2025 Visual Recognition Deep Learning –HW 3

313553014 廖怡誠

NYCU ACM

HW3 GitHub Link

1 Introduction

This homework focuses on the Instance Segmentation task[40, 6], a challenging problem in computer vision that requires both object detection and semantic segmentation capabilities. Unlike object detection, which only outputs bounding boxes, or semantic segmentation, which classifies each pixel without distinguishing between object instances, instance segmentation aims to assign a unique label to each object of interest and accurately delineate its shape at the pixel level. This dual requirement makes it a more complex task, especially in dense or overlapping regions—common in medical imagery. In this assignment, I'm provided with 209 colored medical images for training/validation and 101 test images, each containing varying numbers and shapes of cells from four different classes. Each cell must be segmented as a separate instance, even if it belongs to the same category. The ground truth is given as multiple binary .tif masks per image (e.g., class1.tif, class2.tif, ...), where pixel values indicate different instances.

To solve this task effectively under the provided constraints (no external data, model size $<$ 200M, vision-only models), I adopt and improve upon the Mask R-CNN framework[39], a well-established baseline for instance segmentation. My core strategy includes:

- **Data preprocessing and tiling**: To handle large image sizes and increase effective batch size under GPU memory limitations[31, 22, 37, 13, 17, 16, 12, 36, 41, 32], I divide images into manageable tiles and filter out empty tiles to focus computation on informative regions.

- **Augmentation strategies**: I employ Albumentations-based augmentations (flip, rotation, color jitter, etc.) to improve model robustness and generalization.

- **Loss and training refinements**: I explore training stabilization through carefully chosen learning rates, warm-up schedules, and weighted loss composition.

- **Evaluation pipeline**: I ensure the predicted masks are converted into COCO-compatible Run-Length Encoding (RLE) format, and we locally verify results using pycocotools[24, 2, 23] before submission to the CodaBench leaderboard.

These combined techniques aim to balance accuracy, computational efficiency, and robustness, helping us achieve high AP[27] scores on the benchmark while respecting all competition constraints.

2 Method

This section outlines four key aspects of the methodology: data processing, model architecture, training details and the inference strategies employed in the experiments.

In this howework3, I created virtural environment using anaconda[29, 47]. I use the pytorch[28, 45] training my model. To efficiently calculate and visualize the results, I imported the numpy[30] and opencv libraries[1].

2.1 Data Preprocessing

To address GPU memory constraints and improve the model's ability to focus on local features, we preprocess the medical images by tiling each image into fixed-size patches (e.g., 512×512). This approach ensures that the model can process high-resolution details without downscaling the entire image. We filter out empty or mask-less tiles to avoid wasting computation on irrelevant regions. During training, we apply the following data augmentations using Albumentations [20]:

- Horizontal and vertical flips (p=0.5)
- Random rotation (0°, 90°, 180°, 270°)
- Shift, scale, and rotate with small limits (p=0.5)
- Color jitter: brightness/contrast adjustment, hue/saturation shift (p=0.3)
- All images and masks are normalized and converted to tensors before feeding into the model.

Some pictures after data augmentation:

2.2 Model Architecture

I build our instance segmentation model by customizing Mask R-CNN [39] with a more powerful backbone:

- **Backbone**: I use ResNeXt-101-32x8d[18], a high-capacity architecture known for its group convolution design, which improves representation learning especially on complex textures like medical images. The pretrained weights from ImageNet are loaded for better initialization.

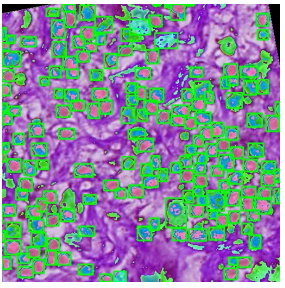

Fig. 1. sample 1

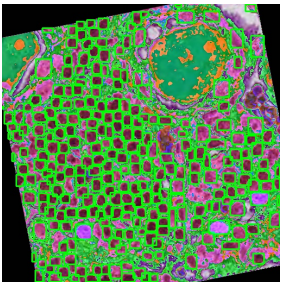

Fig. 2. sample 2

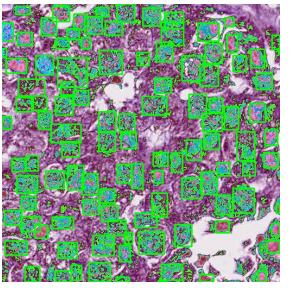

Fig. 3. sample 3

– **Beck (FPN)**: I integrate a Feature Pyramid Network (FPN)[9, 19] on top of the backbone using PyTorch's resnet fpn extractor, which combines multiscale features from different backbone stages to improve small and large object detection.

– **Region Proposal Network (RPN)**: Generates object proposals across FPN feature maps with different aspect ratios and scales.

– **RoI Align**: Aligns region proposals into fixed spatial dimensions before feeding into the heads.

– **Box Head**:
  - **Classifier**: Replaced with FastRCNNPredictor to support our custom number of cell classes (4 + 1 background).
  - **Regressor**: : Predicts bounding box coordinates for each instance.

– **Mask Head**:
  - Replaced with MaskRCNNPredictor, a lightweight FCN structure using a hidden dimension of 256.
  - Outputs a binary mask per class for each detected object.

This architecture maintains the total parameter count under 200M, as required, and leverages the enhanced capacity of ResNeXt-101[18, 8] for better instance differentiation in cluttered medical scenes.

2.3 Training Details

The model is trained using Stochastic Gradient Descent (SGD)[14, 4] with a learning rate of 0.005, momentum 0.9, and weight decay 0.0001.I removed the optimizer scheduler because fine-tuning the weights requires generalizing to the current training data distribution. Therefore, very small learning rates can make my model stuck in a local minimum and limit performance. The training is

conducted for 200 epochs with a batch size of 2 (adjusted depending on the GPU memory, I used two NVIDIA 4090 GPUs with 24GB of VRAM and used a distributed data parallel[34] architecture to distribute the training data). I monitor validation mAP@0.5 and save the best-performing model accordingly.

2.4 Inference Pipeline

During inference, the trained MaskRCNN_ResNeXt101 model[18, 8] is loaded along with its checkpoint and set to evaluation mode. Each test image is read using OpenCV[1], converted to RGB format, normalized with the same parameters used during training, and transformed into a 4D tensor suitable for model input.

The model outputs include bounding boxes, class labels, predicted masks, and confidence scores. We iterate through each prediction and apply a confidence threshold (default: 0.5). For every detection above the threshold, we extract the binary mask (thresholded at 0.5) and convert it to Run-Length Encoding (RLE) using pycocotools.mask.encode. Each result is stored with its image_id, category_id, RLE-encoded segmentation, and score.

Finally, all predictions are compiled into a single JSON file (test-results.json) that conforms to the COCO submission format and can be validated locally using the COCO evaluation API before submitting to Codabench.

3 Final & Additional Experimental Result

This section presents a comparative analysis of several experimental configurations. The ablation studies focus on five key factors:

3.1 The Impact of Data Augmentation

The training loss curve presented above offers a clear comparison between two data augmentation strategies applied to the instance segmentation task. The magenta line represents the training process using minimal data augmentation, where only horizontal and vertical flipping were applied. In contrast, the cyan line corresponds to a model trained with richer augmentation, including horizontal and vertical flips, random rotation, ShiftScaleRotate, and normalization.

From the curves, we observe that both models initially experience a steep drop in loss during the early training steps, which is expected as the model rapidly learns basic spatial features. However, after around step 5, the two training curves begin to diverge. The model with less augmentation (magenta) shows higher overall loss and eventually plateaus at a higher value (0.92), while the more augmented model (cyan) continues to descend and converges to a lower final loss (0.87). The smoothed loss values also confirm this: 0.8751 for the more augmented model versus 0.9167 for the less augmented one.

This behavior can be attributed to the richer augmentation' s ability to introduce more diversity and robustness into the training data. By exposing

the model to more geometric variations (rotation, scale changes, and shifts), it becomes better at generalizing to unseen examples and less prone to overfitting on specific spatial layouts. In contrast, the less augmented model likely overfits more quickly to limited visual patterns and thus struggles to continue improving.

Another key point is that the training duration (relative time) for the more augmented model is slightly longer, which is expected due to the computational cost of applying more complex transformations. However, this additional cost results in a better-optimized model, as reflected by the consistently lower loss curve.

Overall, this experiment highlights that appropriate and diverse data augmentation is critical in medical instance segmentation tasks, where the appearance and orientation of instances (e.g., cells) can vary significantly. Augmentation not only improves generalization but also accelerates convergence by helping the model learn invariant and transferable features.

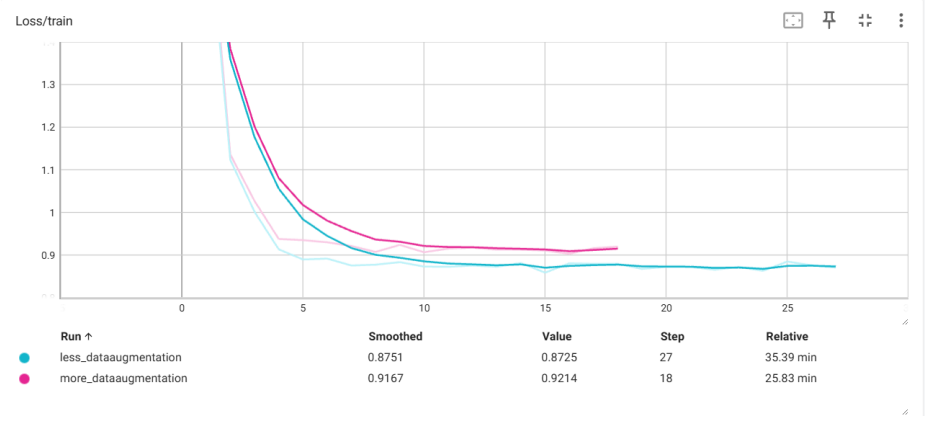

Fig. 4. The comparison of different data augmentation

3.2 Selection of Scheduler

Based on the training loss curves shown above, we observe a striking performance gap between two learning rate strategies: one using StepLR scheduler (green curve) and the other with no learning rate scheduler (orange curve).

Initially, both models exhibit similar behavior during the very early steps. However, the model using StepLR[38] quickly plateaus at a high loss value (0.93) and remains stagnant throughout training. In contrast, the model without scheduler continues to improve steadily, achieving a much lower final loss (0.54) after 151 steps, suggesting significantly better convergence.

In earlier experiments, StepLR[38] was consistently applied, yet performance gains were limited regardless of how the decay rate or step size was tuned. This

led to the suspicion that the scheduler was hindering rather than helping. A plausible explanation is that StepLR[38] reduced the learning rate too early, causing the model to converge prematurely to a suboptimal solution and restricting its ability to explore a better loss landscape. Once the learning rate decayed, the model lacked the momentum to escape poor minima, effectively "freezing" learning progress.

After removing the scheduler entirely, the model exhibited stable and gradual improvement, suggesting that the fixed learning rate was more appropriate for this setting. Without abrupt learning rate drops, the optimizer was able to continue making consistent progress and fully exploit the training signal across epochs.

In conclusion, although learning rate scheduling is often beneficial in deep learning, this experiment reveals that a poorly tuned scheduler can severely degrade performance. In this case, eliminating StepLR led to dramatically better convergence, likely because the model retained sufficient learning capacity throughout training without being prematurely stalled by aggressive decay.

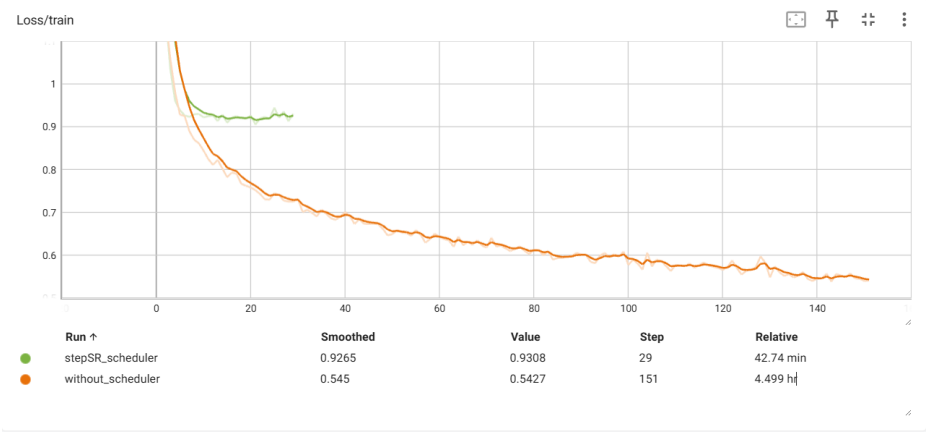

Fig. 5. Comparison of whether the scheduler is used

3.3 The Impact of Different Anchor Sizes in the Region Proposal Network

This training loss comparison explores the effect of modifying anchor sizes in the Region Proposal Network (RPN)[10] on model performance. The purple curve represents the training loss when using smaller anchor sizes—specifically adjusted to (32, 64, 128, 256, 512)—while the green curve corresponds to the default anchor size configuration provided in torchvision' s Mask R-CNN[39].

As shown in the plot, the model with smaller anchor sizes exhibits consistently higher loss throughout training, converging to a final loss of 1.08. In

contrast, the default anchor configuration leads to a significantly lower final loss of 0.93, indicating better optimization and likely superior detection quality.

The performance degradation from using smaller anchors may stem from a mismatch between anchor scale and object scale in the dataset. Since the task involves high-resolution medical images—where cells can occupy relatively large regions—smaller anchors may fail to adequately match and cover these larger objects. This mismatch results in poorer region proposals, less effective learning signals for the RPN, and ultimately degraded overall training performance.

Additionally, the training curves indicate that not only is the loss higher for smaller anchors, but convergence is also slower and more stagnant, suggesting that the model struggles to adapt under the modified anchor configuration. This further supports the hypothesis that the original anchor sizes are better aligned with the object scales present in the dataset.

In summary, this experiment shows that choosing appropriate anchor sizes is critical, especially when dealing with high-resolution inputs. Reducing anchor scales indiscriminately can harm proposal quality, reduce detection coverage, and ultimately degrade model performance—particularly in domains like medical imaging where object sizes tend to be large and variable.

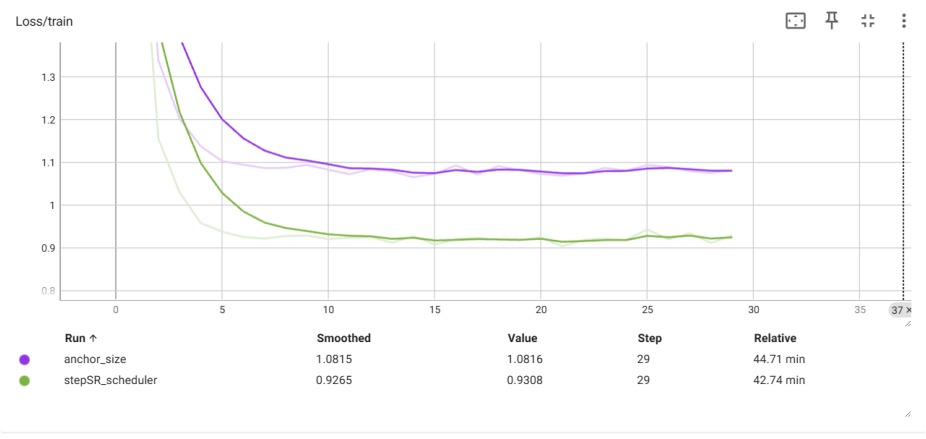

Fig. 6. The comparison of different anchor sizes in the region proposal network

3.4 Selection of Training Model

The two sets of training loss curves above illustrate the effect of replacing the backbone network from ResNet-50 (yellow)[8] to a deeper ResNet-101 (pink)[18], and further investigate the effect of tiling with or without overlap on training performance and dataset richness.

In the first figure, we compare the two backbones directly. The ResNet-101 model (pink), though slower to train due to increased computational complexity,

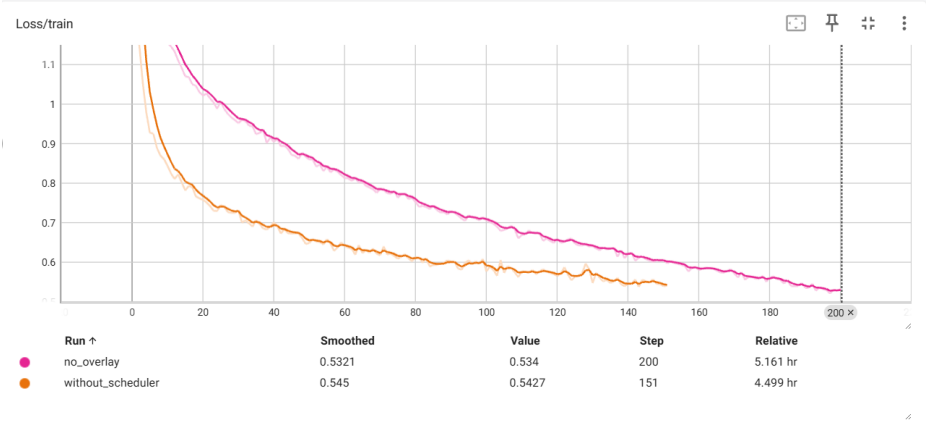

Fig. 7. The comparison of backbone between ResNet50 and ResNet101

demonstrates a steadier and more consistent loss decrease. In contrast, ResNet-50 (yellow) converges faster in early iterations but begins to plateau earlier. This behavior is expected: ResNet-101, with its deeper architecture and higher representational capacity, learns features more effectively—especially in a complex domain like medical imaging—yet requires more training steps to fully realize its potential.

However, due to memory limitations, full-resolution images could not be used when training with ResNet-101. Instead, the images were tiled into 512×512 patches, increasing training time but allowing high-resolution input to remain intact. Despite the added training cost, ResNet-101 demonstrates promising results. Although the loss curves haven't crossed, the pink curve continues to decrease smoothly, suggesting that given more time or epochs, ResNet-101 could eventually outperform ResNet-50.

The second figure explores the impact of tiling strategy using ResNet-101. The pink curve represents training without tile overlap, while the yellow curve introduces tile overlap during patch extraction, effectively increasing the training dataset size. We can clearly observe that the overlapped tiles (yellow) yield faster convergence and better loss values early in training. The richer, more redundant dataset likely provides the model with more diverse views of cell boundaries and contexts, helping it generalize better and reduce overfitting. Though training with overlap is more computationally intensive (6.058 hours vs. 5.161 hours), the performance gain in early-stage loss suggests a worthwhile tradeoff.

Together, these experiments demonstrate two key insights. First, deeper backbones like ResNet-101 are beneficial in terms of model capacity, but they demand careful input handling (e.g., tiling) and longer training schedules. Second, strategic tiling with overlap acts as a form of data augmentation, enriching the training distribution without altering the original data, leading to meaningful gains in learning stability and early convergence.

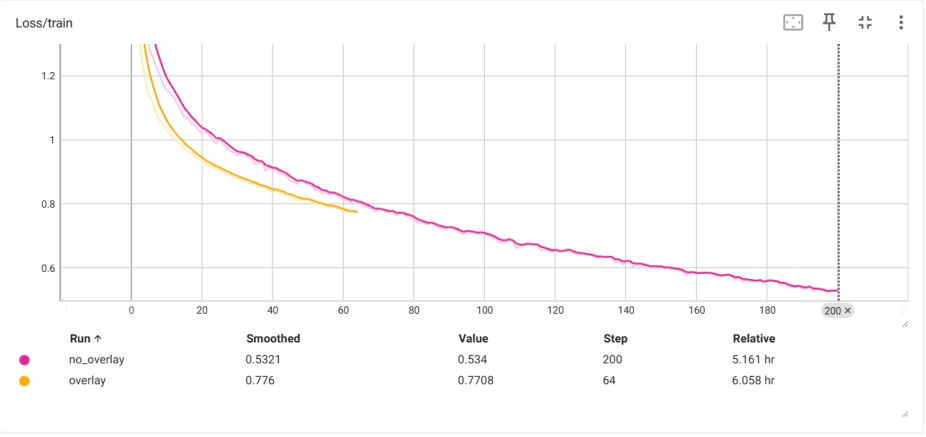

Fig. 8. The comparison of whether tileing images are overlap

In summary, even though the full potential of ResNet-101 has yet to be explored due to limited training time, the results point toward its effectiveness, especially when paired with overlap tiling strategies that compensate for input size constraints and boost training diversity.

3.5 Summary

After a series of ablation studies and training experiments, I finalized my model configuration based on a combination of performance, stability, and training time considerations. The final submitted model uses a ResNet-50 backbone, default anchor sizes, and no learning rate scheduler.

Through analysis of the training loss curves, I found that removing StepLR significantly improved convergence. When StepLR was used, the model often plateaued early due to premature learning rate decay, which hindered the ability to escape suboptimal minima. Without the scheduler, the model maintained a consistent learning pace throughout training, ultimately resulting in better optimization and lower final loss.

Regarding anchor design, I experimented with reducing anchor sizes but observed that this negatively affected performance. Given that the medical images used in this task were high-resolution and often contained large cell structures, the default anchor sizes were more suitable for capturing full object extents, leading to better region proposals and mask quality.

Although I attempted to use a ResNet-101 backbone for its greater feature capacity, this required tiling the images into 512×512 patches due to GPU memory constraints. While initial results were promising—with early validation loss reaching as low as 0.35—the model could not be fully trained due to time limitations. From these early results and the behavior of the loss curve, it is likely that a fully trained ResNet-101 model would outperform the ResNet-50 configuration.

My final submission, trained with ResNet-50 (no scheduler, default anchors), achieved a private score of 0.39. Based on preliminary experiments with ResNet-101, I estimate that with sufficient training time and the use of overlapping tiles to enhance data richness, the score could further improve beyond 0.39.

Some final result show at below:

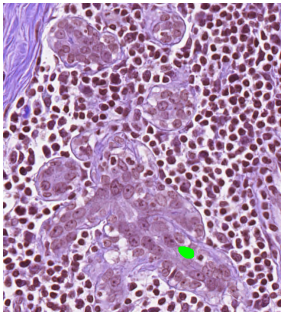

Fig. 9. Final result 1

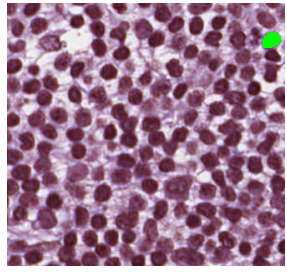

Fig. 10. Final result 2

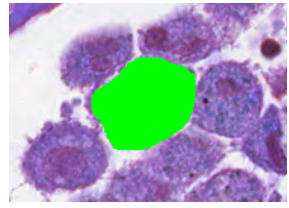

Fig. 11. Final result 3

References

[1] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[2] Stefan Hoops et al. "COPASI—a COmplex PAthway SImulator". In: *Bioinformatics* 22.24 (Oct. 2006), pp. 3067–3074. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btl485. eprint: https://academic.oup.com/bioinformatics/article-pdf/22/24/3067/48838880/bioinformatics_22_24_3067.pdf. URL: https://doi.org/10.1093/bioinformatics/btl485.

[3] Kai Ming Ting. "Confusion Matrix". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 209–209. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_157. URL: https://doi.org/10.1007/978-0-387-30164-8_157.

[4] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). URL: https://api.semanticscholar.org/CorpusID:6628106.

[5] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[6] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[7] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *CoRR* abs/1411.4038 (2014). arXiv: 1411.4038. URL: http://arxiv.org/abs/1411.4038.

[8] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[9] Sergey Ioffe and Christian Szegedy. "Batch normalization: accelerating deep network training by reducing internal covariate shift". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, 2015, pp. 448–456.

[10] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

[11] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[12] Tianqi Chen et al. "Training Deep Nets with Sublinear Memory Cost". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems (NeurIPS)*. 2016, pp. 4129–4137.

[13] Tianqi Chen et al. "Training Deep Nets with Sublinear Memory Cost". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems (NeurIPS)*. 2016, pp. 4129–4137.

[14] Ilya Loshchilov and Frank Hutter. "SGDR: Stochastic Gradient Descent with Restarts". In: *CoRR* abs/1608.03983 (2016). arXiv: 1608.03983. URL: http://arxiv.org/abs/1608.03983.

[15] TorchVision maintainers and contributors. *TorchVision: PyTorch's Computer Vision library*. https://github.com/pytorch/vision. 2016.

[16] Minsoo Rhu et al. "vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design". In: *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE Computer Society, 2016, 18:1–18:13. DOI: 10.1109/MICRO.2016.7783721.

[17] Minsoo Rhu et al. "vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design". In: *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE Computer Society, 2016, 18:1–18:13. DOI: 10.1109/MICRO.2016.7783721.

[18] Saining Xie et al. "Aggregated Residual Transformations for Deep Neural Networks". In: *CoRR* abs/1611.05431 (2016). arXiv: 1611.05431. URL: http://arxiv.org/abs/1611.05431.

[19] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[20] Alexander V. Buslaev et al. “Albumentations: fast and flexible image augmentations”. In: *CoRR* abs/1809.06839 (2018). arXiv: 1809.06839. URL: http://arxiv.org/abs/1809.06839.

[21] Kiri Choi et al. “Tellurium: An extensible python-based modeling environment for systems and synthetic biology”. In: *Biosystems* 171 (2018), pp. 74–79. DOI: 10.1016/j.biosystems.2018.07.006.

[22] Yujun Lin et al. “Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training”. In: *International Conference on Learning Representations (ICLR)*. 2018. URL: https://openreview.net/forum?id=SkhQHMW0W.

[23] Jeremiah K Medley et al. “Tellurium notebooks—An environment for reproducible dynamical modeling in systems biology”. In: *PLoS Computational Biology* 14.6 (2018), e1006220. DOI: 10.1371/journal.pcbi.1006220.

[24] Ciaran M Welsh et al. “PyCoTools: a Python toolbox for COPASI”. In: *Bioinformatics* 34.21 (May 2018), pp. 3702–3710. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bty409. eprint: https://academic.oup.com/bioinformatics/article-pdf/34/21/3702/48921301/bioinformatics_34_21_3702.pdf. URL: https://doi.org/10.1093/bioinformatics/bty409.

[25] Casper O. da Costa-Luis. “‘tqdm‘: A Fast, Extensible Progress Meter for Python and CLI”. In: *Journal of Open Source Software* 4.37 (2019), p. 1277. DOI: 10.21105/joss.01277. URL: https://doi.org/10.21105/joss.01277.

[26] Hao Fu et al. “Cyclical Annealing Schedule: A Simple Approach to Mitigating KL Vanishing”. In: *NAACL*. 2019.

[27] Huizi Mao, Xiaodong Yang, and William J. Dally. “A Delay Metric for Video Object Detection: What Average Precision Fails to Tell”. In: *CoRR* abs/1908.06368 (2019). arXiv: 1908.06368. URL: http://arxiv.org/abs/1908.06368.

[28] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *CoRR* abs/1912.01703 (2019). arXiv: 1912.01703. URL: http://arxiv.org/abs/1912.01703.

[29] *Anaconda Software Distribution*. Version Vers. 2-2.4.0. 2020. URL: https://docs.anaconda.com/.

[30] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[31] Hengyuan Hu et al. “Memory-Efficient Adaptive Optimization for Large-Scale Learning”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/44feb0096faa4d52a3af1df8f8b84903-Abstract.html.

[32] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. “Reformer: The Efficient Transformer”. In: *Proceedings of the 8th International Conference on Learning Representations (ICLR)*. 2020.

[33] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. "Reformer: The Efficient Transformer". In: *Proceedings of the 8th International Conference on Learning Representations (ICLR)*. 2020.

[34] Shen Li et al. "PyTorch Distributed: Experiences on Accelerating Data Parallel Training". In: *CoRR* abs/2006.15704 (2020). arXiv: 2006.15704. URL: https://arxiv.org/abs/2006.15704.

[35] Colin Raffel et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *Journal of Machine Learning Research* 21.140 (2020). Originally presented as arXiv:1910.10683, pp. 1–67. URL: http://jmlr.org/papers/v21/20-074.html.

[36] Samyam Rajbhandari et al. "ZeRO: Memory Optimizations Toward Training Trillion Parameter Models". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE, 2020, pp. 1–16. DOI: 10.1109/SC41405.2020.00024.

[37] Samyam Rajbhandari et al. "ZeRO: Memory Optimizations Toward Training Trillion Parameter Models". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE, 2020, pp. 1–16. DOI: 10.1109/SC41405.2020.00024.

[38] Chiheon Kim et al. "Automated Learning Rate Scheduler for Large-batch Training". In: *CoRR* abs/2107.05855 (2021). arXiv: 2107.05855. URL: https://arxiv.org/abs/2107.05855.

[39] Yanghao Li et al. "Benchmarking Detection Transfer Learning with Vision Transformers". In: *CoRR* abs/2111.11429 (2021). arXiv: 2111.11429. URL: https://arxiv.org/abs/2111.11429.

[40] Ze Liu et al. "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows". In: *CoRR* abs/2103.14030 (2021). arXiv: 2103.14030. URL: https://arxiv.org/abs/2103.14030.

[41] Bin Ren, Jiajia Li, Xipeng Shen, et al. "Offloading Activations with Tiling for Large-Scale Deep Learning". In: *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2021, pp. 145–158. DOI: 10.1145/3445814.3446710.

[42] Bin Ren, Jiajia Li, Xipeng Shen, et al. "Offloading Activations with Tiling for Large-Scale Deep Learning". In: *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2021, pp. 145–158. DOI: 10.1145/3445814.3446710.

[43] Yao-Yuan Yang et al. "TorchAudio: Building Blocks for Audio and Speech Processing". In: *arXiv preprint arXiv:2110.15018* (2021).

[44] Jeff Hwang et al. *TorchAudio 2.1: Advancing speech recognition, self-supervised learning, and audio processing components for PyTorch*. 2023. arXiv: 2310.17864 [eess.AS].

[45] Jason Ansel et al. "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation". In: *29th ACM International Conference on Architectural Support for Programming Lan-*

guages and Operating Systems, Volume 2 (ASPLOS '24). ACM, Apr. 2024. DOI: 10.1145/3620665.3640366. URL: https://pytorch.org/assets/pytorch2-2.pdf.

[46] Jiachen Zhu et al. "Transformers without Normalization". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2025.

[47] conda contributors. *conda: A system-level, binary package and environment manager running on all major operating systems and platforms.* URL: https://github.com/conda/conda.