

Finding a Faster UNet for Satellite Image Segmentation

Michael Cai, Lantian Zhang

Abstract

Our project focuses on enhancing the efficiency of the UNet architecture for satellite image semantic segmentation while retaining accuracy. Our motivation is driven by the limited computational resources and the vast number of satellite images available; as well as the potential need of live segmentation tasks. We explored several optimization techniques with UNet's components to accelerate both training and inference times. Using a specially trimmed model, our modified UNet achieves faster speed. We evaluated our modified UNet on two popular datasets, LandCoverAI and LoveDA. Our work aims to provide insights on how UNet can adapt to satellite image segmentation tasks with an emphasis on the spatial details and speed. Our code for this project can be found here: https://github.com/lantianzhang/CSCI2271_Final_Project

Introduction

Satellite image semantic segmentation is a critical task in the field of computer vision, playing a pivotal role in various applications such as environmental monitoring, urban planning, and disaster management. In recent years, the application of deep learning techniques has significantly advanced the accuracy and efficiency of semantic segmentation tasks. Our project tries to contribute to the ongoing progress of this field by focusing on the enhancement of existing architectures for semantic segmentation of satellite images.

The challenges presented in satellite image segmentation include multi-scale objects, as the dataset includes both urban and rural scenes in different geographical landscapes from different locations; complex background samples, as the high-resolution and different complex scenes bring more rich details and intra-class variance; and inconsistent class distributions,

UNet, introduced in 2015, has proven to be a robust choice for image segmentation tasks. It features using skip connection to connect the encoder and decoder stages. One crucial feature is the use of up-sampling instead of pooling to reconstruct the full picture during the decoding phase[7]. This ensures that spatial information, especially fine-grained details, is preserved, which is essential for accurate land cover classification in satellite images.

In addition to UNet, we took a look at some influential segmentation models such as UNet++ and DeepLabV3. UNet++ enhances the skip connection by combining and upsampling from the skip connections upwards (overall into smaller), this means the input to the final decoder is a skip connection that is a combination of all previous inputs. From the paper UNet++ performs 3% better in testing than the original UNet[8]. DeepLab is based on atrous convolution combined with pyramid pooling to construct a deeper network that retains more high level information at finer resolutions without increasing the number of parameters[2].

Methodology

Datasets used

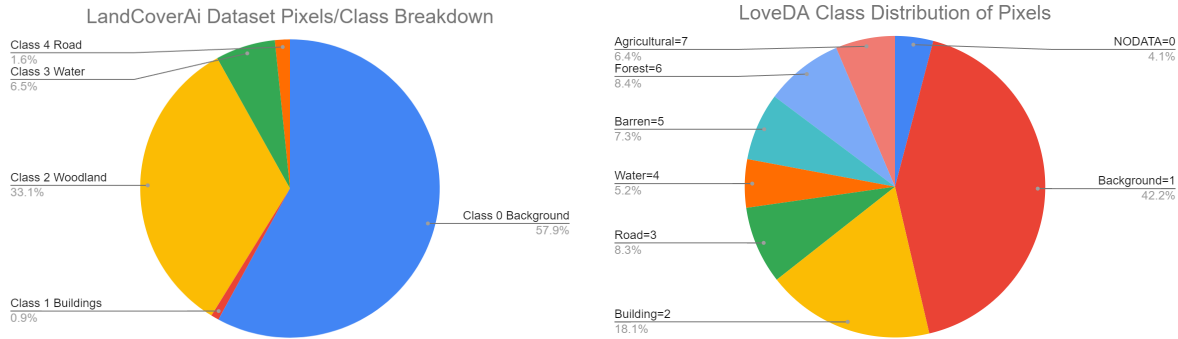
To establish a robust benchmark for our approach, we utilized two publicly available datasets: LoveDA and LandCoverAI.

The LoveDA dataset provides a diverse collection of annotated satellite images tailored for land cover segmentation and classification tasks in Chinese cities of Nanjing, Changzhou and Wuhan in July 2016, totaling 536.15 km²[5]. There are 7 annotated classes in this dataset: building, road, water, barren, forest, agriculture, and background; in addition to a class where there is no data/blank. The spatial resolution is 0.3 meter, with red, green and blue bands[5].

The LandCoverAI dataset covers 216.27 km² of land in Poland from images taken from 2015 to 2018[4]. There are 4 annotated classes: building, woodland, water, road, and background. The spatial resolution is 0.25 or 0.5 meter, with red, green and blue bands[4].

Both datasets have actual images with corresponding segmentation masks (ground truth). The images in the LoveDA dataset are 1024 by 1024 pixels, whereas images in the LandCoverAI dataset are 512 by 512 pixels. In order to make the two datasets uniform in terms of resolution, we split each LoveDA image into four 512 by 512 pixel tiles.

For training, we have 7470 images in the LandCoverAI dataset and 1512 images in the LoveDA dataset. The LandCoverAI is accompanied with a python script to split the source images into training, validation, and testing images. Then, the incorporation of data augmentation becomes imperative, particularly when dealing with smaller datasets, to enhance the model's ability to handle variations and improve its robustness. We applied multiple image augmentation techniques, including randomness in hue, brightness, contrast, and, flip and horizontal flips.



Models Used

For the models that we test, we are using a combination of UNet models that we define and UNet++ and Deeplabv3+ models with “resnet34” encoders and “imagenet” weights from the segmentation-models-pytorch library[9]. We also used the torch-summary library to get a profile of our models for parameter count and memory size. We targeted lower parameter counts and memory size for our custom UNet models to improve speed when compared to the base implementation as memory size used is often correlated to model speeds in UNet[6].

We define our own models by implementing the following classes: Encoder, Decoder, and ConvBlock that can be parameterized with the target input/output depth for features as well as the amount of convolutions per block.

Our implementations of U-Net models consist of the following, a base model that closely follows the original UNet implementation, a “short” model that removes a layer from the UNet, a “skinny” model that reduces the depth and (3x3) convolutions per layer but adds an additional layer, and a “modified convolution” model that redistributes the convolution of all layers in a way that we think could help in the satellite segmentation task. In figures 1-4 we show how these models are implemented.

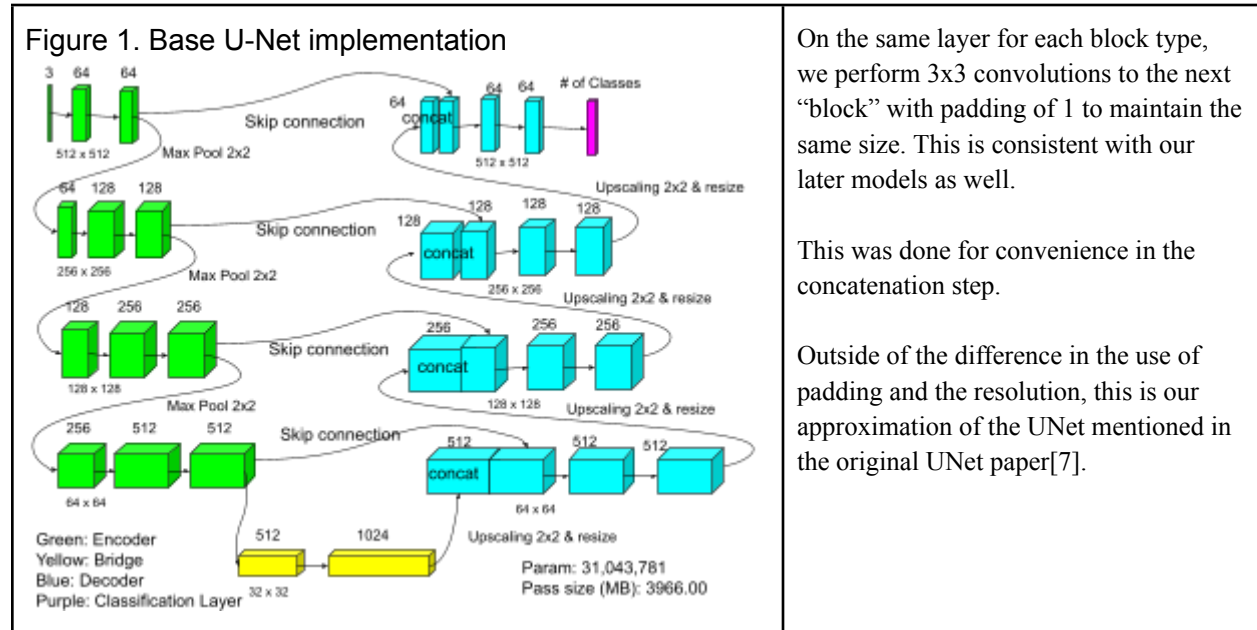
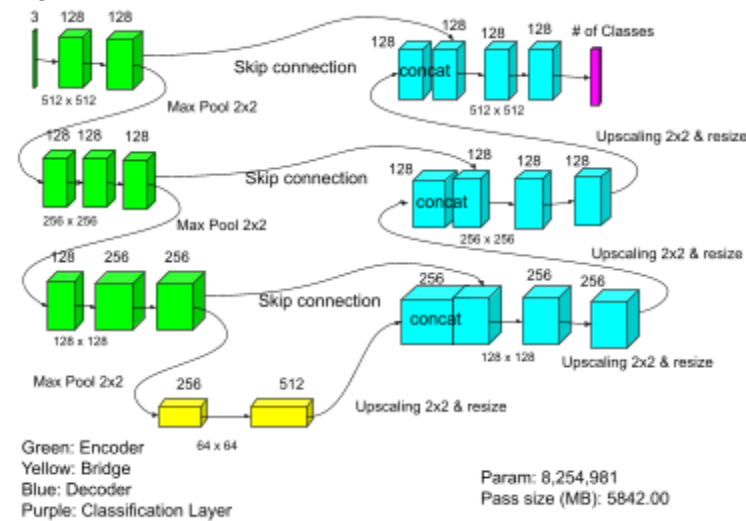


Figure 2. Shorter U-Net implementation

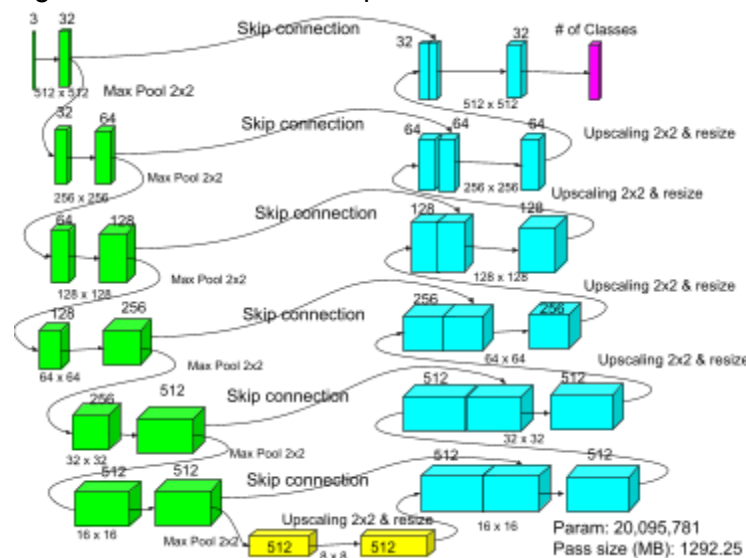


In our shorter UNet implementation we wanted to allocate more resources in earlier stages to offset the loss in layers.

This was mostly an educational experiment to see the performance impact of leaving out the larger context layers and focusing on local patterns.

The memory used is higher despite the lower parameter count due to each pixel of the higher resolution layers having more depth, increasing the memory dramatically.

Figure 3. Skinnier U-Net implementation



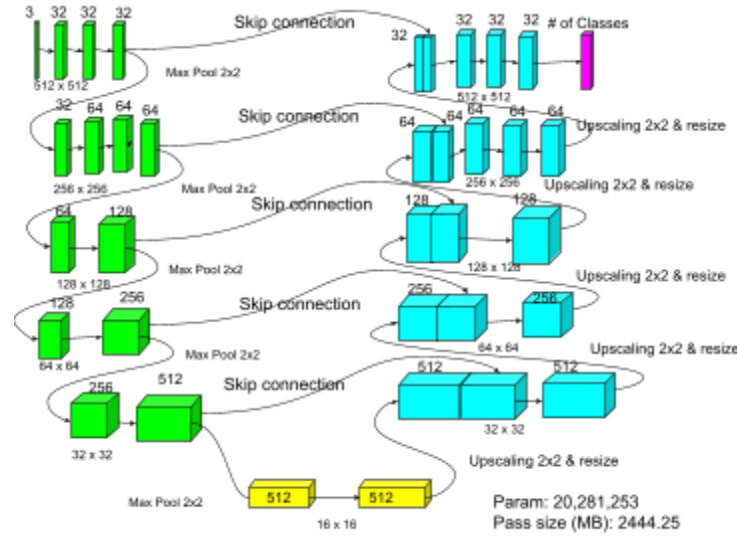
For this implementation, we have 6 encoding/decoding layers which is 1 more than the baseline we had.

We reduced the number of convolutions per layer as well as decreased feature depth.

The bottleneck is now working with a 8x8 space to effectively break the image down to 64 parts at the lowest level.

We also dramatically decreased the memory footprint despite retaining 20M parameters in order to achieve faster performance.

Figure 4. Convolution Modified U-Net (Less Conv)



For this implementation we had 3 convolutions in the top 2 rows to assist in finding smaller features in a satellite image.

In order to achieve the speed requirements, we did reduce the convolutions in the later stages to only 1.

Thus we achieve a 20.3M param model with a modest memory footprint.

Training and metrics

For the training loop, we used the Adam optimizer with a custom loss function “JDTLoss” from the paper “Jaccard Metric Losses: Optimizing the Jaccard Index with Soft Labels”[3]. The reason we chose JDTLoss is because it can simulate Jaccard, Dice, and Tversky loss all in one function based on the parameters given. This will allow us to fine tune and experiment with different options in our models. The main advantage of JDTLoss is the ability to work with soft labels using the Jaccard metric, however we are not using this feature in this study[3].

We chose ‘JDTLoss’ in its default ‘Jaccard’ mode over Cross Entropy, as the referenced study reported a 2% improvement with their default metric[3]. Additionally, we adapted the function to run efficiently on GPUs and accept weight parameters.

To enhance GPU memory usage, we implemented GradScaler() and employed accumulated losses to simulate larger batch sizes, addressing potential limitations when the actual batch size exceeds GPU capacity.

Table 1. Standard training configuration

Epoch limit	50 due to time considerations. The fastest variation still takes ~5 hours to train.
loss function	JDTLoss(alpha=1, beta=1, gamma = 1.0 ,device = device).to(device)
optimizer	torch.optim.Adam(model.parameters(), lr=0.001 , weight_decay = 0.001)
early_stopping	5
lr_scheduler	ReduceLROnPlateau(optimizer, mode='min', patience=2, factor=0.5)
Logical Batch Size	8

Table 2. Metrics explanations

Average Epoch Time	time in seconds from the start of the training epoch to the end of the validation epoch in the loop. This should be considered as a relative measurement as it is machine / environment dependent.
mIOU per class	add IOU for the class if they are present in the image for every image and then divide by the number of images they are present in.

overall mIOU	Sum of mIOU per class / all classes. Thus all have equal impact.
pass size (MB)	The amount of memory measured by torchsummary that is needed to perform a forward / backward pass on the model.

Experiments / Results

Table 3. Love DA Results (Run on Google Colab V100) *Best, Second, Third*

Model s	Param _count	Memor y_Footp rint (MB)	Avg_T rain_T ime_p er_ep och (s)	mIOU all classes	noDat a class 0 mIOU	Backg round class 1 mIOU	Buildi ng class 2 mIOU	Road class 3 mIOU	Water class 4 mIOU	Barren class 5 mIOU	Forest class 6 mIOU	Agricu ltural class 7 mIOU
UNet_ Base	31,043, 976	3966	60.784 6	0.2217	0.6109	0.4128	0.2875	0.1543	0.0802	0.0735	0.1463	0.008
UNet_ Short	8,254,9 81	5842	78.002 9	0.2432	0.7718	0.4122	0.2835	0.1528	0.0808	0.0863	0.1494	0.0091
UNet_ Skinny	20,095, 781	1292.25	27.951 4	0.2639	0.6872	0.4457	0.3383	0.2134	0.1319	0.111	0.1672	0.0165
UNet_ Less_ Conv	20,281, 253	2444.25	35.741 9	0.2305	0.4996	0.4411	0.349	0.184	0.1182	0.1075	0.1347	0.0099
UNet+ +	26,079, 189	3333	39.911 2	0.2881	0.5497	0.4669	0.3845	0.2654	0.2639	0.1471	0.1895	0.0376
Deepla bv3+	22,438, 485	872.63	22.512	0.2571	0.6935	0.4261	0.2919	0.1604	0.2068	0.0963	0.1508	0.0307

Table 4. LandCoverAi Results (Run on r5 5600G, 48GB Ram, RTX 4060ti 16GB) *Best, Second, Third*

Models	Param_c ount	Memory_ Footprint (MB)	Avg_Trai n_Time_ per_ep och (s)	mIOU all classes	backgrou nd class 0 mIOU	Building class 1 mIOU	Woodlan d class 2 mIOU	Water class 3 mIOU	Road class 4 mIOU
UNet_Ba se	31,043,97 6	3966	730.9526	0.3246	0.6469	0.0812	0.5562	0.195	0.144
UNet_Sh ort	8,254,981	5842	468.471	0.2558	0.5893	0.0697	0.4353	0.077	0.1078
UNet_Ski nny	20,095,78 1	1292.25	312.0205	0.3983	0.6693	0.375	0.5011	0.1833	0.2631
UNet_Le ss_Conv	20,281,25 3	2444.25	443.4832	0.3195	0.6602	0.0865	0.5416	0.1657	0.1434
UNet++	26,079,18 9	3333	582.9711	0.3953	0.6848	0.2976	0.5839	0.134	0.2764
Deeplabv 3+	22,438,48 5	872.63	259.2879	0.3846	0.6827	0.2833	0.5761	0.1392	0.2419

Our results show that the average training speeds are indeed correlated to memory footprint, thus our attempt to simplify the model in UNet_Short has fallen short of the goal of being faster. Instead, this leads to the slowest model across all models tested. It also generally performed the worst, so we do not recommend skimping on layers for satellite segmentation.

Out of the custom models we ran, UNet_skinny stood out at having one of the highest mIOU across both datasets and the second fastest model only losing to Deeplabv3+ for training speeds. We conclude that this is due to the extra level of context provided in the lowest layer. In satellite segmentation tasks, it is critical to understand the environment and the larger picture in order to classify correctly. Take buildings and roads for example, they are more likely found together than not and that information can help discern a rectangle as a building rather than a rectangular patch of off colored grass. Similarly, many terrain are not singular plots but a continuous patch that can span multiple frames.

Challenges in getting higher mIOU


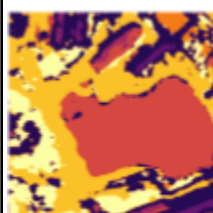


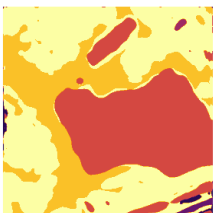

Outside of these general trends, we noted that these outcomes are in general pretty low. We have gone through many lengths to identify points where we could improve training performance.

We have experimented with many tuning parameters including simulating dice loss, using tversky loss at $\beta = 0.3, 0.5, 0.7$, Jaccard loss, cross entropy with weights, JDTLoss with weights, adding a dropout layer to the classification head, and implemented the augmentations that we could in the time constraints. Including inverse weights caused over-prediction of the lower frequency classes so we also tried square root weights but the outputs appear further off than having no weights. In the end, we did not find tuning to provide much improvement in mIOU performance that would benefit both datasets during our limited time.

We would also like to point out how underrepresented features generally did poorly. In LoveDA this would include water, barren and agriculture. In LandCoverAI it includes buildings, water, and roads. These features are often the ones that drag down the average mIOU score. The model could improve if the weights are based on an encoder that already works well for those weaker classes. In the LandCoverAI paper, they used weights from an encoder trained on Cityscapes to achieve their higher mIOU scores. This would be a valid strategy, but also means that learning from scratch would be a great disadvantage.

Another part of the challenge was the randomized augmentation. We generate the augmentations randomly when using the data loader, however this may not be the ideal approach as some augmentations may not look realistic such as severe hue changes. In the LandCoverAI paper they were able to use augmentations that we were not able to implement such as those that specifically simulates the seasons. With their augments, they were able to produce 74700 samples for the training set, however with our set of augmentation it is effectively less than half of that. Those specially made augments would greatly improve the ability for these models to learn the nuances in satellite imagery.

The data itself also has proven to be a challenge to work with. To exemplify this, we include samples from LoveDA and LandCoverAi:

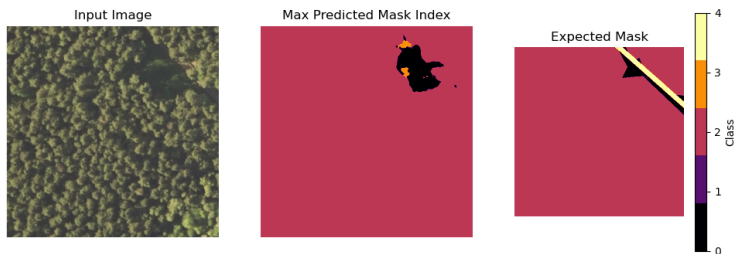
Original (LoveDA)	Base	Skinny	UNet++	Deeplabv3+	Target Mask
					

There is some discrepancy between what the target mask is and our own understanding of what the mask should be. For example, we cannot recognize that there is another waterbody to the left of the image, or can we annotate the whole of the upper left part of the image as background. UNet++ has the best predicted mask result, our skinny UNet model is producing a decent mask that captures the two bodies of water. while the original UNet is

capturing too many fine grain details. Interestingly, DeepLab does not capture any of the pixels as background class (class 1). This might explain why our mIOU is lower than expected, as the target mask's annotation is different from what one might expect.

Even in the LandCoverAI dataset we could notice challenging situations where a human would not be able to draw the correct mask with the given image.

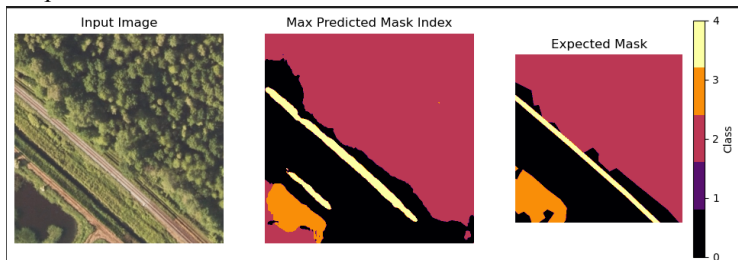
UNet++ example where the road is hidden in the shadows



UNet Skinny for comparison



DeepLabV3+ where another area looks like a road



UNet Skinny for comparison



These challenges, as well as the hours of training to measure and validate improvements of tuning, made it very difficult to produce a well performing situation for any of these models. But we were able to find distinctions between model speeds and general performance in our measurements.

Brief Conclusion / Discussion

In this project, we have learned several points on the task of a faster satellite segmentation using a UNet. The first is the importance of having the correct levels of context, if the UNet is too shallow then the larger picture would be missing and thus lead to a less accurate model. Additionally, memory usage is strongly correlated with the speed of training the model and it's less about the number of parameters in this UNet case as the majority of the memory is used by the number of features at each level and it would be detrimental if one were to have excessive width in the earlier levels because of the higher resolution and the per-pixel nature. And finally, the datasets that include satellite imagery would have to face visual challenges like identifying areas hidden within shadows and patterns that require even greater context than what a single image may provide. Having proper augmentation and importing weights from specific models to offset the imbalance classes could also help greatly when dealing with a smaller dataset.

Future Work

Given more time and resources, we would like to experiment with revising the skip connections between encoding and decoding. This is a popular field for optimizing a UNet as the connections can do much more than contribute to the corresponding level. In UNet++ we have seen improvements gained by having higher level connections be integrated by the lower ones. In a more recent paper, "UNetv2" is an implementation of a UNet that incorporates all levels of the skip connections in an efficient manner and allows for a greater level of context while being efficient in memory usage. We would also like to spend time to figure out optimal tuning for training UNet based models given without the hand-crafted approach mentioned in the papers for these datasets.

References

- [1] S. R. Hashemi, S. S. M. Salehi, D. Erdogmus, S. P. Prabhu, S. K. Warfield, and A. Gholipour, "Asymmetric Loss Functions and Deep Densely Connected Networks for Highly Imbalanced Medical Image Segmentation: Application to Multiple Sclerosis Lesion Detection," *IEEE Access*, vol. 7, pp. 1721–1735, 2019, doi: [10.1109/ACCESS.2018.2886371](https://doi.org/10.1109/ACCESS.2018.2886371).
- [2] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation," in *Computer Vision – ECCV 2018*, vol. 11211, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., in Lecture Notes in Computer Science, vol. 11211, Cham: Springer International Publishing, 2018, pp. 833–851. doi: [10.1007/978-3-030-01234-2_49](https://doi.org/10.1007/978-3-030-01234-2_49).
- [3] Z. Wang, X. Ning, and M. B. Blaschko, "Jaccard Metric Losses: Optimizing the Jaccard Index with Soft Labels." arXiv, Oct. 29, 2023. Accessed: Dec. 10, 2023. [Online]. Available: [http://arxiv.org/abs/2302.05666](https://arxiv.org/abs/2302.05666)
- [4] A. Boguszewski, D. Batorski, N. Ziemba-Jankowska, T. Dziedzic, and A. Zambrzycka, "LandCover.ai: Dataset for Automatic Mapping of Buildings, Woodlands, Water and Roads from Aerial Imagery." arXiv, Apr. 21, 2022. Accessed: Dec. 15, 2023. [Online]. Available: [http://arxiv.org/abs/2005.02264](https://arxiv.org/abs/2005.02264)
- [5] J. Wang, Z. Zheng, A. Ma, X. Lu, and Y. Zhong, "LoveDA: A Remote Sensing Land-Cover Dataset for Domain Adaptive Semantic Segmentation." arXiv, May 31, 2022. Accessed: Dec. 15, 2023. [Online]. Available: [http://arxiv.org/abs/2110.08733](https://arxiv.org/abs/2110.08733)
- [6] Y. Peng, M. Sonka, and D. Z. Chen, "U-Net v2: Rethinking the Skip Connections of U-Net for Medical Image Segmentation." arXiv, Nov. 29, 2023. Accessed: Dec. 06, 2023. [Online]. Available: [http://arxiv.org/abs/2311.17791](https://arxiv.org/abs/2311.17791)
- [7] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation." arXiv, May 18, 2015. Accessed: Dec. 15, 2023. [Online]. Available: [http://arxiv.org/abs/1505.04597](https://arxiv.org/abs/1505.04597)
- [8] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, "UNet++: A Nested U-Net Architecture for Medical Image Segmentation." arXiv, Jul. 18, 2018. Accessed: Dec. 14, 2023. [Online]. Available: [http://arxiv.org/abs/1807.10165](https://arxiv.org/abs/1807.10165)
- [9] "qubvel/segmentation_models.pytorch: Segmentation models with pretrained backbones. PyTorch." Accessed: Dec. 15, 2023. [Online]. Available: https://github.com/qubvel/segmentation_models.pytorch