Extraction of Building Footprints from Satellite Imagery

Elliott Chartock
elboy@stanford.edu

Whitney LaRow
Stanford University
wlarow@stanford.edu

Vijay Singh
vpsingh@stanford.edu

Abstract

We use a Fully Convolutional Neural Network to extract bounding polygons for building footprints. Our network takes in 11-band satellite image data and produces signed distance labels, denoting which pixels are inside and outside of building footprints. Finally, we post-process the data to produce bounding polygons. When a similar dataset was first released as part of the first SpaceNet Challenge, the winning implementation produced an F1 score of 0.25 and used no deep learning; our approach outperforms this with an F1 score of 0.34.

1. Introduction

This project aims to extract structured information (i.e. bounding polygons) for building footprints from high-resolution satellite imagery. We believe this problem is interesting because it is fundamental to the larger problem of creating automated maps from satellite imagery. These maps are useful for tasks like disaster response and locating eligible rooftops for solar panels.

Because of its applicability, this problem has been widely explored in the past. Recently, it has become even more accessible through Topcoder's SpaceNet Challenge, which provides a high-quality labeled dataset and specific evaluation tools for contestants to submit their solutions for building footprint finding.

In the past, this specific problem has usually been approached as a segmentation problem with two classes - "building" and "not building". We decided to explore an alternate route, instead using signed distance labels (where pixel values indicate the distance to the closest building wall, with pixels inside buildings having positive values and pixels outside buildings having negative values). This creates a regression segmentation problem - instead of classifying pixels, we assign them real valued scores. Our pipeline consists of image pre-processing to create features and labels, a Fully Convolutional Neural Network to label pixels, and then post-processing to decide on the final polygons.

Last year, the winning implementation of TopCoder's SpaceNet Challenge produced an F1 score of 0.25 and did not utilize any Neural Networks (see Section 2.3 for further detail). With our approach, we hope to exceed this score given the recent success Convolutional Neural Networks have seen in image segmentation.

2. Related Work

2.1. FCN Semantic Segmentation

Object classification of satellite imagery has largely been approached as a semantic segmentation problem. In [6], the authors apply two segmentation techniques to the ISPRS Vaihingen 2D datset to classify the pixels of the image into five classes of interest: road, building, car, vegetation, tree. The first approach is patch-based, which trains a CNN on small image patches, and then predicts each pixel as the center pixel of an enclosed region. The second approach is a pixel-to-pixel Fully Convolutional Neural Network (FCNN). The architecture uses four sets of layered 3x3 convolutions, with each set followed by a 2x2 max pooling layer that down-samples the input to lower spatial resolution. Each convolution is separated by a ReLU nonlinearity and Batch normalization. A transpose convolution is then applied to up-sample back to the original pixel input size, and finally a softmax layer predicts the class for each pixel.

The authors find that both models effectively extract larger targets, such as buildings and roads, but the patch-based model yielded low F1 on the car class due to a large number of false positives (predicting cars that don't exist) and also mislabeled small patches of trees as vegetation. They conclude that the pixel-to-pixel FCNN technique is the superior architecture for segmentation of satellite imagery.

[2] builds off the findings of [6] by applying the pixel-to-pixel FCNN architecture to the SpaceNet dataset. Since the SpaceNet Challenge is only concerned with building footprints, [2] adapts the architecture to learn only two classes: "building" and "non-building". Post-processing then takes the predicted binary image and creates polygonal borders.

This implementation yielded a Building Footprint Metric Score of 0.168605, which took 5th place in the first iteration of the SpaceNet Challenge.

2.2. Semantic Segmentation with SSD Polygon Proposal

[9] also considers the SpaceNet Challenge as a semantic segmentation problem, but adds an additional learning layer to the pipeline to optimize the polygonization step. The initial FCNN architecture makes pixel-wise predictions as either inside a building, outside a building, or within a small threshold (four pixels) distance of a border. The segmentation step produces a heatmap prediction where each pixel takes on one of three values corresponding to the three target classes. This implementation then uses the Single Shot MultiBox Detector (SSD) method proposed in [7] to learn which bounding boxes are good polygon approximations of building footprints.

The SSD approach discretizes the sample space of bounding polygons and then learns which polygons produce the best overlap with building footprints. In the work of [9], each predicted heatmap is divided into a 50x50 grid. Each grid region has a set of 16 default rectangular footprint proposals. The SSD algorithm uses a feed-forward convolutional network to produce scores for a building being present in each proposal region. To adhere to the requirements of the SpaceNet Challenge, [9] predicts whether each rectangle has an IoU (discussed in Evaluation) above 0.5 with some building footprint.

By appending the FCNN with a polygonization learning layer, [9] produced a Building Footprint Metric Score of 0.245420, which took second place in the first round of the SpaceNet Challenge. Our emphasis on the heatmap post-processing stage via clustering and marching squares builds on the success of the SSD method to learn good polygon predictions from the heatmap.

2.3. Random Decision Forests

While the nature of this problem lends itself to CNN-based solutions, surprisingly, the SpaceNet Challenge winning implementation does not use a convolutional network. [13] produces a Building Footprint Metric Score of 0.255292 via a series of Random Decision Forests. In the image classification step, this implementation builds two random forests of binary classification trees, one of which predicts building pixels and the other predicts border pixels. A third random forest then builds regression trees that predicts pixel-wise signed distances to building borders. A flood fill algorithm is applied to the distance heatmaps to distinguish between neighboring buildings, and then a convex hull procedure generates polygons from pixel groups. Finally, a random forest of regression trees is used to predict which polygons will yield the highest IoU.

We draw our inspiration to approach this project as a regression segmentation problem that predicts signed distance heatmaps from the success of this Random Decision Forest implementation. Our hope is that training an FCNN to predict signed distance heatmaps will yield significant boost in footprint extraction performance.

3. Dataset

To train and test our model, we use the data provided by Topcoder's SpaceNet Challenge [10]. Our dataset consists of 250 16-bit GeoTiff images collected by the DigitalGlobe Worldview-3 satellite. The images are all of Las Vegas and each image is provided in 4 different formats: grayscale, RGB, 8-band multi-channel (i.e. 8 different frequency measurements), and higher-resolution 8-band multi-channel. These images all cover a 200 meter x 200 meter area on the ground. In addition to the images, all training data is accompanied by ground truth labels, which takes the form of a CSV file containing unique (polygon, building, image) triplets.

Our dataset is broken up into 60% training, 20% validation, and 20% test.

3.1. Data Augmentation

For the task of training a CNN from scratch, 150 images is relatively few. We conquer this limitation by applying rotational and reflective transformations on the input images to increase our training data eight-fold. We rotate the original image by 0 degrees (identity mapping), 90 degrees, 180 degrees, and 270 degrees to create three new training images. We then apply a vertically aligned reflection on top of each aforementioned rotation to produced four more new training images. As discussed in Results, data augmentation facilitated CNN learning and improved the best model's ability to generalize to unseen satellite images.

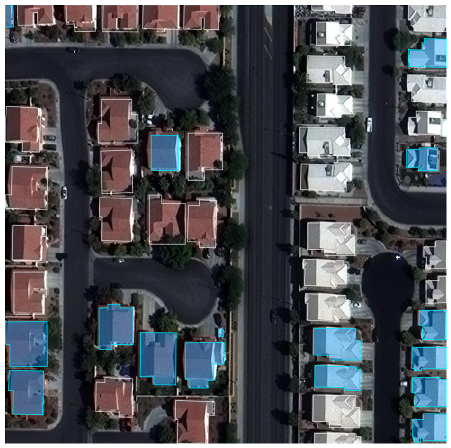

Figure 1: Example image from data set with ground truth building footprints overlain.

4. Evaluation

Our final output is a CSV file containing the polygon outline of each building we find, where a polygon is specified by a list of points that define it, and which image the building is in. This matches the format of the labeled ground truth data. This way, we can overlay our predicted polygons over input images or ground truth polygons to help visualize our model's results and qualitatively evaluate our precision and recall.

Quantitatively, we use IoU (Intersection over Union) to evaluate our model's results [4]:

$$\text{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

IoU presents a normalized, scale-invariant metric that focuses on the area of the regions. We say that an IoU score above 0.5 indicates a detection. Finally, we evaluate our model by combining precision (fraction of proposals that are true positives) and recall (fraction of labeled objects that are detected) into one overall F1 score [1]:

$$F_1 = \frac{2 \times precision \times recall}{precision + recall}$$

We calculate one value for precision and one value for recall for the entire test set of images (not per image) and calculate one F1 score for all the buildings and images at once. F1 score is between 0 and 1, with values closer to 1 indicating a better model.

5. Approach

5.1. Pre-Processing

The first step of our pipeline is image pre-processing. The dataset contains 4 different image formats we can use as input features. For simplicity, we began by using a concatenation of the high-resolution 8-band multi-channel images and the 3-band RGB images. Both formats yield images that are 650 x 650 pixels. This creates an input size of 650 x 650 x 11.

The next step is converting the CSV ground truth labels into GeoJSON data and then raster data (i.e. a matrix of pixels) that we can use to calculate loss while training our model [11]. The GeoJSON data is formatted as a list of points (lattitude, longitude) that form the bounding polygon for each building in a given image. We first translate these coordinates from geospatial values (latitude and longitude) to pixel values, using the GDAL (Geospatial Data Abstraction) library. We then use signed distance transform to assign a real value to each pixel equal to its distance from the closest building boundary (with pixels inside buildings having positive values and pixels outside buildings having negative values) [11]. We then scale this real value to lie between -1 and 1. Our results from implementing this technique on the image in Figure 1 can be seen in Figure 2.

One benefit to using signed distance labels over binary class labels (where a pixel is 1 if it is inside a building footprint and 0 otherwise) is ease in the post-processing step to distinguish multiple adjacent buildings from one large building.

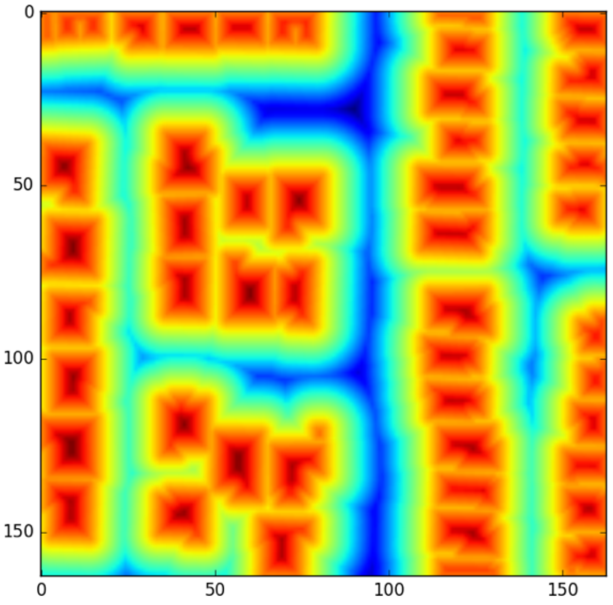

Figure 2: Signed distance transform results on the low-res (163 x 163 pixel) image in Figure 1.

5.2. Model

The next step is to train a model that outputs good heatmap predictions. We feed the input data and labels into a Fully Convolutional Neural Network (FCNN) to train a model that will predict signed distance pixel-wise values. The choice to train an FCNN architecture was inspired by the work in [8], in which the authors use an FCNN to produce state-of-the-art results on semantic segmentation tasks.

At a high level, an FCNN stacks multiple convolutional layers, with additional downsampling and upsampling layers inside the network. We now discuss our baseline and best performing model architectures. For all models described below, each convolution is followed by a ReLU nonlinearity and Batch normalization. We train using Adam optimizer with learning rate = 0.0001, weight decay = 0.1, β_1 = 0.999, and β_2 = 0.9. All experiments in this paper are run using the deep learning framework PyTorch. Our deep convolutional models are trained and tested using an NVIDIA Tesla K80 GPU.

[1]Note that the SpaceNet Challenge refers to the F1 score as Building Footprint Metric Score; the two can be used interchangeably. For the purposes of this paper we will here on out refer to this score as F1.

Baseline Model As a first pass, we propose a simple FCNN architecture, which can be seen in Figure 3. We stack two convolutional blocks, where each block down-samples to lower spatial resolution with 2x2 max-pooling. In the first block we do two convolutions of 16 filters and 32 filters, respectively, both with 3x3 kernels. The second block does a convolution of 32 filters with 3x3 kernels and then a transpose convolution of 1 filter with 3x3 kernels. With two max-pooling layers that down-sample, and no upsampling layers, we map the original 650x650x11 input images to low-res 163x163 pixel heatmaps to match the dimensions of our labels. Note that in this model, all convolutions preserve dimension.

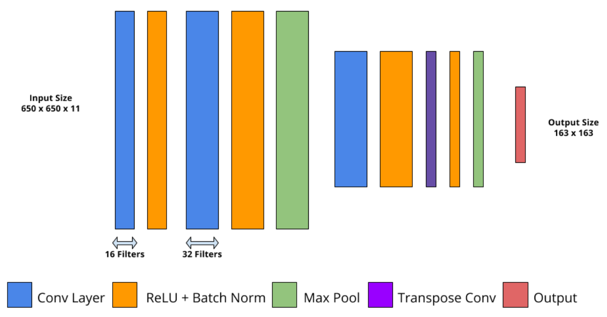

Figure 3: Baseline FCNN architecture.

Final FCNN Architecture The CNN architecture of the best performing FCNN model stacks layers of convolutions and transpose convolutions, separated by two max-pooling layers that each half the dimension, down-sampling from the original 650x650 image dimension to low-res 163x163 signed distance predictions. The explicit architecture can be seen in Figure 4.

In this model, we stack three convolutional blocks, where between each block we down-sample to lower spatial resolution with 2x2 max-pooling. The first block begins like our baseline model, with two convolutions of 16 filters and 32 filters, respectively, both with 3x3 kernels. The second block contains a conv layer and a conv transpose, each with 32 filters and 3x3 kernels. The final block has a convolution and a transpose convolution of 16 filters with 3x3 kernels each. Finally, there is a transpose convolution of 1 filter with a 3x3 kernel, that produces the desired 163x163 dimension heatmap prediction.

All convolutional layers except the final layer are followed by a ReLU and Batch normalization. To deter overfitting, both max-pool layers are followed by dropout layers with dropout probability 0.3.

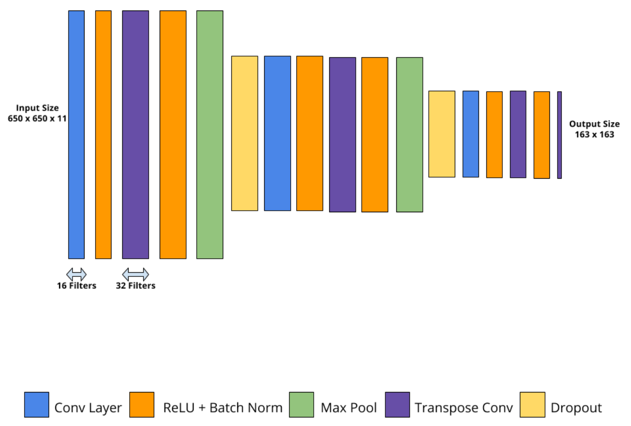

Figure 4: Final FCNN architecture.

5.3. Post-Processing

The last part of the pipeline is converting the FCNN predicted matrix of signed distance pixels into polygonal predictions represented as a list of points with the same format as the ground truth labels.

Greedy Clustering One method for the polygonization step is a greedy algorithm for forming clusters of pixels which represent individual buildings. We then use GDAL to convert the clusters into vectorized polygons represented by a list of points [3]. The vanilla clustering algorithm, which we call Greedy Clustering, works as follows:

1. Add all positive pixels to a candidate set, S of pixels.
2. Select the most positive pixel, $p \in S$, and add it to the current cluster, C.
3. Iterate through S, adding pixels to C if they are (a) adjacent to some pixel, n_i, in C and (b) have a signed distance less than or equal to that of n_i.
4. Stop when C stops growing.
5. Remove all pixels in C from S.
6. Repeat steps 2-5 until S is empty.

Early results indicated that the vanilla algorithm tends to produce mixed results. We witness that it will occasionally break a building up into multiple small clusters as seen in Figure 5. Additionally, at then end of cluster generation, there tends to be a few leftover pixels that get put into their own small clusters, leading to the formation of small erroneous buildings.

To prevent these errors, we adjust the algorithm to create an improved greedy clustering algorithm, which we call Threshold Greedy Clustering. This new algorithm consists of two main differences. First, to prevent building breakup, we relaxed constraint (b) for adding a pixel to the current cluster. Instead of requiring that the new pixel have a signed distance less than or equal to that of n_i, we allow it to have a signed distance less than or equal to that of $n_i + \epsilon$ for some small value ϵ. After trying a variety of different values for ϵ, we found that 0.01 worked particularly well (see section 7.3 below for further discussion). Second, to prevent small cluster formation at the end, we add an early stopping threshold λ (when the set S gets to below λ pixels, stop the cluster generation). Again, after testing out differet values on our validation set, we found that $\lambda = 50$ worked particularly well (quantitative results in section 7.3 below).

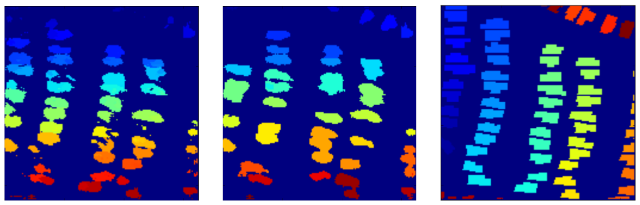
Figure 5: Clusters produced by the vanilla greedy clustering algorithm (left) and the improved greedy clustering algorithm (center) compared to the ground truth polygons (right). Distinct colors represent distinct clusters, but coloring is otherwise arbitrary and has no significance.

Marching Squares The second algorithm we implement for post-processing is a commonly known algorithm for contour-finding called Marching Squares [12]. This algorithm produces smooth outlines based on the signed distance pixel values. We convert these outlines into a mask of building clusters and then use GDAL as before to convert these clusters into vectorized polygons represented by a list of points. We found that marching squares was a more efficient algorithm than greedy clustering, but it tended to perform worse as seen in Figure 6 because the building outlines tended to be less sharp and adjacent buildings frequently got combined into one.

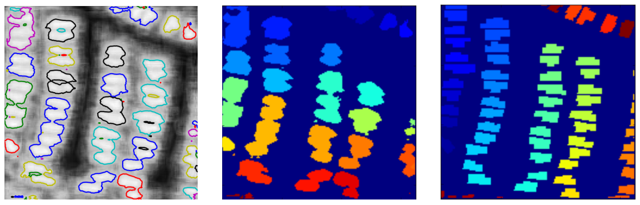
Figure 6: Contours produced by the marching squares algorithm (left) and their associated clusters (center) compared to the ground truth polygons (right). Distinct colors represent distinct clusters, but coloring is otherwise arbitrary and has no significance.

6. Experiments

We run several experiments to test different tasks within the pipeline independently. First, we experiment on decisions within the convolutional architecture that predicts heatmaps. Then, we experiment on the polygonization post-processing step to compare Threshold Greedy Clustering and Marching Squares. The authors believe that producing two separate local optimal solutions for heatmap creation and polygonization, respectively, will lead to a globally optimal solution. Formally, for network architectures A and B and post-processors X and Y, if A produces better heatmap predictions than B, and X converts heatmaps to polygonal building boundaries better than Y, we operate under the assumption that stacking A-X will yield higher F1 score than any of A-Y, B-X, or B-Y.

6.1. FCNN Optimization

In this phase of experimentation we use Threshold Greedy Clustering polygonization with $\epsilon = 0.05$ and $\lambda = 10$ for all network experimentation, for consistency. In this set of experiments, we look to optimize our loss function. Specifically, we test L_2 loss versus MSE loss. We then train two models, one with no regularization, and another with weight decay and dropout regularization.

6.2. Polygonization

In post-processing experimentation, we test Greedy Clustering, Threshold Greedy Clustering, and Marching Squares on the heatmaps produced by the FCNN in Figure 4. Within the Threshold Greedy Clustering experimentation, we tune the ϵ and λ parameters on the same heatmaps, which are produced from out best FCNN model from 6.1.

6.3. Input Bands

After completing the FCNN architecture experimentation, we test our optimized pipeline on different image inputs. We train the best FCNN architecture on RGB inputs, 8-band multi-channel inputs, and the 11-band concatenation discussed in dataset above. We hypothesize that the 11-band concatenation contains redundancies that significantly increase the duration of pre-processing and training, but do not boost performance.

7. Results & Discussion

All results provided are on our test set of 50 images.

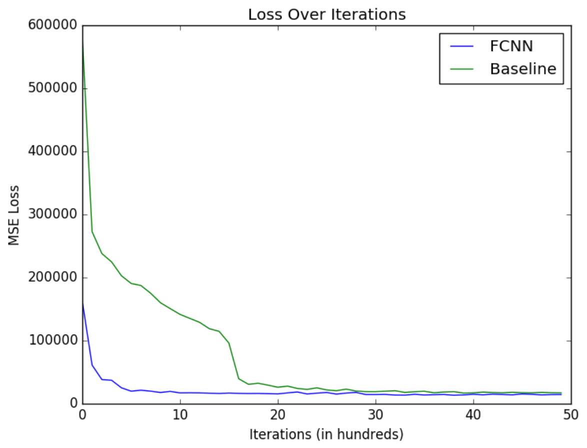

Figure 7: Graph of loss over time for the baseline network and the final FCNN.

7.1. Architectures

We first experimented with the architecture of the FCNN. We created a baseline following a relatively shallow architecture with two layers, each consisting of a convolution followed by a transpose convolution. Our baseline outperformed our expectations, producing an F1 score of 0.11. Importantly though, the baseline was still far from our goal of outperforming the previous winning implementation's F1 score of 0.25.

To improve the model, we built a deeper model. First, we tried utilizing aspects of of high-performing CNN-based models entered in the first iteration of the SpaceNet Challenge. This consisted of adding more convolutional and transpose convolutional layers, along with a linear layer at the end of our model.

Surprisingly, though the model successfully overfit on training data, it yielded dramatically worse test results, producing an F1 score of nearly 0. Of note, adding the linear layer at the end worsened our model, despite being utilized in successful implementations such as [2]. We believe this is due to past implementations discretely categorizing pixels into "building" and "not building" classes, whereas our signed distance output is richer and needs to capture more information, which may have been lost in the last linear layer.

Additionally, adding more convolutional and tranpose convolutional layers did not improve our results at first, either. We realized we did not give proper thought to our hidden dimensions and had no pattern to how we were upsampling and downsampling; once we added more structure to these layers, our F1 score jumped up dramatically to above 0.2.

Of note, past implementations used Cross-Entropy Loss, this worked well because they discretely classified pixels into classes. With our use of signed distance labels, we had to pick a different loss function well-suited to our approach. Initially, we used Mean Squared Error, as this loss function is often used in regression problems. However, we later switched to using L_2 loss: we believed this made more sense as our labels capture actual distances of pixels to building borders, and L_2 loss is distance-preserving. As shown in Table 1, this improved our results as well.

Model Comparison			
	Precision	Recall	F1 Score
Baseline with L_2 Loss	0.11	0.18	0.11
FCNN with L_2 Loss	0.37	0.23	0.28
FCNN with MSE Loss	0.35	0.15	0.21

Table 1: Comparing different loss functions to the baseline model while holding post-processing to Threshold Greedy Clustering with $\epsilon = 0.05$ and $\lambda = 10$.

We also noticed that our model was performing significantly better on training data than test data, so we decided to add regularization in the form of two Dropout Layers and a weight decay, improving our results as shown in Table 2.

Regularization Effects			
	Precision	Recall	F1 Score
With Reg.	0.37	0.23	0.28
Without Reg.	0.23	0.19	0.21

Table 2: Comparing the effects of regularization (dropout and weight decay) while holding post-processing to Threshold Greedy Clustering with $\epsilon = 0.05$ and $\lambda = 10$.

7.2. Input Bands

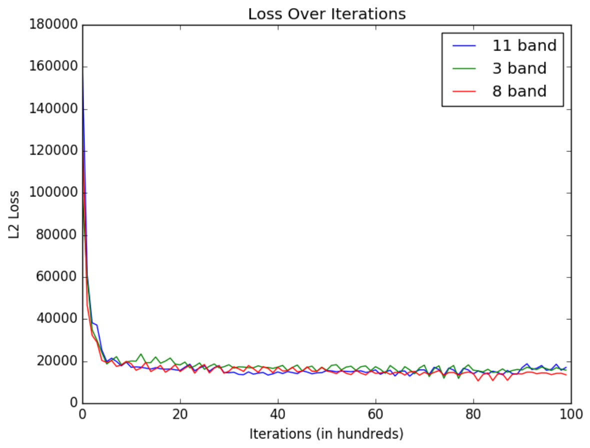

Figure 8: Comparison of loss over time using 3-band, 8-band, and 11-band inputs with our FCNN.

Our initial hypothesis was that creating an 11-band input by concatenating the RGB inputs and 8-band multichannel inputs created redundancies that would not boost performance. While the matching loss curves above initially seemed to corroborate this hypothesis, we ended up being incorrect after computing the final F1 scores. As shown in Table 3, the 11-band concatenation significantly outperformed the 8-band or 3-band alone, with 8-band outperforming 3-band. Clearly, each band does contain unique information important in the task of image segmentation.

Input Band Comparison			
	Precision	Recall	F1 Score
RGB	0.15	0.19	0.17
8-band	0.19	0.32	0.24
11-band	0.36	0.28	0.30

Table 3: Comparing scores from using different input bands.

7.3. Post-Processing

The final piece of our pipeline to experiment with was our post-processing techniques. We first evaluated the vanilla Greedy Clustering algorithm and saw that the poor precision of the algorithm was mainly responsible for the low F1 score (see Table 4).

We amended the algorithm to create Threshold Greedy Clustering as described in section 5.3 above, which greatly improved our precision and thus F1 score. After tuning ϵ (ranging from 0.005 to 0.05) and λ (ranging from 10 to 60), we found values that increased the final F1 score of our best model to 0.34.

We also tried the Marching Squares algorithm described in section 5.3, but found the results to be worse than Threshold Greedy Clustering, due to its lack of flexibility in forming building borders (i.e. no ϵ parameter could be easily introduced to allow the breakup of erroneously combined larger buildings).

Post-Processing Comparison			
	Precision	Recall	F1 Score
Greedy Clustering	0.08	0.28	0.12
Threshold GC $\epsilon = 0.05, \lambda = 10$	0.37	0.23	0.28
Threshold GC $\epsilon = 0.01, \lambda = 50$	0.42	0.28	0.34
Marching Squares	0.25	0.21	0.23

Table 4: Comparing different post-processing techniques and parameter values on our best FCNN model with L_2 loss.

7.4. Final Model

After experimentation, we finalized our model using the FCNN architecture outlined in Figure 4 with L_2 loss. The model includes regularization in the forms of dropout, batch norm, and weight decay, which help it prevent overfitting. It takes in 11-band input images and uses Threshold Greedy Clustering with $\epsilon = 0.01$ and $\lambda = 50$ to form the final building polygons.

Final Model		
Precision	Recall	F1 Score
0.42	0.28	0.34

Table 5: Performance of our final model, which outperforms all previous SpaceNet challenge winners.

8. Conclusion & Future Work

In this work, we experimented with different segmentation and polygonization methods to further research in geospatial computer vision algorithms. Ultimately, we have created a pipeline that extracts building footprints from satellite imagery with high accuracy. We trained a Fully Convolutional Neural Network to perform pixel-to-pixel regression segmentation. We then applied a post-processing step - either Marching Squares or Greedy Clustering - to convert the FCNN output into bounding polygons corresponding to building footprint predictions. We concluded that the signed distance heatmap predictions, combined with Marching Squares polygonization provided the best performance in terms of accurately overlapping predicted footprints with ground truth footprints. Our best model had an F1 score of 0.34 on test data, and we achieved our goal of surpassing the previous winning implemenation's F1 score of 0.25.

In future work, inspired by the success of [1] using transfer learning, we plan to train our FCNN on top of a pretrained CNN such as VGG. The hope is that the VGG model has learned general features from a larger image corpus, and then we can fine-tune the model to the task of satellite image segmentation with our smaller SpaceNet dataset. We also plan to approach building footprint extraction as an instance segmentation problem. We can simplify the pipeline by directly extracting bounding polygons using a Mask R-CNN, rather than first predicting a heatmap and then applying Marching Squares [5]. Finally, dilated convolutions have also seen success in semantic segmentation as they can merge spatial information across inputs more aggressively; experimenting with them in our model could be interesting.

References

[1] H. Azizpour, A. S. Razavian, J. Sullivan, A. Maki, and S. Carlsson. Factors of transferability for a generic convnet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(9):1790–1802, 2016.

[2] Bic-User. Topcoder: bic-user's implementation. `https://github.com/SpaceNetChallenge/BuildingDetectors/tree/master/bic-user`, 2017. Github.

[3] P. Hagerty. Object detection on spacenet. `https://medium.com/the-downlinq/object-detection-on-spacenet-5e691961d257`, 2017. Medium.

[4] P. Hagerty. The spacenet metric. `https://medium.com/the-downlinq/the-spacenet-metric-612183cc2ddb`, 2017. Medium.

[5] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. *arXiv preprint arXiv:1703.06870*, 2017.

[6] M. Kampffmeyer, A.-B. Salberg, and R. Jenssen. Semantic segmentation of small objects and modeling of uncertainty in urban remote sensing images using deep convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–9, 2016.

[7] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37. Springer, 2016.

[8] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[9] C. Marek. Topcoder: marek.cygan's implementation. `https://github.com/SpaceNetChallenge/BuildingDetectors/tree/master/marek.cygan`, 2017.

[10] Topcoder. Spacenet challenge problem statement. `https://community.topcoder.com/longcontest/?module=ViewProblemStatement&rd=16892&pm=14551`, 2017.

[11] A. Van Etten. Getting started with spacenet data. `https://medium.com/the-downlinq/getting-started-with-spacenet-data-827fd2ec9f53`, 2017. Medium.

[12] Wikipedia. Marching squares. `https://en.wikipedia.org/wiki/Marching_squares`, 2017.

[13] Wleite. Topcoder: wleite's implementation. `https://github.com/SpaceNetChallenge/BuildingDetectors/tree/master/wleite`, 2017.