CS 175: Projects in AI

FRONT COVER

CS 175 Technical Memorandum

Challenge Title:
AI for Social Good: Electricity and Settlement Detection

Authors:
Cole Hajek, Jacob Berkel, Jaehoon Song, Nathan Huey

Team Name: The Thunder Men

March 2024

INTRODUCTION

All around the world, individuals and households lack access to sufficient electricity and thus live in energy poverty. More formally, energy poverty occurs when a household is forced to reduce its energy consumption to a level that is harmful to its inhabitants. Energy poverty can be widespread within certain regions. For example, 48% of people living in sub-Saharan Africa live without electricity (Mukhtar M et al., 2023). Additionally, energy poverty is becoming more widespread as the percentage of people experiencing energy poverty has increased by 4% from 2019 to 2021 (IEA, 2022).

Electricity is critical to modern life. Living without a sufficient source of electricity can cause issues with educational opportunities, medical capabilities, internet access, and light at night. Luckily, energy poverty is fixable and efforts to supply electricity to those in need can have a positive impact on a wide scale. To begin this process, settlements that lack electricity need to be discovered so that plans to supply them with electricity can be created.

Detecting which settlements lack electricity can be extremely difficult as the problem can span entire continents. To tackle the large scope of this problem, we have trained machine learning models to predict the location of settlements that lack electricity. To achieve this, we trained our models using satellite imaging data from sub-Saharan Africa. Through this process, we aim to create a reliable and accurate method of detecting settlements that lack electricity. Then, we hope this resource can be used to extend the power grid in Africa to those in need.

MAIN BODY

Identification Need

Identifying settlements that lack electricity requires monumental effort as identification can span entire continents. This further illustrates the need for this research as machine learning models are often relied upon when processing large amounts of data. Given the widespread nature of energy poverty in some regions of the world, this research can be integral to the effort of supplying energy to those in need. When considering training machine learning models for this segmentation task, some challenges include preprocessing the large dataset, the large amount of computation needed to train the models, and the time constraint of the deadline.

A common neural network design for segmentation tasks is a convolution neural network (CNN). These neural networks use pooling layers to detect features within the training data. CNNs are often used as a baseline for many other more complicated and complex model's architecture. One such example of a well-known CNN is ResNet 101, a powerful image classification model with up to 152 layers. Using transfer learning, pre-trained models like ResNet 101 can be used to specialize in different tasks, such as semantic segmentation within a given scope.

Another popular model architecture for semantic segmentation is the U-Net, a type of fully-connected convolutional neural network. The U-Net was originally developed for biomedical image segmentation. The network consists of an expansive path, and a contracting path, which gives the architecture its namesake U shape. While originally developed for biomedical image segmentation, the U-Net can be used in applications such as self-driving cars and satellite imagery analysis.

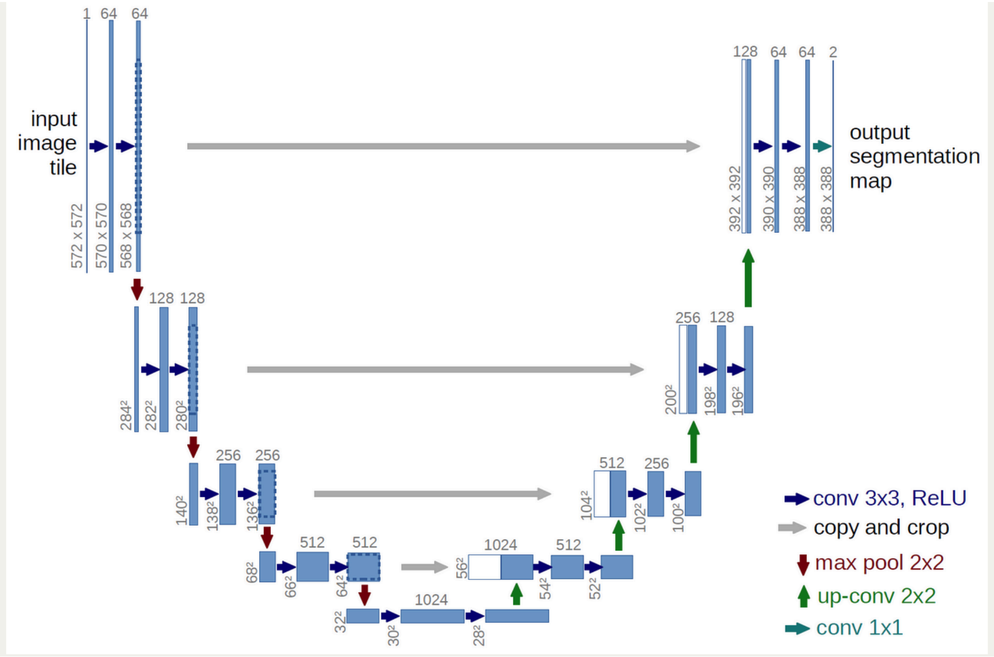


Figure 1. Diagram with a visual representation of the U-Net architecture. Segmentation of Satellite Imagery using U-Net Models for Land Cover Classification (Priit Ulmas et al. 2020).

This work includes using a modified U-Net structure model for creating land cover classification mapping based on satellite imagery. This study used satellite images taken from the Sentinel-1, Sentinel-2, and Landsat-8 to detect land cover.

Fully Convolutional Networks for Semantic Segmentation (Jonathan Long et al. 2014).

The work aimed to prove that fully convolutional neural networks exceed the state-of-the-art in semantic segmentation. This focused on transferring existing convolutional neural network representation and fine-tuning for segmentation tasks.

Data Description

To train our models, we used the 2021 IEEE GRSS dataset. This dataset was a part of the 2021 IEEE GRSS Data Fusion Contest, organized by the Image Analysis and Data Fusion Technical Committee and the Geoscience and Remote Sensing Society. The data contains satellite imaging data and is comprised of 98 tiles of 800x800 pixels. Each tile contains 98 channels, composed together from 4 different satellites. These tiles are split into 60 tiles for training, 19 tiles for validation, and 19 tiles for testing. Each tile has been resampled to a Ground Sampling Distance of 10m, and thus each tile corresponds to a 64km2 area. This dataset contains channels captured from the satellites Sentinel-1, Sentinel-2, Landsat 8, and Suomi NPP.

Sentinel-1 polarimetric SAR dataset

- Contains 4 images with 2 channels which represent the intensity values for VV and VH polarization. This includes 2.1 GB of data, stored as type float32. This data was taken with the acquisition mode of interferometric Wide Swath.

Sentinel-2 multispectral dataset

- Contains 4 images with 12 channels of reflectance data within VNIR and SWIR ranges at a GSD of 10, 20m, and 60m. This includes 6.2GB of data, stored as type uint16.

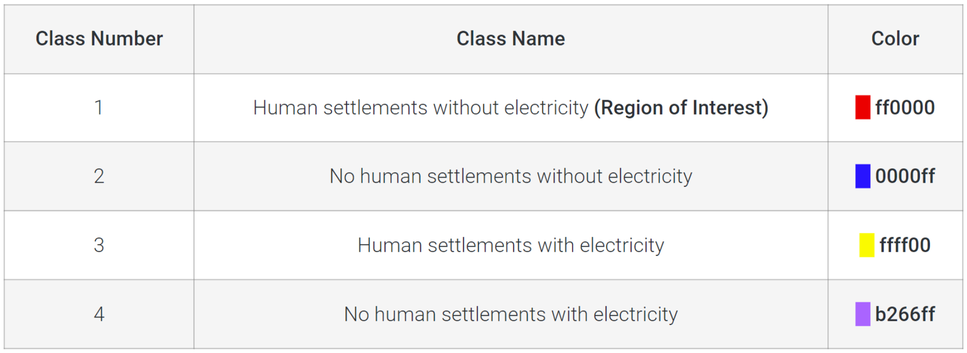
Landsat 8 multispectral dataset

- Contains 3 images with 11 channels of reflectance data covering VNIR, SWIR, and TIR ranges. This includes 8.5GB of data, stored as the type float32. The sensors used include OLI and TIRS sensors.

The Suomi Visible Infrared Imaging Radiometer Suite nighttime dataset

- Contains 9 images with data from the Day-Night band sensor of the Visible Infrared Imaging Radiometer Suits provides on 1 channel. For this channel, the global daily measurements of nocturnal visible near-infrared light at a GSD of 750m. This is a corrected version of the original DNB data. This includes 1.2 GB of data, sorted as type uint16.

This dataset also contains ground truth images, used for reference when training and testing our model. These ground truth images contain 4 unique semantic labels.



Class Number	Class Name	Color
1	Human settlements without electricity **(Region of Interest)**	🟥 **ff0000**
2	No human settlements without electricity	🟦 **0000ff**
3	Human settlements with electricity	🟨 **ffff00**
4	No human settlements with electricity	🟪 **b266ff**

Figure 2. *2021 IEEE GRSS Data Fusion Contest.*

Methodology

Theoretical Aspects

Solving the problem of identifying human settlements and electricity is a problem of semantic segmentation. Semantic segmentation is a computer vision problem with the goal of labeling every pixel in an image as a particular class or object. For this project, our goal was to label pixels in tiles of satellite images as one of four classes shown in Figure 2.

Due to a lack of time and computing power, we were unable to implement any form of dimensionality reduction based on model performance or estimated feature similarity. Instead, we researched the intended function of each of the satellite bands being used as features in our dataset to logically determine what bands would be the most practical for our task. We were able to test a couple of band combinations but the end choice was to use:

VIIRS: max projection - For night-time electricity use detected via light pollution
Sentinel1: VV, VH - For feature detection and p
Sentinel2: 02,03,04 - High definition RGB
Sentinel2: 08,11,12 - Detecting soil moisture and vegetation health/water bodies can be good indirect indicators of human activity or lack thereof
Landsat: 5, 6, 7, 8 - Near Infrared - More analysis on vegetation health penetrating atmospheric haze, and a high-definition panchromatic mapping

Machine Learning Techniques Used

Machine learning requires data to be in a uniform and compact form. We had to clean and format our data to a form that machine learning tools like TorchVision could use.

After cleaning and formatting the data, we injected augmented versions of our base 60 tiles into the dataset to expand the size of our data and help the model learn more generalized features. We had four transformations - add noise, Gaussian blur, horizontal flip, and vertical flip - each of which had a 50% chance of being applied.

Due to overwhelming inequity in class frequency, our models had a hard time identifying areas with electricity. Due to how infrequent classes 3 and 4 were it led our model to ignore them completely. In an attempt to remedy this situation, weigh the loss function of each class to be inversely proportional to its frequency using the formula:

$$\frac{Sum\ of\ Classes}{(number\ of\ classes * frequency\ of\ class\ i)}$$

This function was successful in getting our model to predict the minority classes however it didn't seem to improve the ability of our model to distinguish them. Its effect on the minority tiles was negligible compared to the decrease in the overall performance of the model.

Model Architectures

We developed three supervised learning models: a Segmentation Convolutional Neural Network, a U-Net, and a Fully Convolutional Network with transfer learning from FCN Resnet 101. Each of these models is based on a Convolutional Neural Network or CNN. CNNs are commonly used for image segmentation, as they can identify features in images by learning weighted filters.

SegmentationCNN

Our Segmentation CNN model had a basic CNN architecture consisting of encoding blocks that fed into max-pooling layers, then fully connected layers.

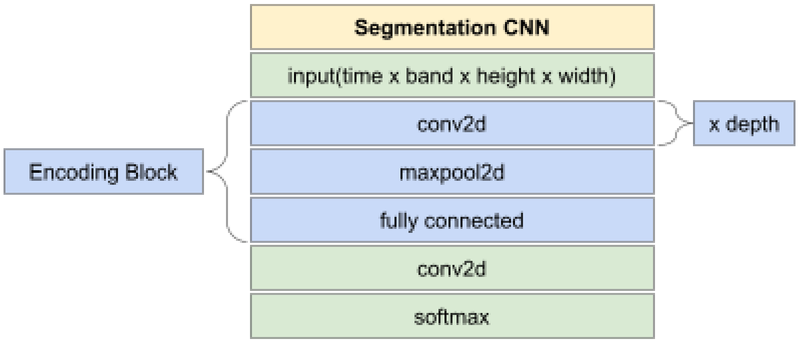


Figure 3. *Segmentation CNN Architecture.*

UNet

Our U-Net model followed the U-shaped architecture consisting of encoders that fed into decoders with skip connections between encoders and decoders of equal size. The architecture is shown in Figure 1.

FCNResnetTransfer

Our Fully Convolutional Network had custom input and output layers. The backbone was taken from the FCN Resnet 101 model with pre-trained weights.

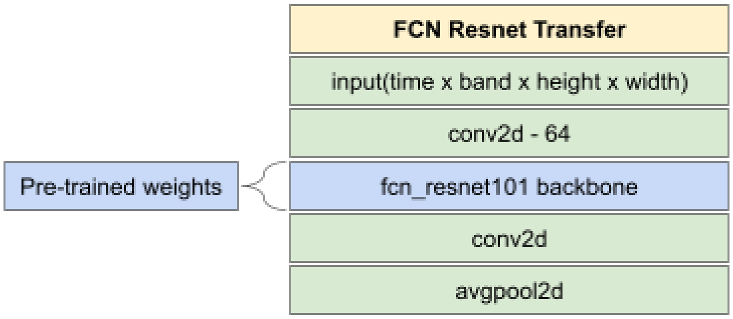


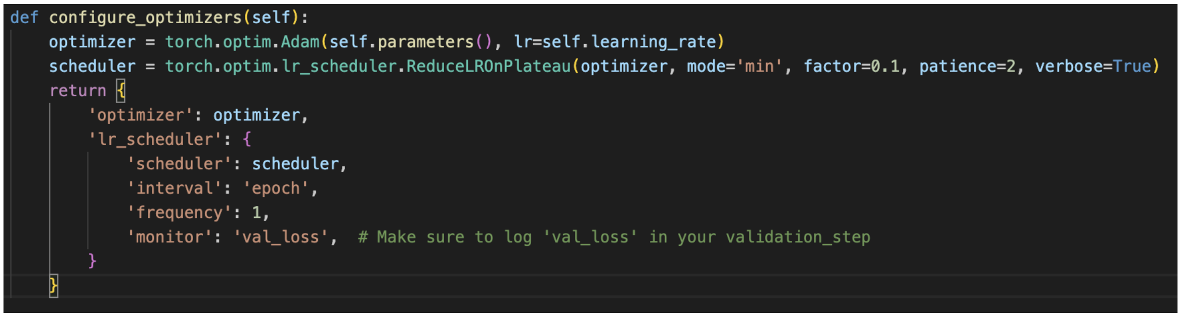
Figure 4. *FCN Resnet Transfer architecture.*

Metrics and Hyperparameters

To quantify the quality of our models, we measured accuracy, F1 scores, and area under curve scores, both per class and on average. Our models used the Adam optimizer with dynamic step sizes and using cross entropy as our loss function. For our experiments using weighted data, we used cross entropy with weights.

We ran sweeps over a variety of hyperparameters to find the optimal combination. The hyperparameters we searched over included learning rate, batch size, depth, embedding size, and number of encoders.

Learning rate optimization



```python
def configure_optimizers(self):
    optimizer = torch.optim.Adam(self.parameters(), lr=self.learning_rate)
    scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=2, verbose=True)
    return {
        'optimizer': optimizer,
        'lr_scheduler': {
            'scheduler': scheduler,
            'interval': 'epoch',
            'frequency': 1,
            'monitor': 'val_loss',  # Make sure to log 'val_loss' in your validation_step
        }
    }
```

Figure 5. *Learning rate configuration.*

To find the best initial learning rate we ran Bayes-style sweeps which indicated the best learning rates were in the range 0.05 and 0.0001. So we decided to use a midpoint (0.001) on the higher end to converge with fewer epochs.

We experimented with a couple of different methods of updating our Learning rate at first having no rescheduling, then we tried decreasing the learning rate on a consistent schedule (every 3 epochs it would be decreased by a factor of 0.1). This was a better approach but ultimately we settled on an adaptable learning rate that decreased if there was no progress after 2 epochs. This had a significant boost to our model accuracy, allowing us to converge on several minima.

Replicability

In order to ensure the replicability of the results achieved in our project, it is crucial to document and provide clear instructions for the future or just for general documentation purposes. In this section, we will not only explain the scripts required to repeat the results but also break them down step by step so for effortless replicability if any readers of this choose to replicate this process.

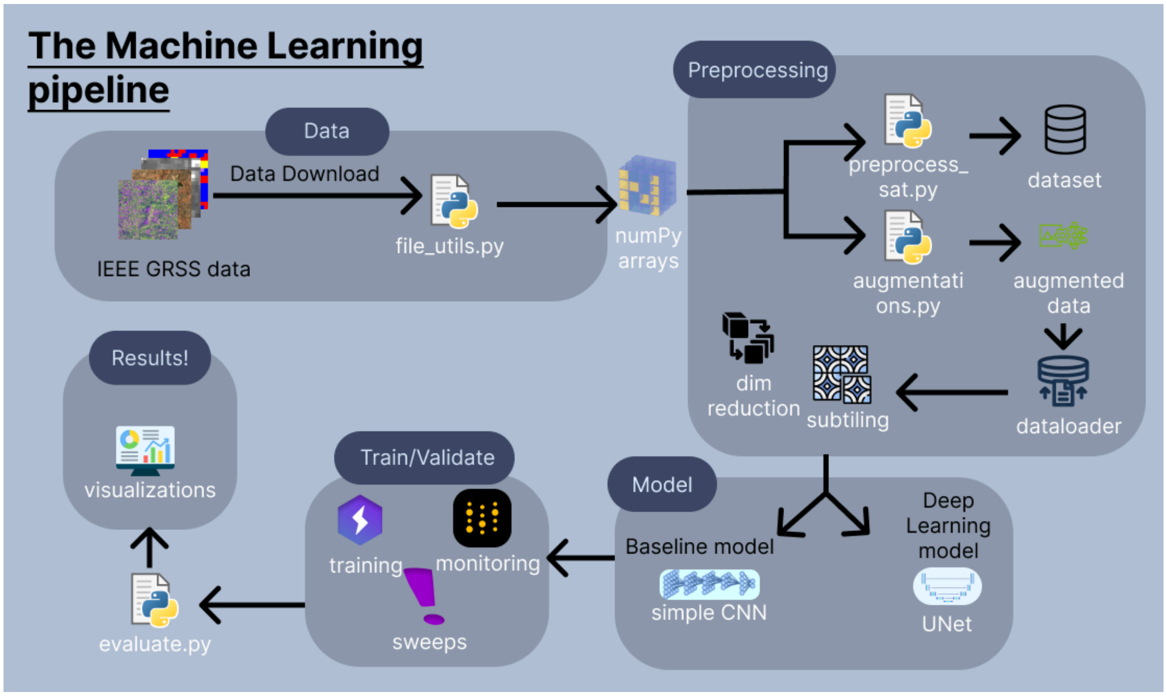


***Figure 6.** Our Machine Learning Pipeline.*

Step 1: Data Acquisition:

In order to obtain the IEEE GRSS 2021 satellite imagery dataset, one must download it here. The link will redirect to a Google Drive page where one can download the training data for our project.

Step 2: Data Utility before Preprocessing:

Before preprocessing raw training data, we use file_util.py to define a module for loading and processing satellite imagery data. The Metadata class encapsulates essential information about each satellite file, such as file type, name, tile ID, bands, and time. This class serves as a container for storing metadata, enabling easy access and manipulation of file-specific information. Various functions are defined for parsing file names of different satellite types to extract date and band information using pattern-matching techniques like Regex expressions. Functions like get_satellite_files and read_satellite_files obtain a collection of satellite files and convert satellite images into NumPy arrays, preparing the data for further analysis and modeling by converting raw files into structured arrays and extracting relevant metadata.

Step 3: Data Preprocessing and Augmentations:

The preprocessing section in preprocess_sat.py enhances satellite imagery for analysis and machine learning. Tailored functions handle specific satellite data types, applying domain-specific preprocessing steps for consistency and compatibility across datasets. Overall, preprocessing improves image quality and usability for analysis, visualization, and machine learning tasks. The main preprocessing section involves preprocessing satellite images, creating custom PyTorch datasets, data loaders, and transforms, and exploring dimensionality reduction as well as augmentation techniques. Provided files like file_utils.py and preprocess_sat.py are utilized for preprocessing. Tasks involve subtitling satellite images for processing and dynamic mosaicking, writing custom PyTorch datasets, data loaders, and transforms, and exploring dimensionality reduction techniques such as PCA, t-SNE, and UMAP to identify patterns in the data. Simplified grid slicing functions, custom Dataset classes inheriting from PyTorch, and transformations for data augmentation are implemented. PyTorch DataLoader handles batch loading and shuffling, while a Lightning DataModule manages training and validation data loaders. Dimensionality reduction techniques are applied to visualize and analyze patterns and feature combinations in sub-tiled satellite images, aiding in understanding the data's structure and relationships.

Step 4: Model Selection, Training/Validating, and Producing Results:

After selecting what model to train and specifying ESDConfig we can train our model using train.py. For our deep learning model, we utilized UNet with the other choices being SegmentationCNN and FCNResnetTransfer. After this, we can evaluate the model by passing in a model_path of the model you would like to run which will save various types of files that we can use to evaluate our model. Additionally, we run a script called train_sweeps.py. This script first trains the selected model, initializes a Weights and Biases, extracts hyperparameters from the config, and runs the sweep with the sweeps.yml file as our input. After that, the results of our sweeps from our trained model are available from the project directory specified in the configuration in Weights and Biases. Overall, train_sweeps automates the process of tuning hyperparameters using Weights and Biases, allowing for efficient exploration of hyperparameter space to improve model performance.

Tools

In this section, we will go over the different technologies that we have utilized for the generation of results.

Technology Used	**Use case**
Python	Our main programming language used to write all machine learning scripts
NumPy	Utilized as main format for preprocessing, converted into tensors later
PyTorch / PyTorch Lightning	Define and train deep learning models
Scikit-Learn	Split dataset into training and validation, dimensionality reduction techniques
CPU	Utilized our laptop’s CPU to perform hyperparameter sweeps

Figure 7. *Technologies utilized in this project.*

Tests

We ran hyperparameter sweeps to optimize the configuration of our models.

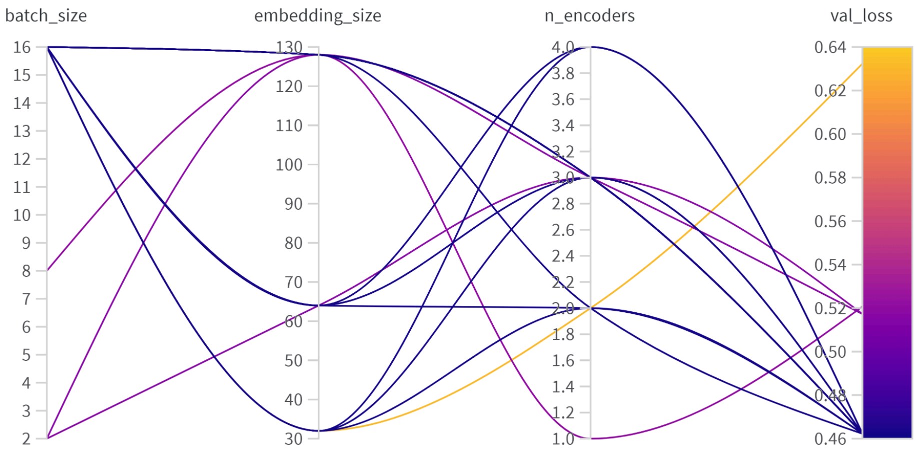


Figure 8. *Hyperparameter sweep results.*

We split our data into 80% training and 20% validation. We evaluated our models on the validation data. Our evaluation criteria were accuracy, the area under the curve (AUC) score, and the F1 score.

Results

Segmentation CNN

Unfortunately, this model was only able to learn to predict the most common class type. As shown in our results in Table A1 in the appendix, this model had 100% accuracy for class 1 and 0% accuracy for all other classes. Class 1 made up nearly 60% of the data, so it was not surprising that such a simple model would fall into a local minimum of always predicting the most likely class.

FCN Resnet Transfer

The FCNResnetTrasnfer architecture did a much better job than the SegmentationCNN. All four classes are being predicted, however accuracy, precision, and recall are still quite low overall. As shown in Table A2 in the appendix, the peak average accuracy was just under 50%. This model may have some potential but was not the most promising architecture we used.

As seen in Figure 9, the accuracy of the four classes was erratic across each step, with the final accuracies falling well before 60%.

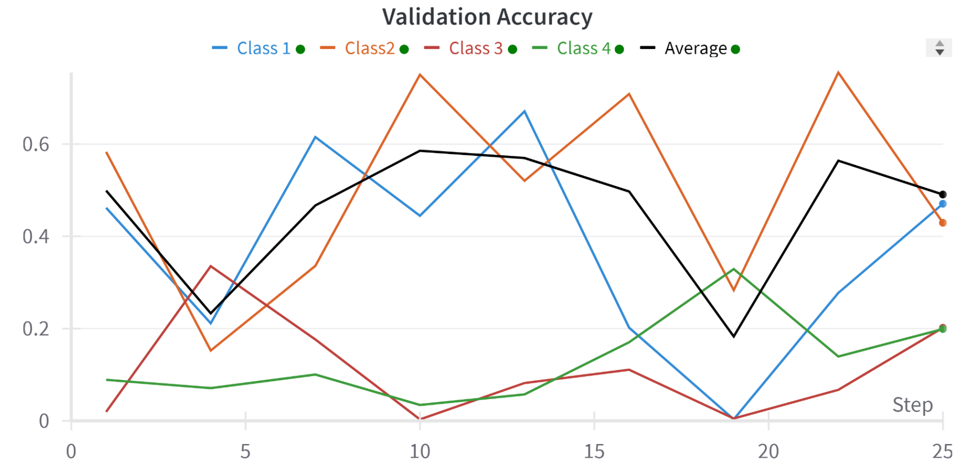


Figure 9. *FCNResnetTransfer final accuracy.*

UNet

Our UNet model did a much better job learning the different classes. It was able to correctly label human settlements and no human settlements very well, as demonstrated in Figure 11 (a), however, this model had trouble distinguishing between electricity and no electricity. Data in classes 3 and 4 - the classes with electricity - are rare in the dataset and the model was unable to correctly identify them. This model achieved a peak average accuracy of over 80%, as shown in Table A3 in the appendix. Classes 1 and 2 have accuracies over 75%, but class 3 had less than 1% accuracy and class 4 had a 0% accuracy.

These results suggest that a two-binary classification approach would be more effective where two binary classification models are created - one to identify human settlements and a second to identify electricity. This could be an area of exploration in continuing this project.

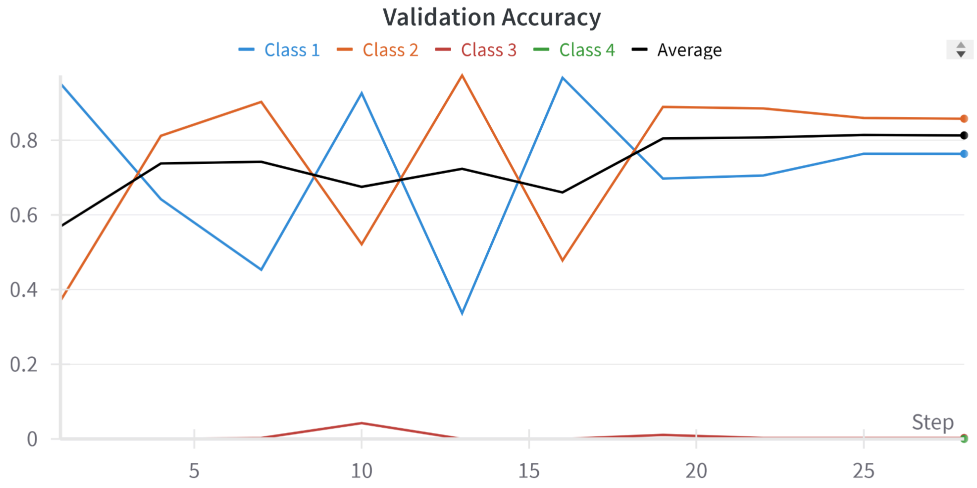
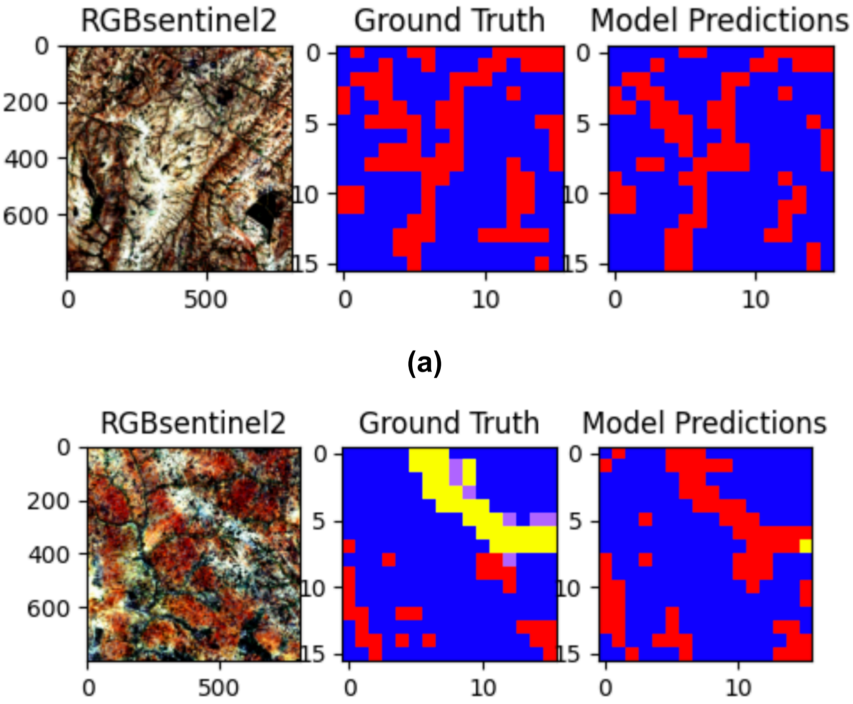


Figure 10. *UNet final validation accuracy score over 9 epochs.*



(a)

(b)

Figure 11. *(a) UNet predictions on data from only classes 1 and 2. (b) UNet predictions on data of all four classes.*

Github source code

All source code is available on our Github page linked below.

https://github.com/cs175cv-w2024/final-project-the-thunder-men

CONCLUSION

This project demonstrates the potential, or lack thereof, of three different common machine learning architectures.

Our most promising model was our UNet model, which reached a maximum average accuracy of 81%. This model had over 70% accuracy in differentiating between human and no-human settlements but was unable to differentiate between electricity and no electricity, instead opting to almost always predict no electricity. The UNet architecture shows great potential in solving semantic segmentation problems on satellite imagery.

Our most basic architecture, the SegmentationCNN, shows that a simple architecture is not up to the task. This model learned to always predict the most common class label and was unable to escape this local minimum.

The FCNResnetTransfer showed some promise. This model had higher accuracy than the UNet on the two rarer class types - the classes with electricity - but had lower overall accuracy, reaching a peak average accuracy of 49%. The fact that this model achieved more balanced accuracy levels than the UNet may suggest there is potential with this model.

The most promising avenue for future work is creating multiple models to solve two separate classification problems: identifying human settlements and identifying electricity. Solving these problems separately may lead to a model that can better predict all four class types, and would allow for more tailored satellite bands to be used for each subproblem.

APPENDIX

Table A1: SegmentationCNN validation metrics

	Accuracy	AUC	F1
Class 1	0.0000	0.4827	0.0000
Class 2	1.0000	0.5213	0.6704
Class 3	0.0000	0.2008	0.0000
Class 4	0.0000	0.1665	0.0000
Average	0.5325	0.3428	0.2799

Table A2: FCNResnetTransfer validation metrics

	Accuracy	AUC	F1
Class 1	0.4706	0.6914	0.4506
Class 2	0.4293	0.6951	0.4893
Class 3	0.2018	0.2823	0.1392
Class 4	0.1991	0.2259	0.0475
Average	0.4906	0.4737	0.2850

Table A3: UNet validation metrics

	Accuracy	AUC	F1
Class 1	0.7635	0.8938	0.7140
Class 2	0.8576	0.9036	0.8466
Class 3	0.0027	0.1563	0.0052
Class 4	0.0000	0.08018	0.0000
Average	0.8128	0.5085	0.7231

REFERENCES

[REF. NO.] 2021 IEEE GRSS Data Fusion Contest. Online: www.grss-ieee.org/community/technical-committees/data-fusion”

Mukhtar M, Adun H, Cai D, Obiora S, Taiwo M, Ni T, Ozsahin DU, Bamisile O. Juxtaposing Sub-Sahara Africa's energy poverty and renewable energy potential. Sci Rep. 2023 Jul 19;13(1):11643. doi: 10.1038/s41598-023-38642-4. PMID: 37468495; PMCID: PMC10356766.

IEA (2022), Africa Energy Outlook 2022, IEA, Paris https://www.iea.org/reports/africa-energy-outlook-2022, Licence: CC BY 4.0

Priit Ulmas, Innar Liiv (2020). Segmentation of Satellite Imagery using U-Net Models for Land Cover Classification. https://doi.org/10.48550/arXiv.2003.02899

Jonathan Long, Evan Shelhamer, Trevor Darrel (2014). Fully Convolutional Networks for Semantic Segmentation. https://doi.org/10.48550/arXiv.1411.4038

ACKNOWLEDGEMENTS

The authors would like to thank the IEEE GRSS Image Analysis and Data Fusion Technical Committee, Hewlett Packard Enterprise, SolarAid, and Data Science Experts for organizing the Data Fusion Contest.