Final Project. Sartorius - Cell Instance Segmentation

Visual Recognition using Deep Learning, 2025 Spring

report by ***Group 4 ICCV***

313540009, Anna Kompan 安娜 (team leader)

313551818, Marie Picquet 克莉絲

313554802, Sophie Fu 傅冰潔

313540002, Mariia Leontieva 馬莉亞

Code link (kaggle): https://www.kaggle.com/code/annakompan/old-test-2

1. Introduction

The goal of “Sartorius - Cell Instance Segmentation” competition[1] on Kaggle is to detect and delineate distinct objects of interest in biological images depicting neuronal cell types commonly used in the study of neurological disorders. More specifically, we aim to use phase contrast microscopy images to train and test the model for instance segmentation of neuronal cells. Researchers may be able to use this to more easily measure the effects of disease and treatment conditions on neuronal cells. As a result, new drugs could be discovered to treat the millions of people with these leading causes of death and disability.

The authors of this competition provide train images with a corresponding train.csv file where each ground truth cell instance is encoded in RLE format. Train dataset contains 606 images and the test set contains 3 images. Number of cells for each cell type in the train set can be seen in Figure 1. There is also a “train_semi_supervised” folder with unlabeled images, provided as additional data for a semi-supervised approach.

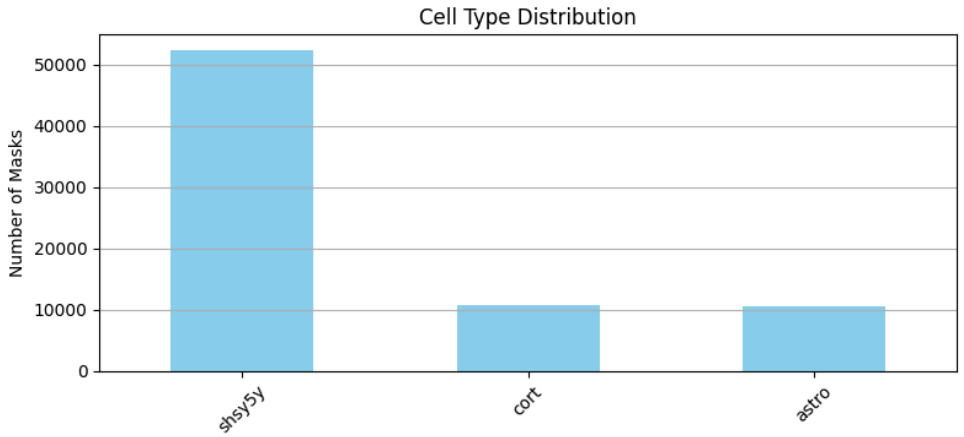

Figure 1. Cell Type distribution in “Sartorius - Cell Instance segmentation” training set.

A sample image with its corresponding masks is plotted in Figure 2. One can see that the target masks in the train set may overlap with each other; however, the overlaps should be removed when submitting predictions on Kaggle, otherwise an error will be generated during submission.

The metric that is used to evaluate results of this competition is mean average precision at different intersections over union (IoU) thresholds. The IoU of a proposed set of object pixels and a set of true object pixels is calculated as:

$$IoU(A, B) = \frac{A \cap B}{A \cup B}$$

The metric sweeps over a range of IoU thresholds, at each point calculating an average precision value. The threshold values range from 0.5 to 0.95 with a step size of 0.05: (0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95).

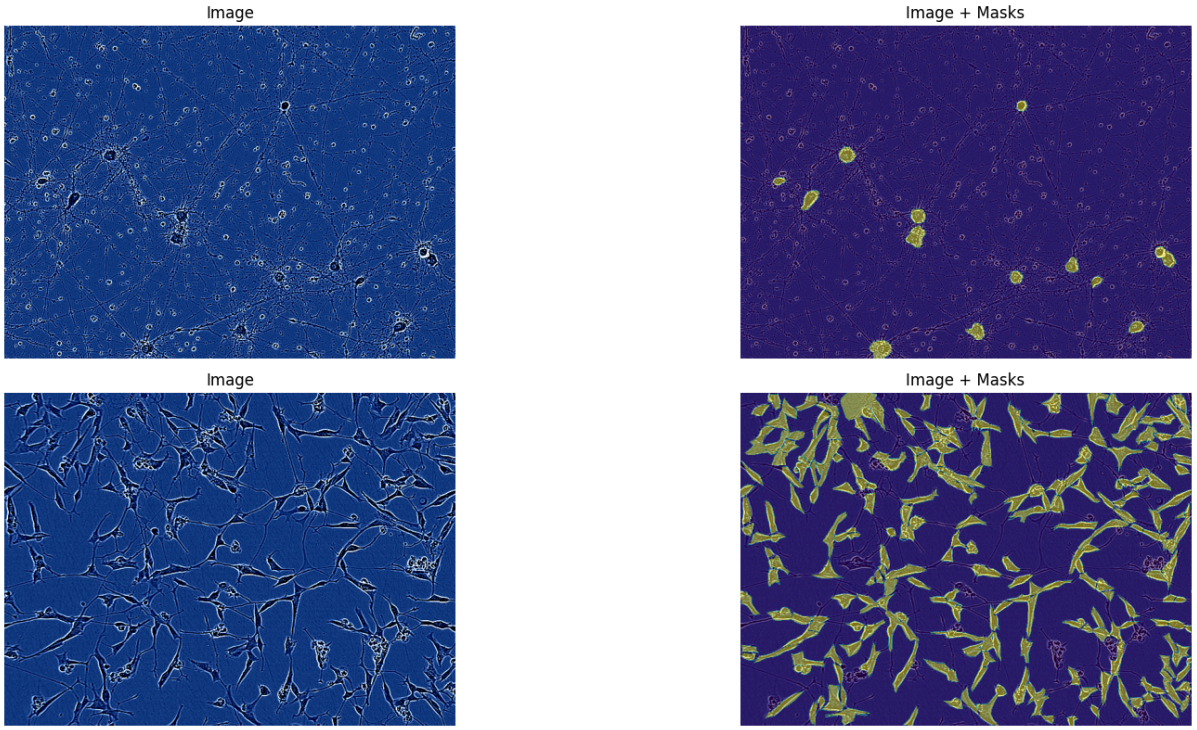

Figure 2. Training sample with its ground truth instance segmentation masks.

2. Related works

One of the popular solutions for this problem is ***Mask R-CNN*** (regional convolutional neural network), which is state-of-the-art in terms of image segmentation and instance segmentation introduced by Kaiming He, Georgia Gkioxari, Piotr Dollár and Ross Girshick in their paper “Mask R-CNN”[2]. Mask R-CNN was built on top of Faster R-CNN, a popular framework for object detection. Its structure is shown in Figure 3. There are two stages in this framework:

- The first stage is called a **Region Proposal Network** (RPN), which proposes candidate object bounding boxes.
- The second stage extracts features from each candidate box and performs classification, bounding-box regression and a **binary mask**.

Mask R-CNN is the model we already used in Assignment 3 for cell segmentation, so it’s a natural solution for this problem. Mask R-CNN can be both trained from scratch or trained from Imagenet-pretrained weights, depending on selected backbone structure. The backbone for this model can also be selected

according to preference, so this solution is convenient and flexible. However, since this task is particularly difficult, it may need a more complex model to fit properly.

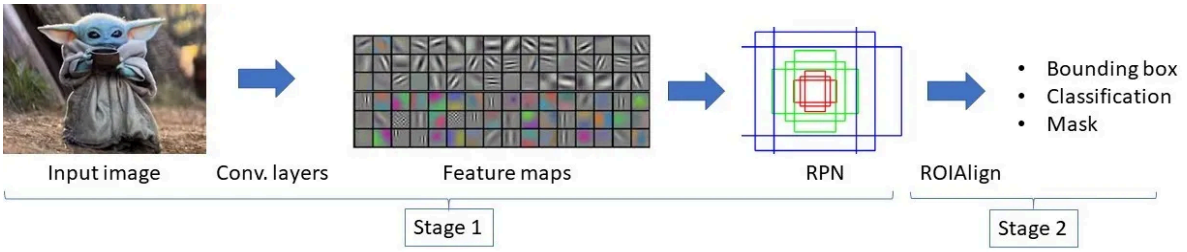

Figure 3. General structure of Mask R-CNN.

Some teams that got good results on this competition have used Detectron2 platform, so we decided to try it out as well. Detectron2 is the successor of **Detectron**[3], which is Facebook AI Research's software system that implements state-of-the-art object detection algorithms, including Mask R-CNN, RetinaNet, Faster R-CNN, RPN, Fast R-CNN and R-FCN. It allows selecting ResNeXt, ResNet, Feature Pyramid Networks (with ResNet/ResNeXt) or VGG16 as backbone network architectures.

3. Method/Approach

For Mask R-CNN implementation, we used the ResNet-50-FPN backbone (from torchvision detection models library) from the "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks"[6] paper. Its structure can be seen in Figure 4.

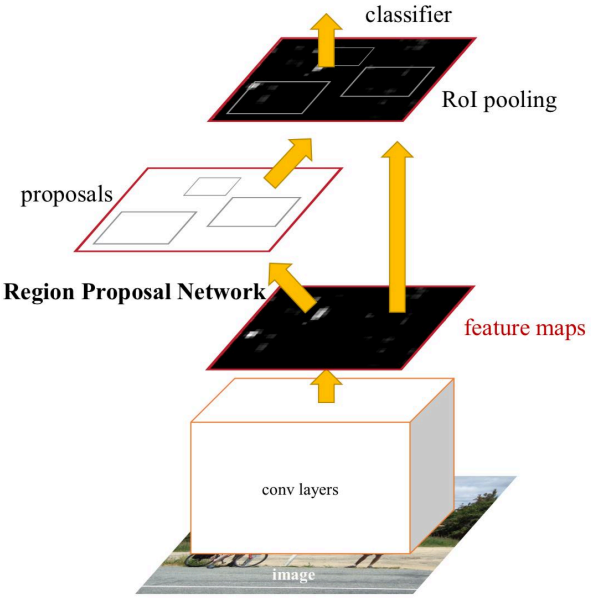

Figure 4. Faster R-CNN.

The train set was split for training and validation as 80/20 or 90/10, the result of each choice is discussed in experiments below. Each image was resized to 512x512 and normalized according to ImageNet mean and variance.

Implementation of Mask R-CNN model using pytorch framework is shown in the screenshot below.

```
NUM_CLASSES = len(CELL_TYPE_DICT)
# Load the COCO-pretrained weights you added
model = torchvision.models.detection.maskrcnn_resnet50_fpn(weights=None, weights_backbone=None)
state_dict = torch.load("//kaggle/input/pretrained/maskrcnn_resnet50_fpn_coco_0.15.1.pth")
model.load_state_dict(state_dict)

# Replace classification head
in_features = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, NUM_CLASSES + 1)

# Replace mask head
in_features_mask = model.roi_heads.mask_predictor.conv5_mask.in_channels
hidden_layer = 128
model.roi_heads.mask_predictor = MaskRCNNPredictor(in_features_mask, hidden_layer, NUM_CLASSES +
1)
```

4. Experimental Results

Experiment 1: choosing optimizer function

Settings:

`weight_decay = 0.0005`

`MOMENTUM = 0.9`

```
optimizer = torch.optim.SGD(params, lr=LEARNING_RATE, momentum=MOMENTUM, weight_dec
ay=WEIGHT_DECAY)
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)
n_batches, n_batches_val = len(train_loader), len(val_loader)
```

Model: As described above, here we used the Mask R-CNN model with a ResNet-50-FPN backbone, pretrained on COCO 80 classes.

Evaluation Metric: mean average precision (mAP) at different intersections over union (IoU) thresholds.

Results:

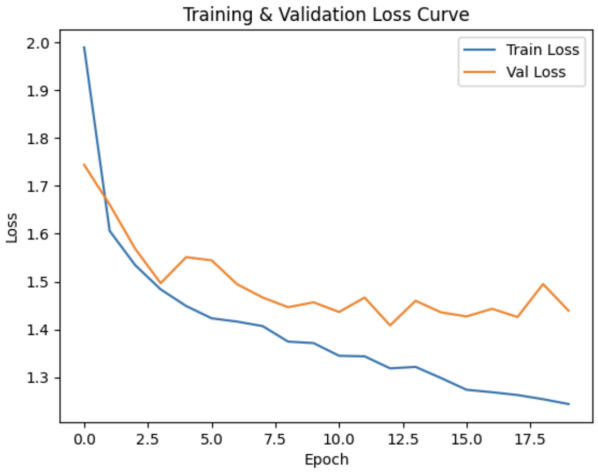

Figure 5.Training and validation loss curves for Experiment 1.

As optimizer scheduler, we used ReduceLROnPlateau; it was expected to adaptively lower learning rate if no improvement is detected, which helps to avoid overfitting. However, training validation loss reached a plateau and training didn't improve after about epoch 10.

Experiment 2: adding data augmentations

In this section we are exploring data augmentation approaches to hopefully reach better convergence of the model.

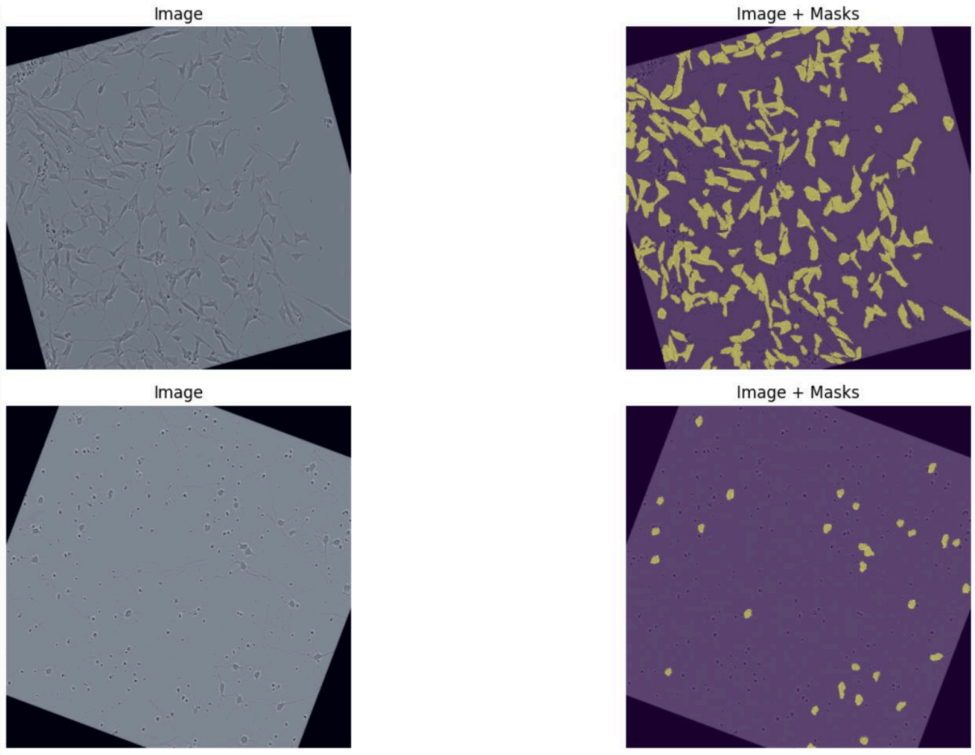

Figure 6. Visual representation of random rotations.

Settings: data augmentation was done using random rotations and flips.

```
transform = T.Compose([
    T.Resize((512, 512)),
    T.RandomHorizontalFlip(0.5),
    T.RandomVerticalFlip(0.2),
    T.RandomRotation(15),
    T.ColorJitter(0.2, 0.2, 0.2),
    T.ToTensor(),
    T.Normalize(mean=RESNET_MEAN, std=RESNET_STD)
])
```

LR scheduler and optimizer: remained like in Experiment 1.

Results:

Figure 7. Training and Validation Loss Curve for Experiment 2.

We decided to implement augmentation under the assumption that increased variability in data helps generalization, especially in small/imbalanced datasets like the cell dataset we have. However, as we can see from Figure 7 above, validation loss became unstable after adding random rotations (goes up and down across epochs), therefore we make a conclusion that random rotation augmentations might be too aggressive, probably because they distort instance masks, and maybe in further work, augmentations should be limited to simpler, not too aggressive transforms (we will stick with vertical/horizontal flips from here on).

Experiment 3: two-stage training

In this experiment, we implement two-stage training: warm-up LR scheduler with SGD optimizer and gradient clipping[5].

Settings:

Data augmentation with horizontal and vertical flip with 0.5 probability.

```
    transforms.append(HorizontalFlip(0.5))
    transforms.append(VerticalFlip(0.5))
```

Stage 1: Freeze the backbone and only update parameters in box_predictor/mask_predictor (focus training on parts that need to adapt to cell dataset). Optimizer used in this stage is

```
opt_stage1 = torch.optim.SGD(
    head_params, lr=1e-3, momentum=MOMENTUM, weight_decay=WEIGHT_DECAY
)
```

Warm-up scheduler for 500 steps. Train heads for 10 epochs.

```python
def warmup_lambda(step):
    return min((step + 1) / 500, 1.0)

warmup_sched = torch.optim.lr_scheduler.LambdaLR(opt_stage1, warmup_lambda)
```

Warmup scheduler is used to increase learning rate during the first few steps (0,249,499). We hope that this prevents instability of training and allows better convergence and lower final loss.

Stage 2: Unfreeze backbone and continue training the entire model (head and backbone), using lower LR for backbone and for the heads to avoid forgetting COCO features.

```
WEIGHT_DECAY2 = 1e-4
```

```python
opt_stage2 = torch.optim.AdamW([
    {"params": head_params, "lr": 1e-4},
    {"params": backbone_params, "lr": 1e-5},
], weight_decay=WEIGHT_DECAY2)
```

Results:

This technique achieves better generalization: backbone retains general features while heads adapt to new dataset. However, the improvements compared to Experiment 2 are not very significant.

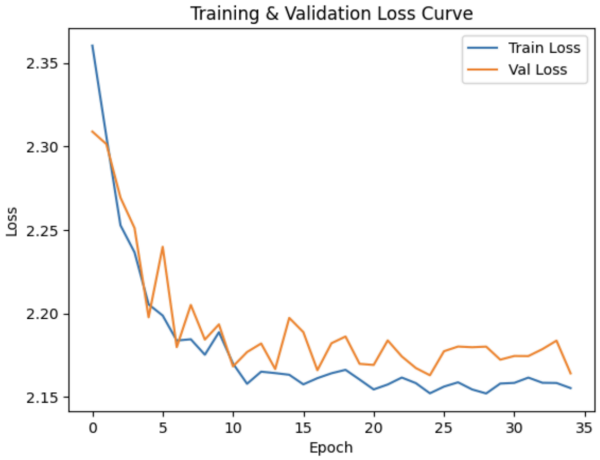

Figure 8. Training and Validation Loss Curve for Experiment 3.

To see how the model performs, below is visualization of one prediction result and its ground truth label. We can see that the model didn't converge properly not just because not all masks are detected, but also all masks have a rectangular shape same as their bounding boxes.

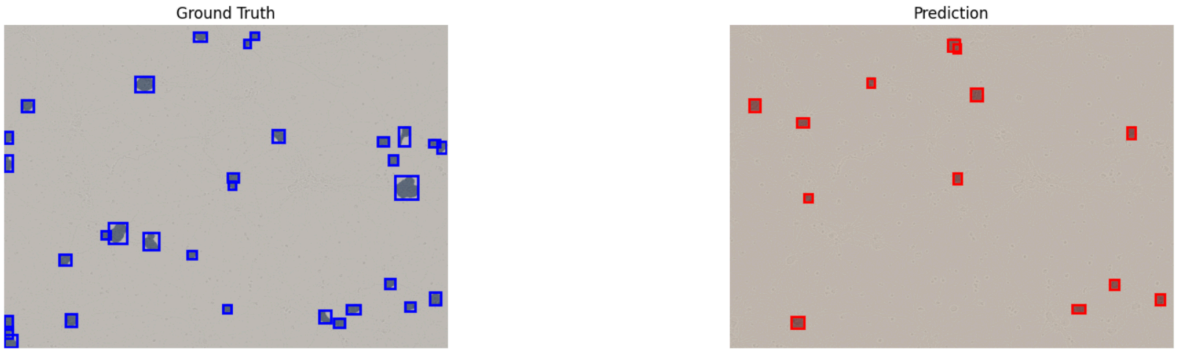

Figure 9. Experiment 3 Prediction Results.

Experiment 4: change second stage optimizer to SGD

Since the last experiment proved that SGD optimizer achieves better results, we decided to use SGD instead of AdamW. Transformations include custom HorizontalFlip and VerticalFlip. Here, we don't do color/rotation augmentations[4].

Settings:

```
MOMENTUM = 0.9
LEARNING_RATE = 1e-3
WEIGHT_DECAY = 5e-4
```

```
optimizer = torch.optim.SGD(params, lr=LEARNING_RATE, momentum=MOMENTUM, weight_decay=WEIGHT_DECAY)
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)
```

Results:

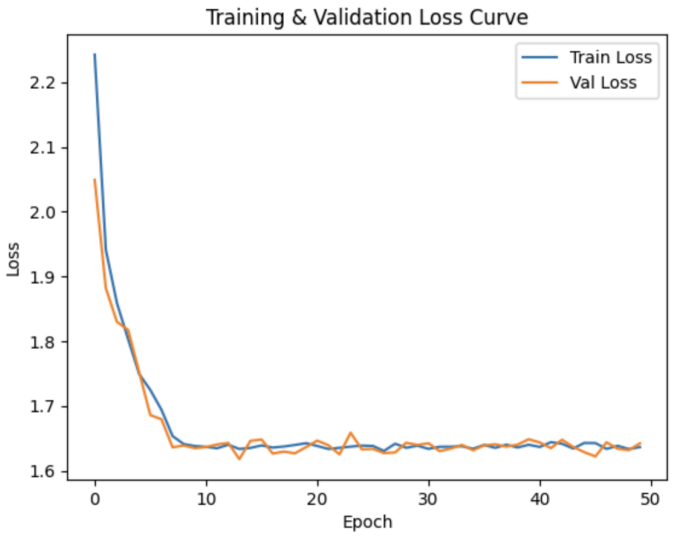

Figure 10. Training and Validation Loss Curve for Experiment 4.

Figure 11 below shows an example of a relatively successful prediction, giving IoU of 0.02. The results on Kaggle after:

Training for 8 epochs yields mAP = 0.025

Training for 50 epochs yields mAP = 0.157

Longer training time yields a significant improvement, but leaves space for an even better convergence.

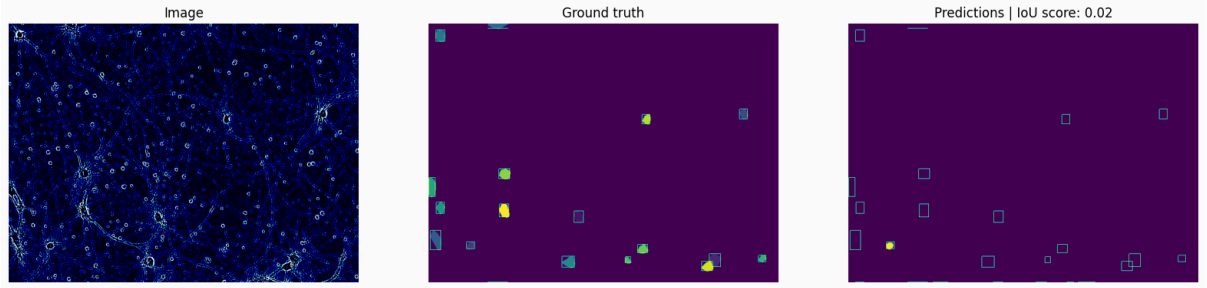

Figure 11. Experiment 4 Prediction Results.

5. Conclusion

In this project we solved instance segmentation problem for Sartorius Cell dataset, using various methods, starting with Mask RCNN model with ResNet-50-FPN backbone pretrained on COCO, we evaluated effects of several training techniques including two-stage training with different optimizers (AdamW, SGD), gradient clipping, loss weighting, data augmentation. Experiment showed that:

- Two-stage training with optimizer switching helped stabilize learning in early epochs and improve convergence
- Gradient clipping helped with handling exploding gradients.
- Less data augmentations is better, best results achieved by only including custom HorizontalFlip and VerticalFlip improved results.

From this project we've learned to implement different training strategies, data preprocessing and augmentation, which gave us better understanding on how these changes might influence results of instance segmentation, improve generalization, especially given a small and imbalanced cell dataset.

Kaggle Snapshot

	313540009	313551818	313554802	313540002

Literature survey	25%	25%	25%	25%
Data pre-processing	10%	80%	10%	
Approach implementation (experiment)	48%	7%	38%	7%
Kaggle submission	45%		45%	10%
Report writing and slide making	40%			60%
Oral presentation	33%	33%	33%	0% (absent)

6. References:

[1] Addison Howard, Ashley Chow, CorporateResearchSartorius, Maria Ca, Phil Culliton, and Tim Jackson. Sartorius - Cell Instance Segmentation. https://kaggle.com/competitions/sartorius-cell-instance-segmentation, 2021. Kaggle.

[2] Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick. “Mask R-CNN”. [1703.06870] Mask R-CNN. https://arxiv.org/abs/1703.06870

[3] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari,Piotr Dollár and Kaiming He. Detectron. https://github.com/facebookresearch/detectron, 2018.

[4] Enzou3. (2022). *Sartorius - Mask R-CNN* [Kaggle Notebook]. Kaggle. https://www.kaggle.com/code/enzou3/sartorius-mask-r-cnn

[5] Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., ... & He, K. (2017). *Accurate, large minibatch SGD: Training ImageNet in 1 hour*. arXiv preprint arXiv:1706.02677. https://arxiv.org/abs/1706.02677

[6] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. 2016. https://arxiv.org/pdf/1506.01497