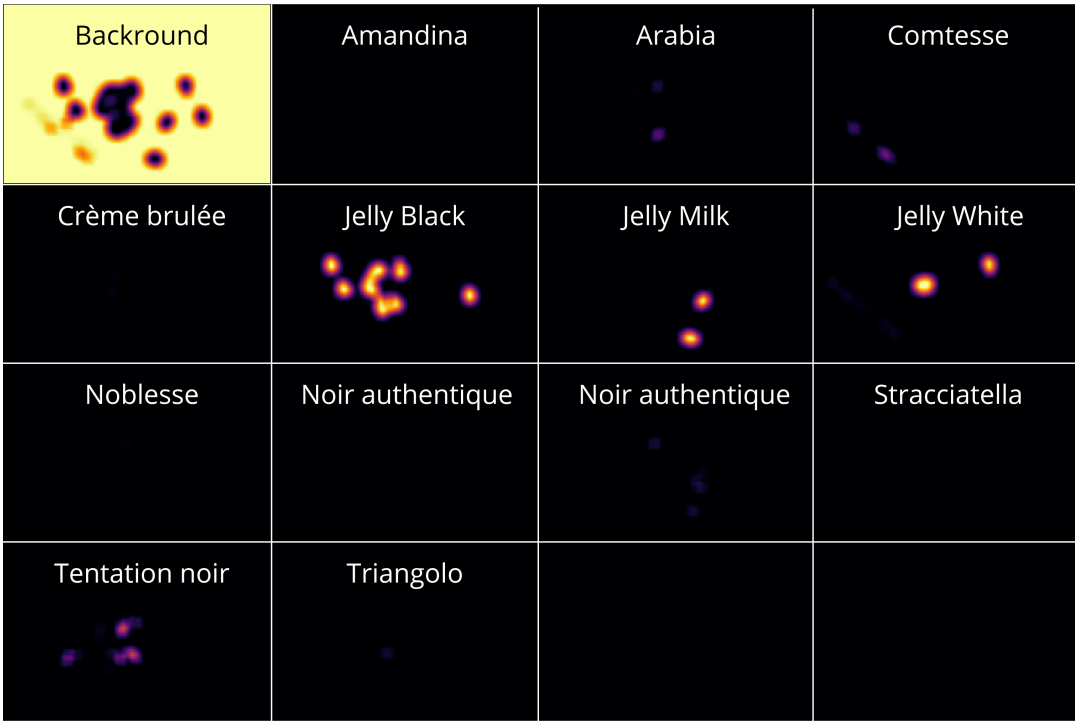


Figure 9: Predicted probabilities visualized separately for each chocolate class.

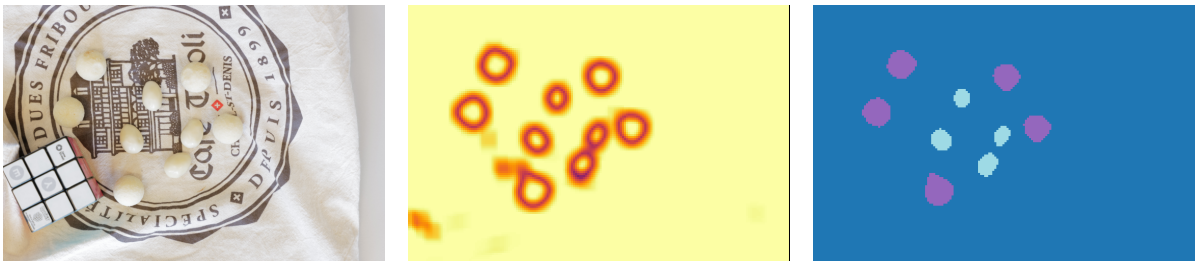
In figure 8, we can follow our prediction pipeline where we first see the models input in (a). This image is then feed to the model that outputs the probabilities for each class as seen in figure 9. In figure 8 image (b) we see the maximum of those probabilities followed by (c) where we see the models outputs.

When examining the model's output for this particularly challenging image, we observe that it behaves as expected. The model successfully segments all of the "Jelly Black" chocolates, even though they are positioned very close to one another. This allows us to later divide the predicted "Jelly Black" region based on the mean area of that chocolate type, resulting in an accurate count. The same holds true for the other chocolate types present in the image: "Jelly Milk" and "Jelly White", which are also correctly identified and segmented.

One issue becomes apparent when analyzing Figure 9, particularly in the predicted probabilities for the “Tentation Noir” class. The model assigns high probabilities to pixels located between some of the chocolates, which leads to those in-between areas being misclassified as “Tentation Noir” in the final prediction. As a result, the predicted region includes background pixels that don’t belong to any actual chocolate, ultimately producing an incorrect count for that class.

The figures above show that the model has learned to segment the different types of chocolates and the background, allowing it to later predict their labels with high accuracy based on the spatial distribution and appearance of each patch.

We show another example in figure 10, which is one of the hardest images with only white chocolates on an already white textured background. Een in this hard situation, the model is still able to pick differentiate between the two types by using small texture and color variations specific to each type.



(a) Original image **(b)** Max class probability **(c)** Predicted class labels

Figure 10: Model outputs for image L1010038, showing the input, maximum probabilities, and class predictions.

5. Discussion

5.1. Model limitations

While our model achieves strong results, as shown in the results section above, it also comes with several limitations.

One of the main challenges is the high computational cost, both during training and inference. Our patch-based classification approach allowed us to extract a large number of training samples from relatively few images, which helped prevent overfitting and improved the model’s ability to generalize. However, this also meant significantly more data to process, increasing training and inference time. For a single image, the model must run a forward pass on every individual patch before we can aggregate the predictions into a final result. This patch-wise processing introduces considerable overhead and slows down the overall prediction pipeline.

Another noteworthy limitation is the model’s limited ability to generalize to new conditions. Both the training and validation datasets were collected in a controlled environment with consistent lighting, fixed distance to the chocolates, and uniform camera angles. While this setup made it easier for the model to learn to count chocolates accurately within that specific setting, it also means the model is unlikely to perform well outside of it.

5.2. Performance

While the model outperformed the baseline in this project, its predictions are not without flaws. One potential reason lies in how we handle the foreground (chocolates) versus the background (non-chocolate areas). In our implementation, instead of predicting only the 13 chocolate classes, we included an additional background category. While this allows the model to distinguish between chocolate and non-chocolate regions, it also means the model must learn what the background looks like. From table 2 we can see that the model is actually the best at predicting the actual background but when the model encounters new images with background elements it hasn’t seen during training this will not hold. In such cases, the foreground may appear out-of-distribution relative to

the learned background, making it harder for the model to separate chocolates from their surroundings, potentially hurting performance.

5.3. Alternative object detection model failure

An object detection model was initially developed as part of our approach but was eventually abandoned due to underwhelming performance. A likely reason for the YOLO model's struggles is the substantial amount of data typically required to train object detectors effectively. Unlike the classification model, which could generate many training examples by dividing images into smaller patches, the YOLO model relied on full-image inputs. This significantly limited the amount of usable training data. Despite applying heavy data augmentation to compensate, the lack of diverse and well-annotated examples ultimately prevented the model from achieving satisfactory results.

References

[1] Sergey Zagoruyko and Nikos Komodakis. *Wide Residual Networks*. 2017. arXiv: 1605.07146 [cs.CV]. URL: https://arxiv.org/abs/1605.07146.

[2] Xia Zhao et al. "A review of convolutional neural networks in computer vision". In: *Artificial Intelligence Review* 57.4 (2024), p. 99. DOI: 10.1007/s10462-024-10721-6.

[3] Abhishek Dutta, Ankush Gupta, and Andrew Zisserman. *VGG Image Annotator (VIA)*. https://www.robots.ox.ac.uk/~vgg/software/via/. Accessed: 2025-05-19.