Detecting Solar Arrays Using a Convolutional Neural Network to Analyze their Development on Agricultural Land in Massachusetts

Chet Jambora, Penn State MGIS Program, 2022

Advisor: Jan Oliver Wallgrün

12/9/2022

Table of Contents

Abstract

The state of Massachusetts has seen a dramatic increase in the number of ground-based solar arrays since the passage of several state and federal bills encouraging their construction. Though solar arrays are tracked by state agencies, there has been no public report on the changes caused by array development on the state's land use, specifically with regard to agricultural land. Economic exigencies have led some farmers to lease sections of their land for solar projects, which provide renewable energy but can be divisive for neighbors [26]. To better understand the growth of solar arrays, I built and trained a machine learning model that can detect arrays in aerial imagery. Image tiles from four MA counties were processed in ArcGIS Pro and then fed to the model, which returned binary classification rasters showing the predicted locations of arrays. These rasters were used to estimate the areas cleared for installation, which were compared to a land-use layer from a year prior to wide-scale array construction to ascertain the previous land-use at occupied sites. Predicted agricultural land overlap was highest in Hampden [248.3 acres] and Hampshire [147.7 acres] counties. Also, 93.4 acres were predicted in Plymouth county and 1 acre in Barnstable county. Though Hampshire and Plymouth contain more farmland, Hampden county had a higher level of solar development, which may reflect more conducive policy conditions in that county [25]. This study serves as a starting point for future investigations into land-use conversion that may help inform policymakers and the general public about the impacts of solar array development in their communities.

Introduction

Solar Array Development in Massachusetts

The state of Massachusetts is a leader in solar development in the nation, ranking 9th among states in the amount of electricity produced from solar [39]. This development was first stimulated by the passage of the Green Communities Act of 2008, which increased state-wide requirements for renewable energy and instituted "net-metering" so homeowners could sell their surplus solar power to utilities [40, 5]. Subsequently, the MA Clean Energy Center (MA CEC) was formed in 2009 which implemented solar rebate programs [24]. Another agency, the MA Department of Energy Resources (MassDOER) also began operating renewable energy certificate programs commonly known as SREC I and SREC II. These programs, which ran from 2010-2014, were targeted towards distributed (utility) solar, and meant to help the state reach its 1600 MW solar capacity goal [24]. In 2018, MassDOER initiated the SMART program to replace SREC which offers subsidies to power companies and currently supports 3200 MW of solar capacity [34]. Though these programs have been effective in encouraging solar development, recent trends suggest that growth has slowed considerably, with new installations falling by 50% in 2019 [26]. This has been attributed to a variety of causes including added restrictions to SMART, electric grid "congestion," and increased opposition by communities to further development and land-use conversion [26].

Figure 1 shows the trend of commercial solar array construction. The year range 2012-2017 saw the highest number of commercial builds, and, according to MassAudubon, their construction accounted for about ¼ of the development of natural lands in that time period [20].

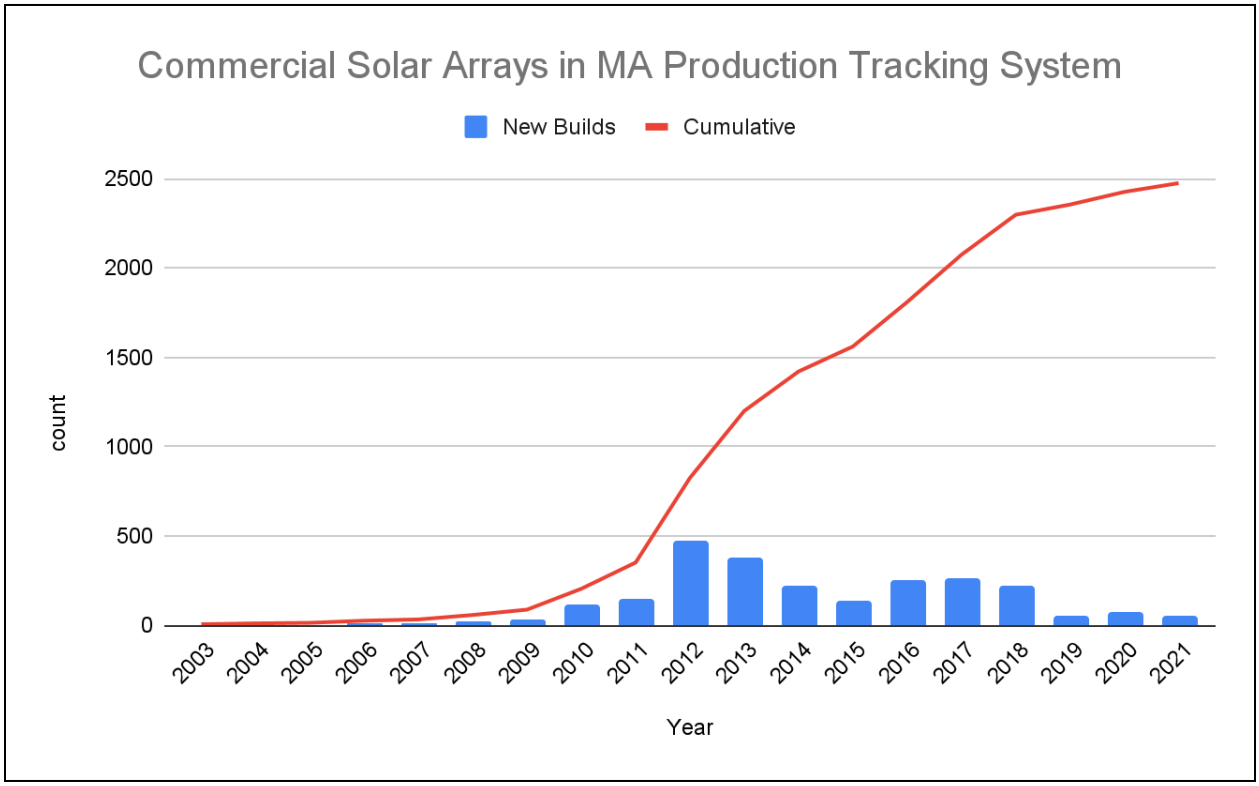

Figure 1: Commercial array installations in MA, showing new builds by year and cumulative builds over time.

Land-Use Conversion and Solar Arrays

The rapid development of ground-based solar arrays has led to some concerns about unchecked land-use change. Ideally, arrays would be situated in areas that have low demand for human use and relatively low ecological value. These include

brownfields, old industrial grounds, and capped landfills, an example being the Sullivan's Ledge property in New Bedford, MA, a contaminated landfill that was converted to a 1.75 MW solar facility [9]. Though locations such as these are preferred, they are often costly to develop and in aggregate do not provide enough surface area to meet capacity goals [6]. Thus, utilities and solar companies often build on other more favored land-use types such as forest, open land, and agricultural land (see Figure 2).

Figure 2: A solar array on old farmland in Rutland, MA.

Agricultural land is ideal for array development because it is usually flat, cleared of obstructions, and near connections to the power grid [29]. Many farmers are incentivized to lease this land to solar companies for the guaranteed passive income they provide [29]. It often makes better sense for them to "harvest the sun" than to try to grow crops on increasingly thin margins with higher fuel and feed costs. Despite this, there is concern in some Massachusetts communities that excessive array development

will fundamentally alter local agricultural landscapes and possibly impact food security.[1] This speculation may be caused by a lack of detailed information on array land-use conversion available to the public. One source which provides some insight is the MassCEC Production Tracking System, a database of renewable energy installations in Massachusetts. Within it there are 3,160 “agricultural” solar entries as of February 2022 [21]. However, these entries do not specify the size of the installations or their coordinates, and therefore cannot be used to estimate the extent of array development on agricultural land.

Detection of Solar Arrays with Machine Learning

To estimate land-use conversion caused by solar array installations, they must first be detected and rendered as a layer in mapping software. This can be done efficiently by using an image classification model to automatically detect arrays in imagery. These models are trained on large datasets by which they learn to generalize features of predefined classes. The model updates its internal parameters by checking the output of a loss function, which assesses its prediction capability against labeled examples. Once a model is properly trained it can be used to predict instances of a class such as solar arrays in new images. Models can be configured to predict the class of a single object in an image, which is called image segmentation, or to predict the class of each pixel in an image, which is known as semantic segmentation.

[1] Interestingly, the period of greatest commercial solar development (2012-2017) also saw a noticeable decline in the total land in farms [-31,864 acres] in the state [38].

One of the most frequent types of machine learning models used for image classification tasks is the convolutional neural network, or CNN. They are ideal for this because they are able to reduce image dimensionality (through convolutions) while limiting information loss [16]. They also need fewer parameters than other models such as feed-forward networks and are better able to handle “noisy” data [27]. CNN’s are inspired by the way humans process images in the visual cortex of their brains [2]. They are classified as a type of deep learning, which is a subset of machine learning based on neural nets [13]. In a CNN, images are rendered as multi-dimensional arrays (tensors) and passed through a convolution layer which has a filter (kernel) that produces another multi-dimensional array called a feature map [27]. A round of convolutions is connected to a pooling layer which downsamples the input, reduces parameters, and filters data noise [14]. The sequence (convolution, convolution, pooling) occurs several times, but this path alone does not result in the reconstruction of the whole image [16]. This shortcoming was resolved in 2015 by Olaf Ronneberger who developed the U-Net configuration of the CNN (see Figure 3) [32]. The U-Net has a U-shaped architecture with a classical CNN path (contracting) on the right side of the “U,” a bottleneck at its bottom, and an added “expansive” path that uses the feature maps for image reconstruction. This expansive path contains convolutions, concatenation layers, and ends in a final output layer (usually softmax) that produces a vector of pixel predictions [2].

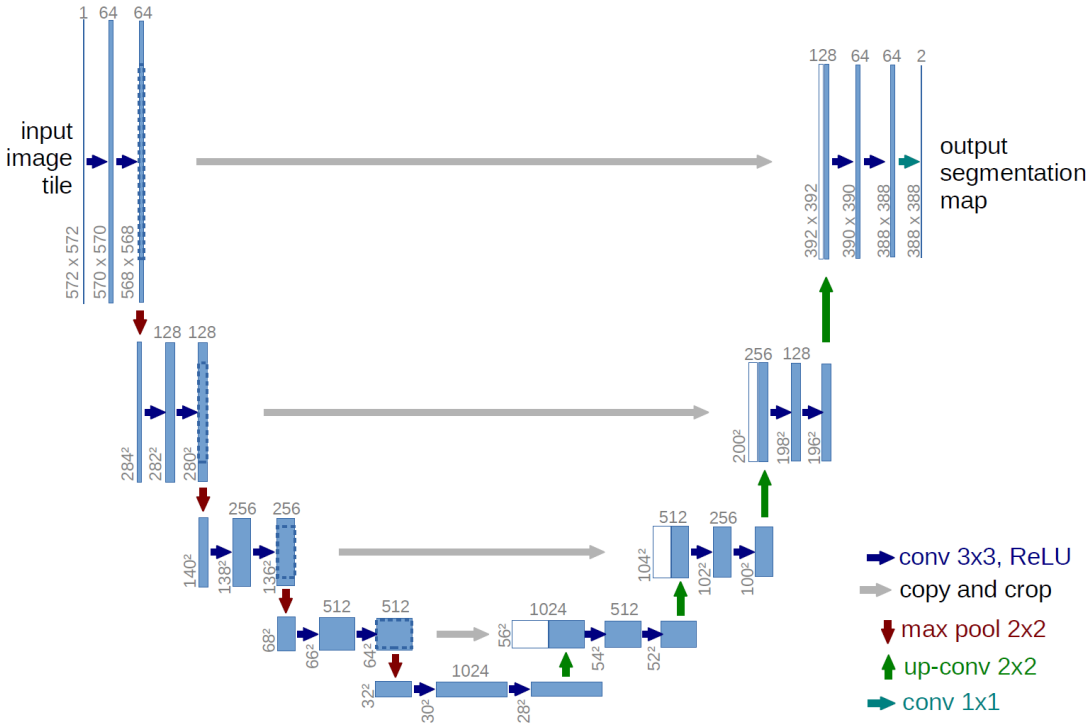

Figure 3: The U-Net configuration of a convolutional neural network. Downsampling occurs in the left-hand “contracting” path of the architecture, while upsampling of the imagery occurs in the right-hand “expansive” path. This figure was retrieved from U-Net: Convolutional Networks for Biomedical Image Segmentation on 11/26/2022 [32]. Used here for educational purposes.

Literature Review

The first paper which described a method to detect solar arrays with machine learning in aerial imagery was published by Malof et al. (2016). The authors used a Random Forest classifier with filtering and post-processing to label arrays in aerial images (0.3 m resolution) of Fresno, CA, and reported results for pixel-based and object-based detection methods [19]. Subsequently, Puttemans et al. (2016) tested four approaches for array classification, including a pixel-based method with a support vector machine classifier, and found that it tended to be less efficient when many

non-array pixels shared the color of pixel arrays [30]. They also found that object-detection techniques had the best performance, but they did not compare them to CNNs which they admitted could produce better results [29].

With further advances in deep learning, CNN's have become increasingly popular for array detection. For instance, Camilo et al. (2018) used a CNN with a SegNet configuration to perform semantic segmentation of PV arrays in aerial imagery [3]. Like Puttemans et al. (2016), they compared the performance of pixel vs. object-based methods, and found that SegNet outperformed older CNNs in both cases. In 2017, Goloko et al. used a CNN trained on a relatively small dataset to label arrays and returned an 87% accuracy rate [11]. Further, Castello et al. (2019) used a CNN with a U-Net configuration to detect rooftop solar panels and calculate their sizes with pixel-based image segmentation [4]. In aerial images of Switzerland they were able to achieve an Intersection-over-Union (IoU) score of 0.64 and an accuracy rate of 0.94 [4].

In 2020, Zech and Ranalli used a U-Net based CNN in Tensorflow enhanced with transfer learning to test several ResNet backbones and attained a range of values for precision (0.70-0.86) and recall (0.76-0.86) [41]. Projects using U-Net have continued to show promising results, with da Costa et al. (2021) finding that it had the best results among 4 tested architectures for segmenting arrays [7]. Parhar et al. (2022) also used a U-Net in combination with a small training dataset and EfficientNet-B7 classifier and were able to achieve an IoU of 0.86 and an F1 score of 0.92 [28].

Project Scope

To detect solar arrays and thereby estimate land use conversion, I built and trained a CNN model with a U-Net configuration. My method follows in a line of recent projects using a U-Net model to perform semantic segmentation of solar arrays (Castello et al. 2019, Zech and Ranalli 2020). However, it is unique in that it focuses on large ground-based arrays as opposed to rooftop panels and analyzes land-use conversion caused by array construction, which is a topic of growing interest [20]. The main outputs of this project are a trained model for detecting solar arrays, vector datasets of arrays, and this report which includes estimates of land-use conversion caused by array construction. Due to time and monetary constraints, I limited the analysis to four Massachusetts counties: Hampden, Hampshire, Plymouth, and Barnstable. They were chosen because they contain a relatively large amount of agricultural land and represent multiple regions of the state. The results of this project, including vector datasets of predicted solar arrays, were uploaded to a Github repository for public use.[2] In future work, I intend to expand the analysis to imagery from the entire state to generate a comprehensive prediction map.

Methods

In this section, I will describe the steps I took to detect solar arrays in aerial imagery and estimate land-use conversion caused by array installations. I will explain my rationale for taking these steps, describe the model I used, and specify areas in

[2] https://github.com/Robert173/solar_array_machine_learning

need of improvement. The workflow (see Figure 4) for this project includes three major stages: model creation, array prediction, and land-use estimation.

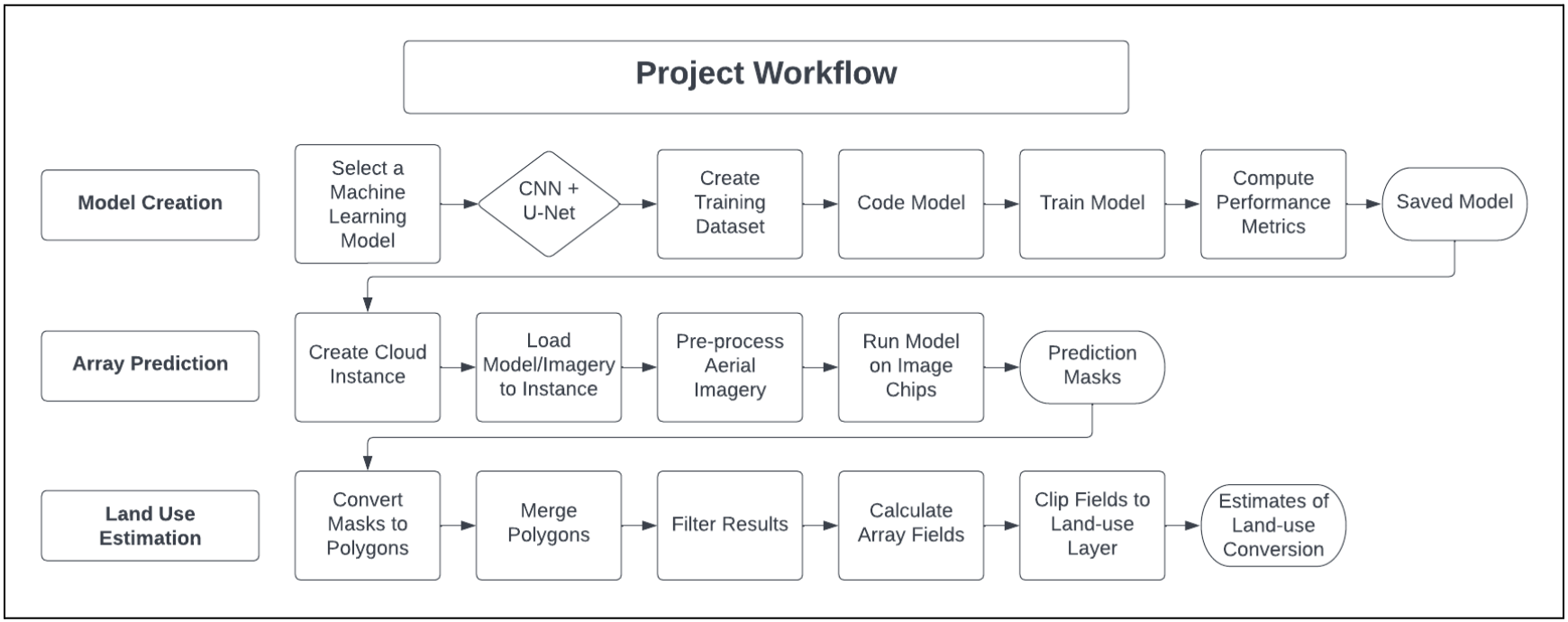

Figure 4: Project workflow, divided into three stages: 1. Model Creation, Array Prediction, and Land-Use Estimation.

Data Description

The images used in this study are from a 2021 orthoimagery dataset commissioned by MassGIS.[3] It includes 10,218 images covering 1500 × 1500 m areas [22]. These image "tiles" were produced at 15 cm resolution with 4 bands (RGB + near infrared). Although they were originally rendered as GeoTIFFs, MassGIS converted them to JPEG2000 format with low-loss compression [22]. For this project, I created image chips from these larger images for training and prediction tasks. They were rendered as GeoTIFFs because this format was the viable choice in ArcGIS Pro for the 4-band images and because it maintains image quality. It should be noted that GeoTIFFs have not been frequently used in array detection projects, probably because of their high disc space requirements and consequent effects on processing speed. However, such high-quality imagery with an extra band likely offers the potential for more accurate training and labeling. A sample image is shown in Figure 5.

Other data I acquired from MassGIS include a counties layer to select images within counties and a 2005 land use layer of Massachusetts.[4] It was produced with semi-automated methods and features a minimum mapping unit of 1 acre (as low as ¼ acre in some cases) [23]. This land use layer was selected because it is a high quality dataset produced for Massachusetts made in a year prior to large-scale array construction, which began in 2008.[5] In a future time series analysis, where installation

[3] 2021 aerial imagery: https://www.mass.gov/info-details/massgis-data-2021-aerial-imagery
[4] Counties layer: https://www.mass.gov/info-details/massgis-data-counties
Land Use (2005) dataset: https://www.mass.gov/info-details/massgis-data-land-use-2005
[5] The layer is based on 0.5 m resolution aerial imagery.

date ranges can be more reliably determined, land-use layers from years closer to the dates of installation might be employed to get an even more accurate perspective of land-use at installation sites.

Figure 5: Sample 2021 aerial image of Massachusetts showing solar arrays

Create a Training Dataset

Machine learning models require a curated dataset of example images to learn how to differentiate classes. They must be sufficiently representative of the classes so that the model can distinguish them in a variety of instances. Thus, they are usually large, consisting of tens of thousands of images. They must also be proportional with enough instances of target classes (in this case, arrays) compared to background instances so that the model can properly learn their features. To speed up this process, some models are pre-trained on generic datasets such as ImageNet to establish baseline model weights and are later run on user-specific imagery. For this project, I trained the model on a dataset of 15,198 image chips without any pre-training.

To create these image chips, I used the Label Objects for Deep Learning tool in ArcGIS Pro. I first selected images from the 2021 dataset which had representative features of both classes, solar arrays and background. Areas in each image were delineated with bounding boxes to target arrays and key features which could generate false positives such as rooftops, parking lot lines, tree shadows, and shimmering water. Within these areas, I used the tool to label each array row as an instance of the array class (1); all other pixels were labeled automatically as background (0). After this, I used the tool to create image chips from the bounding box areas along with masks showing the class of each pixel in the chips. The standard size for these chips is 256 × 256, which has been shown to be optimal for machine learning tasks [33]. Examples of image chips and masks are shown in Figure 6.

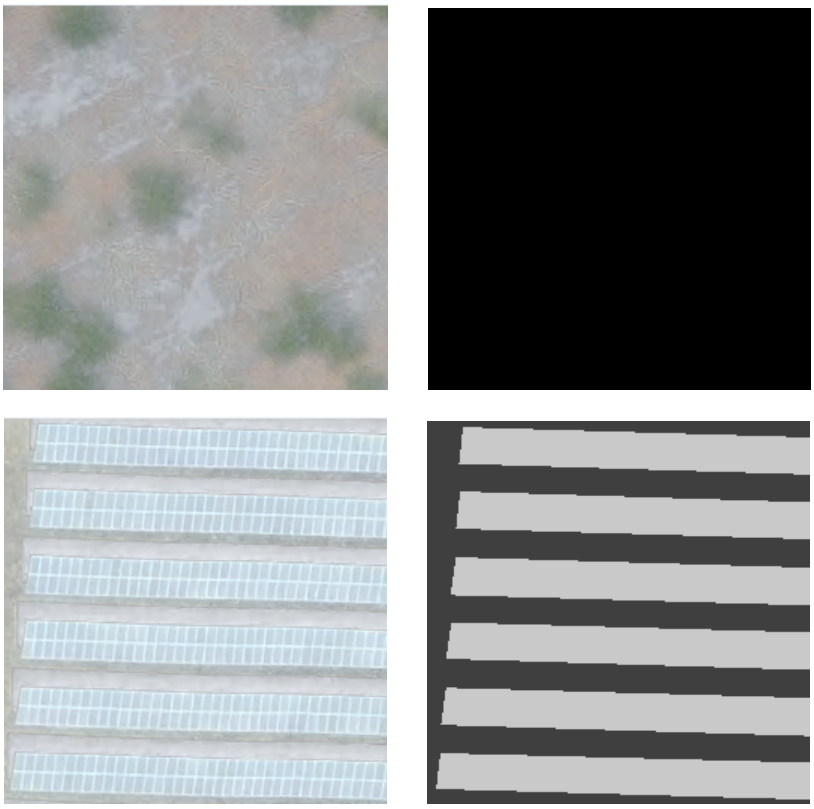

Figure 6: Image chips (left) and their corresponding masks (right). Array pixels (white) are distinguished from the black background.

Reserve Validation Dataset

A portion of the training dataset is not used for actual training but is set aside to refine model weights and evaluate performance at the end of each training stage. Ideally, this reserve data set would be produced by selecting image chips at random. However, because the training data class proportion is unbalanced (high number of background vs. array chips) I could not ensure enough chips with arrays would be included to meaningfully validate the model if I used this method. Thus, I reserved several images at random from the set of images I used to create the training data and used the image chips generated from them as the validation dataset. In future model

runs, I will create a new validation dataset through a random selection of images from training or by K-fold cross validation, where training is conducted on image subsets [1].

Build Machine Learning Model

The model that I used was encoded in Python 2.9 with the Tensorflow and Keras packages.[6, 7] Tensorflow is a popular package from Google that allows users to implement and customize machine learning tasks. On top of Tensorflow, an extensible API called Keras was developed that enables more user-friendly access to machine learning. Both are widely used in the machine learning community but are slowly being replaced by other packages such as pytorch which offer more intuitive and simpler coding.

Some of the foundational code to augment the training dataset, implement the U-Net, and run callbacks were adapted from other sources, most especially the "Semantic Segmentation" tutorial posted by Yann Le Guilly on Gitlab, which has since been taken off the site [17]. However, I adapted and wrote original code for many of the model tasks, such as visualizing sample predictions and reading and decoding the GeoTIFF training images.

In brief, the first step in the model code is reading the GeoTIFF imagery and masks and converting them to tensors. For some image formats such as jpeg this is a simple operation, but Tensorflow does not provide comprehensive support for

[6] Download Tensorflow at https://www.tensorflow.org/install.
[7] Learn about Keras at https://keras.io/getting_started/.

GeoTIFFs. The Tensorflow experimental package contains a function to decode 4-band GeoTIFFs but not one for single band TIFFs such as the training masks. To decode the masks, I had to write a specialized function using the tf.py_function wrapper, since Tensorflow does not integrate with “inefficient” functions of other modules. This allowed me to open the mask with the PIL module, convert it to an array, and then convert the array to a tensor.[8]

Once the images and masks were converted to tensors, they were listed with the list_files method and transformed with the tf.map() function. I then used functions to load, shuffle and normalize the images (dividing by 255 so all values would be on a 0-1 scale). At this time, several hyperparameters were also set for the model.[9] These include the batch size (subset of data run in one forward pass, usually a multiple of 8), learning rate (control changes to weights based on the loss), and dropout rate (sets fraction rate for training updates, helps prevent overfitting) [12, 31, 35]. After this, I added the full code for the CNN model with a U-Net configuration, which is shown in Figure 7.

[8] Tensors are generalized, multi-dimensional arrays [2].
[9] The term hyperparameter denotes a fixed setting for the learning process.

```python
# model parameters
dropout_rate = 0.5
input_size = (256, 256, 4)
N_CLASSES = 2


initializer = 'he_normal'


# U-Net Model

# -- Encoder -- #
# Block encoder 1
inputs = Input(shape=input_size)
conv_enc_1 = Conv2D(64, 3, activation='relu', padding='same', kernel_initializer=initializer)(inputs)
conv_enc_1 = Conv2D(64, 3, activation = 'relu', padding='same', kernel_initializer=initializer)(conv_enc_1)

# Block encoder 2
max_pool_enc_2 = MaxPooling2D(pool_size=(2, 2))(conv_enc_1)
conv_enc_2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(max_pool_enc_2)
conv_enc_2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(conv_enc_2)

# Block  encoder 3
max_pool_enc_3 = MaxPooling2D(pool_size=(2, 2))(conv_enc_2)
conv_enc_3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(max_pool_enc_3)
conv_enc_3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(conv_enc_3)

# Block  encoder 4
max_pool_enc_4 = MaxPooling2D(pool_size=(2, 2))(conv_enc_3)
conv_enc_4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(max_pool_enc_4)
conv_enc_4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(conv_enc_4)
# -- Encoder -- #

# ----------- #
maxpool = MaxPooling2D(pool_size=(2, 2))(conv_enc_4)
conv = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(maxpool)
conv = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(conv)
# ----------- #
```

```
    # -- Decoder -- #
    # Block decoder 1
    up_dec_1 = Conv2D(512, 2, activation = 'relu', padding = 'same', kernel_initializer = initializer)(UpSampling2D(size = (2,2))(conv))
    merge_dec_1 = concatenate([conv_enc_4, up_dec_1], axis = 3)
    conv_dec_1 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(merge_dec_1)
    conv_dec_1 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(conv_dec_1)

    # Block decoder 2
    up_dec_2 = Conv2D(256, 2, activation = 'relu', padding = 'same', kernel_initializer = initializer)(UpSampling2D(size = (2,2))(conv_dec_1))
    merge_dec_2 = concatenate([conv_enc_3, up_dec_2], axis = 3)
    conv_dec_2 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(merge_dec_2)
    conv_dec_2 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(conv_dec_2)

    # Block decoder 3
    up_dec_3 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_initializer = initializer)(UpSampling2D(size = (2,2))(conv_dec_2))
    merge_dec_3 = concatenate([conv_enc_2, up_dec_3], axis = 3)
    conv_dec_3 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(merge_dec_3)
    conv_dec_3 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(conv_dec_3)

    # Block decoder 4
    up_dec_4 = Conv2D(64, 2, activation = 'relu', padding = 'same', kernel_initializer = initializer)(UpSampling2D(size = (2,2))(conv_dec_3))
    merge_dec_4 = concatenate([conv_enc_1, up_dec_4], axis = 3)
    conv_dec_4 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(merge_dec_4)
    conv_dec_4 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(conv_dec_4)
    conv_dec_4 = Conv2D(2, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(conv_dec_4)
    # -- Decoder -- #

    output = Conv2D(N_CLASSES, 1, activation = 'softmax')(conv_dec_4)

```

Figure 7: Code for U-Net structure. Includes encoder (downsampling), decoder (upsampling), and output (softmax). Several convolution layers (Conv2d) with rectified linear unit (ReLU) activation neurons are used to extract image features [2].

After this, a callback function was created that enabled model checkpoints and training times to be saved and the training time to be calculated. The model was instantiated with tf.Keras.model and the loss (SparseCategoricalCrossEntropy), optimizer (Adam), and metrics (accuracy) were established as variables. The learning rate, which controls the model's response with regard to error, was set at 0.0001 [42].[10] These hyperparameters were compiled with the model, which was passed to model.fit().

Train Model

On my home computer, the model requires approximately 3 ½ days (8hrs + for each epoch over 10 epochs) of constant processing to finish training.[11] To be more efficient, I transferred the model code to an AWS cloud instance and downloaded the training imagery and aerial imagery of Massachusetts to that instance. I was unable to leverage GPU acceleration for training on the instance because a bug would cause the model to crash after a few epochs. I therefore trained the model with the instance cpu and saved the best version as a .tf file for later use. In this case, the best version is the model saved at the end of the epoch with the best value for the monitoring metric, which was accuracy. In future iterations, I will consider using a different and perhaps more informative monitoring metric. Figure 8 shows the final code to compile and fit the model.

[10] This means that the model makes relatively small weight changes at each update.
[11] My home computer is an HP laptop running Windows 10. The processor is an 11th Gen Intel Core i5-1135G7 with a base clock speed of 2.42 GHz and 8 logical processors. It has 8 GB of installed RAM.

```
EPOCHS = 10

model = tf.keras.Model(inputs = inputs, outputs = output)

loss = tf.keras.losses.SparseCategoricalCrossentropy()
optimizer = tf.keras.optimizers.Adam(learning_rate = 0.0001)

model.compile(optimizer = optimizer, loss = loss, metrics = ['accuracy', 'MeanSquaredError'])

with tf.device("/cpu:0"):
    model_history = model.fit(dataset["train"], epochs = EPOCHS,
                              steps_per_epoch = STEPS_PER_EPOCH,
                              validation_steps = VALIDATION_STEPS,
                              validation_data = dataset['val'],
                              callbacks = callbacks)
```

Figure 8: Final code section to compile and fit the model, showing loss, optimizer, and metric selections.

Compute Performance Metrics

During the model run, accuracy was calculated after each batch.[12] However, accuracy is not very useful as a metric since the data is highly class imbalanced; it reflects the high number of true positive background predictions, rather than the smaller number of more relevant array true positives [15]. Another metric which was calculated was loss which was determined by the loss function [8]. It measures how well the machine learning algorithm models the data. Other metrics which I attempted to calculate during training include precision, recall, and the F1 score (dice coefficient), but the configuration that I chose does not natively support them. Therefore, I had to calculate them separately by predicting on the validation imagery with the finished model, comparing the output to ground-truth rasters, and computing the metrics manually. These metrics are described in Figure 9:

[12] Specifically, sparse categorical accuracy, which pairs with the sparse categorical cross-entropy loss function.

Performance Metrics for Machine Learning		
Metric	*Description*	*Equation*
Pixel Accuracy	The percent of pixels that are labeled correctly	$\frac{TP + TN}{TP + FN + TN + FP}$
Precision	Ratio of true positives to all predicted positives; measures error due to false positives	$\frac{TP}{TP + FP}$
Recall	Ratio of true positives to actual positives (ground-truth); measures error due to false negatives	$\frac{TP}{TP + FN}$
F1 Score	The harmonic mean of precision and recall; especially useful for imbalanced classes	$2 \times \frac{Precision \times Recall}{Precision + Recall}$

Figure 9: Common metrics calculated in machine learning [37].

For the validation imagery, I calculated the three metrics for output that was filtered and not-filtered of false positives (see “Post-Processing” section for details on filtering). I also calculated performance metrics for an optimized version of the model (v. 2) run on a larger training dataset. This version has higher precision (see Figure 10), implying that less false positive area was predicted, but it also has lower recall, in which more false negatives were produced. Since model v.2 was produced later in the project, I used v.1 to produce most of my output. The results for both model versions are shown below in Figure 10:

Model Version 1			
	Precision	Recall	F1 Score
Filtered	0.88	0.89	0.88
Not-filtered	0.87	0.89	0.88

Model Version 2			
	Precision	Recall	F1 Score
Filtered	0.94	0.87	0.90
Not-filtered	0.92	0.87	0.89

Figure 10: Performance metrics for model v. 1 and 2.

It should be noted that the validation data for both models was not produced by an entirely random process and represented a relatively small area. In future model runs, I will create the validation dataset through cross validation, wherein pixels are randomly selected from validation imagery and predicted on.

Prepare Imagery for Model

The model only accepts images with the same size as the images used for training. Therefore, I had to split every aerial image tile (10,000 × 10,000 pixels) that I downloaded into 256×256 chips. I accomplished this with the Split Raster (Data Management) tool in ArcGIS Pro. To ensure complete coverage of the image with same sized chips, I set an overlap of 24 pixels for the splitting. One of the benefits of this was that over a third of the area of each chip (except edge chips) were overlapped and predicted on multiple times. This produced a range of pixel values [0, 63, 85, 127, 170,

191, 255] that represent various levels of overlap and prediction; for instance, a value of 170 represents 3 chip overlaps with 2 positive array predictions. From the output, I selected only the high confidence pixels [>170] that were predominantly labeled as arrays.

Run Model on Images

The saved model was loaded in Python using the load_model function in Tensorflow. After this, image chips were read and decoded using Tensorflow functions. To do this, I first had to expand the image dimensions, since the model expects a dimension containing the batch number when reading imagery. The images were then fed to the model which calculated the pixel-wise class probabilities. Using the Tensorflow argmax function, I selected the class with the highest probability for each pixel. I then converted the array of pixel predictions to image “masks” using the keras preprocessing array_to_image function. At this stage, I also copied the world file[13] of the original chip to the folder containing the masks so that each mask would have a spatial reference for later merging and analysis. Figure 11 shows image and mask samples.

[13] World files (.tfw) contain information about the location, scale, and rotation of raster imagery.

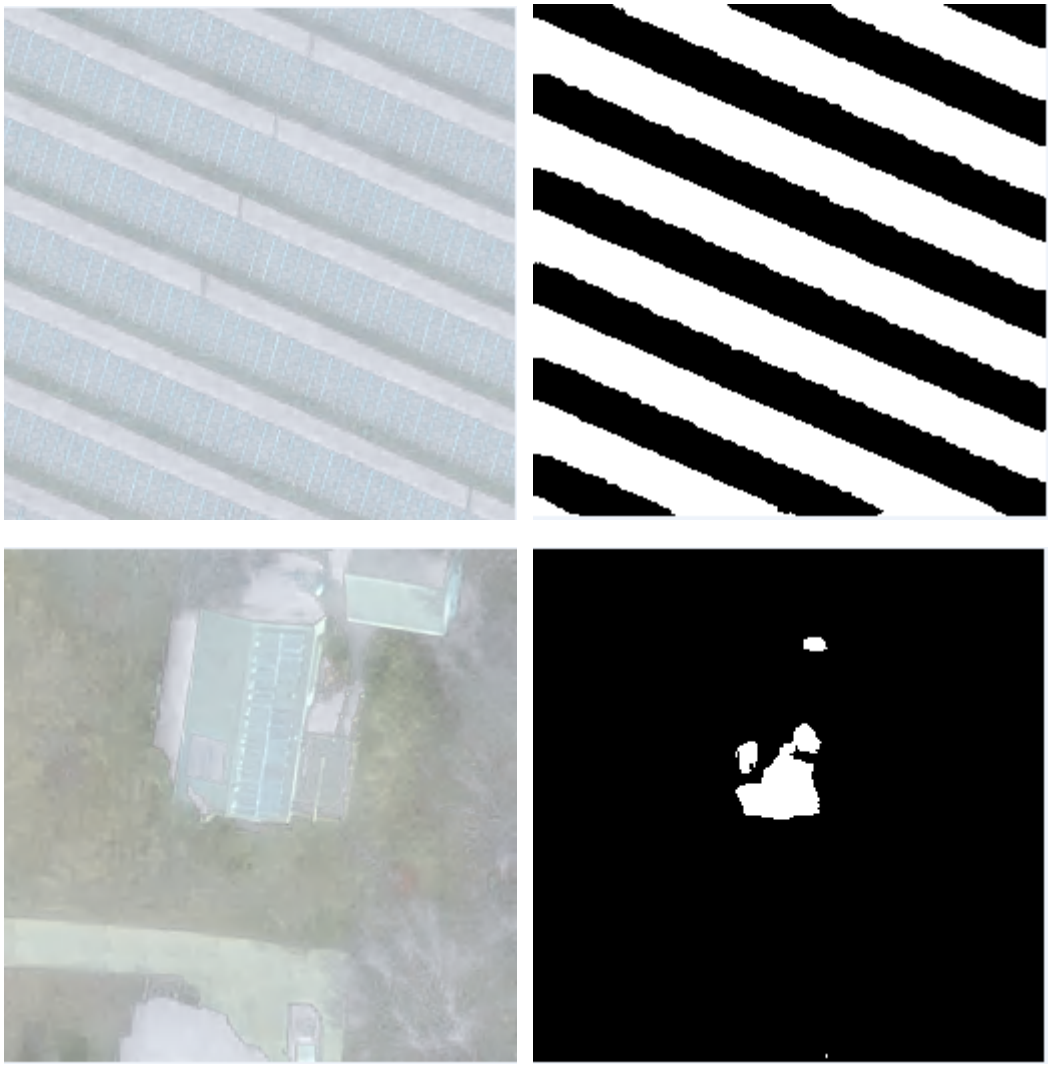

Figure 11: Image chips (left) and their corresponding masks predicted by the model. Array recall is generally high, but a number of false positives are generated by the model.

Post-Processing

After the masks were produced, they were added to a list if their maximum pixel value was greater than zero, which excluded masks lacking array pixels. The individual chip-size masks were then combined into whole image-sized outputs using the Mosaic to New Raster tool in ArcGIS Pro. High confidence pixels [>170] were extracted and

saved to new rasters. These were then converted to vector format using the Raster to Polygon tool. Though I also wrote code to produce aggregate raster outputs, I primarily generated polygons since I needed them to estimate array fields. Converting the masks to polygons also allowed me to conduct another essential step in the process, filtering.

An ideal model would produce a sufficiently accurate output that could stand alone, however the model I used predicted a significant number of small false-positive pixels. I therefore had to impose filtering rules on the output to better isolate true solar arrays. To do this, I exploited some of the common attributes of the false positives (FPs), including their typical size, shape, and degree of separation. In general, FP's were small, irregularly shaped, and spread out (see Figure 11), so I filtered out polygons that were smaller than 10 sq.m., that had a relatively high perimeter area ratio (>3), or were greater than 10 m from their nearest neighbor. By removing these polygons, the resulting masks had a greater proportion of true arrays and could be used to produce feasible estimates of array fields and land-use conversion. The metrics calculated for filtered and non-filtered outputs suggest that the non-filtered FPs impact the results less than they appear to, but in aggregate they would likely skew the calculations for array fields. Figure 12 shows a prediction map with a number of false positives (left) and the results after filtering false positives (right).

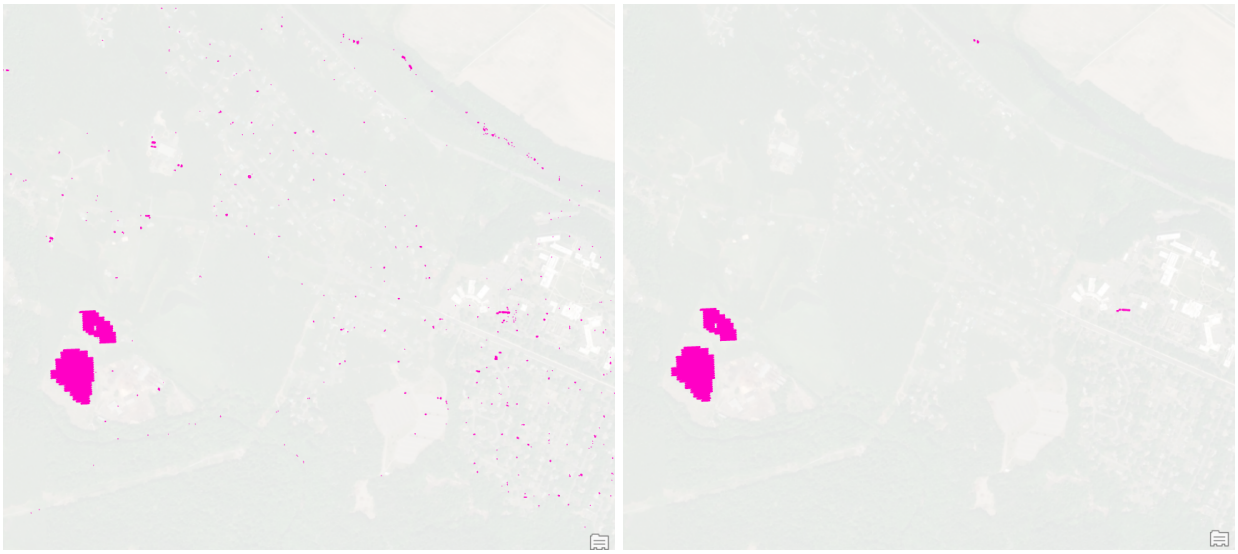

Figure 12: Predictions map of a developed area (left) and the filtered shapefile (right).

Image Processing in the Cloud

One of the issues I encountered was the time it took to run the model on new imagery. On my home PC with CPU,[14] it took about 20 minutes to predict all the image chips of a single image tile. It would therefore be impractical to analyze entire counties, which could consist of 800-1000 tiles, on this machine. Therefore, I downloaded the imagery tiles to an AWS instance [g4dn.2xlarge] and enabled a Tesla T4 GPU with a compute capability of 7.5 to increase performance. GPUs are advantageous in such cases since they support parallel processing and handle images rapidly [18]. With this configuration, I was able to predict on image tiles in under 3 minutes.

[14] My home computer has an Intel Iris Xe Graphics GPU, however this is not compatible with the NVIDIA cuda toolkit needed to harness GPU computation in Tensorflow.

Estimating Array Fields and Land Use Conversion

Once the array polygons were produced I calculated the total area that was converted to array fields. In this context, “array fields” refer to areas occupied by arrays, the gaps in between them, and some surrounding areas. They are an estimate of the land cleared for installation, and are a better reflection of land-use conversion than panels alone. To derive them, I first grouped the solar array polygons by distance using the Group By tool in ArcGIS Pro. Next, I created convex hull polygons[15] with the Minimum Bounding Geometry tool for each array group (see Figure 13). In some cases, these polygons extended outside the cleared areas, but in general they closely approximated the installations.

Figure 13: An array “field”, which is the smallest convex polygon that can be drawn around an input [10].

[15] A convex hull polygon is a minimum bounding area that contains all vertices of the input.

These array fields were then used to clip a 2005 land-use layer to determine what the previous land-use was at array sites. Though it cannot be said with certainty that the immediate land-use prior to construction is the land-use in the dataset, it is highly likely in most cases, especially for uses such as agriculture that change little over long periods. The resulting land-use polygons were grouped by category and analyzed for each county. Figure 14 illustrates the converted land uses of two solar array groups:

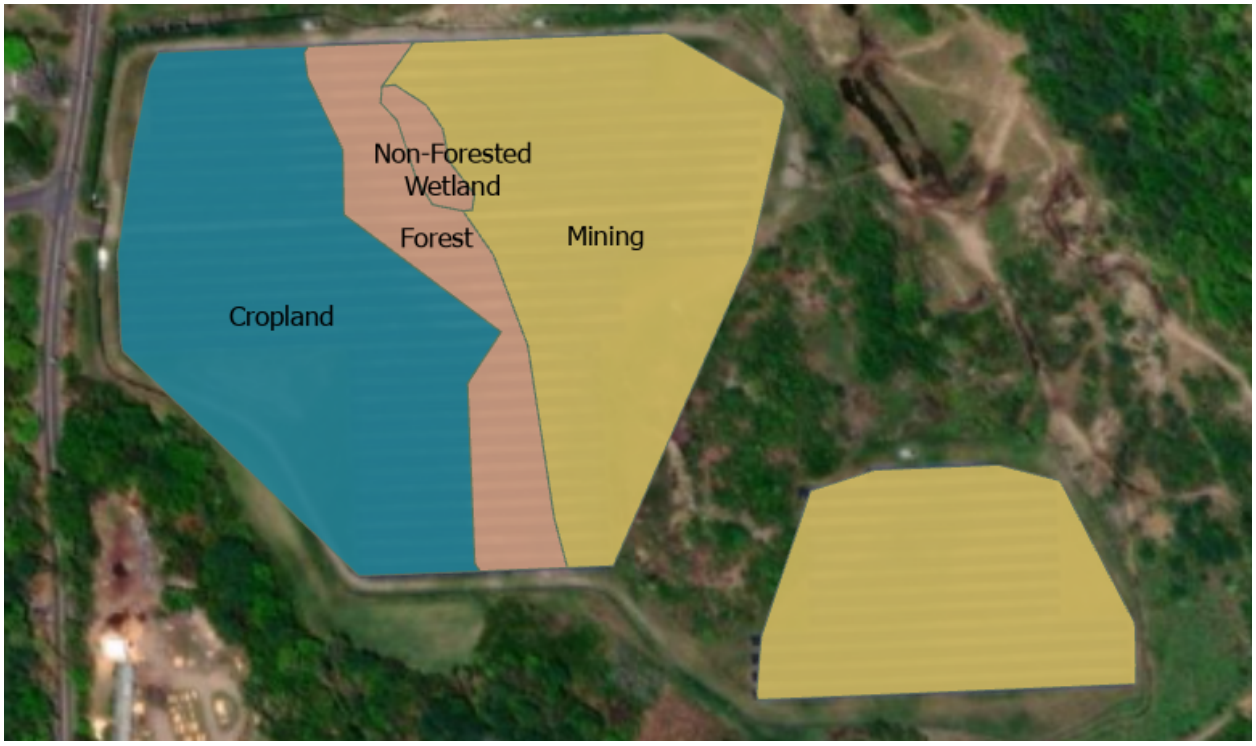

Figure 14: These solar array groups replaced several land use types, including cropland, forest, and what appears to be an abandoned mining complex.

Results

Barnstable County

In Barnstable County the land use types with the greatest acreage of overlap were open land [100.4], industrial/commercial/transportation [45.8], and forest [42.9]. The findings for the industrial/commercial category suggest that a large proportion of the arrays were on rooftops, although some of this can be ascribed to the area overlapped by arrays at Barnstable Municipal Airport which is classified as “transportation.” These arrays are circled in Figure 15. Only 1.2 acres of agricultural overlap was detected, which may be partly due to the relatively small number of acres farmed in the county [25]. Figure 16 shows the proportional occurrence of each land use type:

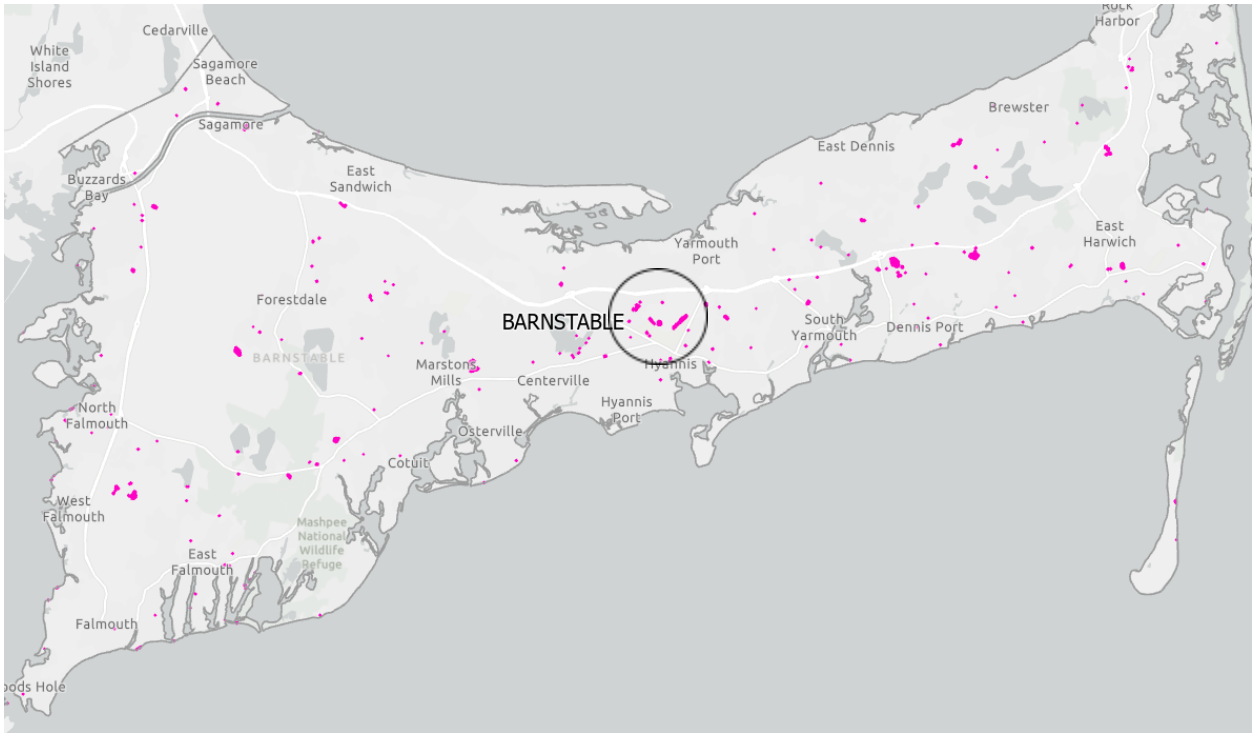

Figure 15: Prediction map for Barnstable County, MA. The arrays at Barnstable Municipal Airport are circled.

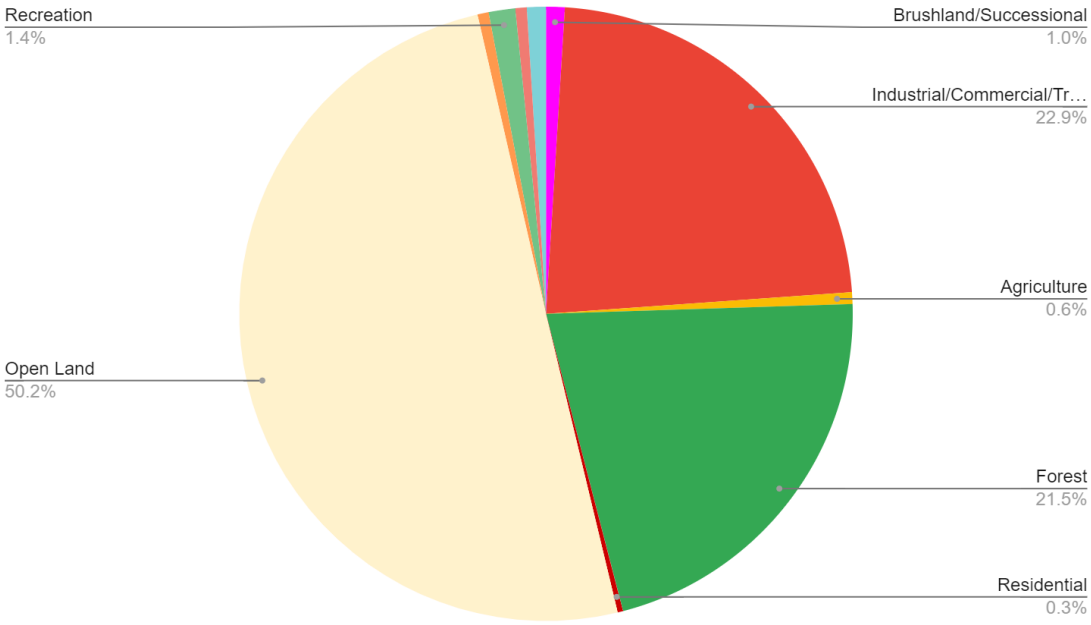

Figure 16: Percentage of each land use overlapped by solar arrays in Barnstable County

Plymouth County

In Plymouth County, the land use types with the greatest acreage overlap were forest [369.5], open land [142.9], and agriculture [93.4]. The areas of converted forest occurred primarily in the southern portion of the county. Cranberry bogs, which are characteristic of the area, account for 37% of the total agricultural overlap, with the rest being cropland, pasture, and nursery. Also, 38.2 acres of land in the junkyard/waste disposal category was occupied. The prediction map includes a noticeable number of false positives (see Figure 17), however some of these may also be rooftop panels that were not filtered. Figure 18 shows the proportional occurrence of each land use type:

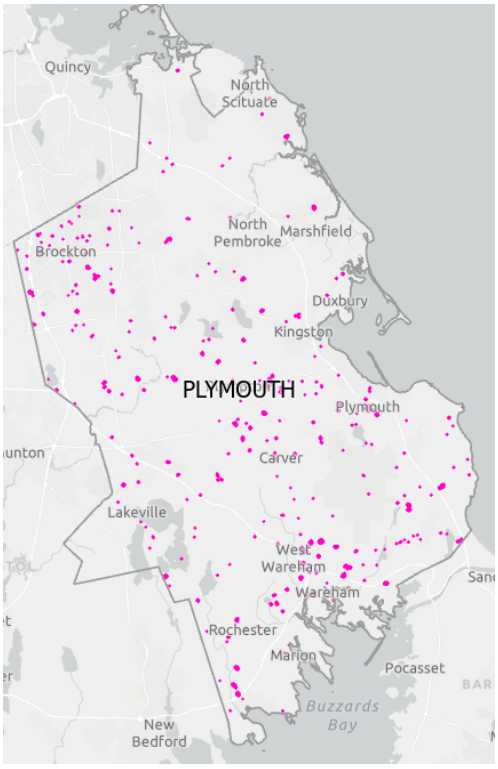

Figure 17: Prediction map for Plymouth County, MA

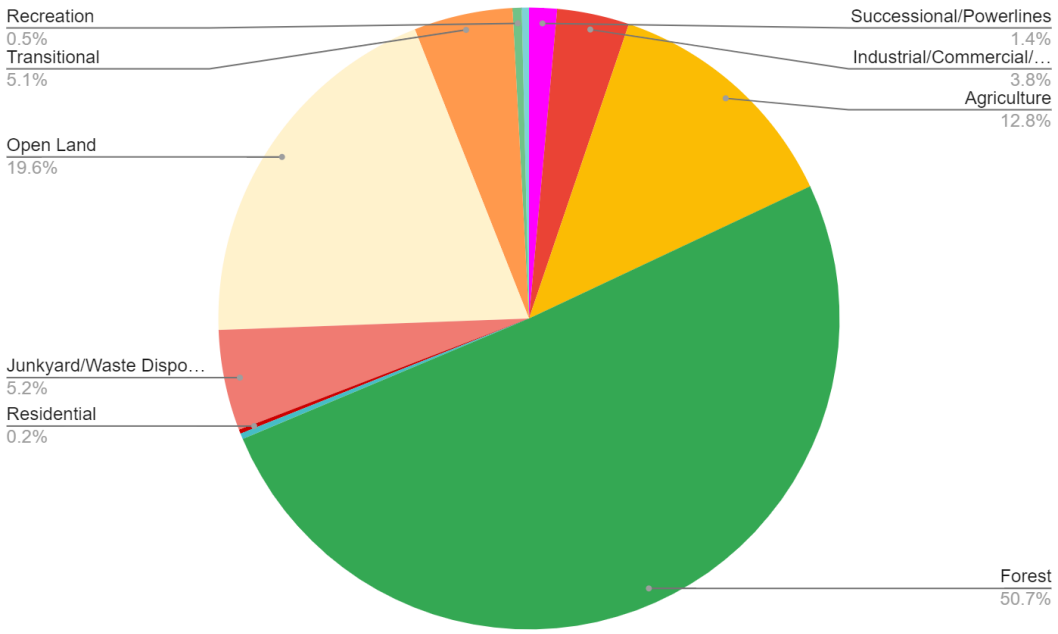

Figure 18: Percentage of each land use overlapped by solar arrays in Plymouth County

Hampshire County

In Hampshire County, the land use types with the greatest overlap were agriculture [147.7], forest [108.9], and water [71.0]. This county had the highest proportion of agricultural land overlap [32.2%] of the four analyzed. Areas of agricultural overlap predominated in the Connecticut River Valley where farming is most prevalent. Water generated a noticeable number of false positives in both the Westfield River and Quabbin Reservoir (see Figure 19). Also, the developed areas of cities such as Amherst tended to produce false positives that were not filtered out. Figure 20 shows the proportional occurrence of each land use type:

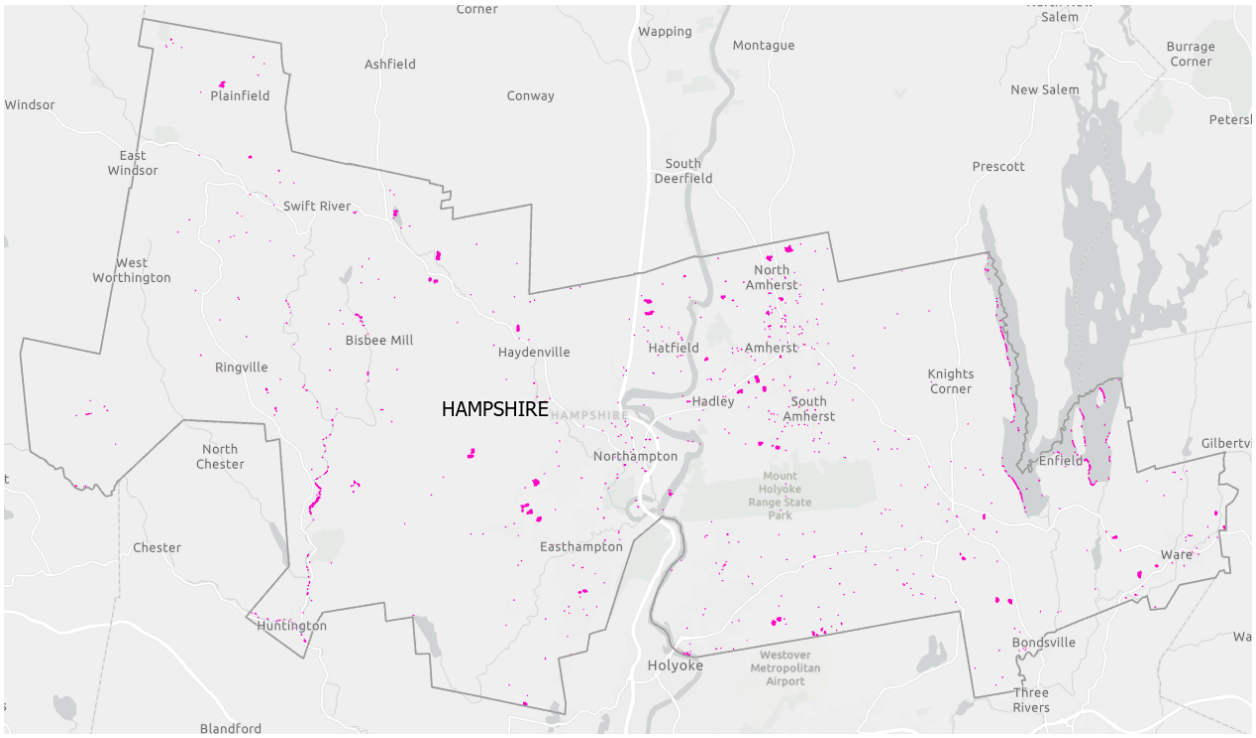

Figure 19: Prediction map for Hampshire County, MA

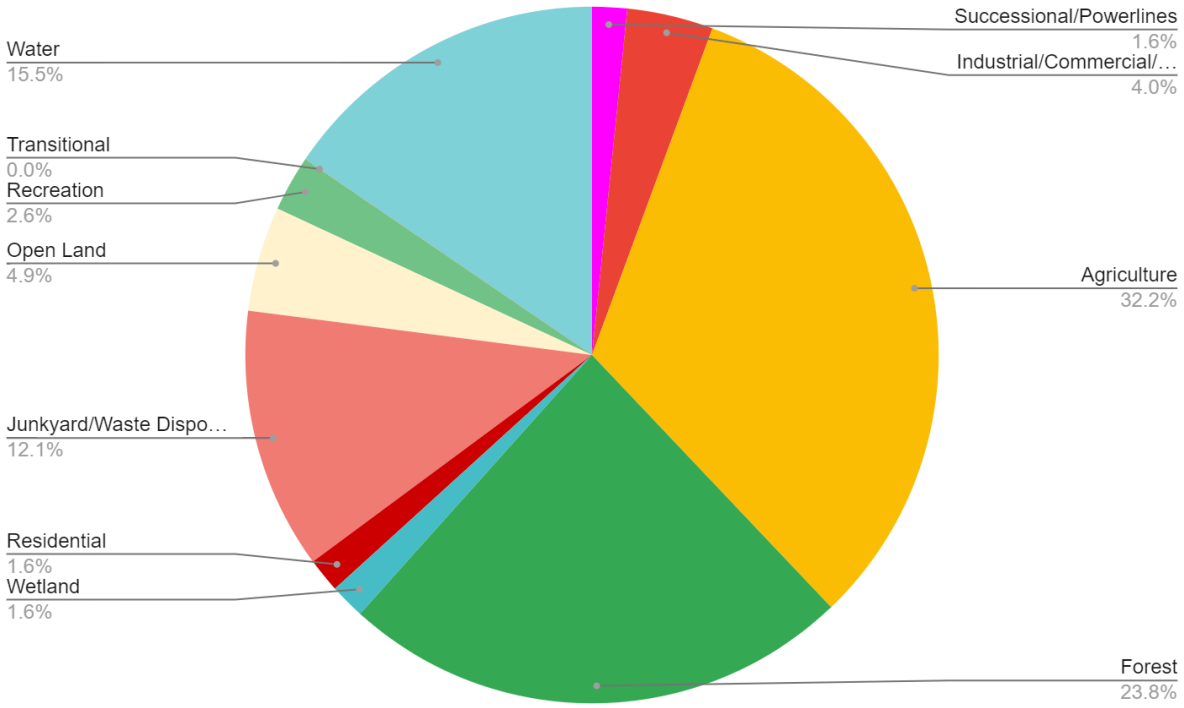

Figure 20: Percentage of each land use overlapped by solar arrays in Hampshire County

Hampden County

In Hampden County, the land use types with the greatest overlap were forest [351.1], agriculture [248.3], and open land [95.2]. Solar arrays again predominated in the densely populated river valley and eastern half of the county (see Figure 21). Hampden produced the largest total area labeled as array [946.1], though a portion of this is accounted for by false positives of water and developed areas.

The imagery was subsequently run on a newer model (v. 2) which predicted 850.5 acres labeled as array with slightly less but similar figures for the three prevalent land use types (see “Compute Performance Metrics” section). The new model largely

excluded false positives generated by water (see Figure 23, 24). Due to time constraints I was not able to run model v.2 on other counties but I intend to do so with an updated version in the future. Figure 22 shows the proportional occurrence of each land use type:

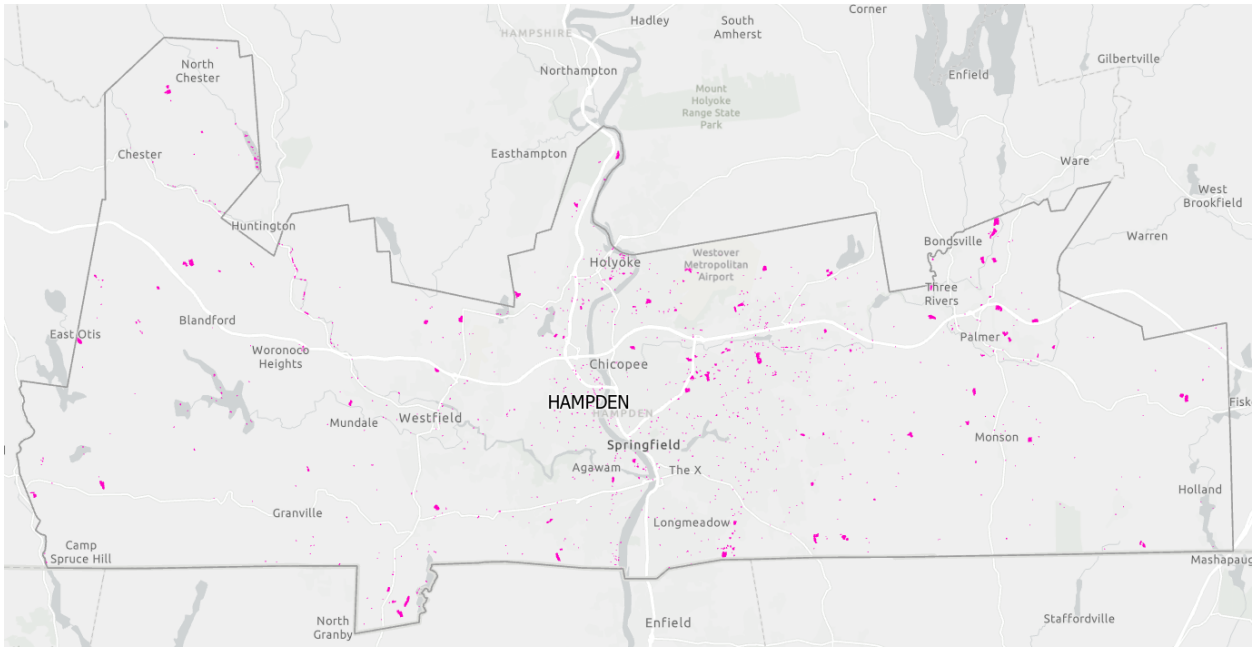

Figure 21: Prediction map for Hampden County, MA.

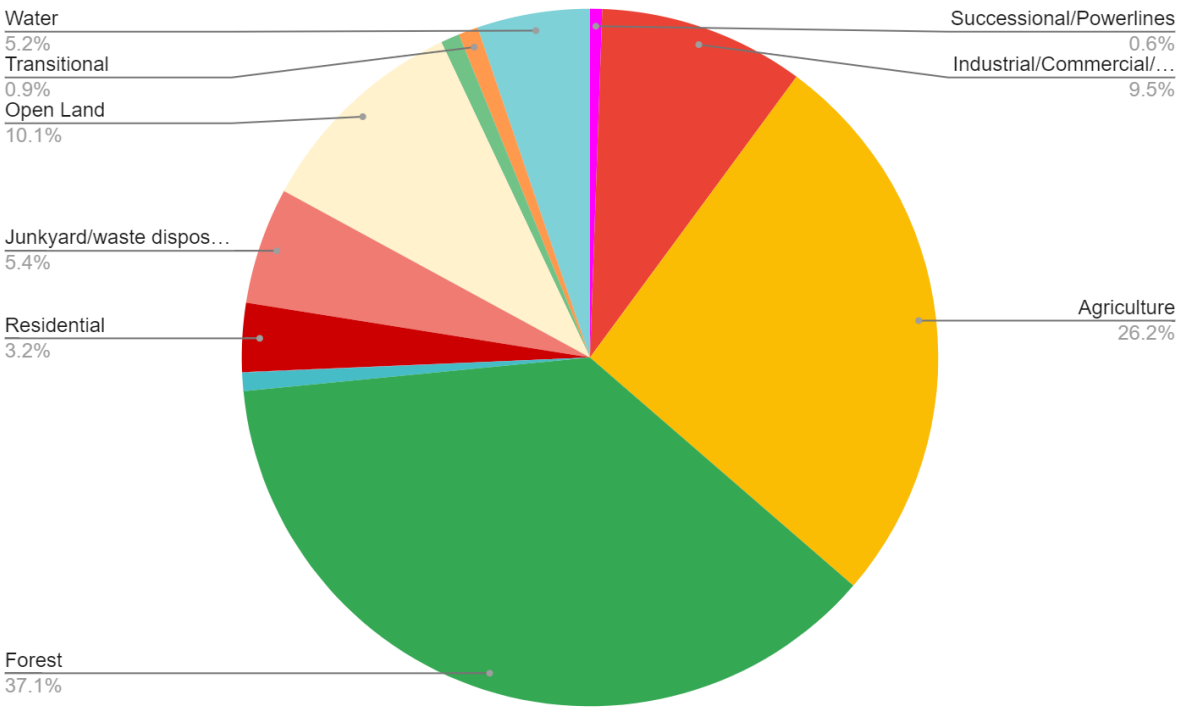

Figure 22: Proportion of each land use type overlapped by solar arrays in Hampden County.

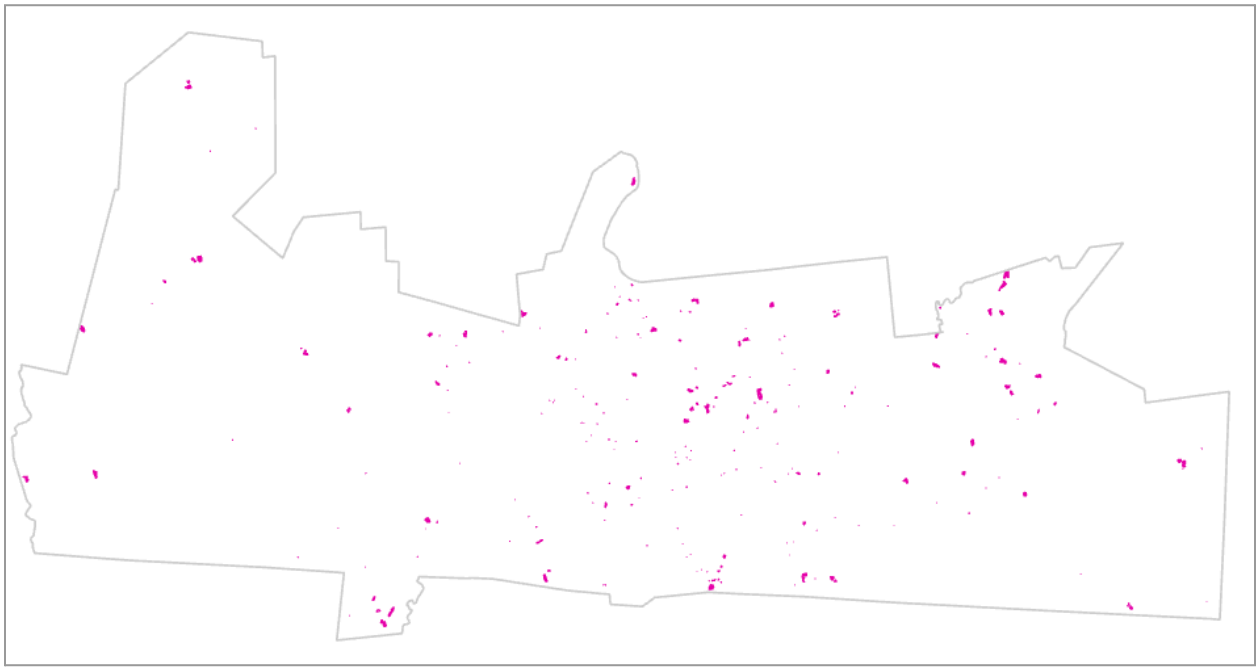

Figure 23: Prediction map for Hampden County using model v. 2 with filtering.

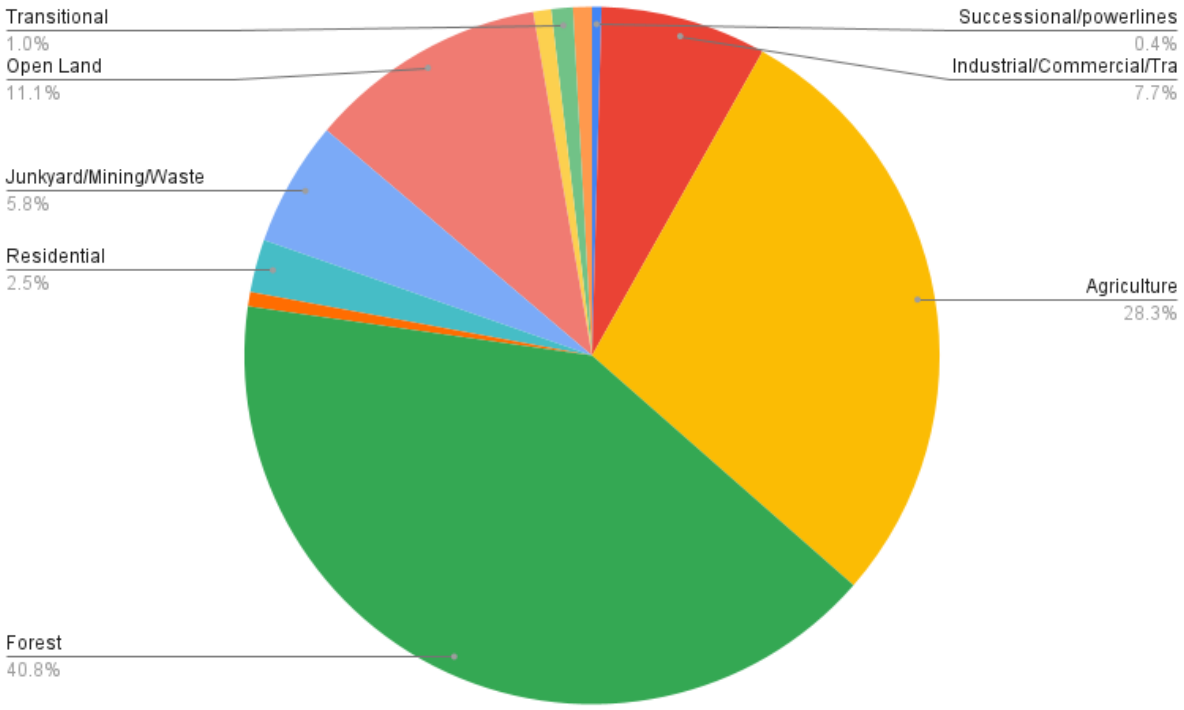

Figure 24: Proportion of each land use type overlapped by solar arrays in Hampden County.

Discussion

Hampshire and Hampden counties had the largest areas of former agricultural land overlapped by solar arrays [248.3, 147.7]. While Plymouth County contains more total agricultural land, it had less overlap [93.3] [25]. These differences may be due to a number of factors, including local zoning policy and receptivity to solar development, which invite further investigation. Barnstable County had noticeably little array overlap of farmland, though it has less agricultural land overall [25]. It should be stated that land-use figures such as these are derived from estimations of array fields, and that in some cases they over or underestimate the actual land cleared for installation. In future

work, a more precise bounding method, coupled with manual editing, could provide more accurate accounts of array fields and land-use conversion.

The capability of the model to accurately predict ground-based array pixels is constrained by several factors. For one, the model sees individual image chips but does not observe the context from which they were extracted. This limits its ability to learn array backgrounds, which is important for distinguishing rooftop arrays from ground-based arrays. A potential way of mitigating this is to use a larger chip size, such as 512×512, or even passing whole images to the model, but this would require a considerable amount of computer memory.[16] Another difficulty is that the imagery data is highly class imbalanced. Though the training dataset includes a large proportion of array pixels for proper training, solar arrays are a relatively rare occurrence in actual images. Due to this, the model may be biased to label a greater number of pixels as array than would be expected for a given image. Finally, the model will necessarily generate some errors due to imperfect learning, however these are usually small and can be easily filtered from the final output.

These factors might be mitigated by refining the model and further optimizing the training dataset. The loss function which was used, sparse categorical cross entropy, is suitable for the classification task, but might be less effective than a more discriminating function such as focal sparse crossentropy. Also, comparing runs with different learning rates or training durations could reveal better combinations that more accurately model the data. This would be informative if performance metrics were calculated at the end of

[16] Training images must have dimensions that are multiples of 32.

each epoch, however this is difficult to implement with the current configuration. In the future, I intend to write a custom callback that calculates metrics and evaluates computational performance while the model is training.

Conclusion

Summary Results and Outlook

Overall, 490.6 acres of agricultural land were found to be overlapped by solar array fields and about 80% of this overlap occurred in the two western counties.[17] Also, agriculture was among the top three converted land-use types in 3 of the counties (see Figure 25). The predicted area of array fields by no means represents a large percentage of the total cropland in Massachusetts. However, since a large proportion of array fields overlap agricultural land, it does suggest that more farmland could be vulnerable to development, especially as the state presses forward with its renewable energy initiative.[18] Ultimately, the growth of new solar installations will be determined by the availability of land going forward, and it is likely that arrays will continue to be installed but at a slower pace than during the 2012-2018 period. Given that solar arrays can cover many acres and remain on the landscape for decades, it is imperative that their development be monitored and balanced against losses of ecosystem services, natural habitat, and landscape value.

On the completion of this project, I uploaded summary results for counties on Github for public use. In the future, I intend to add some of my code base, prediction

[17] To put this in perspective, the average size of a Massachusetts farm in 2017 was about 68 acres. [38].
[18] About 171,496 acres [38].

maps of MA counties produced with optimized models, and an analysis of land-use conversion for the whole state.

County	Most Common Land Use Types			Area of AG Land Overlap by Array Fields (acres)	Total Predicted Array Field Area
	First	Second	Third		
Barnstable	Open Land	Industrial/ Commercial	Forest	1.2	198.1
Plymouth	Forest	Open Land	Agricultural	93.4	729.4
Hampshire	Agricultural	Forest	Water	147.7	458.2
Hampden	Forest	Agriculture	Open Land	248.3	946.1

Figure 25: Summary results for land-use conversion estimates. Forest and open land were commonly replaced in all four counties.

Lessons Learned

The most important lesson I learned from this project is the true nature of machine learning, that it can be invaluable for automating tasks and generating new knowledge, but that it is also an emerging technology that requires a deep expertise to fully understand and implement. Often, I had to go through a trial and error process to find the proper functions and module versions that worked together to get the desired results. However, by doing so I vastly improved my coding and problem-solving abilities.

One of the major challenges of this project was reading the training masks in Tensorflow. Though there is an experimental function that decodes TIFFs, it only supports 4 band (RGB+A) images, so it was not suitable for parsing the single-band

masks. To decode them, I created a function using numpy and PIL, but these packages were not compatible with tf.map() which transforms the dataset elements [36]. I therefore had to pass the function to the special tf.py_function() wrapper which allows non-native functions to be used in Tensorflow but causes some loss in performance. The use of this function is likely the cause of an error when running the model in GPU which crashes the run after a few epochs. I later overcame this by saving the model after each epoch and restarting the training with the GPU.

In a later case, PIL caused the kernel to crash when displaying a sample image with the dataset.take() method. The error was noted by developers in older versions of Spyder and was corrected in a later version not directly available in Anaconda. I resolved this by creating a new environment with conda, installing the latest spyder release and simultaneously installing conda-forge which is a compilation of compatible package versions. Though I later used pip to download other key packages, including Tensorflow, it did not cause any conflicts in this case. However, one package in particular, tf.addons, corrupted my Anaconda environment after installation. Though I wanted to use this to implement some useful features, I later learned that it was only compatible with a deprecated version of Tensorflow. These instances illustrate how trade-offs or future issues might have to be accepted when one tries to adapt Tensorflow to unique situations.

Another area of concern was the high processing requirements that my project demanded. To address this, I uploaded my model to an AWS cloud instance where I

could run long-term tasks such as model training in the background. Though this was definitely helpful and cost-effective, the speed of basic ArcGIS processes did not noticeably improve. Also, running imagery through the image splitting-predicting-reconstruction pipeline still took several days for imagery of an entire county. For other machine learning practitioners, the reality of image processing in the cloud is important to realize, since basic services are charged by the hour, and costs can unexpectedly amass if not regularly checked.

A final obstacle was the long time it took for Tensorflow to predict on image chips with a CPU, about 20 minutes. To do this for imagery of an entire county would be inefficient and expensive in the cloud with CPU alone, so I enabled a GPU on the instance. However, this was not a straightforward task in AWS since I had to find the proper combinations of several drivers and change the display adapter in the instance Device Manager.[19] I also had to request a vCPU increase to gain access to the GPU, which may be required to limit the exploitation of extra processing power.

Future Work

I intend to refine the current model and use it to create a prediction map of solar arrays for the entire state of Massachusetts. To do this, I will train updated model versions with a GPU and experiment with different hyperparameter settings. My goal will be to improve the precision of the model, which could be achieved by refining the

[19] cudatoolkit = 11.2, cudnn = 8.1.0

training dataset or changing the learning rate. When this is done, I will use the model to estimate land-use conversion due to array construction throughout the state.

Credits

Thank you to my advisor, Professor Jan Oliver Wallgrün, for his guidance throughout this project. Also, thank you to Professor Joe Ranalli of Penn State Hazleton, PhD candidate Matthias Zech, and Professor Jordan Malof of Duke University for providing some initial insights and advice.

References

[1] Bose, Amitrajit, “Cross Validation - Why & How,” *towardsdatascience*, 1/30/2019, https://towardsdatascience.com/cross-validation-430d9a5fee22, retrieved on 12/4/2022.

[2] Buduma, Nithin, et al., “Fundamentals of Deep Learning: Designing Next Generation Machine Intelligence Algorithms,” *O’Reilly Media*, Sebastopol, CA, 5/2022.

[3] Camilo, Joseph, et al., “Application of a semantic segmentation convolutional neural network for accurate automatic detection and mapping of solar photovoltaic arrays in aerial imagery,” 2017 IEEE Applied Imagery Pattern Recognition (AIPR) Workshop, Washington D.C., 10/10/2016, https://arxiv.org/abs/1801.04018, retrieved on 5/26/2022.

[4] Castello, Roberto, et al., “Deep learning in the built environment: automatic detection of rooftop solar panels using Convolutional Neural Networks,” *Journal of Physics: Conference Series* 1343 012034, https://iopscience.iop.org/article/10.1088/1742-6596/1343/1/012034, retrieved on 5/25/2022.Conservation

[5] Conservation Law Foundation, “Summary of S.2768, The Green Communities Act,” http://www.clf.org/wp-content/uploads/2011/09/CLF-Green-Communities-Summary_6-24-08.pdf, retrieved on 12/2/2022.

[6] Cough, Kate, “Maine’s prime farmland is being lost to solar. Is ‘dual use’ the answer?,” *Sun Journal*, 1/22/2022, https://www.sunjournal.com/2022/01/22/maines-prime-farmland-is-being-lost-to-solar-is-dual-use-the-answer/, retrieved on 11/18/2022.

[7] da Costa, Marcus, et al., “Remote Sensing for Monitoring Photovoltaic Solar Plants in Brazil Using Deep Semantic Segmentation,” *Energies* 14(10): 2960, 2021, https://www.mdpi.com/1996-1073/14/10/2960, retrieved on 5/22/2022.

[8] DataRobot, “Introduction to Loss Functions,” https://www.datarobot.com/blog/introduction-to-loss-functions/, retrieved on 12/5/2022.

[9] EPA, “Utility-Scale Solar Energy Development: The Sullivan’s Ledge Superfund Site in New Bedford, MA,” https://web.archive.org/web/20210502031836/https://semspub.epa.gov/work/01/586232.pdf, retrieved on 11/17/2022.

[10] ESRI, “Minimum Bounding Geometry (Data Management),” https://pro.arcgis.com/en/pro-app/latest/tool-reference/data-management/minimum-bounding-geometry.htm, retrieved on 12/5/2022.

[11] Golovko, Vladimir, et al., “Convolutional neural network based solar photovoltaic panel detection in satellite photos,” 2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems, 9/21/2017, https://ieeexplore.ieee.org/xpl/conhome/8086093/proceeding, retrieved on 5/22/2022.

[12] itdxer, “What is batch size in neural network,” *stackexchange*, 5/22/2015, https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network, retrieved on 12/3/2022.

[13] Kelleher, John D., “Deep Learning,” *The MIT Press*, Cambridge, MA, 9/10/2019.

[14] Keras, “MaxPooling2D Layer,” https://keras.io/api/layers/pooling_layers/max_pooling2d/, retrieved on 12/6/2022.

[15] Kolassa, Stephan, “Why is accuracy not the measure for assessing classification models?,” *stackexchange*, https://stats.stackexchange.com/questions/312780/why-is-accuracy-not-the-best-measure-for-assessing-classification-models, retrieved on 12/4/2022.

[16] Lang, Niklas, “Using Convolutional Neural Network for Image Classification,” *towardsdatascience*, 12/4/2021, https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4, retrieved on 12/2/2022.

[17] Le Guilly, Yann, “Semantic Segmentation with tf.data in Tensorflow 2 and ADE20K dataset,” *gitlab*, 12/14/2019, https://web.archive.org/web/20210512085519/https://yann-leguilly.gitlab.io/post/2019-12-14-tensorflow-tfdata-segmentation/, retrieved on 5/7/2022.

[18] Lee, Kevin C., “Parallel Computing - Upgrade Your Data Science with GPU Computing,” *towardsdatascience*, 10/26/2020, https://towardsdatascience.com/parallel-computing-upgrade-your-data-science-with-a-gpu-bba1cc007c24, retrieved on 11/5/2022.

[19] Malof, Jordan M. et al., “Automatic detection of solar photovoltaic arrays in high resolution aerial imagery,” *Applied Energy* 183: 229-240, 2016, https://www.sciencedirect.com/science/article/abs/pii/S0306261916313009, retrieved on 5/18/2022.

[20] Mass Audubon, “Losing Ground: Nature’s Value in a Changing Climate,” 2020, https://www.massaudubon.org/content/download/41477/1007612/file/Losing-Ground-VI_2020_final.pdf, retrieved on 11/5/2022.

[21] Mass CEC, “Solar PV Systems in Massachusetts,” 2/2022, https://www.masscec.com/sites/default/files/documents/Solar%20PV%20Systems%20in%20MA%20as%20of%20Feb%202022.xlsx, retrieved on 12/5/2022.

[22] MassGIS, “MassGIS Data: 2021 Aerial Imagery,” https://www.mass.gov/info-details/massgis-data-2021-aerial-imagery, retrieved on 12/6/2022.

[23] MassGIS, “MassGIS Data: Land Use (2005),” 6/2009, https://www.mass.gov/info-details/massgis-data-land-use-2005, retrieved on 12/6/2022.

[24] MassSolar, “History of Solar in Massachusetts,” http://solarisworking.org/history-of-solar-in-massachusetts, retrieved on 12/6/2022.

[25] Massachusetts Department of Agricultural Resources, “Agricultural Resources Facts and Statistics,” https://www.mass.gov/info-details/agricultural-resources-facts-and-statistics#:~:text=Current%20Statistics,agricultural%20 products%20 on%2068%20 acres, retrieved on 11/25/2022.

[26] Morehouse, Catherine, “As Massachusetts solar installs plummet, stalled interconnections, land use questions are key hurdles,” *UtilityDive*, https://www.utilitydive.com/news/as-massachusetts-solar-installs-plummet-stalled-interconnections-land-use/572925/, retrieved on 10/6/2022.

[27] nbro, “Why do we need convolutional neural networks instead of feed-forward neural networks,” *AI Stackexchange*, 12/12/2020, https://ai.stackexchange.com/questions/21394/why-do-we-need-convolutional-neural-networks-instead-of-feed-forward-neural-netw, retrieved on 12/6/2022.

[28] Parhar, Poonam, et al., “HyperionSolarNet: Solar Panel Detection from Aerial Images,” 1/6/2022, https://arxiv.org/abs/2201.02107, retrieved on 5/23/2022.

[29] Peters, Xander, “Solar energy is a new cash crop for farmers- when the price is right,” *The Christian Science Monitor*, 10/4/2021, https://www.csmonitor.com/Environment/2021/1004/Solar-energy-is-a-new-cash-crop-for-farmers-when-the-price-is-right, retrieved on 11/5/2022.

[30] Puttemans, Stephen., et al., “Detection of Photovoltaic Installations in RGB Aerial Imagery: A Comparative Study,” GEOBIA2016 Conference, Enschede, Netherlands, [conference paper], 9/2016, https://www.researchgate.net/publication/305395692_Detection_of_Photovoltaic_Installations_in_RGB_Aerial_Imaging_A_Comparative_Study, retrieved on 5/21/2022.

[31] Rakhecha, Aditya, “Understanding Learning Rate,” *towardsdatascience*, 6/28/2019, https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network, retrieved on 12/4/2022.

[32] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox, “U-Net: Convolutional for Biomedical Image Segmentation,” 5/18/2015, https://arxiv.org/pdf/1505.04597.pdf, retrieved on 11/6/2022.

[33] Rukundo, Olivier, “Effects of Image Size on Deep Learning,” 2021, https://arxiv.org/ftp/arxiv/papers/2101/2101.11508.pdf, retrieved on 10/5/2022.

[34] SolarReviews, “2022 Massachusetts SMART Program Explained,” 3/17/2022, https://www.solarreviews.com/blog/massachusetts-smart-program-replaces-srecs, retrieved on 12/6/2022.

[35] Tensorflow, “tf.compat.v1.layers.Dropout,” https://www.tensorflow.org/api_docs/python/tf/compat/v1/layers/Dropout#:~:text=Dropout%20consists%20in%20randomly%20setting,training%20time%20and%20inference%20time, retrieved on 12/3/2022.

[36] Tensorflow, “tf.data.Dataset,” https://www.tensorflow.org/api_docs/python/tf/data/Dataset, retrieved on 12/6/2022.

[37] Tiu, Ekin, “Metrics to Evaluate your Semantic Segmentation Model,” *towardsdatascience*, 8/10/2019, https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2, retrieved on 8/12/2022.

[38] United States Department of Agriculture, “2017 Census of Agriculture,” 4/2019, https://www.nass.usda.gov/Publications/AgCensus/2017/Full_Report/Volume_1,_Chapter_1_US/usv1.pdf, retrieved on 11/5/2022.

[39] US Energy Information Administration, “Massachusetts: State Profile and Energy Estimates,” https://www.eia.gov/state/analysis.php?sid=MA, retrieved on 11/15/2022.

[40] Wool, Joel, “Bright history, but dark clouds threaten Massachusetts solar policies,” *Clean Water Action*, 7/17/2017, https://cleanwater.org/2017/07/07/bright-history-dark-clouds-threaten-massachusetts-solar-policies, retrieved on 12/1/2022.

[41] Zech, Matthias and Joe Ranalli, “Predicting PV Areas in Aerial Images with Deep Learning,” 2020 47th IEEE Photovoltaic Specialists Conference (PVSC), 1/5/2021, https://ieeexplore.ieee.org/document/9300636, retrieved on 5/21/2022.

[42] Zulkifli, Hafidz, “Understanding Learning Rates and How It Improves Performance in Deep Learning,” *towardsdatascience*, 1/21/2018, https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10, retrieved on 12/4/2022.