Instance Segmentation on LIVECell dataset

Description of Our Approach

Aleksander Hański, Jakub Radziejewski

Description of the Dataset

LIVECell (Label-free In Vitro image Examples of Cells) dataset - designed for the automated segmentation of cells in phase-contrast microscopy. Over 1.6 million individual annotated cells. In total 5 239 images, each of resolution 704 x 520. Annotations (in COCO format) require 2.1GB of space.

Preprocessed dataset configuration:

- Total original images: 100
- Training split: 70 images (70%)
- Validation split: 15 images (15%)
- Test split: 15 images (15%)
- Image dimensions: 704 × 520 pixels
- Tiling strategy: 7×7 grid with 3×3 sliding window
- Tiles per image: 25, each of size 300x222px
- Total tiles: 1,750 (train), 375 (val), 375 (test)

Data Version Control (DVC)

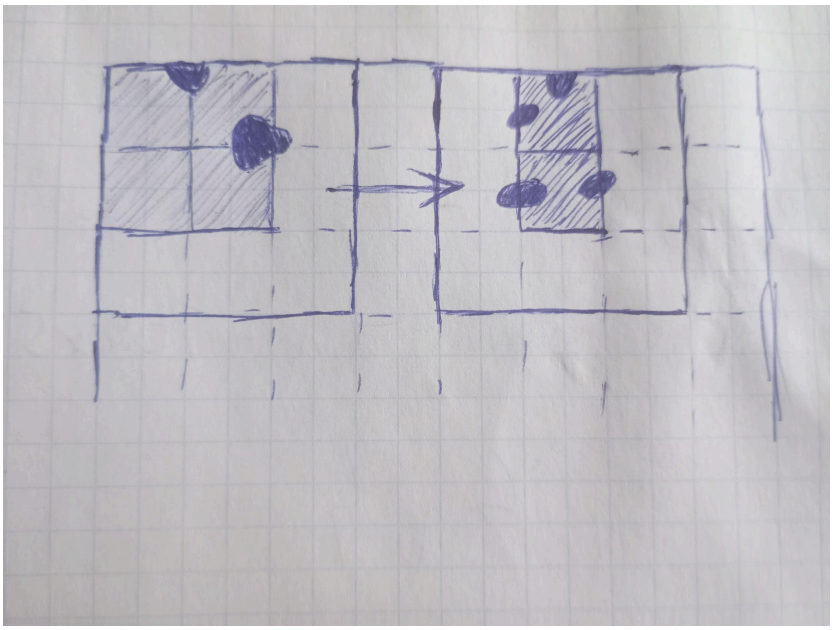
This dataset is managed using *Data Version Control* to efficiently track and version large data files. DVC Tracked Files:

- Original Dataset (data.dvc):
  - Path: data/
  - Size: 2.18 GB (2,180,595,526 bytes)
  - Files: 5,242 files
  - MD5 hash: 57c3e7af52bf34561a772f05dec4bcfc.dir
- Preprocessed Split Dataset (data_split.dvc):
  - Path: data_split/
  - Size: 237.5 MB (237,534,631 bytes)
  - Files: 2,503 files
  - MD5 hash: 652ffa1b940420156ad9b151f996f081.dir

Description of the Problem

We performed instance segmentation of individual cells. It turned out that individual images with annotations take up too much space in RAM memory, we therefore preprocessed data by

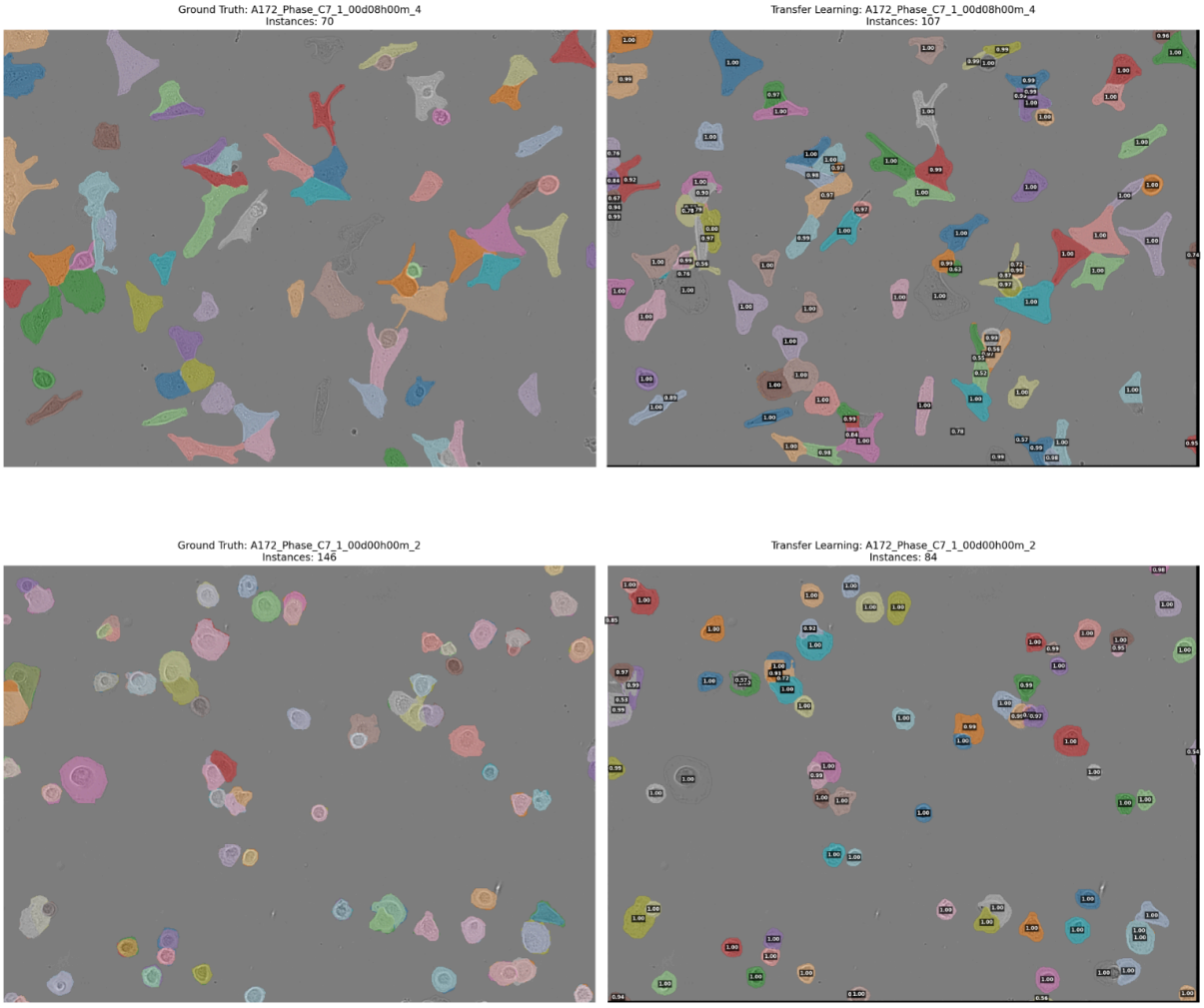
dividing each image and corresponding annotations into smaller sub-images, and performed training on them. For each original image we store corresponding 25 sub-images that overlap (every one still has size bigger than 200x200px), and we perform final prediction and visualization by summing model's predictions on smaller sub-images into the original image. As sub-images overlap, we do not cut the model's predictions at the sub-image borders. We perform sliding windows. Each sub image is divided into 3x3 mini-tiles, and for each sub-image we save predictions of cells that lie within mini-tiles that either lie on the border of original image, or are center mini-tile of sub-image (or rather save prediction of cells whose threshold% of area lie within correct mini-tiles).



Description of used architectures with diagrams showing the layers

Transfer learning model

It is Mask R-CNN (Regional Convolutional Neural Network) architecture. The model has ResNet50 with Feature Pyramid Network (FPN) backbone pretrained on COCO dataset, RPN (Region Proposal Network) and ROI (Region of Interest) heads also already pretrained. It has replaced box and mask predictors that detect whether there is a cell or not instead of original 80 COCO classes. Box and mask predictors have random weights at the beginning.



Custom Mask R-CNN Architecture

Based on: He et al., "Mask R-CNN".

The first stage proposes regions that might contain cells. The second stage classifies these regions, refines their boundaries, and generates pixel-level masks. The model uses ResNet-18 as backbone, adds CBAM attention modules, and processes features through a Feature Pyramid Network before detection and segmentation heads.

Backbone ResNet-18

ResNet-18 extracts features from input images through four stages with progressively lower resolution and more channels: 64, 128, 256, and 512 channels. Each stage uses residual blocks with skip connections that help gradients flow during training. We use ResNet-18 instead of deeper variants because of our computational limitations.

CBAM Attention Module

Based on: Woo et al., "CBAM: Convolutional Block Attention Module".

CBAM is applied after each ResNet layer to refine intermediate feature maps before they are passed to the FPN. Channel attention is applied first to emphasize informative feature channels using global average and max pooling followed by a small network. Spatial attention is then applied to the channel-refined features to highlight important spatial locations using pooled channel statistics and a convolution. This improves feature discrimination and localization with minimal additional computation.

Feature Pyramid Network

FPN creates features at multiple scales so the model can detect both small and large cells. It takes the four ResNet outputs, converts them all to 256 channels with 1x1 lateral convolutions, then combines them top-down by upsampling coarse features (using nearest-neighbour interpolation) and adding them to fine features discovered in early layers. The output is four feature maps with consistent 256 channels but different resolutions. This is needed because cells vary significantly in size.

Region Proposal Network

RPN proposes regions that likely contain cells. At each location in the feature maps, it considers nine anchor boxes with three sizes (32, 64, 128 pixels) and three aspect ratios (0.5, 1.0, 2.0). A 3x3 convolution processes features, then a 1x1 convolution predicts objectness scores for each anchor. During training, anchors with high overlap to ground truth are positive, low overlap are negative. We sample 128 of each and compute binary cross-entropy loss. During inference, we keep top proposals by score, filter by threshold, and remove overlaps with NMS. This narrows down hundreds of thousands of possible locations to a few hundred good proposals.

ROI Align

ROI Align extracts 7x7 feature maps from each proposal. It divides each proposal into a 7x7 grid and samples points using bilinear interpolation. This preserves spatial precision compared to older ROI pooling methods. The output is fixed-size features that can be batch-processed by the detection and segmentation heads. Precision matters for accurate masks. In our implementation, we utilize the torchvision.ops module.

Box Head

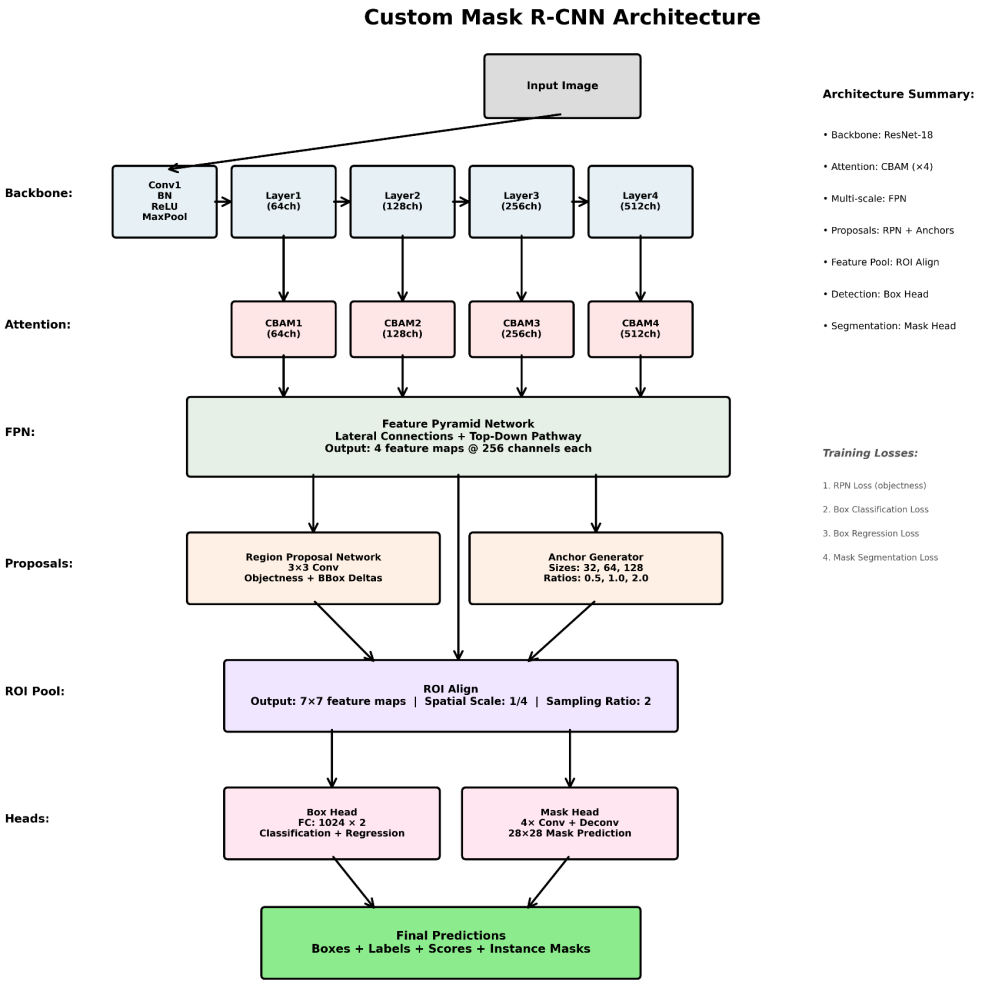
The Box Head is responsible for the final classification and geometric refinement of each region proposal. It receives the 7x7x256 feature maps from RoI Align, which are flattened into a one-dimensional vector to be processed by two fully connected layers of 1024 units each. This branch produces two specific outputs: a classification score determining if the proposal is a cell or background, and a set of coordinate adjustments that refines the initial RPN proposal to fit the cell's boundaries more accurately.

During training, the network uses a multi-task loss approach. Positive proposals are used to train both the classification branch via Cross-Entropy loss and the regression branch via Smooth L1 loss. Negative proposals are used only to train the classification branch to recognize background noise.

Mask Head

The Mask Head operates in parallel to the Box Head to generate pixel-level segmentations. Unlike the Box Head, it utilizes a Fully Convolutional Network (FCN) structure to preserve spatial information. It processes the ROI features through four 3x3 convolutional layers, maintaining 256 channels to extract deep morphological patterns. To restore the spatial resolution necessary for a mask, the features are upsampled using a 2x2 transposed convolution, followed by a 1x1 convolution that outputs the final logits.

During training, Binary Cross-Entropy loss is calculated specifically for the mask corresponding to the ground-truth class, preventing the different class masks from competing with one another. During inference, the output is upsampled to a 28x28 resolution using bilinear interpolation and thresholded at 0.5.



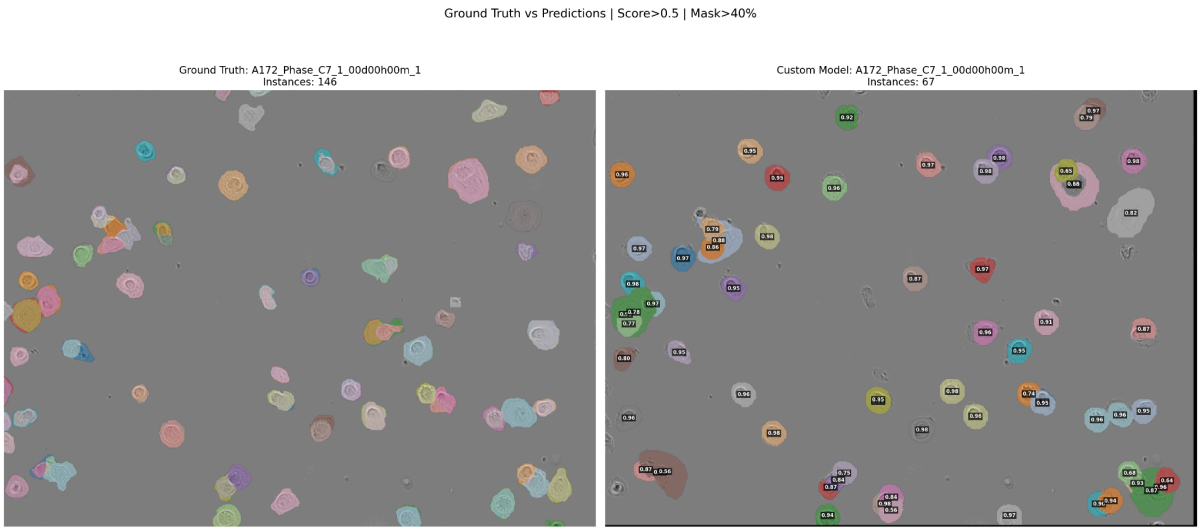
Inference

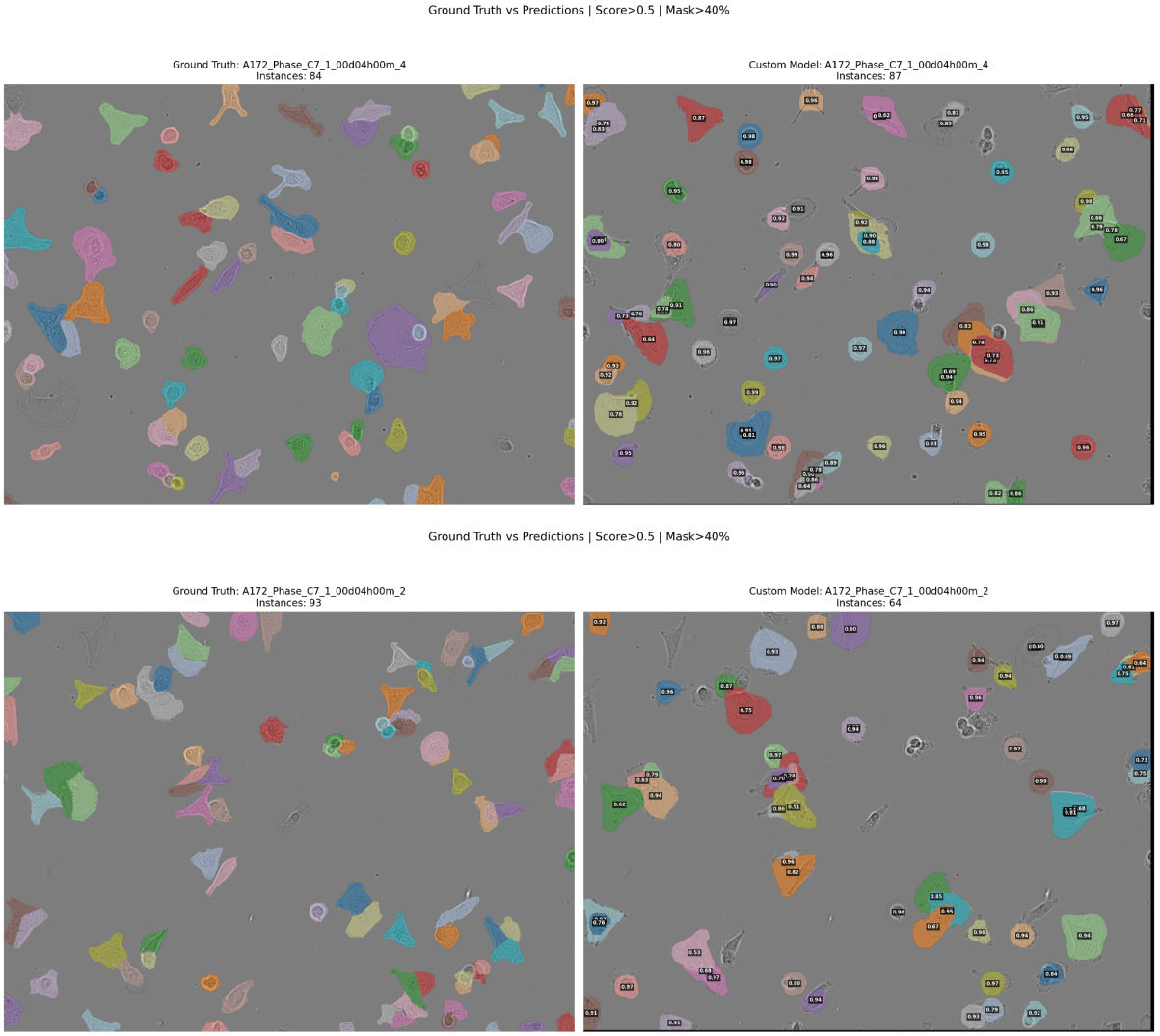
The inference process begins with feature extraction through a ResNet-18 backbone enhanced with CBAM attention modules, followed by Feature Pyramid Network (FPN) fusion to create multi-scale representations. The Region Proposal Network (RPN) then generates candidate object locations using a grid of 9 anchors per position, filtering proposals by objectness scores and Non-Maximum Suppression (NMS). ROI Align extracts fixed 7×7 features for each proposal, which are processed by the box head for classification and bounding box refinement, with detections filtered by confidence threshold (0.4) and NMS (IoU 0.5).

The mask head generates 28×28 binary masks for remaining detections, which are upsampled to match each bounding box size and placed at the corresponding image location. The final output contains bounding boxes, class labels, confidence scores, and pixel-level segmentation masks for all detected cells.

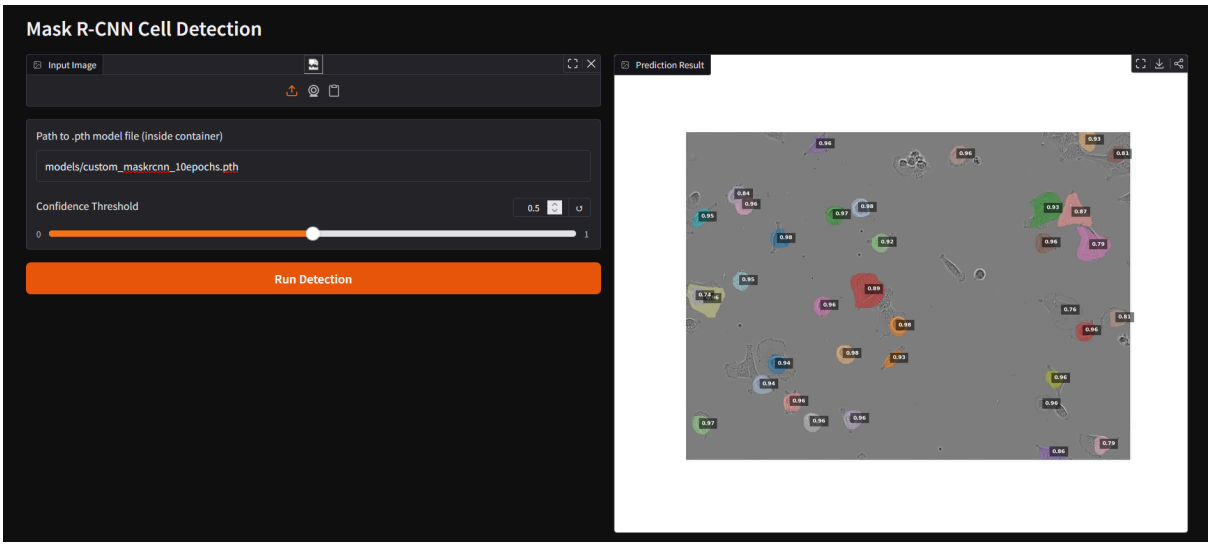
Visualization of Results

3 outputs from a test set run by our custom architecture by our run which is described below in the appropriate section.





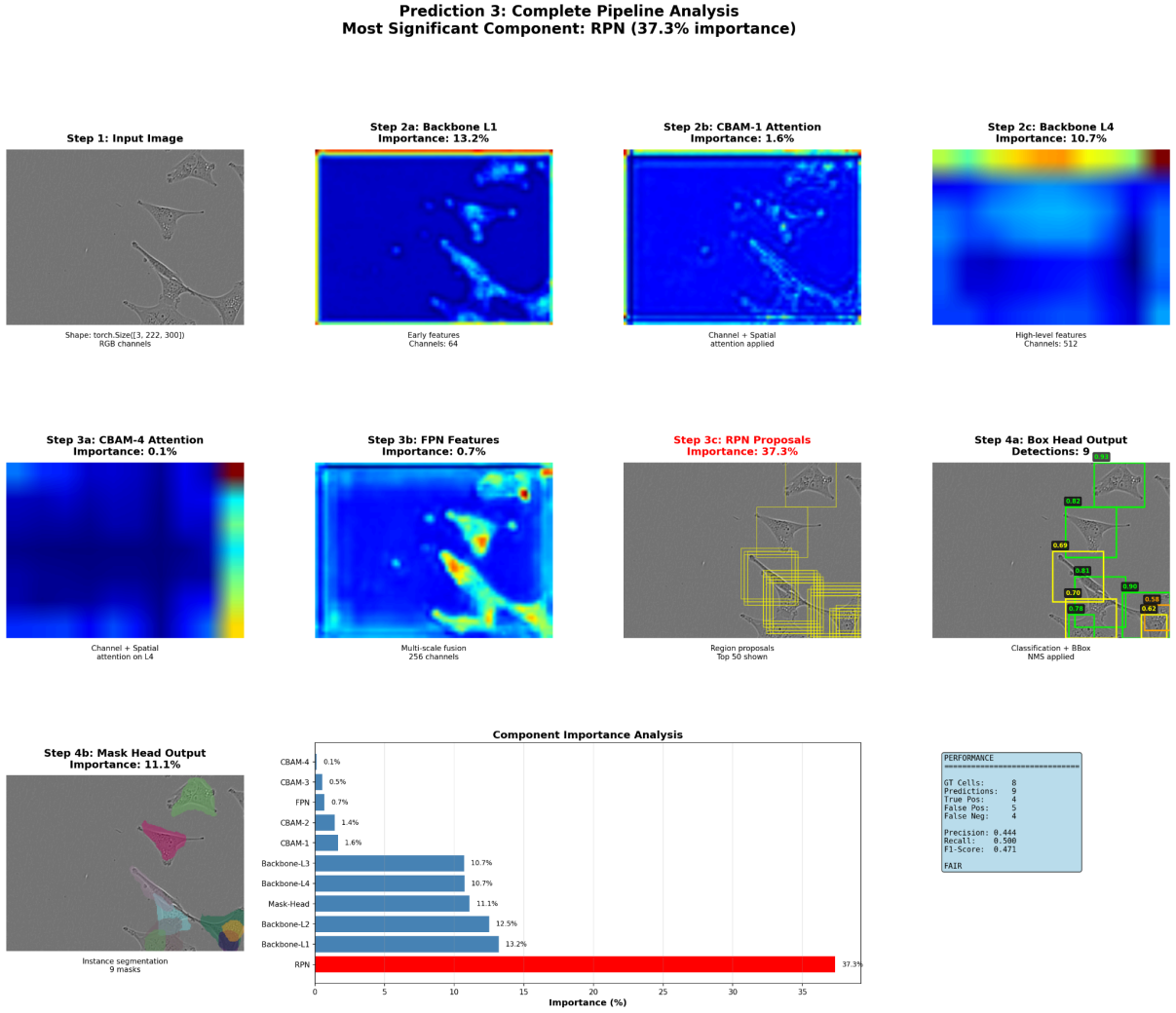
We also included a *Gradio* interface where the end user can choose which file to upload and which checkpoint from which model to use, processing takes around 5 seconds to get model prediction. Also the confidence threshold can be adjusted.



Explanation of Predictions

The explanation system uses forward hooks to capture intermediate activations at key points throughout the inference pipeline, including backbone layers (layer1-4), CBAM attention modules, FPN, RPN, box head, and mask head. During forward pass, these activations are automatically recorded and stored for subsequent analysis.

Feature importance is calculated by measuring average activation strength for each component and converting to percentages. Feature maps are visualized by averaging channels and applying colors to show activation intensity.



The RPN consistently emerges as the most important component across all predictions (37-38% importance), indicating that proposal generation is the critical stage in the detection pipeline.

Model analysis: size in memory, number of parameters

Transfer Learning:

Memory size:	167.6 MB
Total parameters:	43,922,395
Trainable parameters (before freezing):	43,699,995
Trainable parameters (after freezing):	16,529,164

Custom model:

Memory size:	118.10 MB
Total parameters:	30,958,337
Backbone (ResNet-18):	11,766,592 (38.0%)
Custom layers:	
- FPN:	2,607,104
- RPN:	601,645
- CBAM Attention:	43,912
- Box Head:	13,905,930
- Mask Head:	2,623,234
Total custom:	19,191,745 (62.0%)

Description of the training and the required commands to run it

Transfer learning:

We first freeze the backbone for 5 epochs, and then perform fine-tuning on the whole model for 5 epochs.

Training command:

python src/train_transfer.py

Optimizer: SGD with momentum

- Stage 1 LR: 0.005
- Stage 2 LR: 0.001
- Momentum: 0.9
- Weight decay: 0.0005

Custom model:

1. Initialize ResNet-18 backbone without any pretrained weights

2. Initialize custom components (CBAM, FPN, RPN, heads) randomly

3. Train all components

4. No frozen layers - full model is trainable

Training command with W&B:

python src/train_custom.py --use_wandb

Optimizer: AdamW

- Learning rate: 0.0001
- Weight decay: 0.0001
- Betas: (0.9, 0.999)

Scheduler: StepLR

- Step size: 2 epochs
- Gamma: 0.1 (10× reduction)
- LR schedule: 0.0001 → 0.00001 → 0.000001

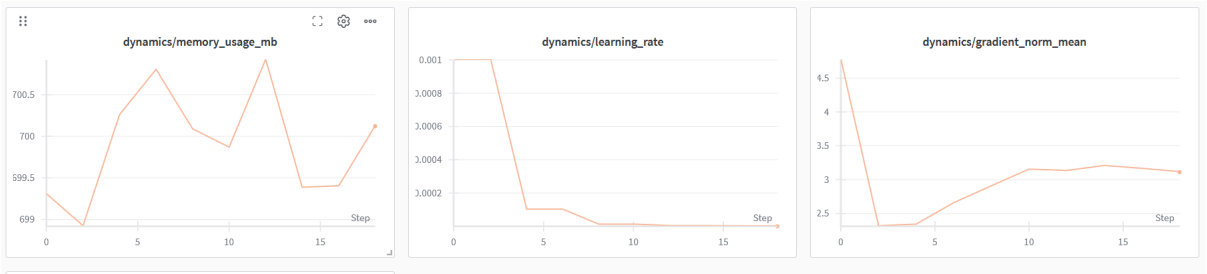
Training duration: 10 epochs

Description of used metrics, loss, and evaluation for both models

Custom Model

Training Dynamics Metrics

Gradient Norm -monitoring training stability and convergence
Learning Rate - tracked per epoch
Memory usage of GPU (in MB)
Epoch Time (in seconds)



Loss

Total Loss = L_rpn_cls + L_box_cls + L_box_reg + L_mask

Component losses:

1. RPN Classification Loss (L_rpn_cls)
   • Type: Binary cross-entropy
   • Purpose: Train RPN to distinguish objects from background
   • Applied to: All anchor boxes

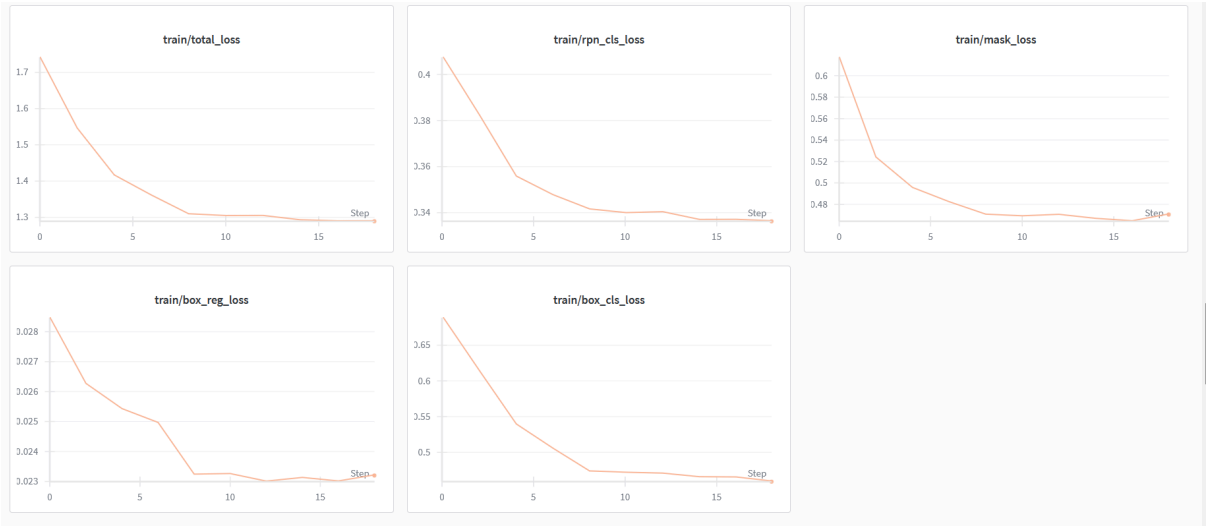
2. Box Classification Loss (L_box_cls)
   • Type: Cross-entropy (2 classes)
   • Purpose: Classify proposals as cell vs background
   • Applied to: RPN proposals

3. Box Regression Loss (L_box_reg)
   • Type: Smooth L1 loss
   • Purpose: Refine bounding box coordinates
   • Formula: SmoothL1(pred_box, gt_box)

4. Mask Loss (L_mask)
   • Type: Binary cross-entropy
   • Purpose: Generate accurate segmentation masks
   • Applied to: 28×28 mask predictions for positive boxes only

Plots of Losses

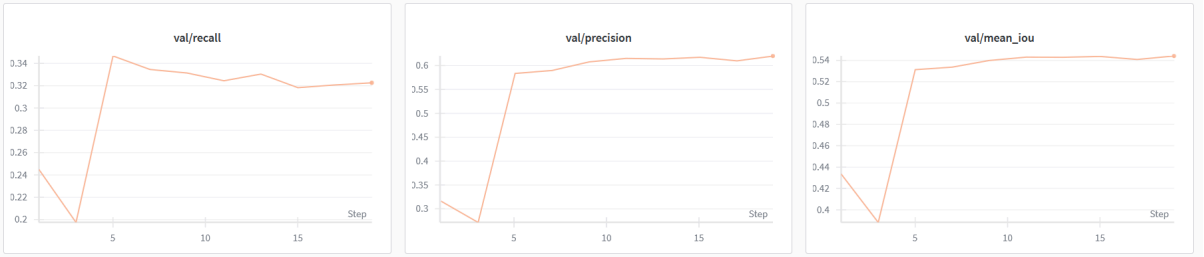
Custom architecture:



Evaluation metrics

1. Mean Intersection over union(mIoU)
   Formula: IoU = (Predicted ∩ Ground Truth) / (Predicted ∪ Ground Truth)
   Range: [0, 1], higher is better
2. Precision
   Formula: TP / (TP + FP)
3. Recall
   Formula: TP / (TP + FN)
4. F1 score
   Formula: 2 × (Precision × Recall) / (Precision + Recall)
5. Confidence score
   Range: [0, 1]
   Purpose: Model's confidence in each detection
   Filtering: Detections with score > 0.5 kept for evaluation

Plots of Evaluation metrics from Validation



Transfer Learning

The same losses as in the custom model.

The same metrics, and also:

Evaluation Count Metrics:

- Total Ground Truth Instances
- Total Predicted Instances
- Total True Positives

Training Dynamics Metrics

- Gradient Norm - tracks gradient magnitudes (mean ± std)
- Avg Predictions per Image - counts detections above 0.5 threshold per image
- Epoch Time

Used hyperparameters along with an explanation of each why such value was chosen

---

Custom model:
Optimizer: AdamW with learning rate 0.0001, weight decay 0.0001, and betas (0.9, 0.999). The learning rate balances stability and convergence speed - 0.001 caused instability while 0.00001 was too slow. AdamW was chosen for better regularization than Adam and adaptive per-parameter rates versus SGD.

Scheduler: StepLR with step size 2 and gamma 0.1, reducing learning rate by 10× every 2 epochs (1e-4 → 1e-5 → 1e-6). This creates three training phases: rapid learning, refinement, and convergence. StepLR provides clear phase separation and reproducibility for our 10-epoch training.

Training Configuration: Batch size of 4 balances memory and stability. Training for 10 epochs with no gradient clipping (stable gradients) and no augmentation (consistent evaluation protocol).

Transfer Learning:

Optimizer: SGD with Momentum

- Stage 1: Learning rate 0.005, momentum 0.9, weight decay 0.0005
- Stage 2: Learning rate 0.001, momentum 0.9, weight decay 0.0005

SGD with momentum is the standard optimizer for Mask R-CNN and object detection models, widely used in the original Detectron implementations. The two-stage learning rate strategy allows faster learning when training only the prediction heads (Stage 1, LR=0.005), then more conservative fine-tuning when the entire network is unfrozen (Stage 2, LR=0.001). This 5× reduction prevents destabilizing the pretrained backbone features. Momentum (0.9) accelerates convergence and smooths gradient updates, while weight decay (0.0005) provides L2 regularization to prevent overfitting.

Training Configuration

- Batch size: 2
- Stage 1 epochs: 5 (frozen backbone, train heads only)
- Stage 2 epochs: 5 (unfreeze backbone, full fine-tuning)

The small batch size (2) accommodates GPU memory constraints while still allowing gradient accumulation. The two-stage approach with frozen/unfrozen backbone is a transfer learning best practice: Stage 1 quickly adapts the task-specific heads without disturbing pretrained features, while Stage 2 allows the backbone to fine-tune to the target domain.

Short training (5 total epochs) leverages strong COCO pretraining to avoid overfitting on the small cell dataset.

Comparison of models

Results from Transfer learning:
Final Validation Metrics (IoU threshold: 0.5):

Mean IoU:	0.6841
Mean Precision:	0.8223
Mean Recall:	0.8220
F1 Score:	0.8222

Results got from Custom architecture:
Test Results:

IoU:	0.5649
Precision:	0.6423
Recall:	0.4742
F1 Score:	0.5456

Github repository

https://github.com/jakubradziejewski/livecell-instance-segmentation.git

Bibliography

Selected Dataset: LIVECell.
Link to the page where data and annotations can be downloaded from:
https://sartorius-research.github.io/LIVECell/
The dataset was prepared for the following article: Edlund, C., Jackson, T.R., Khalid, N. et al. LIVECell—A large-scale dataset for label-free live cell segmentation. Nat Methods 18, 1038–1045 (2021). https://doi.org/10.1038/s41592-021-01249-6.

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

Convolutional Block Attention Module

Sanghyun Woo, Jongchan Park, Joon-Young Lee, In So Kweon
https://arxiv.org/abs/1807.06521