Using Deep Learning and Synthetic Aperture Radar for UXO Detection

Michael Prihoda

Center for Geospatial Information Science, University of Maryland

GEOG-797 Capstone

Jonathan Hathaway

November 13, 2023

Abstract 3

1 Introduction

1.1 Background 4

1.2 Literature Review 5

1.3 Objectives 6

2 Material and Methods

2.1 Study Area 7

2.2 SAR Imagery 8

2.3 Training Data 9

2.4 Semantic Segmentation

2.4.1 Pre-trained Models 10

2.4.2 Custom U-Net Model 11

2.5 Evaluation Criteria 12

2.6 Risk Classification 13

3 Results

3.1 Model Metrics 14

3.2 Mapped Segmentation Outputs 15

4 Discussion 16

References 18

Appendix A (Code Samples) 24

Abstract

During the war in Ukraine, millions of artillery shells have been fired by Russia and Ukraine combined, as shells sometimes fail to detonate, this means a significant amount of unexploded ordnance falls among the shelled-out towns and fields. The munitions that fail to detonate are likely to land near those that do explode. As a result, locations with a potential for explosive hazards can be identified by detecting craters. X-band synthetic aperture radar provides any time/any weather opportunities for Earth observation at sub-meter resolutions and the number of X-band platforms in orbit has increased significantly in recent years. Craters can be resolved in imagery from these platforms, however, manually locating them is unfeasible at larger scales. The suitability of deep learning models for binary segmentation of synthetic aperture radar imagery is tested and shown to work even on this atypical type of source imagery. For greater control over model configuration and parameters, a new model is built and evaluated against more generalized models. Ultimately, segmentation outputs are collected from the models and post-processed into visualizations showing areas at higher risk of unexploded ordnance relative to others.

Using Deep Learning and Synthetic Aperture Radar for UXO Detection

1 Introduction

1.1 Background

Modern warfare often results in widespread destruction of infrastructure, facilities, and the landscape in locations where active combat occurs. Significant displacement of population occurs in these areas as many seek safety elsewhere, in hopes of returning once the acute threat of war is no longer present. Many hazards persist in the environment following the cessation of active hostilities and continue to pose a threat to returning populations for generations, particularly unexploded ordnance (UXO). Significant amounts of UXO are the result of artillery shells and similar projectiles failing to detonate, becoming embedded in the soil, and being unearthed later, sometimes with deadly consequences. Explosive munitions from World War I continue to appear each tilling season in parts of Europe, this annual unearthing of war remnants is known as the “Iron Harvest” (Note et al., 2018).

One hundred years after the outbreak of World War I, in 2014 a new conflict began in Europe between Ukraine and the Russian Federation which escalated significantly following Russia’s full-scale invasion in February 2022. This war between Ukraine and Russia, much like the First World War a century prior, is characterized by the heavy use of artillery: Joseph Stalin’s “god of war” (Hunder, 2023). Between both Ukraine and Russia, 30,000 to 60,000 artillery shells are fired per day (millions per year), in addition to rockets and other explosive projectiles (Levy, 2023). Given the sheer number of shells fired and with each shell facing some probability of failure, a significant number of munitions are emplaced as UXO. Stockpiles of North Korean artillery shells have demonstrated particularly low reliability and high rates of failure with up to one in five shells failing to detonate; Russia hopes to sustain their millions-per-year rate of artillery fire, at least in part, by tapping into these stockpiles (Smith, 2023). As a result of artillery-centric warfare, explosive remnants of war are nearly certain to persist in Ukraine’s environment long-term until it is removed or destroyed.

Removal or destruction of UXO is a time and resource-intensive task, as well as dangerous to those tasked with the work. Fortunately, modern technologies continue to offer ways to locate/detect as well as dispose of UXO with greater efficiency and less risk to explosive ordnance disposal teams and bystanders. Magnetometers, ground penetrating radar, and electromagnetic induction technologies are among those used to precisely identify and locate potentially hazardous UXO. Implementing these technologies effectively first requires knowing where to look. In recent years the deployment of a variety of remote sensing platforms with ever-improving capabilities presents ways to contemporaneously identify areas impacted by artillery and other munitions through the observation of craters which, in turn, reveal areas where UXO is more likely to be present. While current sensors and platforms allow the identification of individual craters by a user, the very high spatial and temporal resolution of remote sensing data makes it unfeasible to manually identify all crater locations along the 1,000km+ line of contact. Prior research demonstrated the applicability of binary segmentation deep learning models on very high-resolution multispectral imagery to automate the identification of munition craters (Duncan et al., 2023). This project and the following pages present a process based on the work of Duncan et al. also for the automated detection of craters but using synthetic aperture radar (SAR) data as the source imagery rather than multispectral/optical data.

1.2 Literature Review

Existing literature regarding craters and SAR imagery largely focused on landform-scale impact and volcanic craters on Earth as well as on other celestial bodies. Angarita et al. (2022) demonstrated the use of differential interferometric SAR (DInSAR) whereby multiple SAR images acquired at different times and containing phase information of the returned radar signal are stacked to reveal displacement over time. The displacement investigated by Angarita et al. (2022) is related to an eruption event; measuring volcanic and seismic events such as these is a typical use case of SAR in Earth observation. Outside of Earth's atmosphere, SAR has seen widespread use investigating other planets in the solar system as well.

Herrick et al. (2023) demonstrated the ability of SAR to make observations and collect data even through the opaque atmosphere of Venus to reconstruct the geochronology of the Venusian surface by identification of craters. In terrestrial environments, this ability to see through otherwise opaque atmosphere translates to an all-weather, day or night sensing platform able to observe change by reimaging a location. These capabilities are often relevant when deep learning is applied to SAR imagery for a variety of interpretation, classification and segmentation tasks.

Hafner et al. (2022) SAR imagery in conjunction with multispectral for urban change detection. To accomplish that, a dual-stream U-Net was created for binary segmentation of multitemporal products from Sentinel 1 and Sentinel 2. The GitHub repository for the model created by Hafner et al. provided an early template for building a custom U-Net. Yue et al. (2020) explore the creation of custom, purpose-built neural networks in greater detail. The authors not only outline specific design choices made for their SAR image classifier but also provide information explaining their rationale for those choices. The discussion related to loss functions provides a good starting point to consider when exploring potential loss functions for this project. Additionally, the rationale for substituting an average pooling function in place of the more commonly used max pooling function in the downsampling layer provided some concrete ways generic computer vision architectures could be modified to better accommodate SAR data as an input. Zhang et al. (2023) present additional considerations relevant to building deep learning networks for SAR-based computer vision tasks. To improve the ability of computer vision models to extract non-linear features present in SAR imagery, they developed components which can be included in existing models that utilize texture as a source of information which models can be trained on.

Prior works seen in the literature related to SAR and deep learning often concern methods to improve or specialize a model's performance at a specific task rather than general usage. Building a U-Net specialized at detecting craters in SAR imagery is no different and is dependent on the data available and how the targets (craters) appear in it. Rather than using change over time or textural information to train a neural network on a task like past research, decisions for a custom U-Net focused on using information in SAR imagery which captures the geometric characteristics of craters.

1.3 Objectives

The ability of SAR remote sensing platforms to operate independently of weather conditions and time of day allows them to fill gaps in optical/multispectral coverage as well as augment data collected by these sensors. As additional SAR platforms are launched, more opportunities arise to extract value from the data they produce. Demonstrating compatibility between SAR imagery and pre-trained deep learning models shows not only is the nominal task of crater detection feasible, but so are countless additional, analogous tasks. These models are pre-trained on multi-band optical images and offer few configuration options to improve performance with single-band SAR images; building a custom segmentation model allows far more flexibility and extensibility to optimize for this data type. Post-processing models' segmentation outputs ultimately produces visualizations of UXO risk. In summary, this project's objectives are as follows:

- Use pre-trained binary segmentation models to establish their applicability to the crater detection task in SAR imagery.
- Create a custom binary segmentation model purpose-built for the single-band SAR imagery and evaluate its performance relative to the previously run pre-trained models.
- Post-process and visualize the outputs of binary segmentation as maps of potential UXO risk.

2 Material and Methods

2.1 Study Area

In early April of 2022, the Sloviansk Offensive by the Armed Forces of the Russian Federation reached the settlement of Dovhenke on the border of Kharkiv and Donetsk Oblasts (Clark et al., 2022). The front remained largely unchanged in the following months leaving Dovhenke and the nearby villages of Brazhkivka and Sulyhivka directly in the line of contact. On June 12 the settlement fell to Russian forces before being recaptured around August 10 according to the General Staff of Ukraine. Both multispectral and SAR imagery from the early summer of 2022 clearly show the impact of war on Dovhenke, Sulyhivka, and Brazhkivka, as well as their rural surroundings. After confirming the presence of craters and the availability of data, this area and period were targeted as the project's key region of interest. Figure 1 outlines the area of interest for this project.

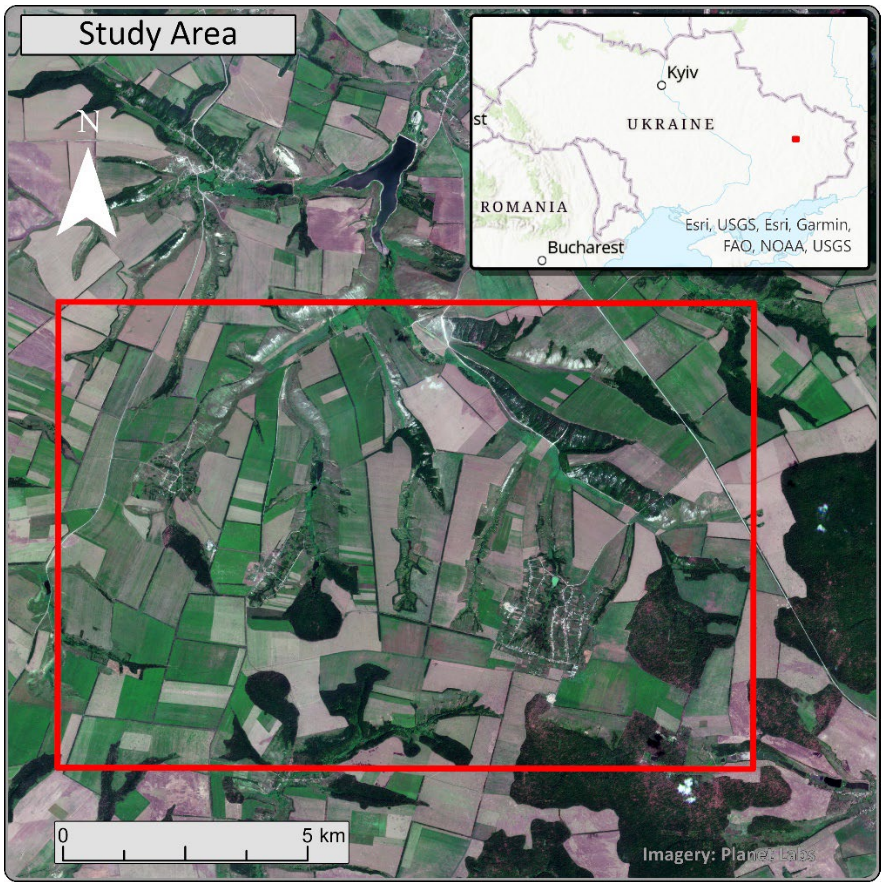

Figure 1: Study Area

2.2 SAR Imagery

Through its Data Cooperative program, Capella Space provided three scenes of very high-resolution SAR imagery from the summer of 2022 collected by one of their second-generation, Whitney class satellites. Scenes were identified using the Capella Explorer application, requested for download, and ultimately delivered digitally in a .zip file. Each scene had the target imagery as well as associated files for image metadata and statistics. Collections occurred on June 27, July 2, and July 18, 2022. The sensing mode for all three collections was “spotlight,” which produces SAR images with a 0.5m ground sample distance. The product type was specified as GEO, Capella’s terrain-corrected, pre-processed product with values of backscatter intensity. In addition, the GEO product contains only backscatter intensities for the HH (horizontal-horizontal) polarization with a cell size of 0.35m. Additional characteristics such as orbital node and look/incidence angle are in Table 1.

The differences in sensor/transmitter angle, look direction and orbital node have an obvious effect on how craters are captured in SAR; this is most clearly seen by high backscatter intensity values corresponding to the inner wall of the crater furthest from the satellite. A diagram illustrating this effect as well as an example from each scene are seen in Figure 2 and Figure 3. As rotation was applied to source image chips to augment training data sets, rotation angles were chosen to align the synthetic features of one set with the source features of another set.

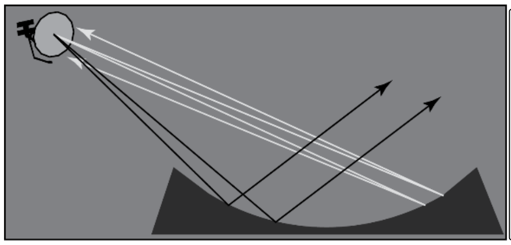
Figure 2: Backscatter Diagram

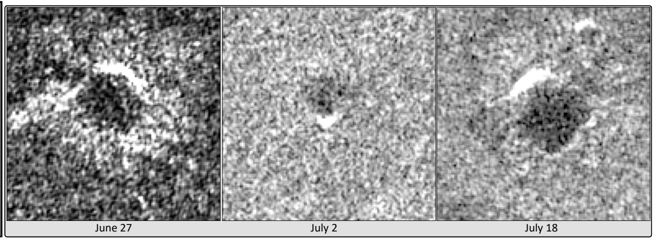

Figure 3: Per Scene Crater Example

Scene	Date	Look Angle	Incidence Angle	Pointing	Orbt	Polarization
CAPELLA_C06_SP_GEO_HH_20220627185019_20220627185043	2022-06-27	28.55	31.36	Left	Descending	HH
CAPELLA_C06_SP_GEO_HH_20220702135523_20220702135550	2022-07-02	36.38	40.24	Right	Ascending	HH
CAPELLA_C06_SP_GEO_HH_20220718055934_20220718060001	2022-07-18	33.86	37.39	Left	Ascending	HH

Table 1: SAR Scene Additional Data

2.3 Training Data

Training data collection and preprocessing occurred in the training data manager tool packaged as part of Esri’s Deep Learning Libraries for ArcGIS Pro. In total, 3,390 craters were manually labeled in the imagery. To facilitate and organize training data collection, SAR images were overlaid by a 5 by 5 grid with each grid cell having a side length of 1km. These cells were used to segregate areas used for training data from areas used for test data. Data from July 2 provided disproportionately fewer craters for sampling than the other two scenes. It is possible craters were truly absent in the imagery, alternatively, effects from the orbital node and steeper look/incidence angles may have made crater identification more difficult compared to the other two locations. Following sample collection, within the Esri tool the features created as class labels along with the imagery they were associated with were exported to a training data directory. Images were chipped to a size of 256px by 256px; label polygons overlaying these images were converted to raster chips of the same size with a value of 1 for the crater class and 0 elsewhere. Training data was split 80:20 following a general rule of thumb in deep learning (Gholamy et al., 2018). Image and label pairs served as training data both for pre-trained models instantiated with Esri's Deep Learning Libraries as well as runs of UNet models with custom configurations running in terminal or Jupyter windows.

SAR Training Data	Initial Craters	Additional Craters	Expanded Crater Count	Augmented Data
June 27, 2022 SAR	611	854	1465	…
July 2, 2022 SAR	49	98	147	…
July 18, 2022 SAR	744	1034	1778	…
Total	1404	1986	3390	3390
Initial Image+Label Pairs:	2,567	**Expanded Image+Label Pairs:**	3,894	5,938

Table 2: Training Data Collection Stats

2.4 Semantic Segmentation

2.4.1 Pre-trained Models

All models used in this project were variations of the same underlying U-Net architecture with different PyTorch Image Model (TIMM) encoders that performed binary segmentation of SAR imagery. Binary segmentation to automate crater detection produced simple, raster outputs where each cell is classified by the model as either belonging to the "crater" class or the "background" class. A panel of five pre-trained models were fine-tuned on the training data sets and subsequently tested on unlabeled imagery. The five models are as follows:

Inception v4: computer vision model created by Google engineers (v1 was known as GoogleNet). The Inception model is a very deep convolutional neural network (CNN) characterized by gridded layers called Inception blocks which perform several convolution operations in parallel with pooling operations in the network (Szegedy, 2016). This model is pre-trained on the ImageNet dataset.

MixNet-S*:* MixNet is characterized by the mixed convolutional layers from which it's built. Mixed convolutional layers use multiple different-sized kernels, or windows, within the same layer in contrast to typical convolution operations which use a single fixed-sized kernel applied across the whole layer (Tan and Le, 2019). MixNet has also been trained on ImageNet, and in benchmarking was less resource-intensive than other models with comparable classification performance.

MobileNet*:* MobileNet is an architecture built with consumer electronics, particularly smartphones, in mind as the computing platform. As a result, MobileNet is intended to be less resource-intensive than other convolutional neural networks, like MixNet (Howard et al., 2019).

ResNet-34*:* ResNet is a widely used model backbone in computer vision tasks, and many deep learning interfaces, such as Esri's, default to a ResNet implementation if no other backbone or encoder is specified. ResNet is characterized by the use of recurrent connections between non-adjacent layers, analogous to the skip connections seen in the generic U-Net architecture (He et al., 2015).

VGG11*:* The VGG model is another very deep CNN and is trained on ImageNet like many other computer vision models. VGG reaches deep network layers through consecutive convolutional layers all with a kernel size of 3. The VGG model generalizes well in several computer vision tasks and was a top achiever when it was first introduced in the 2014 ImageNet challenge (Simonyan and Zisserman, 2015).

2.4.2 Custom U-Net Model

After establishing segmentation of the Capella SAR images is possible, a custom implementation of the U-Net architecture was built in the PyTorch deep learning framework (Paszke et al., 2019). In addition, the PyTorch Lightning libraries were used to modularize model components allowing for easier experimentation and simplifying code (PyTorch Lightning, 2019). By building the model from scratch and incorporating plug-and-play components, the model allowed far greater freedom for customization and parameterization for the single-band SAR images than the models pre-trained on RGB optical images permitted. The model included two PyTorch dataset classes, one for training and one separate for testing. These two datasets were unified in a PyTorch Lightning datamodule class in which they were passed to the appropriate PyTorch dataloader classes for the training, validation, and test steps. The model consisted of four sub-components: a double convolution class, a downsampling class, an upsampling class, and a nearest-neighbor interpolation class. These four components were then brought together in the UNet class to construct the model.

The double convolution class consisted of the sequence: 2D convolution followed by 2D batch normalization followed by ReLU activation function, repeated twice as seen here:

$$2DConv \gg 2DBatchNorm \gg ReLU \gg 2DConv \gg 2DBatchNorm \gg ReLU$$

The UNet model and datamodule were then used to construct a LightningModule which managed the training, validation, and test steps. The criterion for optimization was binary cross entropy between the model’s predicted values and the labeled tiles. Additionally, recall, precision, and F1 were logged for each training epoch’s validation step as well as for the test set following model training. Code samples of the model and datamodule are in *Appendix A.*

2.5 Evaluation Criteria

Suitable criteria to evaluate a model's performance in the segmentation task are recall, precision, and the harmonic mean of recall and precision, F1 (Erickson and Kitamura, 2021). The formulas to calculate these values are in formulas ***a, b,*** and ***c.*** Recall can be summarized as the rate or percentage of known targets that the model identified; in the crater detection task, a recall of .75 would indicate a model identified 75% of the craters labeled. For the crater detection task, precision represents the rate a model is correct when it classifies something as crater. A precision of .75 would indicate that 75% of what the model classified as a crater truly is a crater while 25% is background falsely classified as a crater. The F1 score is the harmonic mean of the two prior values and allows a single metric to capture model performance. Comparisons of models based on the F1 score allow a one-to-one comparison while considering the two separate values.

$$a)\quad Recall = \frac{True\ Positive\ Prediction}{True\ Positive\ Prediction + False\ Negative\ Predictions}$$

$$b)\quad Precision = \frac{True\ Positive\ Predictions}{Ground\ Truth\ Positivs + False\ Positives}$$

$$c)\ F1 = 2 \times \frac{Precision\ \times Recall}{Precision + Recall}$$

2.6 Risk Classification

The initial outputs of the segmentation models are raster tiles of the same shape and cell size as the target labels in the training data set. As the only values for the binary segmentation are 0, background or 1, crater, only cells of value 1 were retained and subsequently merged into a single layer. The craters in the raster layer were then converted into polygons, providing a single shape for each detected crater.

These polygons were then simplified to single points placed at the geometric center of each polygon. To correct some large-scale errors by models, clusters of more than 10 points within 10m of each other were identified. 8 clusters fitting these criteria were present and highlighted areas where the models tended to erroneously identify craters. The eight clusters falsely identifying craters were either manmade structures or streambanks in reality; no other type of feature was associated with similar errors. These false craters were removed manually. A grid with cells measuring 0.25km by 0.25km was constructed for each of the imagery inputs and the number of craters (as points) within each was appended on a cell-by-cell basis. Cells were then symbolized by crater count, with binning steps of 20. The resulting visualization provides a basic idea of where UXO risks may be high relative to other areas. The lack of more specific information regarding the types of munitions fielded and their characteristics limited opportunities to generate more statistically rigorous assessments of UXO risk, though meaningful inferences and conclusions can still be drawn from the available data.

3 Results

3.1 Model Metrics

The calculated metrics per model reveal two tendencies among the pre-trained models and are seen in Table 3. The MixNet and MobileNet models demonstrate very low recall scores, correctly classifying only around 11-12% of craters. At the same time, these two models have the highest scores for precision. This indicates a tendency of these two models to hold back in making positive classifications. The three deep CNNs score significantly higher in recall and at minimal expense to precision score indicating these models not only make a higher number of positive classifications, but they are also often correct when they do. The custom U-Net implementation ultimately achieved the highest recall among the models, at the same time demonstrating the lowest precision. These results suggest the custom UNet has more freedom to make positive classifications and correctly classifies a greater number of craters yet introduces significant numbers of false positives while doing so.

Input: SAR Imagery	Inception v4	MixNet	MobileNet	ResNet-34	VGG11	Custom
Precision	0.792387	0.91363	0.916207	0.853645	0.832741	**0.551117**
Recall	0.395123	0.1183	0.109165	0.382873	0.402163	**0.471522**
F1	0.527305	0.20948	0.195086	0.528642	0.542387	**0.508222**

Table 3: Precision, Recall, and F1 Score By Model

3.2 Mapped Segmentation Outputs

The distribution of bins seen in Figure 4 is largely consistent with what was observed in SAR imagery. Subjectively, the area south of Dovhenke in red appeared in SAR imagery to have the highest concentration of craters within the three scenes. At a high level the grid counts effectively show where craters are relatively more abundant or scarce, however, it is clear when model outputs are overlaid on the source imagery that significant numbers of craters remain uncounted as seen in Figure 5; likewise, this is reflected in the recall scores of the models which indicates all missed more than half of known craters in testing. Improvements remain a future possibility with the use of additional training data and further optimization of models, however, the current results satisfy the objectives of this project.

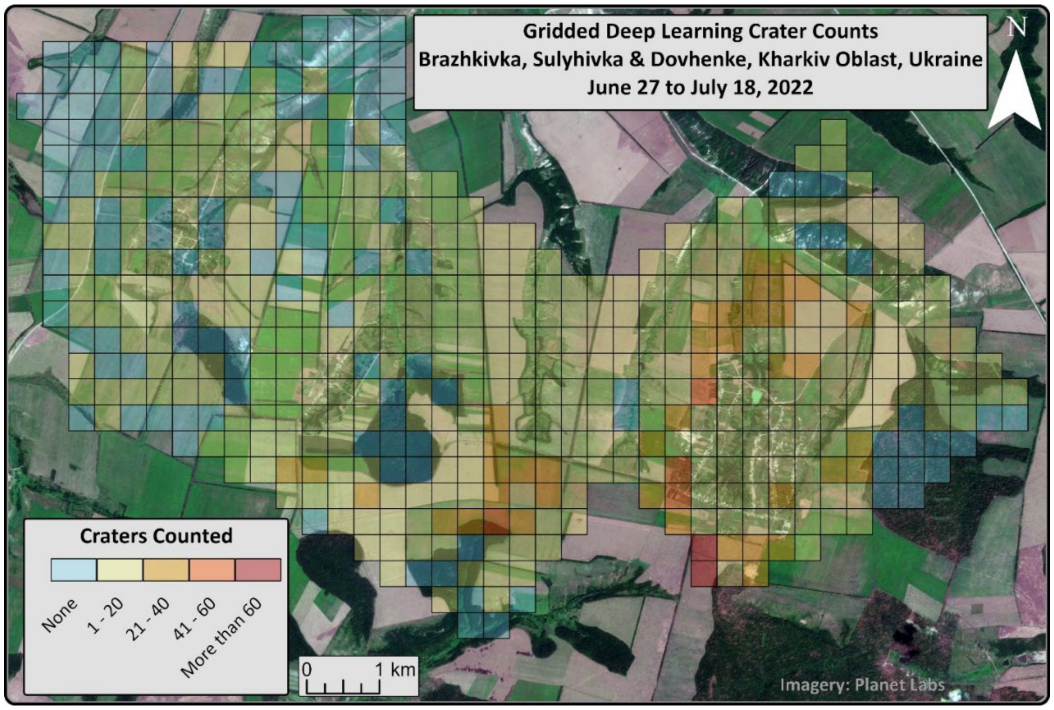

Figure 4: Gridded Crater Counts In Area of Interest

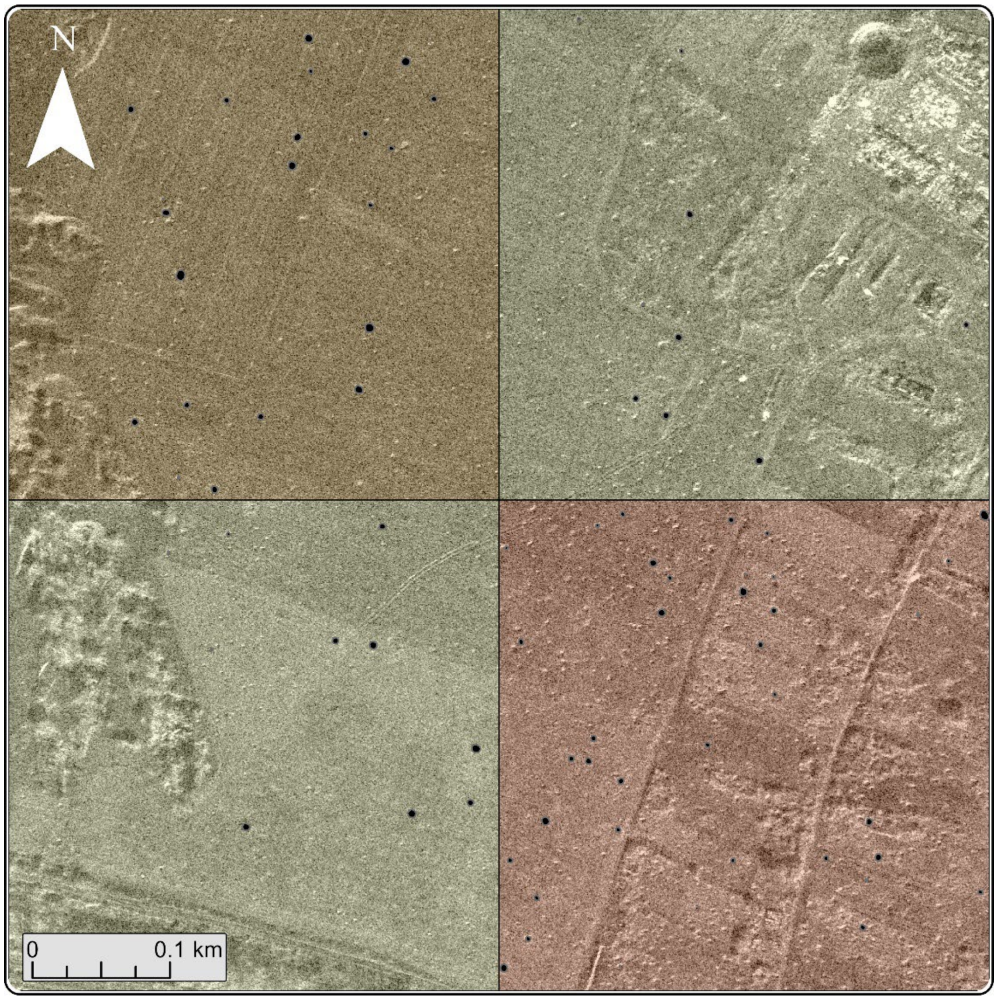

Figure 5: Example Grid Cells With Undercounted Craters, July 18 Image

4 Discussion

This project demonstrated, possibly for the first time, the application of binary segmentation to very high-resolution SAR imagery to automate crater detection. In addition to pre-trained models showing the ability to perform the crater detection task, a custom UNet implementation for binary segmentation created in PyTorch/PyTorch Lightning also showed positive results in a short period. Additionally, the outputs of segmentation made it possible to visually depict areas where UXO concentration may be higher. An area for future investigation is extracting and synthesizing more information regarding target geometry; the size of craters may permit identification of the type of munition it was created by, in turn allowing for more specific assessments of risks and other related analyses. In addition to the crater detection task, the outcome of this project more broadly demonstrates the feasibility of a generic application involving binary segmentation of very high-resolution SAR imagery. As more providers enter the SAR market, and existing providers expand their constellations further, the opportunities to apply deep learning methods to these datasets expand as well. Further areas of inquiry are improvements to recall score, observing whether seasonal or any other differences in collection affect the prominence of craters, and investigating the single-look complex SAR products to see if they can be used more effectively than multi-looked, single-band, single-polarity products.

References

Barata, T., Alves, E. I., Saraiva, J., & Pina, P. (2004). Automatic Recognition of Impact Craters on the Surface of Mars. In A. Campilho & M. Kamel (Eds.), *Image Analysis and Recognition* (pp. 489–496). Springer. https://doi.org/10.1007/978-3-540-30126-4_60

Brimelow, B. (n.d.). *Russia is using its newest and oldest missiles indiscriminately against Ukraine*. Business Insider. Retrieved October 25, 2023, from https://www.businessinsider.com/russia-using-newest-and-oldest-missiles-in-strikes-on-ukraine-2022-7

Byholm, B. (n.d.). *Remote Sensing of World War II Era Unexploded Bombs Using Object-Based Image Analysis and Multi-Temporal Datasets: A Case Study of the Fort Myers Bombing and Gunnery Range* [M.S., Minnesota State University, Mankato]. Retrieved October 22, 2023, from https://www.proquest.com/docview/1916044230/abstract/57802EA8A9684946PQ/1

Chen, S., & Zhang, B. (2022). *RSUnet: A New Full-scale Unet for Semantic Segmentation of Remote Sensing Images* [Preprint]. In Review. https://doi.org/10.21203/rs.3.rs-1211375/v1

Chen, Z., & Jiang, J. (2021). Crater Detection and Recognition Method for Pose Estimation. *Remote Sensing*, *13*(17), Article 17. https://doi.org/10.3390/rs13173467

Clark, M., Barros, G., & Stepanenko, K. (2022, April 6). Russian Offensive Campaign Assessment, April 6. *Institute of the Study of War*. https://www.understandingwar.org/backgrounder/russian-offensive-campaign-assessment-april-6

Diakogiannis, F. I., Waldner, F., Caccetta, P., & Wu, C. (2020). ResUNet-a: A deep learning framework for semantic segmentation of remotely sensed data. *ISPRS Journal of Photogrammetry and Remote Sensing*, *162*, 94–114. https://doi.org/10.1016/j.isprsjprs.2020.01.013

Dozat, T. (2016). *Incorporating Nesterov Momentum into Adam*. https://openreview.net/forum?id=OM0jvwB8jIp57ZJjtNEZ

Duncan, E. C., Skakun, S., Kariryaa, A., & Prishchepov, A. V. (2023). Detection and mapping of artillery craters with very high spatial resolution satellite imagery and deep learning. *Science of Remote Sensing*, *7*, 100092. https://doi.org/10.1016/j.srs.2023.100092

Erickson, B. J., & Kitamura, F. (2021). Magician's Corner: 9. Performance Metrics for Machine Learning Models. *Radiology: Artificial Intelligence*, *3*(3), e200126. https://doi.org/10.1148/ryai.2021200126

Falcon, W. (2019). *PyTorch Lightning* [Computer software]. https://www.pytorchlightning.ai

Fan, X., Yan, C., Fan, J., & Wang, N. (2022). Improved U-Net Remote Sensing Classification Algorithm Fusing Attention and Multiscale Features. *Remote Sensing*, *14*(15), Article 15. https://doi.org/10.3390/rs14153591

Ferrer, L. (2023). *Analysis and Comparison of Classification Metrics* (arXiv:2209.05355). arXiv. http://arxiv.org/abs/2209.05355

Gholamy, A., Kreinovich, V., & Kosheleva, O. (2018). Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation. *Departmental Technical Reports (CS)*. https://scholarworks.utep.edu/cs_techrep/1209

Hafner, S., Nascetti, A., Azizpour, H., & Ban, Y. (2022). Sentinel-1 and Sentinel-2 Data Fusion for Urban Change Detection Using a Dual Stream U-Net. *IEEE Geoscience and Remote Sensing Letters*, *19*, 1–5. https://doi.org/10.1109/LGRS.2021.3119856

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition* (arXiv:1512.03385). arXiv. https://doi.org/10.48550/arXiv.1512.03385

Herrick, R. R., Bjonnes, E. T., Carter, L. M., Gerya, T., Ghail, R. C., Gillmann, C., Gilmore, M., Hensley, S., Ivanov, M. A., Izenberg, N. R., Mueller, N. T., O'Rourke, J. G., Rolf, T., Smrekar, S. E., & Weller, M. B. (2023). Resurfacing History and Volcanic Activity of Venus. *Space Science Reviews*, *219*(4), 29. https://doi.org/10.1007/s11214-023-00966-y

Hicks, S. A., Strümke, I., Thambawita, V., Hammou, M., Riegler, M. A., Halvorsen, P., & Parasa, S. (2022).

On evaluation metrics for medical applications of artificial intelligence. *Scientific Reports*, *12*, 5979. https://doi.org/10.1038/s41598-022-09954-8

Hinton. (n.d.). *Lean on the Barrage: The Role of Artillery in Ukraine's Counteroffensive | Royal United Services Institute*. Retrieved October 27, 2023, from https://rusi.org/explore-our-research/publications/commentary/lean-barrage-role-artillery-ukraines-counteroffensive

Hosna, A., Merry, E., Gyalmo, J., Alom, Z., Aung, Z., & Azim, M. A. (2022). Transfer learning: A friendly introduction. *Journal of Big Data*, *9*(1), 102. https://doi.org/10.1186/s40537-022-00652-w

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications* (arXiv:1704.04861). arXiv. https://doi.org/10.48550/arXiv.1704.04861

Hunder, M. (2023, November 1). The drones fighting cat and mouse battles behind Russian front lines in Ukraine. *Reuters*. https://www.reuters.com/world/europe/drones-fighting-cat-mouse-battles-behind-russian-front-lines-ukraine-2023-11-01/

Iglovikov, V., & Shvets, A. (2018). *TernausNet: U-Net with VGG11 Encoder Pre-Trained on ImageNet for Image Segmentation* (arXiv:1801.05746). arXiv. https://doi.org/10.48550/arXiv.1801.05746

Ioffe, S., & Szegedy, C. (n.d.). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*.

Klabjan, D., & Harmon, M. (2019). Activation Ensembles for Deep Neural Networks. *2019 IEEE International Conference on Big Data (Big Data)*, 206–214. https://doi.org/10.1109/BigData47090.2019.9006069

Klambauer, G., Unterthiner, T., Mayr, A., & Hochreiter, S. (2017). *Self-Normalizing Neural Networks* (arXiv:1706.02515). arXiv. https://doi.org/10.48550/arXiv.1706.02515

Kumar, S. (n.d.). *Semantic hand segmentation using Pytorch | by Saurabh Kumar | Towards Data Science*. Retrieved November 11, 2023, from https://towardsdatascience.com/semantic-hand-segmentation-using-pytorch-3e7a0a0386fa

Levy, M. (2023, April 23). *Unprepared for long war, US Army under gun to make more ammo*. AP News. https://apnews.com/article/us-army-ukraine-russia-ammunition-war-75a9ca2e3be09578c65f1198ba5b72e5

Magnini, L., Bettineschi, C., & De Guio, A. (2017). Object-based Shell Craters Classification from LiDAR-derived Sky-view Factor. *Archaeological Prospection*, *24*(3), 211–223. https://doi.org/10.1002/arp.1565

Maxwell, A. E., Warner, T. A., & Guillén, L. A. (2021). Accuracy Assessment in Convolutional Neural Network-Based Deep Learning Remote Sensing Studies—Part 1: Literature Review. *Remote Sensing*, *13*(13), Article 13. https://doi.org/10.3390/rs13132450

Misra, D. (n.d.). *[1908.08681] Mish: A Self Regularized Non-Monotonic Activation Function*. Retrieved November 1, 2023, from https://arxiv.org/abs/1908.08681

Note, N., Gheyle, W., den Berghe, H. V., Saey, T., Bourgeois, J., Van Eetvelde, V., Van Meirvenne, M., & Stichelbaut, B. (2018). A new evaluation approach of World War One's devastated front zone: A shell hole density map based on historical aerial photographs and validated by electromagnetic induction field measurements to link the metal shrapnel phenomenon. *Geoderma*, *310*, 257–269. https://doi.org/10.1016/j.geoderma.2017.09.029

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library* (arXiv:1912.01703). arXiv. https://doi.org/10.48550/arXiv.1912.01703

Ronneberger, O., Fischer, P., & Brox, T. (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation* (arXiv:1505.04597). arXiv. http://arxiv.org/abs/1505.04597

Schneider, M. (n.d.). *Lessons from Russian Missile Performance in Ukraine | Proceedings—October 2022*

Vol. 148/10/1,436. Retrieved October 27, 2023, from https://www.usni.org/magazines/proceedings/2022/october/lessons-russian-missile-performance-ukraine

Sebring, H. C. (n.d.). *THE NORMAL BIVARIATE DENSITY FUNCTION AND ITS APPLICATIONS TO WEAPON SYSTEMS ANALYSIS, A REVIEW*.

Simonyan, K., & Zisserman, A. (2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition* (arXiv:1409.1556). arXiv. https://doi.org/10.48550/arXiv.1409.1556

Smith, J. (2023, September 12). North Korean ammunition could offer Russian troops flawed but useful support. *Reuters*. https://www.reuters.com/article/russia-northkorea-military-analysis-idAFKBN30I06N

Sommerville, Q. (2023, June 19). Ukraine war: BBC on the front line as Ukraine attacks Russian trenches. *BBC News*. https://www.bbc.com/news/world-europe-65921377

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. (2016). *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning* (arXiv:1602.07261). arXiv. https://doi.org/10.48550/arXiv.1602.07261

Tan, M., & Le, Q. V. (2019). *MixConv: Mixed Depthwise Convolutional Kernels* (arXiv:1907.09595). arXiv. https://doi.org/10.48550/arXiv.1907.09595

Wang, H., Jiang, J., & Zhang, G. (2018). CraterIDNet: An End-to-End Fully Convolutional Neural Network for Crater Detection and Identification in Remotely Sensed Planetary Images. *Remote Sensing*, *10*(7), Article 7. https://doi.org/10.3390/rs10071067

Yan, C., Fan, X., Fan, J., & Wang, N. (2022). Improved U-Net Remote Sensing Classification Algorithm Based on Multi-Feature Fusion Perception. *Remote Sensing*, *14*(5), Article 5. https://doi.org/10.3390/rs14051118

Yue, Z., Gao, F., Xiong, Q., Wang, J., Hussain, A., & Zhou, H. (2020). A Novel Attention Fully Convolutional

Network Method for Synthetic Aperture Radar Image Segmentation. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *13*, 4585–4598. https://doi.org/10.1109/JSTARS.2020.3016064

Zeng, Y.-X., Hsieh, J.-W., Li, X., & Chang, M.-C. (2023). *MixNet: Toward Accurate Detection of Challenging Scene Text in the Wild* (arXiv:2308.12817). arXiv. https://doi.org/10.48550/arXiv.2308.12817

Zhang, Q., He, C., Fang, X., Tong, M., & He, B. (2023). A Statistical-Texture Feature Learning Network for PolSAR Image Classification. IEEE Geoscience and Remote Sensing Letters, 20, 1–5. https://doi.org/10.1109/LGRS.2023.3306373

Appendix A

PyTorch & PyTorch Lightning Code Samples

```python
class SARTrainData(Dataset):
    IMAGE_PATH = "images"
    MASK_PATH = "labels"
    data_path = "DATAPATH"

    def __init__(
            self,
            data_path: str,
            img_size: tuple = (128, 128)
    ):
        self.img_size = img_size

        self.data_path = data_path
        self.img_path = os.path.join(self.data_path, "train", self.IMAGE_PATH)
        self.mask_path = os.path.join(self.data_path, "train", self.MASK_PATH)
        self.img_list = self.get_filenames(self.img_path)
        self.mask_list = self.get_filenames(self.mask_path)

    def __len__(self):
        return len(self.img_list)

    def __getitem__(self, idx):
        tfimg = v2.Compose([
            v2.Resize(self.img_size, 2),
            v2.ToDtype(torch.float32),
            v2.Normalize(mean=[0], std=[1])])

        img = Image.open(self.img_list[idx]).convert('L')
        img = np.asarray([img], dtype=np.float32)
        img = torch.Tensor(img)
        img = tfimg(img)

        msize = v2.Resize(self.img_size, 2)

        mpng = Image.open(self.mask_list[idx]).convert('L')
        mpng = msize(mpng)
        m1 = np.asarray([mpng], dtype=np.int32)
        mask = torch.from_numpy(m1)

        return img, mask


    def get_filenames(self, path):
        files_list = []
        for filename in os.listdir(path):
            files_list.append(os.path.join(path, filename))
        return files_list

```

```python
class DoubleConv(nn.Module):

    def __init__(self, in_ch: int, out_ch: int):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv2d(in_ch, out_ch, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(out_ch),
            nn.Mish(inplace=True),
            nn.Conv2d(out_ch, out_ch, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(out_ch),
            nn.ReLU(inplace=True),
        )

    def forward(self, x):
        return self.net(x)

```

```python
class Down(nn.Module):

    def __init__(self, in_ch: int, out_ch: int):
        super().__init__()
        self.net = nn.Sequential(
            nn.MaxPool2d(kernel_size=2, stride=2),
            DoubleConv(in_ch, out_ch)
        )

    def forward(self, x):
        return self.net(x)

```

```python
class Interp(nn.Module):
    def forward(self, x):
        return F.interpolate(x, scale_factor=2, mode="nearest-exact")

```

```python
class Up(nn.Module):

    def __init__(self, in_ch: int, out_ch: int):
        super().__init__()
        self.upsample = nn.Sequential(
            Interp(),
            nn.Conv2d(in_ch, in_ch // 2, kernel_size=1),
        )

        self.conv = DoubleConv(in_ch, out_ch)

    def forward(self, x1, x2):
        x1 = self.upsample(x1)

        # Pad x1 to the size of x2
        diff_h = x2.shape[2] - x1.shape[2]
        diff_w = x2.shape[3] - x1.shape[3]

        x1 = F.pad(x1, [diff_w // 2, diff_w - diff_w // 2, diff_h // 2, diff_h - diff_h // 2])

        x = torch.cat([x2, x1], dim=1)
        return self.conv(x)

```

```python
class UNet(nn.Module):

    def __init__(self, num_classes: int = 1, num_layers: int = 5, features_start: int = 64):

        super().__init__()
        self.num_layers = num_layers

        layers = [DoubleConv(1, features_start)]

        feats = features_start
        for _ in range(num_layers - 1):
            layers.append(Down(feats, feats * 2))
            feats *= 2

        for _ in range(num_layers - 1):
            layers.append(Up(feats, feats // 2))
            feats //= 2

        layers.append(nn.Conv2d(feats, num_classes, kernel_size=1))

        self.layers = nn.ModuleList(layers)

    def forward(self, x):
        xi = [self.layers[0](x)]
        # Down path
        for layer in self.layers[1: self.num_layers]:
            xi.append(layer(xi[-1]))
        # Up path
        for i, layer in enumerate(self.layers[self.num_layers: -1]):
            xi[-1] = layer(xi[-1], xi[-2 - i])
        return self.layers[-1](xi[-1])

```

```
class SARDataModule(LightningDataModule):
    def __init__(self, data_dir: str = "DATAPATH",
                 batch_size: int = 32):
        super().__init__()
        self.data_dir = data_dir
        self.batch_size = batch_size

    def setup(self, stage: str):

        if stage == "fit":
            SAR_full = SARTrainData(self.data_dir)
            self.SAR_train, self.SAR_val = random_split(
                SAR_full, [0.8, 0.2], generator=torch.Generator().manual_seed(37)
            )

        if stage == "test":
            self.SAR_test = SARTestData(self.data_dir)


        if stage == "predict":
            self.SAR_predict = SARTestData(self.data_dir)

    def train_dataloader(self):
        return DataLoader(self.SAR_train, batch_size=self.batch_size,
                          num_workers=24, persistent_workers=True, pin_memory=True)

    def val_dataloader(self):
        return DataLoader(self.SAR_val, batch_size=self.batch_size,
                          num_workers=8, persistent_workers=True)

    def test_dataloader(self):
        return DataLoader(self.SAR_test, batch_size=self.batch_size,
                          num_workers=4)

    def predict_dataloader(self):
        return DataLoader(self.SAR_predict, batch_size=self.batch_size,
                          num_workers=4)

```

```
ALPHA = 0.3
BETA = 0.9
GAMMA = 1.7

class FocalTverskyLoss(nn.Module):
    def __init__(self, weight=None, size_average=True):
        super(FocalTverskyLoss, self).__init__()

    def forward(self, inputs, targets, smooth=1, alpha=ALPHA, beta=BETA, gamma=GAMMA):

        inputs = F.sigmoid(inputs)

        inputs = inputs.view(-1)
        targets = targets.view(-1)

        TP = (inputs * targets).sum()
        FP = ((1-targets) * inputs).sum()
        FN = (targets * (1-inputs)).sum()

        Tversky = (TP + smooth) / (TP + alpha*FP + beta*FN + smooth)
        FocalTversky = (1 - Tversky)**gamma

        return FocalTversky

```

```
class DiceBCELoss(nn.Module):
    def __init__(self, weight=None, size_average=True):
        super(DiceBCELoss, self).__init__()

    def forward(self, inputs, targets, smooth=1):

        inputs = F.sigmoid(inputs)

        inputs = inputs.view(-1)
        targets = targets.view(-1)

        intersection = (inputs * targets).sum()
        dice_loss = 1 - (2.*intersection + smooth)/(inputs.sum() + targets.sum() + smooth)
        BCE = F.binary_cross_entropy(inputs, targets, reduction='mean')
        Dice_BCE = BCE + dice_loss

        return Dice_BCE

```

```python
class LitUNet(LightningModule):

    def __init__(
            self,
            data_path: str = "DATAPATH",
            batch_size: int = 32,
            lr: float = 0.001,
            num_layers: int = 5,
            features_start: int = 64,
            **kwargs,
    ):
        super().__init__(**kwargs)
        self.data_path = data_path
        self.batch_size = batch_size
        self.lr = lr
        self.num_layers = num_layers
        self.features_start = features_start
        self.save_hyperparameters(),

        self.net = UNet(
            num_classes=1, num_layers=self.num_layers, features_start=self.features_start
        )

        self.acc = TM.BinaryAccuracy(threshold=0.9)

        pos_weight = torch.Tensor([60])
        self.loss_fn = nn.BCEWithLogitsLoss(pos_weight=pos_weight)
        #self.loss_fn = FocalTverskyLoss()
        #self.loss_fn = DiceBCELoss()
        threshold=0.49
        self.recall = TM.BinaryRecall(threshold=threshold)
        self.precision = TM.BinaryPrecision(threshold=threshold)
        self.f1 = TM.BinaryF1Score(threshold=threshold)


    def forward(self, x):
        return self.net(x)

    def training_step(self, batch, batch_idx):
        img, mask = batch
        out = self.net(img)
        loss = self.loss_fn(out, mask.float())
        t_acc = self.acc(out, mask)
        log_dict = {"train_loss": loss, "train_acc": t_acc}
        self.log_dict(log_dict, prog_bar=True)

        return loss

    def validation_step(self, batch, batch_idx):
        img, mask = batch
        out = self.net(img)
        val_loss = self.loss_fn(out, mask.float())
        v_acc = self.acc(out, mask)
        recall = self.recall(out, mask.long())
        precision = self.precision(out, mask.long())
        f1 = self.f1(out, mask.long())
        v_log_dict = {"val_loss": val_loss, "val_acc": v_acc,
                      "val_recall": recall, "val_prec": precision,
                      "val_f1": f1}
        self.log_dict(v_log_dict, prog_bar=True)
        return val_loss


    def test_step(self, batch, batch_idx):
        img, mask = batch
        out = self.net(img)
        recall = self.recall(out, mask.long())
        precision = self.precision(out, mask.long())
        f1 = self.f1(out, mask.long())
        log_dict = {"recall": recall, "precision": precision, "f1": f1}

        self.log_dict(log_dict, logger=True, on_step=True)

    def predict_step(self, batch):
        inputs, target = batch
        return self.net(inputs)

    def configure_optimizers(self):
        opt = torch.optim.ASGD(self.net.parameters(), lr=self.lr)
        sch = torch.optim.lr_scheduler.CosineAnnealingLR(opt, T_max=8)

        return [opt], [sch]

```

```python
model = LitUNet(data_path="DATAPATH")
mlf_logger = MLFlowLogger(
    experiment_name="litUnetLogs",
    tracking_uri="http://localhost:3737",
    log_model=True
    )

dm = SARDataModule(data_dir="DATAPATH")


checkpoint_callback = ModelCheckpoint(
    monitor="val_loss",
    dirpath="DIRPATH",
    filename="sar-{epoch:02d}-{val_loss:.2f}",
    save_top_k=1,
    mode="min"
    )


trainer = Trainer(
    accelerator="gpu", devices=1,
    log_every_n_steps=30,
    # accumulate_grad_batches = 2,
    profiler="simple",
    default_root_dir="ROOTPATH",
    enable_checkpointing=True,
    logger=mlf_logger,
    max_epochs=25,
    callbacks=[checkpoint_callback]
    )


```