École Polytechnique Fédérale de Lausanne

Martin Daigne 340936
Guilhem Destriau 343728
Emanuele Rimoldi 377013

IAPR: Deep Learning Project

May 21, 2025

Contents

1 Abstract

The project aims to develop a system for automatic detection and counting of 13 types of chocolates from heterogeneous images, participating in the Kaggle Deep Learning Challenge. We chose a Faster R-CNN architecture with a MobileNet V3 Large + FPN backbone, equipped with a custom RoI head (`TwoMLPHead` with 512 units) and trained using a OneCycleLR scheduler and extensive data augmentation (crop, flip, color jitter, rotations, and blur). The model achieved an accuracy of 0.99604 on the private test set, surpassing the TA's baseline and demonstrating robustness on both neutral and "noisy" backgrounds. These results demonstrate the effectiveness of the deep learning approach and data augmentation techniques in improving the detector's generalization.

2 Data

2.1 Data Description

We are provided with 13 reference images and 90 train images. There are 13 different chocolates types. There can be multiple chocolates per images, different backgrounds, additional object, occlusion and blur. The chocolates can have different orientations on each image. Each image has weak labels of the number of time each class appears in each image.

2.2 Annotation

As we only have weak labels and we planned on training a detection model, we needed to annotate the data with bounding boxes. To put the bounding boxes with the class for each bounding box we used Roboflow as we could split the annotation task for each member. As we planned on using a YOLO model we exported the data in the YOLO format (`train/images/*.jpg` and`train/labels/*.txt`). Each `.txt` file contains the classes and bounding boxes present on the image e.g

```
1 0.4664066666666667 0.8204425 0.083635 0.1277625
8 0.5281066666666666 0.6384225 0.09620666666666666 0.1446925
9 0.5150216666666667 0.418495 0.10390166666666666 0.16278
```

With `<class_id> <x_center_normalized> <y_center_normalized> <width_normalized> <height_normalized>` at each line

2.3 Data Splitting and Augmentation

Because our dataset was both small and varied, making it prone to overfitting, we needed to augment it. We selected 10 passes (i.e 1403 images in the end) to balance increased diversity with reasonable processing time, resulting in 10+1 distinct versions of each original image.

First, we split our dataset in 80% training and 20% validation and then apply a sequence of random transformations. We run our `run_augmentation.py` script that will do the 10 passes. Each augmentation pass applies:

1. RandomResizedCrop robustness to scale and viewpoint variation
2. Horizontal and Vertical flips
3. random rotations up to 360 for arbitrary object poses to simulate orientation changes because the chocolate can be in any direction
4. brightness jitter [0.5–1.5] and contrast jitter [0.5–1.5] to mimic lighting variation
5. Gaussian blur (kernel 3–7, σ 0.1–2.0, p=0.3) to emulate focus shifts because some images are blurred in the dataset.
6. `SanitizeBoundingBoxes` realigns bounding boxes with the altered images and makes sure not to have degenerated boxes.

2.4 Data Sanitizing

Some transformations are too extreme and result in images that have no chocolates anymore. To tackle this, we run a small `garbage.py` script that removes the empty images and their corresponding labels. As this occurs randomly, we consider it does not change the 80/20 ratio much.

2.5 Data loader

As we initially wanted to train a YOLO model, we sticked to this format even for our Faster R-CNN model and we need to convert the data. Faster R-CNN requires absolute corner coordinates,thus our data loader converts normalized YOLO entries (cx, cy, w, h) into pixel values wrapped in a `BoundingBoxes` object with format `CXCYWH`, then transforms these into the `(xmin, ymin, xmax, ymax)` (XYXY) format within the custom `collate_fn`.

3 Pipeline Overview

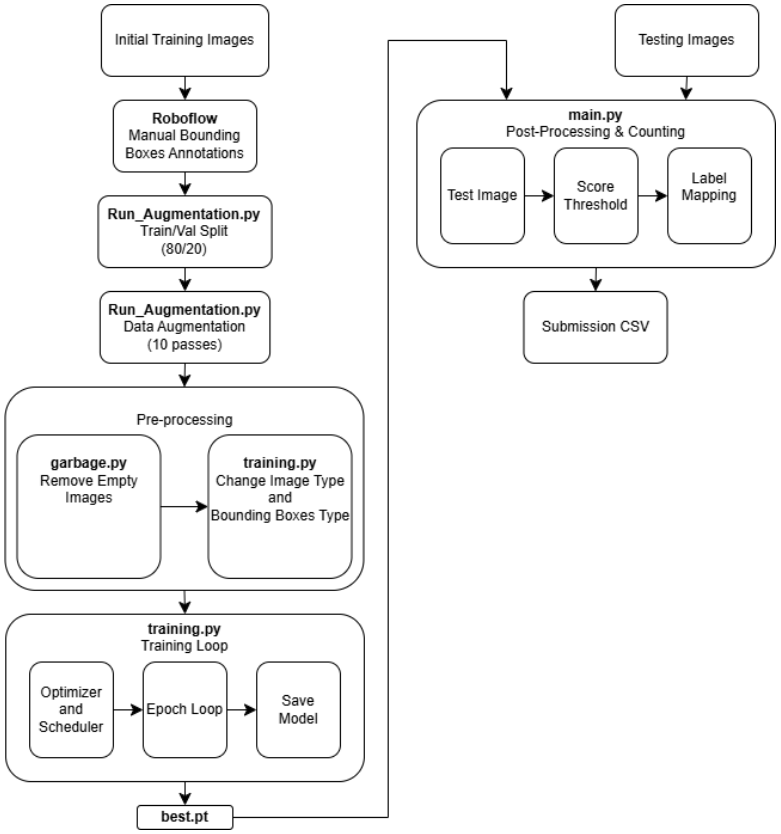


Figure 1: Pipeline

4 Model Architectures

4.1 YOLO12s

Initially, we experimented with YOLOv12, a state-of-the-art one-stage detector from the YOLO family, known for real-time detection and a good balance between speed and accuracy. YOLOv12 consists of a backbone for fast feature extraction, a neck that fuses multi-scale spatial features, and a detection head predicting bounding boxes, class probabilities, and objectness scores in one pass. The architecture is well-designed for general object detection tasks and benefits from pretrained weights and an end-to-end optimized training pipeline, as provided by the Ultralytics framework. However, we faced significant challenges in adapting YOLO12 to our specific problem. Since we were restricted to using Ultralytics only for architecture import and could not leverage its full training pipeline, implementing a custom training loop proved difficult. The model does not provide loss outputs by default, complicating optimization and hyperparameter tuning. Making it much harder to train manually compared to native pytorch models. Despite extensive efforts applying targeted data augmentations—including random resized cropping to 640×640 pixels, flips, brightness and contrast jittering, random rotations, and Gaussian blur—and training with SGD (momentum 0.9, initial LR 1e-3, weight decay 5e-4) combined with a warm-up and cosine annealing scheduler, the results remained suboptimal, with the best accuracy plateauing around 0.84

YOLOv12's underperformance in our task likely stems from several factors. Its design as a one-stage detector prioritizes speed and ease of deployment, but it struggles in scenarios typical of our dataset: densely packed chocolates, heavy occlusions, and numerous small objects with varying appearances and lighting. Additionally, without full access to the original training infrastructure, extensive hyperparameter tuning and large-scale pretraining—which YOLO models rely on heavily—were not feasible. Consequently, the model showed limited adaptability to the intricate visual complexity of our data.

4.2 Faster R-CNN + MobileNetV3 + FPN

To address the challenges posed by our dataset, we selected a two-stage detection approach with Faster R-CNN combined with a MobileNetV3-Large backbone and FPN. This architecture prioritizes precision, particularly in difficult conditions like small, occluded, or densely packed chocolates. While it operates at a slower inference speed compared to YOLOv12, it consistently delivered accuracy above 94% and exhibited more stable and reliable training behavior. The flexibility inherent to the two-stage design allowed us to better tailor the detection process to the complex visual characteristics and variable lighting of our images, ultimately leading to stronger generalization on the chocolate counting task.

Internal Preprocessing

Resizing the image for the architecture input (min 800pix^2 max 1333pix^2)

MobileNet V3 Large backbone

Depthwise separable convolutions to have a good parameters/performance ratio. Inverted residual blocks preserve representational capacity even at low width multipliers also for good parameters/performance ratio. Squeeze-and-excitation recalibrates channel importance, so the network can focus on chocolate-specific textures and colors surrounded by varied backgrounds. [Sandler et al. ("MobileNetV3," ICCV 2019)]

Feature Pyramid Network (FPN)

Builds a pyramid of multi-scale features to detect chocolates at different scales (small jellys vs. Amandino), especially under random crops and rotations.

Region Proposal Network (RPN)

By generating a limited set of candidate regions via an AnchorGenerator and a lightweight Conv->ReLU head, the RPN focuses computation on promising areas. Basically, it learns where chocolates likely are, even in clutter or partial occlusion, by scoring anchors at multiple aspect ratios and scales.

RoIAlign

Extracting fixed-size (7×7) feature patches from each proposed region, RoIAlign preserves spatial alignment even when chocolates overlap or lie at the image edges. This precise pooling is essential for the future classification and bounding-box tightning.

TwoMLPHead

The high-dimensional pooled features (12 544 values) are compressed to a 512-dimensional embedding through two ReLU-activated fully connected layers. This dimensionality reduction lowers overfitting risk on our small dataset while retaining enough capacity to distinguish among 13 chocolate classes.

FastRCNNPredictor

Mapping the 512-dimensional embedding to class scores (14 outputs: 13 chocolates + background) and bounding-box deltas (56 outputs) in a single linear layer keeps the head lightweight and efficient.

Internal Non-Maximum Suppression (NMS) and Intersection over Union (IoU) threshold

Minimal NMS and IoU to reduce the number of False Positive at output.

5 Training

We used PyTorch's OneCycleLR learning rate scheduler with a SGD optimizer and Nesterov momentum. The learning rate increases quickly during a short warm-up phase, then gradually decreases following a cosine schedule. This strategy improves convergence and helps prevent overfitting. All parameters can be found in the code below.

```
optimizer = torch.optim.SGD([
        {"params": decay, "weight_decay": 5e-4},
        {"params": no_decay, "weight_decay": 0.0},
    ], lr=0.002, momentum=0.9, nesterov=True)

scheduler = torch.optim.lr_scheduler.OneCycleLR(
    optimizer,
    max_lr=0.02,# peak LR
    total_steps=total_steps,
    pct_start=0.1, # 10% of steps for LR warm-up
    anneal_strategy="cos", # cosine annealing down after the peak
    div_factor=25,# initial LR = max_lr/div_factor
    final_div_factor=1e4, # final LR = max_lr/final_div_factor
)
```

The model was trained for up to 100 epochs with early stopping triggered after 10 epochs without validation improvement. Both the training and validation losses were computed from a weighted sum of the four standard Faster R-CNN loss components:

- **Classification loss:** penalizes incorrect class predictions for each proposed region
- **Objectness loss:** evaluates whether a region contains an object or not, used by the Region Proposal Network (RPN).
- **Box regression loss:** measures how well the predicted bounding boxes align with the ground truth.
- **RPN box regression loss:**penalizes inaccurate bounding box proposals generated by the RPN. It helps the model generate high-quality candidate regions before classification.

We experimented with different weight combinations for these components to find the best trade-off between detection accuracy and bounding box precision. In the end, we empirically selected weights that placed more emphasis on classification and objectness, as these were more critical to the overall detection quality in our specific context. The final loss is :

```
loss = (1*loss_dict["loss_classifier"]+ 0.8*loss_dict["loss_objectness"]
                    + 0.5* loss_dict["loss_box_reg"]+ 0.1*loss_dict["loss_rpn_box_reg
                        "])
```

During training, we monitored both the training and validation losses and saved the model checkpoint that gave the best validation performance. We also stored the training history to visualize the learning curves and determine the best epoch for final evaluation.

In figure 2 is the loss during the training of our best model:

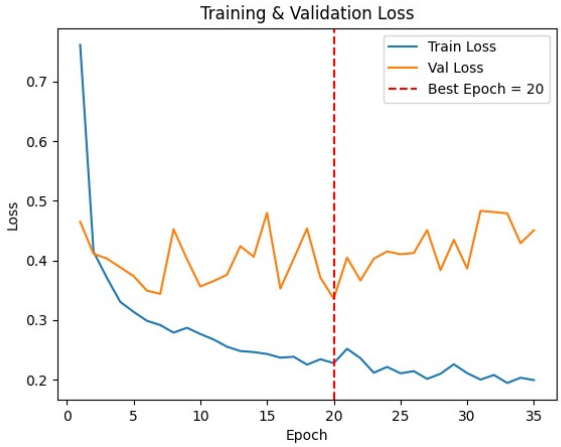


Figure 2: Training losses

The graph shows the evolution of training and validation loss over the epochs. We observe that the training loss decreases steadily, indicating that the model is learning effectively on the training data. However, the validation loss fluctuates and begins to increase after epoch 20, suggesting the onset of overfitting. Epoch 20, marked by a red dashed vertical line, represents the point where the model achieves its best performance on the validation set1. Therefore, early stopping at epoch 20 would help maintain a good balance between learning and generalization.

6 Inference - Submission

For inference, we used the trained Faster R-CNN model to predict the presence of chocolates in the test images. All test images were first resized and normalized to match the input format expected by the model. We used a custom `TestDataset` to load the test images and associate each one with its corresponding ID.

The model was loaded in evaluation mode with the best weights obtained during training. For each image, the model returned a set of predicted bounding boxes along with class labels and confidence scores.

To convert the model outputs into submission format, we applied a confidence threshold of 0.6 to filter out low-confidence predictions. For each remaining detection, we incremented the corresponding chocolate class count using a predefined mapping (`model2idx`) to align internal class indices with the expected Kaggle submission format. The confidence threshold has ben found by comparing manually the results of multiple thresholds (0.3,0.4,0.5,0.6,0.7), 0.5 detected too much and 0.7 not enough and 0.6 has been found to be the best.

The final predictions were stored in a DataFrame, where each row corresponds to one image and contains the predicted counts for all 13 chocolate types. The output was saved as a CSV file named `RCNN_submission.csv`, sorted by image ID and ready for submission.

7 Analysis

Now that we have a well performing model we can analyze how it responds to an input to see if it learned meaningful features

7.1 Convolution Heat Maps

It is interesting to look at how the convolution layers react to an input image:

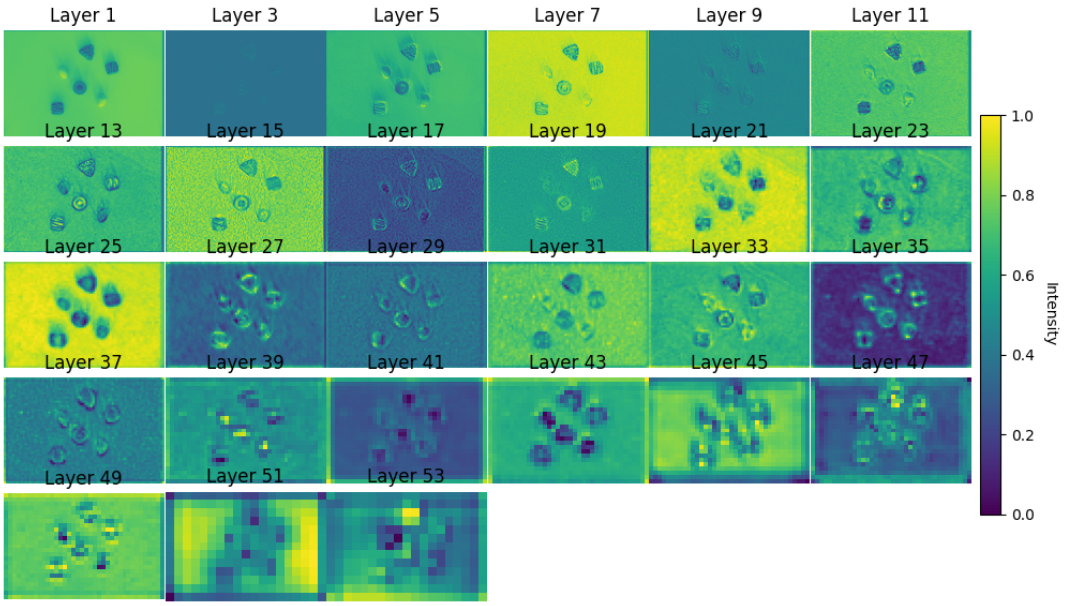


Figure 3: Convolutional Layer Activation Heatmaps

We observe that

- Early layers (Layer 1–9) highlight simple edges and color gradients.
- Mid-level layers (Layer 11–25) begin to focus on shape primitives like circles
- Deep layers (Layer 27–53) are increasingly coarse spatially, but respond strongly in the regions of interests. Either only the background or the chocolates specifically.

This tells us that our model is learning appropriate hierarchies of features: low-level features first (textures/edges), then larger but focused on interest regions activations deeper on. If the deepest layers were still activated at random corners of the image, that would signal that the model has not specialized. But this is "luckily" not the case. We thus don't need a deeper model.

7.2 Box predictor feature space

We can also look at other layers, especially at the last ones where detection has been done and we have each chocolate proposition represented as a big feature vector. In our case we will look at the box predictor of the ROI head which represent each proposition as a vector of size 512. Because such high dimension is difficult to visualize we will use Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE) to map the data to a 2D or 3D space.

PCA projects the data on the axes of maximum variance, t-SNE on the other hand tries to preserve local structures by modeling the pairwise similarities between points and embedding them in a lower-dimensional space where similar points stay close together. t-SNE has the advantage of separating nonlinearly contrary to PCA.

On figure 4 we observe that PCA did not separate the classes, they are all confused whereas on the t-SNE pot we clearly see well separated classes. Indeed, PCA explained variance of the two first components is only around 5% so these two directions alone capture only a tiny fraction of the total data variance, resulting in overlapping clusters when projected onto the plane. In contrast, t-SNE prioritizes the preservation of local neighborhood relationships. On the plot, each class forms a compact, well-separated group in the two-dimensional t-SNE embedding, showing the low intra-class and high inter-class variance encoded by the network weights. With such distinctly clustered representations, the final softmax layer can assign class labels with high confidence and minimal misclassifications.

On figure 5 we see the same result but with an addition dimension.

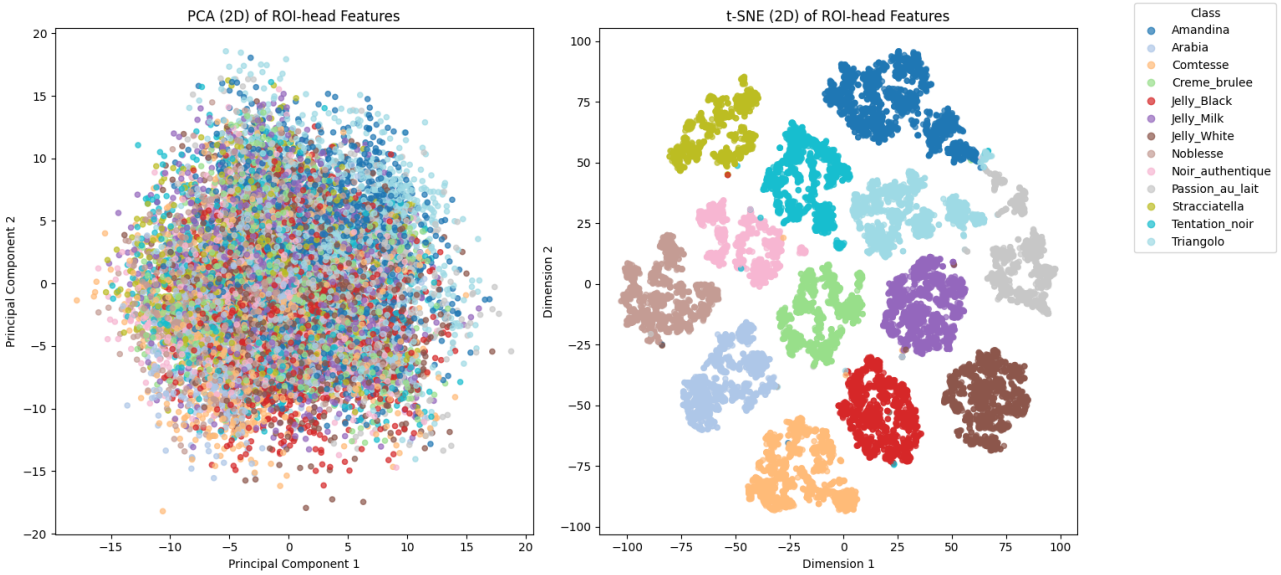


Figure 4: 2D plot

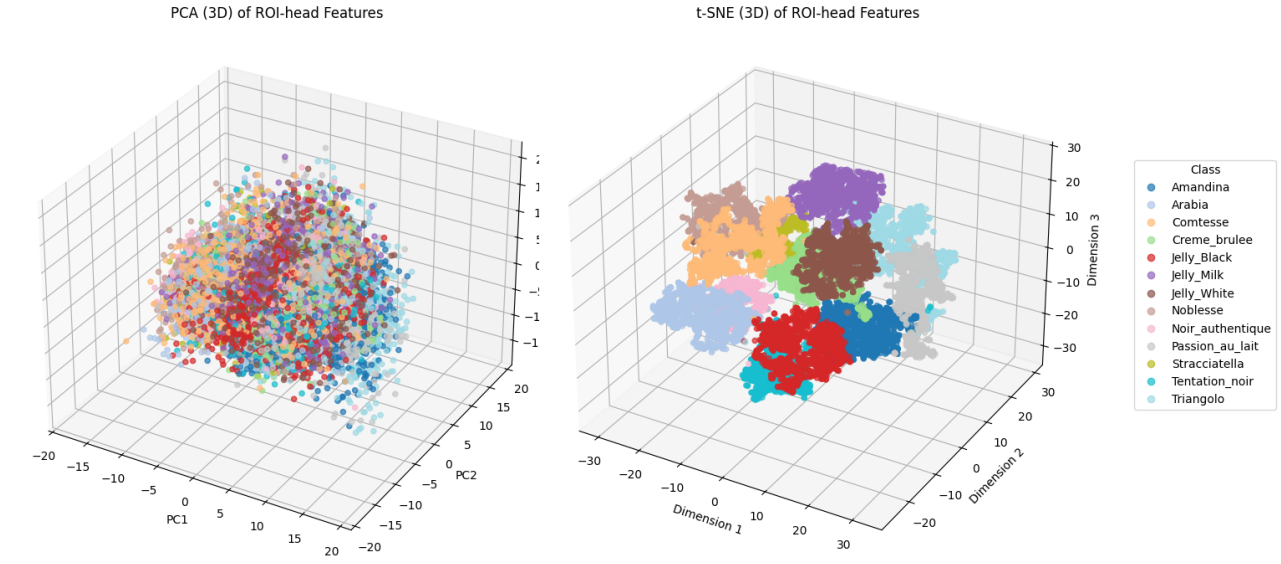
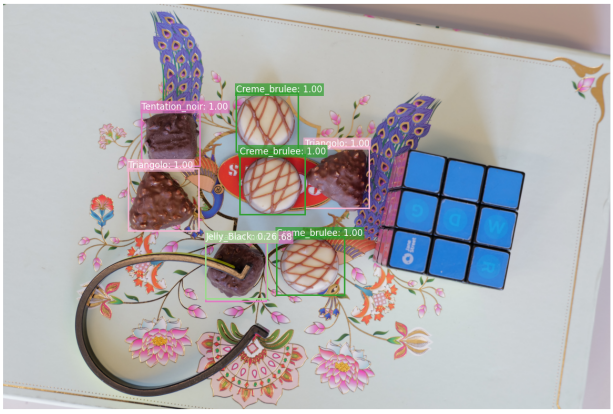
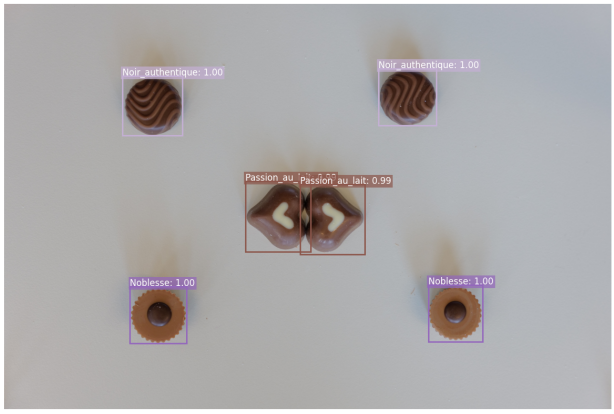


Figure 5: 3D plot

8 Gallery



(a) Colors, background and obstruction



(b) Close

(c) Not fooled by the shape, texture is important

(d) Misclassification, thinks it is comptesse

Figure 6: Raw model output **before** the confidence thresholding