Deep Learning Project Proposal

Written by
Sayam Dhingra(sd5292), Yami Naik(yn2224), Bhautik Sudani(bgs8255)[1]

[1]Tandon School of Engineering, New York University

Abstract

Satellite images are widely used for various applications such as urban planning, environmental monitoring, and disaster response. In this project, we aim to develop a deep learning model based on Faster R-CNN to accurately detect and classify objects in satellite images. We will train the model on the spacenet 7 dataset of satellite images and evaluate its performance. Additionally, we explored various techniques such as data augmentation, transfer learning, and hyperparameter tuning to improve the model's performance. Our code can be found at https://github.com/yaminaik/ObjectDetection

Introduction

Applications for human development and catastrophe response are numerous for satellite imagery analytics, especially when time series methods are used. For instance, 67 of the 232 Sustainable Development Goals depend on measuring population figures, yet the World Bank estimates that more than 100 nations currently do not have efficient civil registration systems. In recent years, the application of deep learning techniques has revolutionized the field of computer vision, enabling significant advancements in various domains, including satellite imagery analysis. With the availability of high-resolution satellite imagery, the extraction and detection of buildings from such datasets have become essential for urban planning, disaster response, and environmental monitoring. This technical introduction presents a report on the development and training of a mask-based neural network specifically designed to detect buildings in the SpaceNet 7 dataset.

The primary objective of this project is to develop an accurate and efficient solution for building detection by leveraging the power of deep learning and utilizing the SpaceNet 7 dataset. The utilization of a mask-based neural network architecture allows us to generate pixel-wise segmentation masks, providing fine-grained information about building boundaries and spatial extents.

The SpaceNet 7 dataset is a widely used benchmark dataset in the field of remote sensing and computer vision. It encompasses a large collection of high-resolution satellite imagery with corresponding pixel-level annotations for building footprints. The dataset presents significant challenges due to the diversity of building types, varying lighting conditions, occlusions, and complex urban environments. This project aims to leverage this rich dataset to train a neural network model capable of accurately detecting buildings.

To address the task of building detection, we employed a mask-based neural network architecture, specifically designed for pixel-wise semantic segmentation. This architecture enables us to generate binary masks where each pixel is classified as building or non-building. Due to the significant class imbalance, a standard accuracy metric alone is not informative. We need a metric that evaluates the model's ability to accurately segment buildings in an image. Therefore, we focus on the Dice metric, which is commonly used for segmentation tasks. It measures the similarity between the predicted and ground truth masks and provides an F1 score-like measurement, calculated as $\frac{2TP}{2TP+FP+FN}$, where TP represents true positives, FP represents false positives, and FN represents false negatives.

Building masks serve as valuable tools for visualization and training deep learning segmentation algorithms. However, the vector labels provided in the SpaceNet 7 dataset offer an additional advantage by allowing for the assignment of a unique identifier, such as an address, to each building. This unique identification enables the matching of building addresses across different time steps, which constitutes a fundamental aspect of the SpaceNet 7 challenge [**?**]. Because of this we changed to a mask based network instead of a bounding box based network.

Due to the large size of the images (1024x1024 pixels),

we divided them into 16 smaller tiles, each measuring 255x255 pixels. This approach allows us to accommodate larger batch sizes and/or deeper models within the GPU's memory. While this method discards some pixels in most images, it maintains uniform tile sizes across the dataset.

It's worth noting that certain images in the dataset have 1023 pixels in one dimension instead of 1024. As a result, we chose 255 as the tile size to ensure consistency across all images. An improvement to this approach would involve creating overlapping tiles to prevent buildings from being cut in half, thereby ensuring the model observes their complete shape.

Related Work

Several papers have contributed to the field of object detection in satellite imagery, addressing various challenges and proposing novel approaches. In this section, we discuss relevant papers that discuss different aspects of object detection in satellite imagery.

In a study by Cheng G. et al [**?**] they provide a comprehensive review of the recent progress in generic object detection in optical remote sensing images. Unlike previous surveys that focus on specific object classes, this review covers a wide range of object categories, including roads, buildings, trees, vehicles, ships, airports, and urban areas. The paper surveys different approaches such as template matching, knowledge-based methods, object-based image analysis, and machine learning-based methods. Additionally, publicly available datasets and standard evaluation metrics are discussed. The paper also identifies two promising research directions: deep learning-based feature representation and weakly supervised learning-based geospatial object detection.

Similarly, Etten [**?**] addresses the challenges of detecting small objects in large satellite imagery. The authors propose a pipeline called YOLT, which enables the evaluation of satellite images of arbitrary size at a high rate. The pipeline utilizes deep learning techniques and demonstrates the ability to detect objects of varying scales with relatively little training data across multiple sensors. The evaluation results show high localization scores for vehicles, even for objects as small as five pixels in size. The paper highlights the efficiency and effectiveness of the YOLT pipeline in object detection tasks.

FrRNet-ERoI is a deep learning model for object detection in satellite images. It is a combination of two models: FrRNet (Feature Pyramid Refined Residual Network) and EROI (Efficient Region Proposal Generation Network with Object Interaction). Therefore, Pazhani et al. [**?**] suggest these techniques to train these models to detect objects and classify them.

In other works, Tahir et al. [**?**] have done a comparative study between different CNN's like faster RCNN (faster region-based convolutional neural network), YOLO (you only look once), SSD (single-shot detector) and SIMRDWN (satellite imagery multiscale rapid detection with windowed networks). Their study concluded that SIMRDWN had the highest accuracy and YOLOv3 had the fastest speed and efficiency.

Methodology

We have developed and implemented a neural network-based solution to detect building footprints on the SpaceNet 7 dataset. In our approach, we primarily focus on performing segmentation to identify buildings within individual images, disregarding the temporal aspect of the original challenge. To facilitate the development process, we utilize fastai, a PyTorch-based deep learning library. This powerful framework offers advanced functionality for training neural networks, incorporating modern best practices, and minimizing the need for cumbersome boilerplate code.

Neural Networks

Neural networks, also known as artificial neural networks (ANNs), are computational models inspired by the structure and functioning of the human brain. They are a subset of machine learning algorithms designed to recognize patterns and make predictions based on input data.

At the core of a neural network are interconnected nodes, called artificial neurons or "units," organized into layers. The input layer receives the input data, which is then processed through one or more hidden layers, and finally produces an output layer with the desired predictions or classifications.

Each artificial neuron takes multiple inputs, applies a weighted sum of those inputs, adds a bias term, and passes the result through an activation function. The activation function introduces non-linearities and enables the neural network to learn complex patterns in the data. We can see a viusualization of these in Fig.**??**.

FastAI

fastai is a high-level deep learning library built on top of PyTorch, which is one of the most popular and powerful open-source frameworks for machine learning and deep learning. fastai aims to make deep learning more accessible by providing a simple and intuitive API while incorporating state-of-the-art techniques and best practices.

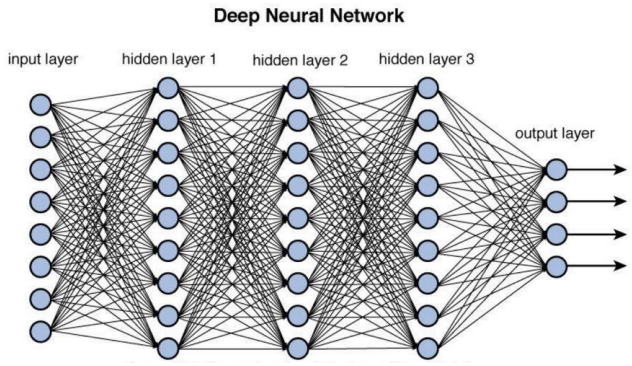

Figure 1: Basic idea of artifician neural networks

Fastai includes a comprehensive set of tools and utilities for various stages of the deep learning workflow, including data preprocessing, model creation, training, and evaluation. It offers a wide range of pre-implemented architectures, such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers, which can be easily customized and adapted to specific tasks.

Furthermore, fastai provides high-level abstractions for common deep learning tasks, such as image classification, object detection, text classification, and natural language processing (NLP), making it easier for users to tackle these tasks with minimal effort.

Google Colab

Google Colab (short for Google Colaboratory) is a cloud-based integrated development environment (IDE) that allows users to write, run, and share Python code collaboratively. It is a free service provided by Google and is widely used by researchers, students, and developers for various tasks, including data analysis, machine learning, and deep learning.

Colab provides a Jupyter Notebook interface, which allows users to create interactive notebooks that combine code, text, and visualizations. These notebooks are stored on Google Drive and can be easily shared with others, making it convenient for collaborative work and sharing code with colleagues or collaborators.

Overall, Google Colab provides a convenient and accessible platform for running Python code in the cloud, leveraging powerful computing resources, and facilitating collaboration and sharing of code and notebooks.

Pre-processing

We create binary masks using the dataset provided. Binary masks refer to a binary representation that indicates the presence or absence of a specific object or feature in an input image. We create this using the geojson files in the spacenet7 dataset.

The dataset exhibits a notable degree of similarity among the approximately 24 images per scene. Although there are seasonal variations in vegetation and sporadic instances of building activity, the overall similarities outweigh the differences. Consequently, a decision was made to disregard the majority of the images. Initially, the intention was to retain every fifth image from each scene, capturing the variability across seasons such as January, June, November, April, and September. However, it was discovered that achieving comparable results could be accomplished by selecting only one image per scene, substantially reducing the required training time.

To accommodate the large size of the images, a strategy was implemented to divide them into smaller tiles measuring 255x255 pixels, and subsequently store them on disk. This approach was chosen based on the observation that most structures within the scenes are relatively small compared to the overall image. Consequently, the division into smaller tiles is not expected to significantly hinder the training process. Additionally, the utilization of smaller tiles offers the advantage of enabling larger batch sizes and accommodating deeper models within the available GPU memory.

It is important to note that while the majority of the images have dimensions of 1024x1024 pixels, a subset of images possesses dimensions of 1023 pixels in one dimension. As a result, a tile size of 255 was specifically chosen instead of 256 to maintain uniformity across all images. This decision does entail the omission of a small number of pixels in most images, although it ensures consistency in the tile size throughout the dataset.

As an improvement for future work, it is recommended to explore the creation of overlapping tiles. By incorporating overlapping regions, the issue of buildings being severed in half and consequently not fully captured by the model can be mitigated. This would enhance the model's ability to comprehend the complete shape and structure of buildings during training.

Finally we can now see the tiles with their corresponding binary masks in the Fig.**??**

Dataset Challenges

The visualization of the dataset reveals several challenges that need to be addressed.

1. The dataset presents the difficulty of dealing with buildings that are often tiny, consisting of only a few pixels, and located in close proximity to one another.
2. Conversely, there are large structures within the

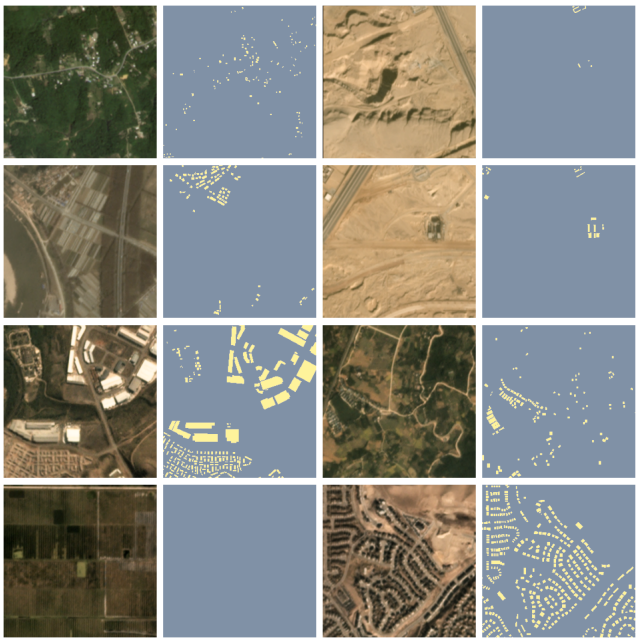

Figure 2: Tiles with their mask after pre-processing

dataset that occupy significantly greater areas compared to the small buildings.

3. Certain buildings pose challenges for recognition, even for the human eye, making it crucial to develop robust algorithms.
4. The dataset exhibits significant variations in building density. Some tiles contain no buildings at all, while others portray urban scenes with numerous buildings.
5. The images in the dataset exhibit remarkable diversity, encompassing variations in topography, vegetation, and urbanization levels.
6. A subset of tiles is partially or completely covered with UDM (User-Defined Mask) masks, necessitating strategies to handle and interpret this partial or missing information effectively.

To thoroughly examine the data's class imbalance, we conducted an analysis focusing on the percentages of building pixels within each tile. To facilitate this analysis, we developed a straightforward dataloader specifically designed for loading and evaluating the masks.

The results reveal a significant prevalence of tiles that either lack buildings entirely or contain an exceedingly small number of building pixels. Notably, 75 images, constituting nearly 10% of the dataset, do not possess a single pixel assigned to the building class. These instances can be attributed to areas characterized by vacant land, bodies of water, or tiles obscured by cloud cover.

On average, a mere 6.5% of a tile's pixels represent the building class. Moreover, the median value is merely 3.4%. Consequently, it is evident that this dataset exhibits a substantial class imbalance, further underscoring the need for careful consideration and effective handling of this inherent imbalance.

For examination we can see that the tile with the highest percentage of buildings is represented in Fig.**??**

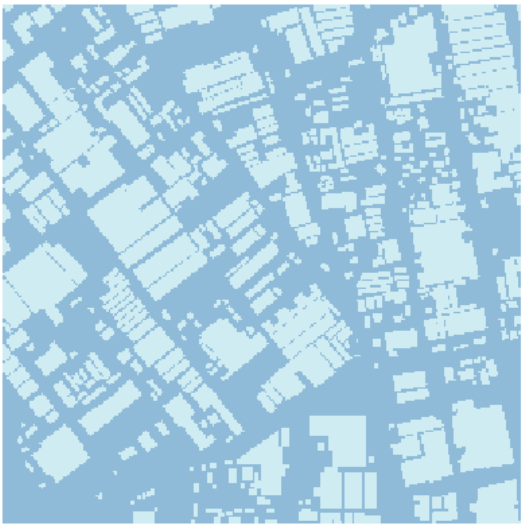

Figure 3: Tiles with the highest percentage of buildings

To help mitigating the imbalanced classes, we remove all tiles that contain no buildings at all from the training set. This reduces the amount of samples by 10%, thereby accelerating the training while helping the model perform better.

Data Loading

The chosen transformations for satellite images are deemed reasonable and appropriate. These transformations encompass vertical and horizontal flipping, rotation, slight adjustments to brightness, contrast, and saturation. Furthermore, normalization is applied based on ImageNet statistics, facilitating the utilization of pretrained models in subsequent stages.

To streamline the dataset creation process, we leverage the convenient DataBlock API provided by fastai. Notably, only 16 tiles per scene are loaded, resulting in the inclusion of just one image per region.

The training set consists of 741 tiles, while the validation set comprises 144 tiles.

As anticipated, the dimensions conform to the following specifications:

1. Each batch contains 12 images.
2. The input images consist of 3 channels.

3. The target masks lack color channels.
4. The image size is set to 255x255 pixels.

Model

The current task involves tackling an image segmentation problem. In the original competition, the objective entails assigning distinct labels to individual buildings to track their evolution over time (known as instance segmentation). However, in this particular approach, the focus is shifted towards semantic segmentation, which involves classifying each pixel as either belonging to a building or not.

The fastai library offers a remarkably straightforward solution through the utilization of a U-Net, a well-established architecture for image segmentation tasks. The DynamicUNet [**?**] module, coupled with an encoder architecture, automatically constructs a decoder and establishes cross connections. This streamlined approach enables the creation of a U-Net using various pretrained architectures, affording more time for experimentation as opposed to writing code from scratch. Several aspects were considered in this decision-making process:

1. Encoder: A xResNet34 model pretrained on ImageNet was chosen as the encoder. This 34-layer encoder strikes a favorable balance between accuracy and memory/compute requirements.
2. Loss function: The selection of an appropriate loss function is crucial for segmentation problems. In this case, a weighted pixel-wise cross-entropy loss will be employed. The incorporation of weights is of paramount importance due to the imbalanced nature of the dataset.
3. Optimizer: The default optimizer, Adam, will be utilized for training the model.
4. Metrics: Given the highly imbalanced classes, a simple accuracy metric would not be meaningful. For instance, a model could predict "no building" for every pixel in an image with only 3% buildings and still achieve 97% accuracy. As an alternative, the focus will be on the Dice metric, which is commonly used for segmentation tasks. This metric, equivalent to the F1 score, measures the ratio of $\frac{2TP}{2TP+FP+FN}$. Additionally, the foreground_acc metric provided by fastai will be included, measuring the percentage of correctly classified foreground pixels (in this case, the building class), akin to Recall.

By considering these factors and making informed decisions, we aim to address the challenges of the image segmentation problem effectively.

Training the model

First we need to decide the learning rate. To do this we can use fastai's learning rate finder to pick a reasonable learning rate. We can see the resulting graph in the Fig.**??**

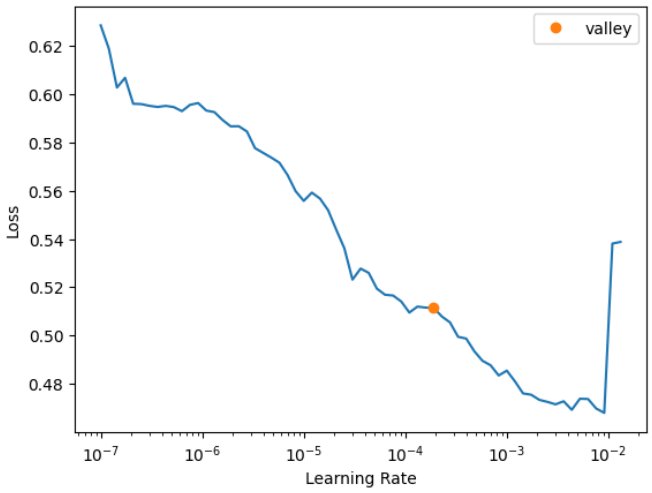

Figure 4: Learning rate discovery graph

After reviewing the graph we can see that the suggested learning rate is somewhere around 1e-4, where the loss decreases steadily.

To train both the encoder and decoder simultaneously, we unfreeze the model. It is important to note that the pretrained features within the encoder should be altered minimally. To achieve this, we set a lower learning rate specifically for the encoder, ensuring that the fine-tuning process does not excessively modify the pretrained features. This is accomplished by defining the learning rate as 'slice(lr_max/10, lr_max)'.

For the training process, we employ the 'fit_one_cycle' method, which encompasses a progressive learning rate schedule. Initially, the learning rate starts at a low value during a warm-up period, gradually increases to a maximum value of 'lr_max', and then anneals to 0 towards the end of the training process. This approach facilitates a balanced and effective optimization process for the model.

In the Fig.**??** we can see the momentum graph. In the momentum plot the Y-axis represents the momentum and the x-axis represents the steps/epochs.

The ADAM optimizer in neural networks incorporates a form of momentum through the use of exponential moving averages for estimating the first moment of the gradients. This allows ADAM to have momentum-like behavior without a separate momentum parameter. Additionally, ADAM combines adaptive learning rates based on the estimates of the first and second moments

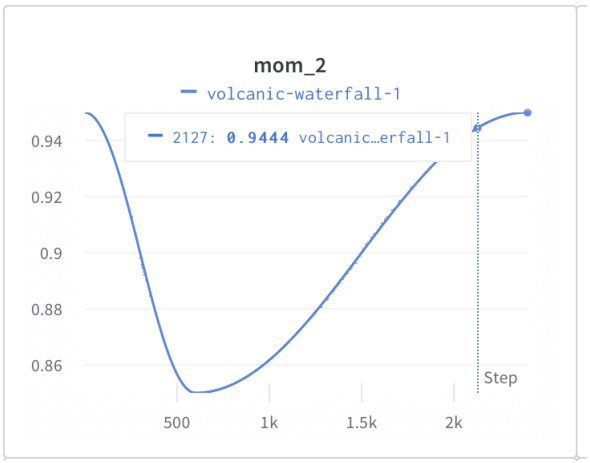

Figure 5: Momentum plot

of the gradients.

In the second graph in Fig.**??** we can see the loss value. Y-axis shows the loss value and the X-axis shows the steps/epochs. This represents the loss on the validation stage. The valid loss represents the discrepancy or error between the predicted output of the model and the true output on the validation set. It quantifies how well the model is performing on unseen data and provides insights into its generalization capability.

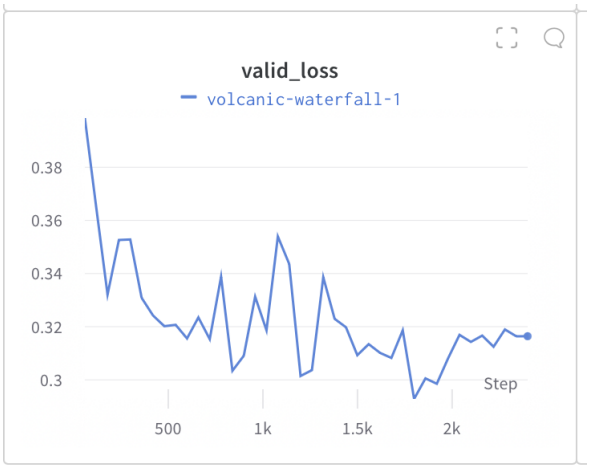

Figure 6: Valid Loss Plot

Next in the Fig. **??** we can see the leraning rate plot. On the Y-axis we have the learning rate vlaue and the X-axis has the steps/epochs. The learning rate controls the magnitude of these parameter updates. A higher learning rate results in larger updates, potentially leading to faster convergence but also risking overshooting the optimal solution. On the other hand, a lower learning rate results in smaller updates, potentially leading to slower convergence but increased precision.

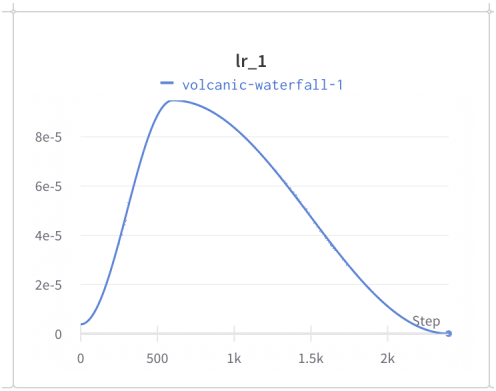

Figure 7: Learning Rate Plot

In the next plot in Fig.**??** we can see two graphs. The first one on top is the epsilon value. The epsilon is represented on the Y-axis while the steps/decay is on the X-axis. Similarly, the second graph on the second shows weight decay on the Y-axis and the steps/epochs on the X-axis.

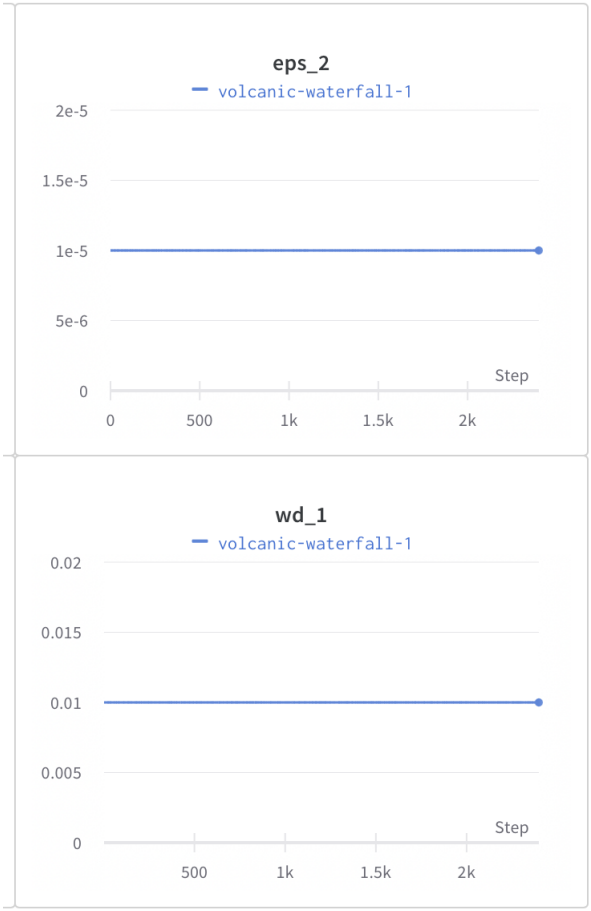

Figure 8: Epsilon and Weight Decay

In the following Fig.**??** here we can see the train_loss, valid_loss, dice score, foreground_acc and time for each epoch. The best model at epoch 29 has a Dice score of '0.539882', which is a good score for us to be able to successfully detect the buildings. We can also observe the valid_loss and train_loss decrease as the epochs increase. They start to even out at epoch 33. Therefore, it was safe to stop after epoch 39.

Now, since the training has finished we can view the loss plot graph. In Fig.**??**. On the X-axis we can see the steps/epochs and on the Y-axis we can see the loss values.

epoch	train_loss	valid_loss	dice	foreground_acc	time
0	0.404860	0.398518	0.272620	0.198581	00:43
1	0.370387	0.364988	0.418141	0.433954	00:45
2	0.350420	0.332094	0.466836	0.500462	00:45
3	0.333788	0.352622	0.420421	0.348120	00:44
4	0.321968	0.352808	0.468778	0.471579	00:44
5	0.319575	0.330865	0.482259	0.671936	00:44
6	0.321750	0.324249	0.496640	0.608572	00:44
7	0.314132	0.320183	0.485234	0.457416	00:44
8	0.306817	0.320697	0.478045	0.426950	00:45
9	0.297017	0.315569	0.508352	0.541224	00:44
10	0.291387	0.323476	0.499862	0.493073	00:44
11	0.290464	0.315321	0.508853	0.665376	00:44
12	0.290465	0.338803	0.487243	0.440306	00:44
13	0.285099	0.303411	0.522775	0.625072	00:44
14	0.296334	0.309071	0.513666	0.666180	00:44
15	0.284487	0.331292	0.494421	0.472607	00:45
16	0.274660	0.318634	0.512042	0.515265	00:44
17	0.271314	0.353927	0.449654	0.365648	00:44
18	0.266733	0.343598	0.522459	0.579844	00:44
19	0.265468	0.301454	0.523147	0.658687	00:44
20	0.263264	0.303652	0.522425	0.580128	00:44
21	0.258987	0.338603	0.517195	0.531966	00:44
22	0.254720	0.322951	0.500843	0.473963	00:44
23	0.252452	0.319787	0.525539	0.600118	00:44
24	0.255464	0.309261	0.529464	0.582459	00:44
25	0.250059	0.313433	0.525148	0.593785	00:44
26	0.245789	0.310150	0.530160	0.560972	00:44
27	0.244661	0.308239	0.531504	0.594034	00:44
28	0.243281	0.318637	0.531588	0.573959	00:45
29	0.244392	0.292818	0.539884	0.641453	00:44
30	0.243672	0.300553	0.532537	0.623098	00:44
31	0.242652	0.298536	0.539017	0.616894	00:44
32	0.236934	0.308161	0.537574	0.588438	00:44
33	0.235580	0.316925	0.536643	0.567474	00:44
34	0.231235	0.314272	0.535318	0.600534	00:44
35	0.233214	0.316655	0.532042	0.574054	00:45
36	0.230536	0.312454	0.536133	0.599066	00:44
37	0.232683	0.318958	0.533479	0.576591	00:44
38	0.231720	0.316382	0.534990	0.576035	00:44
39	0.230117	0.316413	0.534193	0.578171	00:44

```
Better model found at epoch 0 with dice value: 0.2726195451280211.
Better model found at epoch 1 with dice value: 0.418140866724756.
Better model found at epoch 2 with dice value: 0.46683647181516885.
Better model found at epoch 4 with dice value: 0.468778418688751.
Better model found at epoch 5 with dice value: 0.482258796888729.
Better model found at epoch 6 with dice value: 0.4966399055971167.
Better model found at epoch 9 with dice value: 0.5083522636581396.
Better model found at epoch 11 with dice value: 0.5088525203101505.
Better model found at epoch 13 with dice value: 0.5227746724999071.
Better model found at epoch 19 with dice value: 0.5231473880360039.
Better model found at epoch 23 with dice value: 0.5255393070338903.
Better model found at epoch 24 with dice value: 0.5294640277415638.
Better model found at epoch 26 with dice value: 0.5301597394194945.
Better model found at epoch 27 with dice value: 0.5315044863890428.
Better model found at epoch 28 with dice value: 0.5315881821573211.
Better model found at epoch 29 with dice value: 0.5398840157557844.
```

The best model at epoch 29 has a Dice score of `0.539882`.

Figure 9: Training Values

Results

In this section we will show the results of the training. We will also show the result images after testing our model on the test dataset. We store the values of the outputs in descending value of loss.

Samples with highest value of loss

The analysis of images exhibiting the highest losses reveals a consistent pattern of dense urban areas. It is evident that the model encounters challenges in accurately identifying and segmenting large buildings, as they are frequently overlooked or disregarded. Additionally, it is notable that the model tends to merge very small buildings, resulting in conglomerations or "blobs" of multiple buildings. Consequently, the task of tracking individual buildings within this context is expected to be inherently difficult. We can observe Fig.**??**

Samples with medium value of loss

The model exhibits a tendency to merge small buildings into larger conglomerations, leading to the formation of indistinct blobs. Additionally, there are instances where false positives occur, indicating the misclassification of non-building elements as buildings. However, it is worth noting that amidst these limitations, the model demonstrates noteworthy performance by accurately identifying certain buildings that pose challenges even for human observers. We can obser this in Fig.**??**

Samples with low values of loss

In images with a sparse distribution of buildings, the model exhibits varied performance. Generally, the accuracy in such cases appears to be comparatively better than in densely populated areas. However, it is important to note that the model still generates false positives, erroneously classifying non-building elements

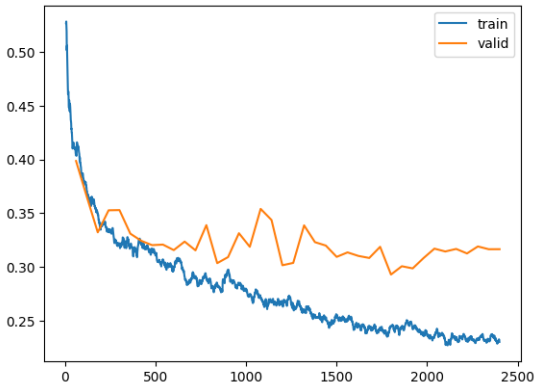

Figure 10: Loss value plot

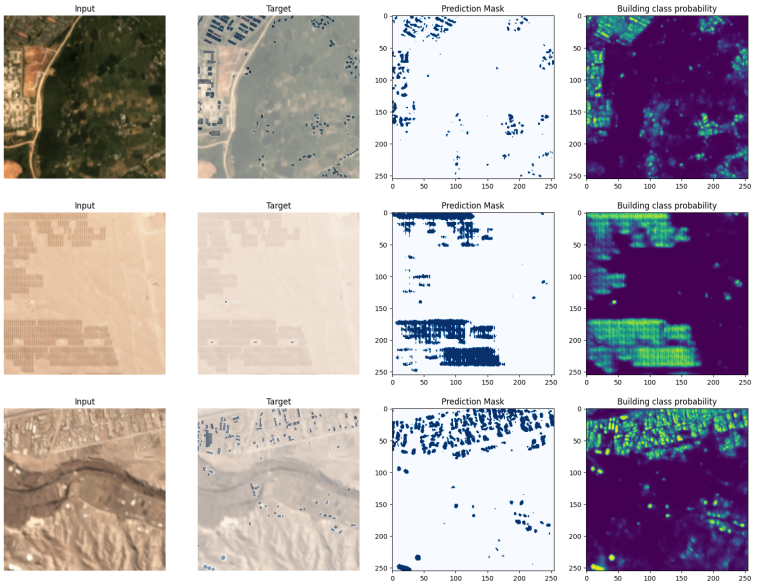

Figure 13: Medium Loss

as buildings. Furthermore, certain tiles exhibit peculiar artifacts, particularly in the corners. This suggests that the model may interpret the corners themselves as buildings, particularly on tiles encompassing water areas. We can observe this in the Fig.**??**

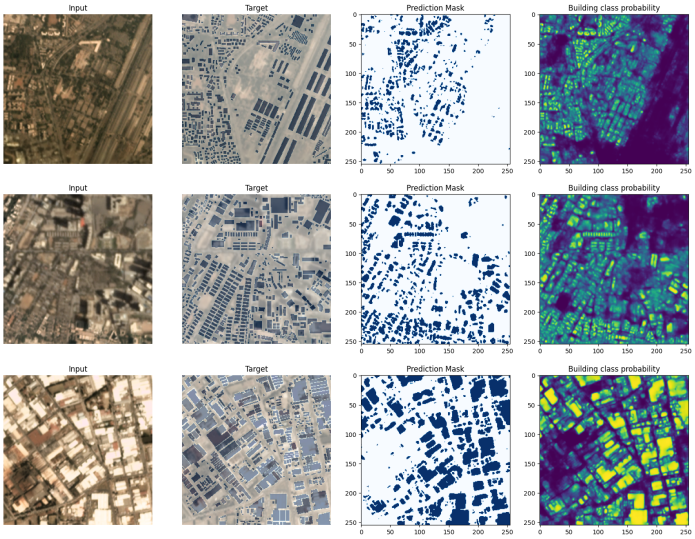

Figure 12: High Loss

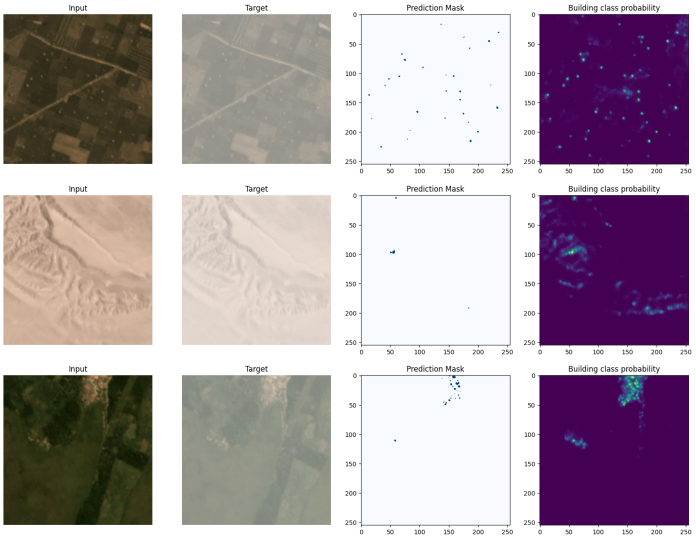

Figure 14: Low Loss

Full image results

Previously we showed the results for each tile image. In this section we will discuss the entire image and show the figures with their prediction masks. We can observe one entire image in the Fig.**??**.

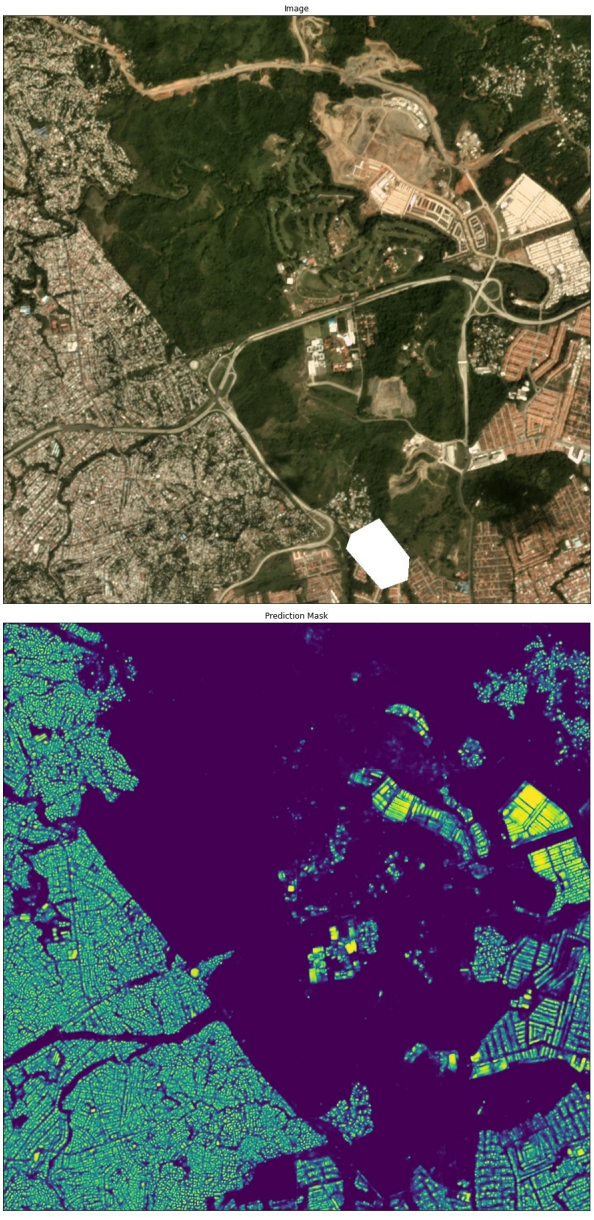

Figure 15: Full Scene

What worked?

1. Utilizing a pretrained encoder: The incorporation of a pretrained encoder, specifically a xResNet34 model pretrained on ImageNet, proved to be beneficial. This approach leveraged the learned features from a large-scale dataset, enhancing the model's ability to capture relevant information and improving overall performance.
2. Selective utilization of images: Through experimentation, it was observed that focusing on a subset of images per scene, specifically one image instead of five, yielded comparable results without significant improvement. This approach allowed for a reduction in training time without compromising the achieved outcomes.
3. Implementation of standard data augmentations: The application of common data augmentations played a crucial role in preventing overfitting. These augmentations introduced variations in the training data, promoting generalization and enhancing the model's ability to handle diverse and previously unseen images.
4. Employment of weighted cross-entropy loss: By incorporating weights into the cross-entropy loss function, the model's performance improved notably. The introduction of weights mitigated the strong bias towards the predominant background class, enabling the model to better recognize and classify buildings, thereby addressing the imbalanced nature of the dataset.

Additionally, although undersampling had a modest impact, it facilitated a slight improvement in training efficiency and contributed to enhanced accuracy.

Overall, these strategies and techniques demonstrated their effectiveness in improving the model's performance and addressing specific challenges encountered in the image segmentation task.

What did not work?

In this section we discuss possible cases that we tried but they did not yield good results or even made the model worse.

1. Implementation of the Mish activation function: Despite the expectation of faster training, the utilization of the Mish activation function did not yield the desired outcome. Instead, it resulted in unstable training dynamics, potentially indicating that this activation function was not well-suited for the specific image segmentation task at hand.

2. Adoption of Dice loss: The substitution of the cross-entropy loss with the Dice loss proved to be unstable as well. This alternative loss function, aimed at enhancing performance for imbalanced datasets, did not yield the expected improvements and may have introduced training instabilities.
3. Integration of self-attention in the U-Net: The incorporation of self-attention mechanisms within the U-Net architecture was anticipated to facilitate the accurate classification of larger structures. However, no significant improvement was observed, suggesting that the self-attention module did not offer substantial advantages in this particular context.
4. Utilization of a deeper xResNet50 encoder: Despite the increased depth and complexity of the xResNet50 encoder, this modification failed to produce improved results. Training time was substantially prolonged, with a six-fold increase, without corresponding enhancements in model performance. This suggests that the additional computational resources and complexity did not yield significant benefits for the specific image segmentation task.

Conclusion

Evaluating the results presents a challenge as there is no direct basis for comparison. Although a Dice score of 0.57 may not appear particularly impressive, it is important to consider the inherent difficulties posed by the dataset and the fact that the architecture was not customized extensively. Given these factors, we find the achieved result to be relatively satisfactory.

Regarding the aspects that yielded positive outcomes, the employed approach successfully tackled the semantic segmentation task by classifying pixels as building or non-building. The model demonstrated an ability to identify buildings that even posed challenges for human perception. This suggests that the initial segmentation framework was effective in capturing relevant features within the images.

Moving forward, we aspire to explore and enhance the capability of recognizing individual buildings and tracking them over time, as required in the original SpaceNet7 challenge. This represents an exciting avenue for future development and improvement in the field of building detection and tracking within satellite imagery analysis.

In the future, to further enhance the results, several strategies can be explored. Firstly, data processing techniques such as utilizing overlapping tiles and scaling up image tiles can be employed to capture finer details and context. Secondly, implementing dynamic thresholding methods can adaptively convert predicted probabilities into a binary mask, facilitating improved differentiation between building and non-building regions. Additionally, exploring recent advancements in segmentation models, such as UNet with ASPP or Eff-UNet, can leverage advanced architectural features to enhance feature representation. Allocating more computational resources to train deeper models can also allow for the capture of complex patterns and representations. Furthermore, employing cross-validation with multiple folds can effectively utilize the entire dataset and enhance generalization. Lastly, ensemble learning, which combines predictions from multiple models, can mitigate biases and boost overall segmentation accuracy.

These strategies offer promising avenues for future research and development, aimed at addressing limitations and achieving further improvements in the accuracy and robustness of the segmentation results.

References

00

[1] Pazhani, A.A.J., Vasanthanayaki, C. Object detection in satellite images by faster R-CNN incorporated with enhanced ROI pooling (FrRNet-ERoI) framework. Earth Sci Inform 15, 553–561 (2022). https://doi.org/10.1007/s12145-021-00746-8

[2] Tahir, A.; Munawar, H.S.; Akram, J.; Adil, M.; Ali, S.; Kouzani, A.Z.; Mahmud, M.A.P. Automatic Target Detection from Satellite Imagery Using Machine Learning. Sensors 2022, 22, 1147. https://doi.org/10.3390/s22031147

[3] Avanetten GitHub user https://github.com/avanetten/CosmiQ_SN7_Baseline

[4] Rim-Chan GitHub user https://github.com/Rim-chan/SpaceNet7-Buildings-Detection

[5] Gong Cheng, Junwei Han, A survey on object detection in optical remote sensing images, ISPRS Journal of Photogrammetry and Remote Sensing, Volume 117, 2016, Pages 11-28, ISSN 0924-2716, https://doi.org/10.1016/j.isprsjprs.2016.03.014

[6] https://doi.org/10.48550/arXiv.1805.09512

[7] Blog: Adam Etten The SpaceNet 7 Multi-Temporal Urban Development Challenge: Dataset Release https://shorturl.at/rSUV5

[8] https://docs.fast.ai/vision.models.unet#DynamicUnet