Kaggle challenge: Sartorius – Cell Instance Segmentation

Hamdi Bel Hadj Hassine
MVA 2022
hamdi.belhadjhassine@ensae.fr

Sami Amrani
MVA 2022
amsami97@gmail.com

Abstract

In this project we provide an implementation of a cell segmentation model that we developed as part of the Sartorius Cell Segmentation Kaggle challenge. We present in this report the main models used in the competition and our methodology for the model choice and optimization. We also provide a series of experiments that allowed us to assess how different model configurations and data processing techniques impact the segmentation performance.

1. Introduction and motivation

This project was our first step into Deep Learning and we chose to work on the Sartorius Kaggle Challenge because it was an opportunity to work on an interesting computer vision challenge that allows us to try state-of-the-art segmentation models and to compare our results with other competitors.

This Kaggle challenge is proposed by the biopharmaceutical company Sartorius and ran from October 15 to December 31. The goal of this challenge is to build an instance segmentation model that can segment 3 types of neuronal cells in microscopy images. Cell detection and segmentation is a recurrent task in computer vision with important applications in biology and pharmaceutical research ranging from disease detection to drug effect monitoring.

2. Problem Definition

2.1. Instance Segmentation

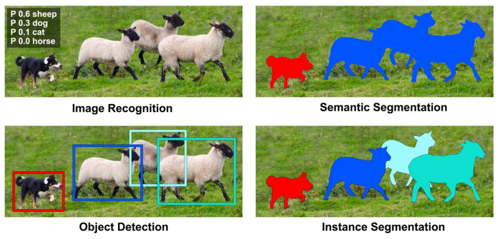


Figure 1: Different computer vision tasks

Our to task in this challenge is to build an instance segmentation model. This means that our model should be able to produce a separate binary mask (i.e. partition of pixels) for every instance in the image, in opposition to image recognition which only provides the probabilities of objects being in the image, semantic segmentation which provides one mask per detected class, and object detection which provides bounding boxes of the objects. In this respect, instance segmentation is the most challenging task since we need to provide a precise and separate mask for every instance in the image.

2.2. Used Data

The data provided in this competition consists in microscopy pictures of cells. Each picture contains multiple cells belonging to one of three types: Neuroglioblastoma, Neurons and Astrocytes. The images are grayscale and have the same size 704×520.

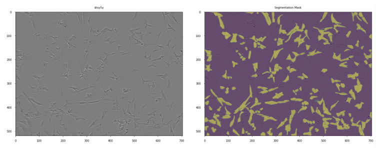


Figure 2: Neuroglioblastoma

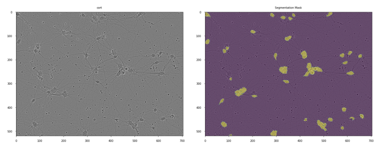


Figure 3: Neurons

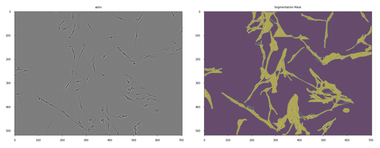


Figure 4: Astrocytes

Training data consists in 606 images and their mask annotations provided in RLE (run length encoding) format. Although 606 seems to be a low number of images to train a neural network, the actual number of instances (cells) in those images is 73585 which should be high enough to train a model to efficiently recognize them. The competition host also provides additional data that can be used to build the models:

- 1972 images without annotations, mainly for self-supervised learning.
- The LIVECell dataset containing 5239 annotated images, but it does not contain the same types of cells.

2.3. Model Evaluation

In this project we will evaluate our models using the mean Average Precision (mAP) metric which is the proposed evaluation metric in the Kaggle challenge and a standard metric in recent image segmentation research. First we define the IoU (intersection over union) of a segmented object A relative to a ground truth object B as $IoU(A, B) = \frac{A \cap B}{A \cup B}$. For a given threshold t, the detection is considered a true positive (TP) if $IoU(A, B) > t$ for any ground truth B in the image, otherwise it counts as a false positive (FP). We also define false negatives (FN) as the ground truth objects B that do not match any predicted object A. The precision for an image is then given by $\frac{TP(t)}{TP(t)+FP(t)+FN(t)}$. By averaging the precision over multiple threshold values $t \in \{0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95\}$ we obtain the Average Precision AP and then the mAP by averaging the AP over all the images in the test set. This results in a representative evaluation metric for the predictions.

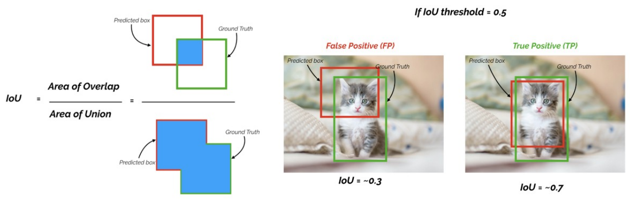


Figure 5: IoU computation

It should be noted however that the choice of the thresholds is important and we believe that the competition organisers shouldn't have included thresholds above 0.8 for all cell types. In fact, for small cells, the cell boundaries can be ambiguous and annotations aren't pixel-perfect. As a result this randomness can cost more than 0.2 IoU for the same prediction as shown in Fig. 6. This means that reaching the 0.8, 0.85, 0.9 and 0.95 thresholds can be mostly random for small cells, therefore the reliability of the competition's scores is questionable.

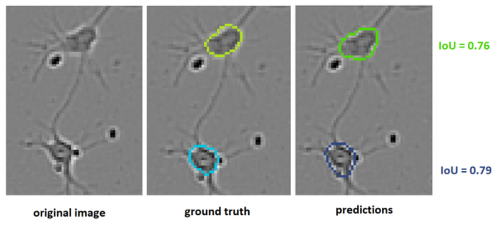


Figure 6: Example of IoU variability on small cells

3. Related work

3.1. LIVECell

The competition host company Sartorius has previously released the LIVECell dataset containing microscopy cell images in the same format as this competition's data (but with other types of cells) and published the article LIVECell—A large-scale dataset for label-free live cell segmentation[1] which discusses cell instance segmentation and provides the results obtained by two instance segmentation models on the LIVECell dataset: Cascade Mask R-CNN and CenterMask, which both achieved good and similar results.

3.2. Data Science Bowl 2018

The Data Science Bowl 2018 was a Kaggle competition aiming to segment cell nuclei in microscopy images. The article Nucleus segmentation across imaging experiments: the 2018 Data Science Bowl [2] provides an analysis of the competition's results. The main takeaway is that most high-performing solutions relied on heavily engineered models like the first solution which used an ensemble of 32 U-Net neural networks, the second solution which used a hybrid model incorporating a Mask R-CNN encoder with a U-Net decoder, and the third solution relied on heavy data augmentation for both train and test data and used a Mask R-CNN model.

3.3. Cellpose

A relevant paper for our task is Cellpose: a generalist algorithm for cellular segmentation [3]. This paper introduces a new cell segmentation model ”Cellpose” and claims that it beats state-of-the-art models like Mask R-CNN and Stardist. Cellpose is based on the U-Net architecture and uses a flow field representation of the masks to better disentangle the cells. A more thorough description is provided in the next section.

4. Methodology

Our methodology in this project consists in selecting a good-performing model then performing a series of experiments with its parameters and configuration in order to improve its performance.

4.1. Model review

From the literature review we identified the most promising models: Mask R-CNN, Cascade R-CNN , U-Net and Cellpose. We also reviewed the Kaggle discussions of the competition and it appears that U-Net didn't perform as well as the other models with the competition's data, so we decided to discard it. First we studied the models, and we provide a description of their architecture and characteristics below (we provide a more thorough description of Mask R-CNN since it is the main model that we retained).

4.1.1 Mask R-CNN

Mask R-CNN [4] is one of the most popular approaches for instance segmentation today. It was developed by building on the on successive improvements of object detection models starting from RCNN [5], then Fast RCNN [6], then Faster RCNN [7], then Mask R-CNN [4] (and incorporating an FPN network [8]), and it is generally considered as state-of-the art in instance segmentation tasks. Mask R-CNN performs instance segmentation over two stages: First it searches for ROIs (regions of interest) in the image, then it classifies the objects in the ROIs and generates their masks.

Backbone and FPN:

The first component of the model is the backbone, which is a convolutional neural network (e.g. ResNet50, ResNet101, VGG, etc..) that converts the RGB input image into a feature map, typically using convolution and pooling layers. Instead of directly using the feature map from the backbone, Mask-RCNN uses a more sophisticated architecture called Feature Pyramid Network (FPN) [8] which also extracts intermediate features from the backbone as illustrated in Fig. 7[1].

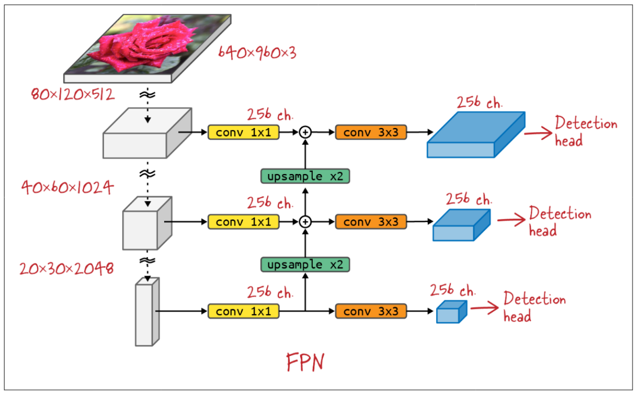


Figure 7: Feature Pyramid Network architecture

The reasoning behind this idea is that feature maps from the first layers of the CNN backbone are higher resolution and contain better localization of the objects, while features from deeper layers contain higher-level semantic information that can detect complex objects. In order to combine those two advantages, feature maps from deeper layers are brought to the same shape as the ones from preceding layers using 1x1 convolutions (to reduce the number of channels) and nearest-neighbour upsampling (to increase spatial resolution), then they are summed up. A 3x3 convolution is also performed at the end to smooth out the upsampling. By doing this, we obtain multi-level feature maps that combine activations from both the first, higher resolution layers and the deeper layers. Mask R-CNN then dynamically chooses and passes one of the feature maps to the detection head, depending on the size of the objects to be detected.

Region Proposal Network:

The next component of Mask R-CNN is the Region Proposal Network (RPN). It is a simple network whose purpose is to classify whether a given anchor contains objects or background. Anchors are defined as boxes with varying sizes and aspect ratios at regularly spaced locations within the input image (or feature map). A simplified example of anchors is provided in Fig. 8, but in practice the number of anchors is in the order of 10^5.

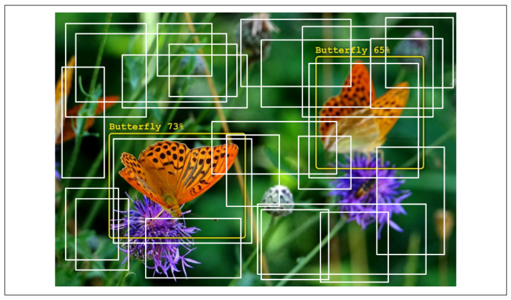


Figure 8: Examples of anchor boxes

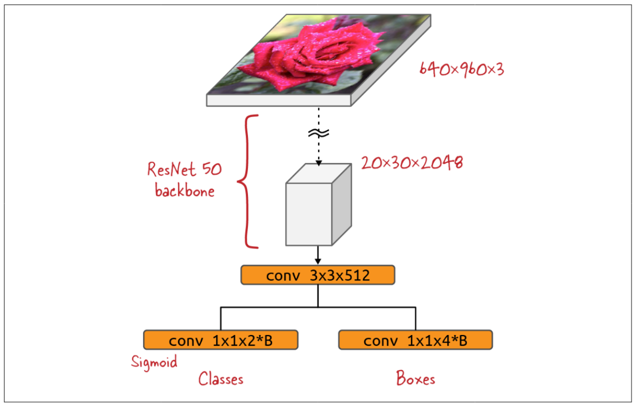


Figure 9: Region Proposal Network architecture (B is the number of anchors per location)

[1]Figures of Mask R-CNN section are taken from the Practical Machine Learning for Computer Vision book

An example RPN is illustrated in Fig. 9. It takes as input a feature map, either directly from the backbone as in the figure or from one scale of the FPN outputs, and an anchor location. It passes the anchor features through a simple convolutional net and two heads: one which outputs a binary class (object/background) and one which predicts the bounding box for the object. The loss of the first head is cross-entropy, while then second uses a smooth L1 ("Huber") loss: $L_{\mathrm{loc}}\left(t^{u}, v\right)=\sum_{i \in\{\mathrm{x}, \mathrm{y}, \mathrm{w}, \mathrm{h}\}} \mathrm{L}_{1,smooth}\left(t_{i}^{u}-v_{i}\right)$ where $\mathrm{L}_{1,smooth}(x)= \begin{cases}0.5 x^{2} & \text{if } |x|<1 \\ |x|-0.5 & \text{otherwise}\end{cases}$

Since the number of anchors is high, predicted bounding boxes are likely to overlap. To only keep one (best fitting) bounding box per object, non-maximum suppression (NMS) is applied to the obtained boxes: For every class, for every two boxes, compute their IoU (i.e. overlap) and if it is above a given threshold (usually 0.5), discard the box with lower confidence (lower object probability in the object/background classification).

After solving the overlapping problem, the output of the RPN is the top N box proposals (default N is 512), which are called regions of interest (ROIs) and serve as input for Mask R-CNN's prediction head. It is worth noting that the purpose of the RPN is not to produce high-quality bounding boxes but to provide "good enough" ROIs that will be refined in the next stage.

Prediction heads:

In stage 2, the first step is to choose which FPN level to take the feature map from, and it is computed as: $n = \mathrm{floor}(n_0 + log_2(\sqrt{wh}/224))$ where w and h are the width and height of a given ROI and n_0 is the FPN level where typical anchor box sizes are closest to 224 (usually $n_0 = 4$). The next step is to crop the ROI from the FPN feature map of level n and resize it to the default prediction head input size. Mask R-CNN paper authors remark here that when cropping the ROI from the feature map, its coordinates shouldn't just be rounded but the ROI should be resampled. For example if the ROI shape is 25x25 and its assigned FPN feature map shape is half the shape of the original image, we shouldn't crop a 12x12 or 13x13 ROI from the feature map, but instead we should sample a 12.5x12.5 ROI from the feature map using bilinear interpolation. This is also applicable to the ROI feature map resizing. The authors refer to this method as ROI alignment. As output, we obtain 7x7 and 14x14 feature maps for each ROI. The first is passed to a CNN with similar architecture to the RPN but deeper, and produces the classes of the detected objects and their bounding boxes. The notable difference is that this time the classification includes all the provided training classes and a background class (regions classified as background are discarded). The second feature map is passed to another CNN that, for each class, performs binary classification of every pixel of a resized version (28x28) of the ROI (the pixel belongs to the given class or not), resulting in one binary mask for each class. While this might seem redundant with the first classification done with the 7x7 feature map, it allows the mask prediction head to learn the features of every class and improves accuracy. Finally, the outputs of Mask R-CNN are the class and bounding box given by the first prediction head and the mask corresponding to the same class given by the second prediction head, resized to the original ROI shape. The complete architecture of Mask R-CNN is given in Fig. 10.

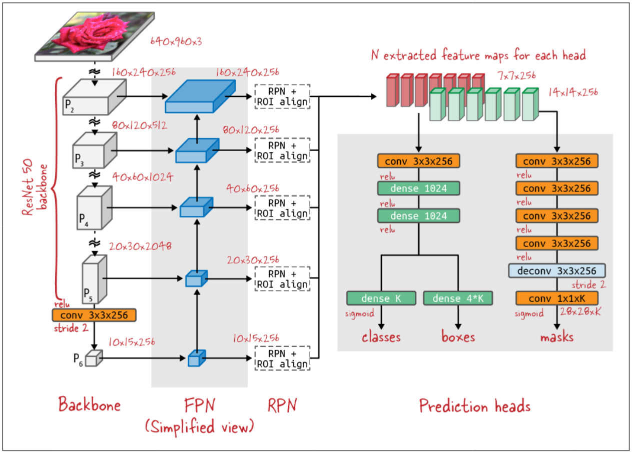


Figure 10: Mask R-CNN architecture

4.1.2 Cascade R-CNN

Cascade R-CNN [9] is another variation of Faster R-CNN which instead of using a single IoU threshold (usually 0.5), uses a sequence of detectors trained with increasing IoU thresholds, to be sequentially more selective against close false positives. The detectors are trained stage by stage, leveraging the observation that the output of a detector is a good distribution for training the next higher quality detector. The architecture of the prediction heads is similar to that of Mask R-CNN.

4.1.3 Cellpose

Cellpose [10] is a novative image segmentation model created specifically for cell segmentation. It is based on the U-Net architecture but instead of directly predicting binary masks, it creates an intermediate representation of a flow field directed towards the center of each cell. To generate that, it first calculates the center of mass of each mask, then all the other points represent an angle to flow from that point towards the center. This makes it easier to disentangle touching or overlapping masks - which is the main issue a regular U-Net faces.

4.2. Model selection

In order to select one of the three previously described models, we tested them and chose the model with the highest mAP on the public leaderboard data.

Testing the models is not trivial since there are many configuration parameters, hyperparameters and preprocessing steps that can influence the results. We had to make a compromise between optimising the models before testing them, which would be counterproductive because we would optimise models that in the end would be discarded, and testing them with the default settings which might not reflect the models' performance if they were well optimized. In the end, we chose to compromise by taking inspiration from the Kaggle discussions which contained evaluations of the models' parameters choice, e.g. the learning rate and the prediction score thresholds etc.. Therefore we trained a version of the models with most of the configuration either taken as the model's default or as the values suggested by other Kaggle competitors if we found such suggestions.

To train Mask R-CNN and Cascade R-CNN we used the Detectron2 library, which is a popular Python module developed by Facebook to bundle a set of computer vision processing tools and pretrained models. For Cellpose, we used the official Cellpose module to download and fine-tune the model.

Before training the models, we split the data into a training set (80%) and a validation set (20%) which will allow us to monitor our model's score during training and throughout the experiments that will be described below.

4.3. Model backbone

After the model selection step, for which the results and discussion are provided in the Evaluation section, we decided to retain Mask R-CNN as our segmentation model. The next step was to evaluate whether we should use a deeper convolutional neural network as the backbone (image feature extractor) of the model, and if the choice of the model backbone heavily impacts the performance of the model.

In the previous model selection step we have used the R50-FPN backbone which is a feature pyramid network (FPN) that uses a 50-layer ResNet for feature extraction. We changed the backbone to an R101-FPN instead which uses a 101-layer ResNet and assessed the impact of using a deeper feature extraction network.

4.4. Transfer Learning

We wanted to evaluate whether our model can benefit from transfer learning if we train it on the LIVECell dataset (which contains other types of cells) or if training on other types of cells wouldn't improve its performance. To do this we created two models with the same configuration and weights and trained one of them only on the competition's data and the other on LIVECell then on the competition's data. This allowed us to evaluate the usefulness of transfer learning in our task.

4.5. Choice of optimizer

The default optimizer used by the Detectron library is SGD. In recent literature, it is often mentioned that the Adam optimizer tends to perform better than SGD, hence we decided to assess how much does the choice of the optimizer affect the training convergence and the final score. To do this, we used the model pretrained on LIVECell as a starting basis and we trained it on the competition's training data, once with SGD and once with the Adam optimizer. We used the same optimizer parameters (such as learning rate) for a fair comparison. The results are presented in the Evaluation section.

4.6. Optimizing detection thresholds and minimum mask area

In this experiment we optimized the detection threshold for each of the 3 classes of cells: When making predictions, Mask R-CNN doesn't only return the masks of the instances but also a prediction score between 0 and 1 for every predicted instance representing the confidence in that segmented instance. By setting a score threshold below which the predicted instances are discarded, we can limit low-confidence predictions and therefore decrease the number of false positives.

Similarly, we can choose a minimum mask area threshold (in pixels) such that predicted masks having a smaller area than this threshold are discarded. This allows us to avoid some false positives where a tiny artefact or stain is segmented.

We optimized the detection threshold and the minimum mask area for each class of cells by evaluating the mAP of the model on the validation data for a range of threshold and minimum area values, and we chose the values that yield the highest mAP.

4.7. Tuning the optimizer

In order to assess the impact of the learning rate on the performance of the model and the training time, we experimented with different learning rates and learning schedules, then we chose the learning schedule offering the highest and most stable validation score.

4.8. Tuning the model hyperparameters

One of the most important steps to improve the model's performance is the hyperparameter tuning. Ideally a grid search over the hyperparameter space would allow us to choose the most appropriate hyperparameters, but in this case the hyperparameter space is huge (there are tens of hyperparameters to tune) and every training process takes hours to run. As a result we resorted to manual hyperparameter tuning where we experimented with changing different hyperparameters and keeping the changed values when we observe an improvement.

4.9. Data Augmentation

By default, the Detectron2 library applies basic augmentations to the training images consisting in horizontal flip and random resizing. To assess the impact of this augmentation, we first trained a model without any augmentation and compared the results. Then we tried adding more augmentations and retained the configuration that yield the best validation score.

4.10. Final model

Before making our final submission for the competition, we trained the model on the validation dataset to improve its generalization ability. Since the validation dataset is relatively small and to avoid overfitting, we also retrain the model on the training dataset with a lower learning rate afterwards. Once the final model is ready, we use it for our main submission for the competition.

5. Model evaluation and discussion

5.1. Model selection

We tested the three promising models Mask R-CNN, Cascade R-CNN and Cellpose as described in the methodology section, and we achieved a public leaderboard mAP score of 0.307 for Mask R-CNN, 0.304 for Cascade R-CNN and 0.300 for Cellpose. Those results are similar and therefore all those three models can potentially be used and improved to win the competition. For instance, it was revealed after the end of the competition that all top 5 winners of the competitions used at least one of those 3 models in their final solution (and conversely all the 3 models figured in at least one of the top 5 solutions).

Since we achieved the highest mAP score with Mask R-CNN, and since it seems to be the most popular instance segmentation model and can be used in other instance segmentation tasks (unlike Cellpose which is specialized in Cell segmentation), we decided to select Mask R-CNN to do our next experiments and try to improve its performance.

5.2. Model backbone

In the backbone selection experiment, we obtained a validation mAP score of 0.2704 with the R50-FPN backbone and 0.2725 (0.77% higher) with the R101-FPN backbone. To further validate the result we submitted both models and obtained a public leaderboard score of 0.307 with R50-FPN and 0.295 (3.9% lower) with R101-FPN. This difference between validation and leaderboard results seems odd, but we believe it is caused by the noisiness of the evaluation metric as explained in part 2.3.

As for training speed, we observed that training with the R50-FPN backbone was 15% faster (with respectively 106 and 122 minutes for 10000 iterations). This is expected since the ResNet-101 is 51 layers deeper and has 19M more parameters to train.

We see that there is no clear winner between the two backbones and the difference between them is marginal. In the end we decided to choose the R50-FPN backbone since it makes our experiments a little faster.

5.3. Transfer Learning

In the transfer learning experiment we compared two models: one trained only on the competition's data, and another one that was also trained on the LIVECell data beforehand. We obtained a validation mAP score of 0.2622 for the first model and 0.2725 for the second, which is 4% higher. We also obtained a similar difference of scores on the public leaderboard (0.291 vs. 0.307 i.e. 5% higher). We conclude that our model benefits from training it on data similar to the competition's, even if the classes of the instances are not the same.

5.4. Choice of optimizer

In this section we compare the performance of two optimizers for training our Mask R-CNN model: SGD (the default Detectron optimizer) and Adam. The validation mAP during training is provided in Fig. 11 :

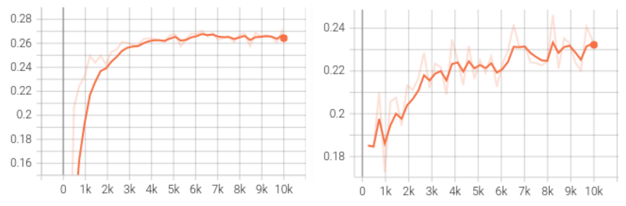


Figure 11: Validation mAP during training. Left: SGD, right: Adam

Interestingly, we observe that the SGD optimizer provides more stable performance and converges in fewer iterations over the training images. Although the total running time for 10000 iterations is higher for SGD (141 vs. 108 minutes), we obtain a significantly higher maximum mAP (0.2725 vs 0.246). We tried lowering the learning rate for Adam which slightly improved its performance (0.2580 maximum mAP) but it remained inferior to SGD.

This result was unexpected since most recent papers suggest that Adam tends to perform better than SGD. Our hypothesis for a possible explanation is that Detectron's default optimizer parameters are better adjusted for SGD. Our conclusion from this experiment is that we should keep the SGD optimizer for our model.

5.5. Optimizing detection thresholds and minimum mask area

In this experiment we optimize the detection score thresholds and the minimum mask area as described in the methodology section. We observe that the detection thresholds have a significant impact on the validation mAP, for example for the third cell class we obtain the results illustrated in Fig. 12. With a poorly set threshold value (close to zero) we obtain mAP lower than 0.22, while the maximum mAP we can achieve is 0.287. On the other hand, the minimum mask area parameter has little impact on the score and only allows us to improve it to 0.289.

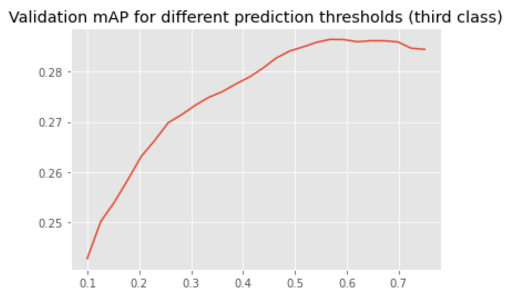
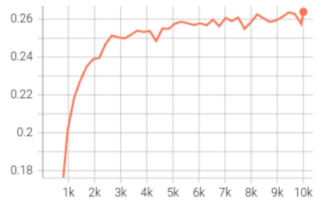


Figure 12: Validation mAP for different detection thresholds

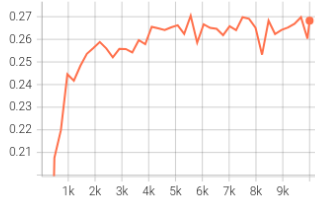
However, this validation score improvement didn't translate into the leaderboard, as the optimized threshold values only achieved 0.307 mAP, which is exactly the same as we had before. We believe this was due to the randomness of the data and the noisiness of the evaluation metric.

5.6. Tuning the optimizer

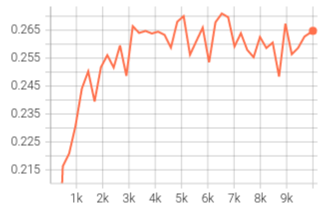
We experimented with training the model for 10000 iterations with learning rates of 0.001, 0.0007, 0.0005, and 0.0001. We compared them based on the mean validation score of the last 10 epochs of training, which was the highest for $LR = 0.0005$ (we also checked the whole validation score curve to verify that the number of epochs is sufficient and that the learning rate 0.0005 has stable performance). Then we tried enabling AMP (Automatic Mixed Precision) to speed up the training process but the difference was small (¡7%) so we disabled it to ensure that we get the best performance. Then we experimented with a learning schedule with base LR of 0.0005 and with γ (LR multiplier) applied every 1000 iterations starting from iteration 2000. We tried two different values of γ and chose the one that yields the highest mean validation score: $\gamma = 0.8$.



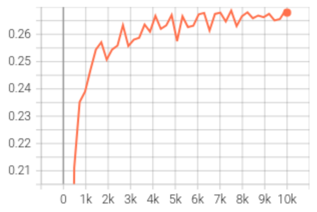
(a) $LR = 0.0001$



(b) $LR = 0.0005$



(c) $LR = 0.0010$



(d) $LR = 0.0005, \gamma = 0.8$

Figure 13: Validation score for different optimizer configurations

5.7. Tuning the model hyperparameters

We experimented with training with different hyperparameters of the Mask R-CNN model, although for computational reasons we couldn't explore the entirety of its very large hyperparameter space. We refer the reader to the corresponding notebook for detailed results. Among the changes that improved the model's score we note: increasing the batch size per image (the number of ROIs per image) to 512, increasing the number of top scoring RPN proposals to keep before applying NMS (15000 for train and 10000 for test) and after applying NMS (3000 for train and 2000 for test), correcting the mean pixel values used for normalisation (127.965 based on the training data), unfreezing the second layer of the backbone (making it trainable), changing the anchor generator sizes from (32, 64, 128, 256, 512) to (24, 40, 80, 128, 256) and aspect ratios from (0.5,1,2) to (0.33, 1.0, 3.0), changing the RPN IOU thresholds to 0.2 and 0.7 and the RPN NMS threshold to 0.75, and increasing the maximum number of detections per image to 700. Most of these changes make sense with respect to the microscopy data we're working with.

With the new hyperparameters, we obtain a validation score of 0.298 and a leaderboard score of 0.310. The increase in the leaderboard score was small, which indicates that the default parameters are already good and that the model is not very sensitive to hyperparameter changes (at least in this case).

5.8. Data Augmentation

We first tried removing the default augmentations (horizontal flip and random resizing) and retraining the model, and we obtained a poor validation score of 0.210, which proves that data augmentation is important. However, when we tried adding more augmentations apart from the default ones, such as random contrast, random brightness, vertical flip, and more random resizing, we only observed a slight improvement of the validation score (0.300), which suggests that after a certain level, adding more augmentations doesn't improve the model anymore.

5.9. Final model

Before making our final submission, we trained the best model from the previous section on the validation dataset, then on the training set again with a lower learning rate. We submitted the model to the competition and obtained a public leaderboard score of 0.314 and a private leaderboard score of 0.323, with a difference of less than 10% compared the top score of the competition (0.356). We provide in Fig. 14 an example of the detection masks produced by the model on an image from the test set. Qualitatively, we can see that most of the cells are correctly detected:

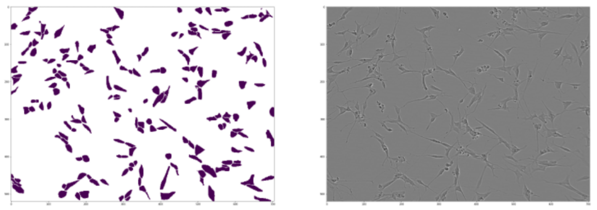


Figure 14: Input image (right) and segmentation mask (left)

6. Conclusion

In this project, we explored different state-of-the-art instance segmentation models, learned their architecture and how they generate the segmentation masks, and after selecting Mask R-CNN as our main model, we fine-tuned its architecture (using a ResNet50-FPN backbone), its optimizer and learning schedule, its hyperparameters, and we experimented with other deep learning techniques such as transfer learning and data augmentation. Those experiments allowed us to improve the performance of the model and to learn which techniques and which parameters are the most important. Our only regret is that the training time of instance segmentation models was long, which limited the number of experiments we could make, but we still managed to build a model that achieved a good score and we are very satisfied with the results, given that this is our first experience in deep learning.

7. References

References

[1] C. Edlund, T. R. Jackson, N. Khalid, N. Bevan, T. Dale, A. Dengel, S. Ahmed, J. Trygg, and R. Sjögren, "LIVECell-A large-scale dataset for label-free live cell segmentation," *Nat Methods*, vol. 18, no. 9, pp. 1038–1045, 09 2021. 2

[2] J. C. Caicedo, A. Goodman, K. W. Karhohs, B. A. Cimini, J. Ackerman, M. Haghighi, C. Heng, T. Becker, M. Doan, C. McQuin, M. Rohban, S. Singh, and A. E. Carpenter, "Nucleus segmentation across imaging experiments: the 2018 Data Science Bowl," *Nat Methods*, vol. 16, no. 12, pp. 1247–1253, 12 2019. 2

[3] C. Stringer, T. Wang, M. Michaelos, and M. Pachitariu, "Cellpose: a generalist algorithm for cellular segmentation," *Nat Methods*, vol. 18, no. 1, pp. 100–106, 01 2021. 2

[4] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," 2018. 3

[5] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013. [Online]. Available: http://arxiv.org/abs/1311.2524 3

[6] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. [Online]. Available: http://arxiv.org/abs/1504.08083 3

[7] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497 3

[8] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," *CoRR*, vol. abs/1612.03144, 2016. [Online]. Available: http://arxiv.org/abs/1612.03144 3

[9] Z. Cai and N. Vasconcelos, "Cascade R-CNN: delving into high quality object detection," *CoRR*, vol. abs/1712.00726, 2017. [Online]. Available: http://arxiv.org/abs/1712.00726 4

[10] K. J. Cutler, C. Stringer, P. A. Wiggins, and J. D. Mougous, "Omnipose: a high-precision morphology-independent solution for bacterial cell segmentation," *bioRxiv*, 2021. [Online]. Available: https://www.biorxiv.org/content/early/2021/11/04/2021.11.03.467199 4