MEF UNIVERSITY

BUILDING FOOTPRINT EXTRACTION USING DEEP LEARNING TECHNIQUES

Capstone Project

OYTUN DENIZ

İSTANBUL, 2018

MEF UNIVERSITY

BUILDING FOOTPRINT EXTRACTION USING DEEP LEARNING TECHNIQUES.

Capstone Project

OYTUN DENIZ

Prof. Dr. Muhittin Gökmen

İSTANBUL, 2018

MEF UNIVERSITY

Name of the project: Building Footprint Extraction Using Deep Learning Techniques.
Name/Last Name of the Student: Oytun Deniz
Date of Thesis Defense: 10/09/2018

I hereby state that the graduation project prepared by Your Name (Title Format) has been completed under my supervision. I accept this work as a "Graduation Project".

10/09/2018

Prof. Dr. Muhittin Gökmen

I hereby state that I have examined this graduation project by Your Name (Title Format) which is accepted by his supervisor. This work is acceptable as a graduation project and the student is eligible to take the graduation project examination.

10/09/2018

Prof. Dr. Özgür Özlük

We hereby state that we have held the graduation examination of Oytun Deniz and agree that the student has satisfied all requirements.

THE EXAMINATION COMMITTEE

Committee Member	Signature
1. Prof. Dr. Muhittin Gökmen	……..
2. ……..	……..

Academic Honesty Pledge

I promise not to collaborate with anyone, not to seek or accept any outside help, and not to give any help to others.

I understand that all resources in print or on the web must be explicitly cited.

In keeping with MEF University's ideals, I pledge that this work is my own and that I have neither given nor received inappropriate assistance in preparing it.

Name	Date	Signature
Oytun Deniz	10.09.2018	

EXECUTIVE SUMMARY

BUILDING FOOTPRINT EXTRACTION USING DEEP LEARNING TECHNIQUES.

Oytun Deniz

Advisor: Prof. Dr. Muhittin Gökmen

SEPTEMBER, 2018, 56 Pages

Geospatial industry is getting bigger and bigger these days in addition to creating massive amount of data. Today map features such as roads, building footprints are created through manual techniques. There is not automated solution that extracts map features such as roads, building footprints from satellite imagery. Advance automated feature extraction techniques will serve important uses of map data including disaster response.

SpaceNet is a commercial satellite imagery and labeled training data to foster innovation in the development of computer vision algorithms. In this paper we will give a brief explanation about image classification, object recognition processes and why deep learning is effective on object recognition, and how we can apply these concepts to our problem which is Building Footprint extraction. And we will use SpaceNet's dataset and apply tensorflow backhand object detection model.

Key Words: Deep Learning, Building, Footprint, Satellite

ÖZET

DERİN ÖĞRENME TEKNIKLERI KULLANILARAK UYDU GÖRÜNTÜLERİNDEKİ BİNALARIN İŞARETLENMESİ.

Oytun Deniz

Tez Danışmanı: Prof. Dr. Muhittin Gökmen

EYLÜL, 2018. 56 Pages

Coğrafi veri endüstrisi gün geçtikçe büyümekte ve ciddi anlamda büyük veri setleri oluşturmaktalar. Günümüzde yollar, binalar gibi harita özellikleri, uygu görüntüleri kullanılarak manuel tekniklerle ayırt edilebiliyor. Bu insan gücünü ortadan kaldıracak bir otomasyon henüz bulunmamakta. Olası bir otomasyon, gelecekte uydu görüntülerinin işlenip insanlık yararına kullanılabilmesine olanak sağlayacaktır.

SpaceNet görüntü işlem algoritmalarını geliştirmek amacıyla her hangi bir ticari amaç gütmeden. test ve train veri setleri sağlamaktadır. Bu çalışmada, resim sınıflandırma ve obje tanıma algoritmaları ile ilgili detaylara ek olarak derin öğrenme tekniklerinin obje tanımlama algoritmalarındaki önemi ve bu teknikleri uydu görüntülerindeki binaları işaretlemek için nasıl kullanıldığı incelenmektedir..

Anahtar Kelimeler: Derin Öğrenme, Bina, Kaplama Alanı, Uydu

TABLE OF CONTENTS

.

1. INTRODUCTION AND LITERATURE REVIEW

In this section, firstly, we will explain details about image classification, and object recognition concepts. Secondly, we will talk about Fundamentals of Deep Learning and why it's better than classic image classification techniques. At the last section, how we can apply these techniques to our problem which is building footprint extraction.

1.1. Image Classification and Object Detection

What is Image Classification? Basically, Image classification takes an image and predicts the object in an image. For example, you have a picture of a dog or cat. The model predicts that its dog or cat. What if we have a dog and cat image in the single picture. What would our model predict? To solve this problem, we can create a multi-label classifier which predicts both. To do that we have to find the location of classes in the image. It is like a drawing rectangle. In a single image, predicting the location of the object and its class is an example of Object Detection.

Simple visualization could make more clear the problem. Let's think that we are working on the automated car system. Our sensors captured the image below. And we are trying to make car system recognize objects and stops at the proper time.

Example input image illustrated in Figure 1:

Figure 1: Example input image

To solve this problem, somehow if we could drive a box around these objects. Car system can mark this boxes and then make a decision. So our aim is something like Figure 2.

Figure 2 – Decision Boxes

Firstly, we have to find the location of all objects and then identify it using some kind of image classifier.

We can start by dividing the picture into pieces, then feed this images our classifier and we can find the location of objects.

We divided the picture into 4 pieces. Upper left illustrated in Figure 3:

Figure 3: Upper of Input Image

Upper right illustrated in Figure 4:

Figure 4: Upper right of Input Image

Lower left of image illustrated in Figure 5:

Figure 5: Lower Left of Input Image

Lower right of image illustrated in Figure 6:

Figure 6: Lower right of Input Image

If we merge these images output illustrated in Figure 7.

Figure 7: Merged Input Image

This is a good approach but we need a more accurate way and confidential system. Another idea can be, increasing pieces.

The result illustrated in Figure 8.

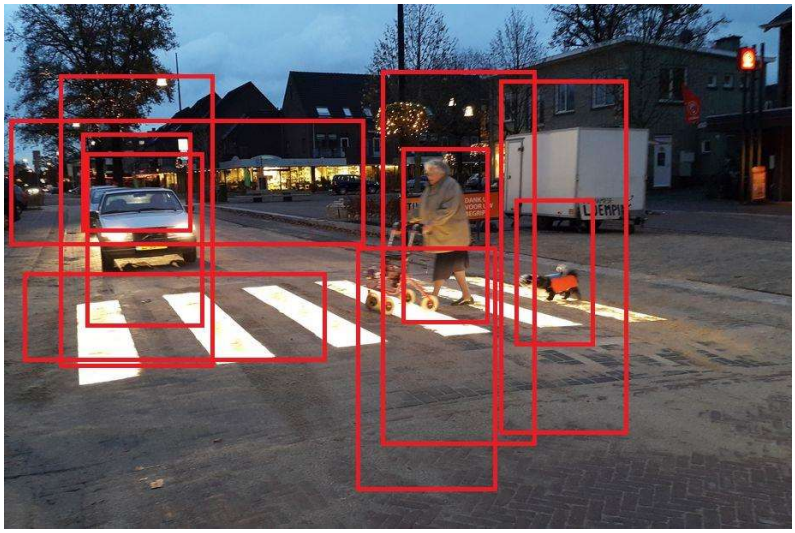

Figure 8: Full input image with object boxes

The solution seems a bit better than previous but our boxes are overlapping. We need the more structured way to solve this problem.

We can perform more structured division. Divide images same small pieces maybe 20x20 grid. As we know our objects in images are connected with other pieces around them. Assume that we dived person piece in 3 if we go around middle pieces in different heights and aspect ratio. After that, we can pass these images to the classifier and get predictions. Results will be more structured but improbable.

The result illustrated in Figure 9.

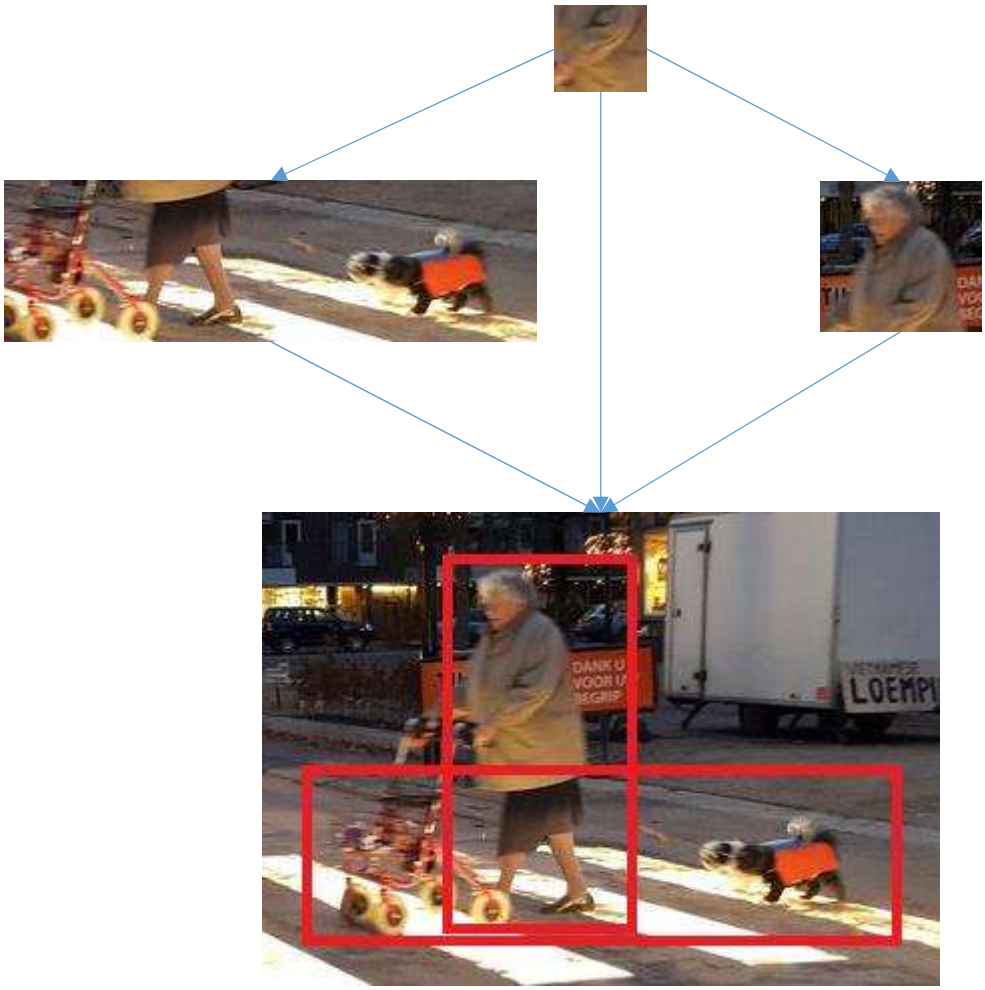

Figure 9: Relationship between sub images

The final solution is promising but we can add some efficient ways to the solution. We can increase grid and patch size. In addition, instead of feeding all pieces of images to image

classifier, we can feed it a different classifier like background prediction, after that, we apply our main classifier. I affect our model accuracy in a good way and speeds up our model run time.

1.2. Neural Networks and Object Detection

The imagenet project is a database which is designed for visual object detection research. It contains over 14 million URLs of images. Since 2010, every year ImageNet publishes competition about visual recognition. The unexpected solution shows up in 2010 ImageNet Large Scale Visual Recognition Challenge. A convolutional neural network, which is developed with CUDA for GPU support achieved a winning top 5 test error rate of 15.3% and achieved by the best second-best entry. Its name is ALEXNET. ALEXNET has an incredible impact on the machine learning field. Especially it is the deep learning application on machine vision field. The popularity of deep learning in computer vision field increases over years, since 2012. [13][15]

1.3. Object Detection Methods

1. Hog Features

In 2005 a paper published by Navneet Dalal and Bill Triggs. It is adopting the linear SVM base model. In theory, model converts pixel –based representation into gradient-based and feeds them SVM for classifying.

To be more clear algorithm illustrated in Figure 10. [22]

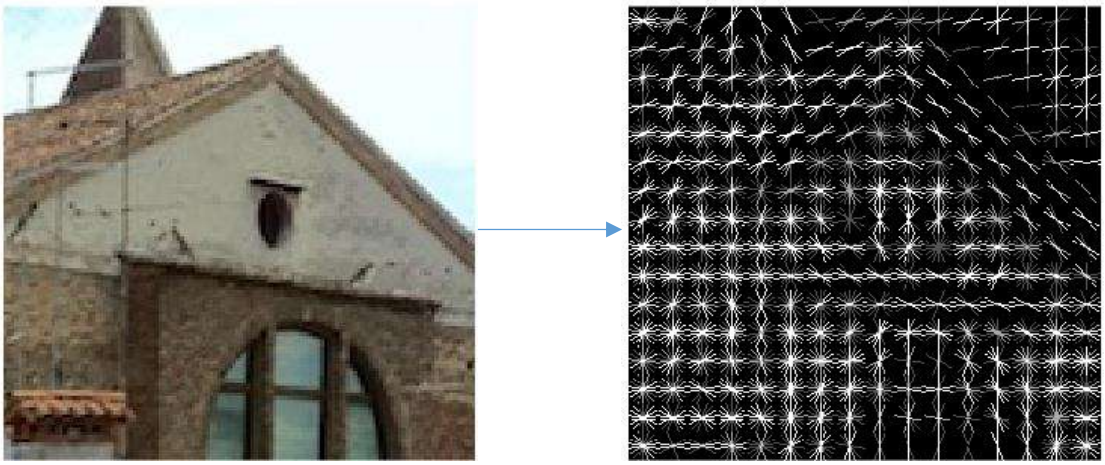

Figure 10: Hog Features

Histogram of Oriented Gradients features are inexpensive on computation power and it can be used for real-time object detection, face detection.

2. Region-based Convolutional Neural Networks (R-CNN)

Histogram of Oriented Gradients features are inexpensive on computation power and it can be used for real-time object detection, face detection.

You can see the visualization of selective search algorithm in Figure 11. [4]

Figure 11: Selective Search

Selective Search uses clues like texture, color etc. to find possible location of the object in the image.

R-CCN has 3 steps:

- Using selective search algorithm, it scans input image and generates ~2000 possible region proposals
- Runs convolutional Neural network (CNN) for each these regions
- Take outputs for each CNN and feed all into SVN to classify the region after that a linear regression for box creation around the objects

These steps are illustrated in Figure 12:

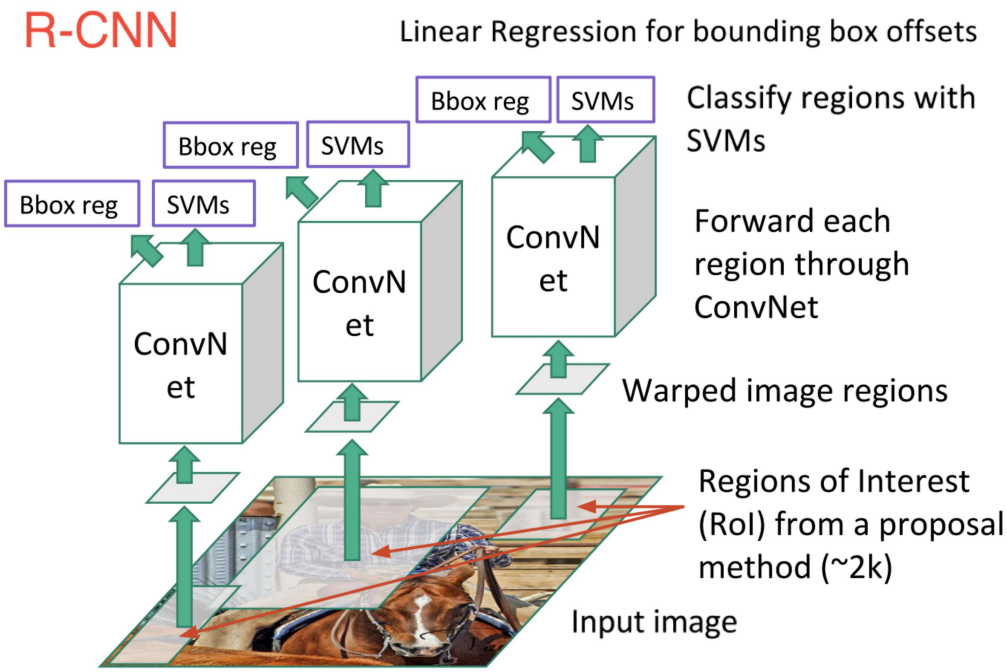

Figure 12: Region-based Convolutional Neural Networks (R-CNN)

R-CCN's are intuitive but slow. [2][3][12]

3. Fast Region-based Convolutional Network (Fast R-CNN)

In 2015, Ross Girshick developed Fast R-CNN. If we compare R-CNN and Fast R CNN, the main idea is the same but Fast R-CNN has better time consumption.

Differences:

- Fast R-CNN performs feature extraction just before the region proposing. So Instead of feeding all regions to CNN, running just one CNN.
- Replacing SVM with softmax layer.

These steps are illustrated in Figure 13

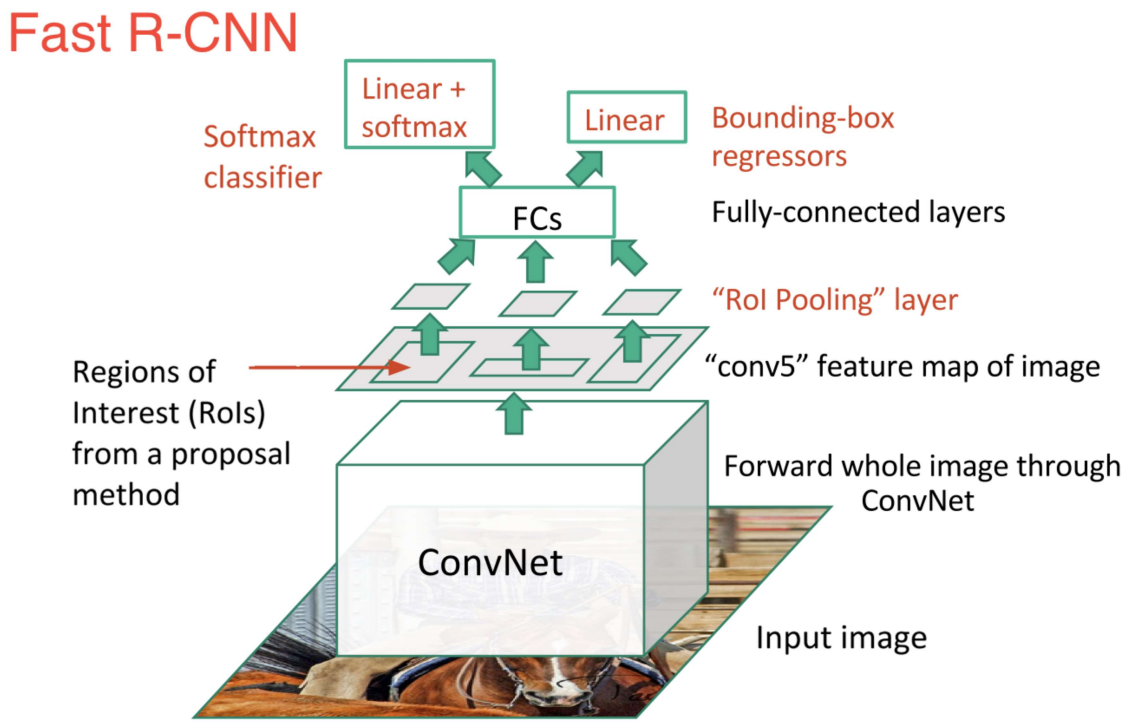

Figure 13: Fast Region-based Convolutional Network (Fast R-CNN)

Fast R-CNN performs much better than R-CNN. [3][2][1][12]

4. Faster Region-based Convolutional Network (Faster R-CNN)

In 2016, Shaoqing Ren developed Faster R-CNN. The only difference between Fast R-CNN was selective search algorithm. Faster R-CNN replaces selective search algorithm with a convolutional network which name is Region Proposal Network. Selective Search is the slowest part of previous algorithms. With RPN approach time consumption is better. [2]

These steps are illustrated in figure 14.

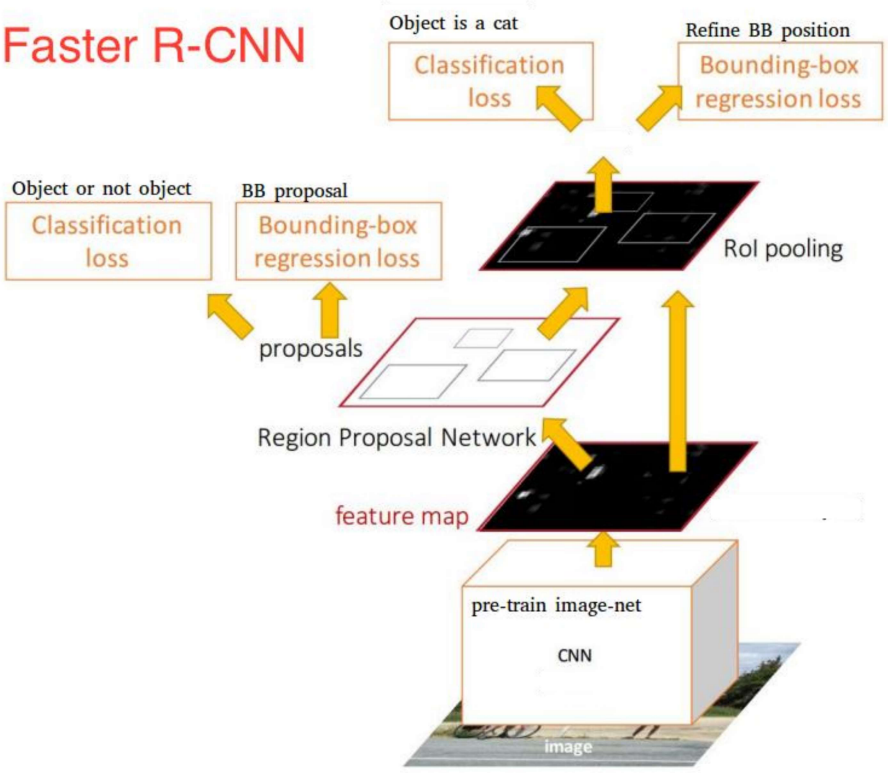

Figure 14: Faster Region-based Convolutional Network (Faster R-CNN)

5. Region-based Fully Convolutional Network (R-FCN)

In 2016, Jifeng Dai developed R-FCN. R-FCN is a model which has only convolutional layers. Developers merged location invariant and location variant in a single model. [1]

R-FCN Steps:

- Model takes input

- Generates a score bank. Last layer outputs feature maps. These maps are unique in the detection of specific image and location. This is called position sensitive score maps.
- RPN to generate the region of interest.
- For each region of interest, model divides are pieces or sub regions which is score maps
- For each sub-regions model checks score bank to see sub region matches of some object. For example, a cat is lower right. We will grab score maps which are related lower right.

An Example illustrated in Figure 14:

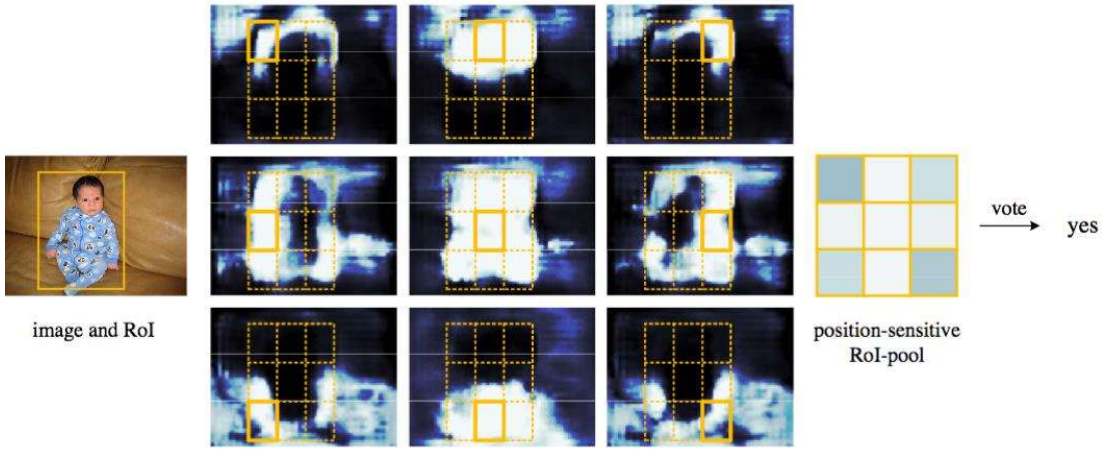

Figure 14: Region-based Fully Convolutional Network (R-FCN) yes vote

We can see that model trying to detect a person. Sub regions in feature maps related to a person. If its true algorithm votes yes. And identifies the person and its location.

In Figure 15, we can see that region of interest could not find object properly. It's shifted right and cannot center person. So the final vote will be no.

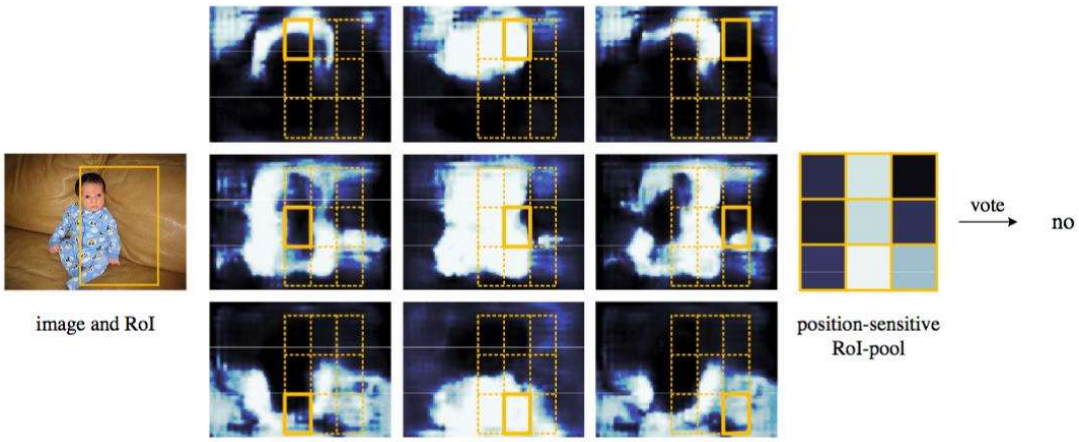

Figure 15: Region-based Fully Convolutional Network (R-FCN) no vote

6. YOLO: You only Look Once

In 2016, J.Redmon et al. developed a model. Its approaches on object recognition problems were different than other methods.

Yolo Detection System:

- Model takes input image
- Resizes the input image 448 X 448
- Runs a single convolutional network.
- Thresholds detectors by model's confidence.

Thus process illustrated in Figure 16:

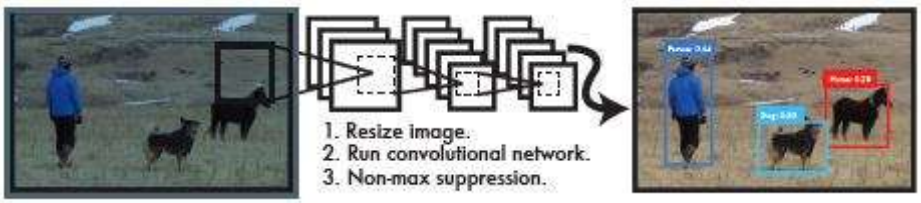

Figure 16: Yolo algorithm steps

Yolo divides each image into an S x S grid. Each grid predicts N bounding boxes and confidence. We can say that confidence is the synonym of the accuracy of bounding boxes.

For example, we have boxes predicted. If the box does not contain any image its score must be lower. So let's set our threshold %40 confidence. Output illustrated in Figure 17.

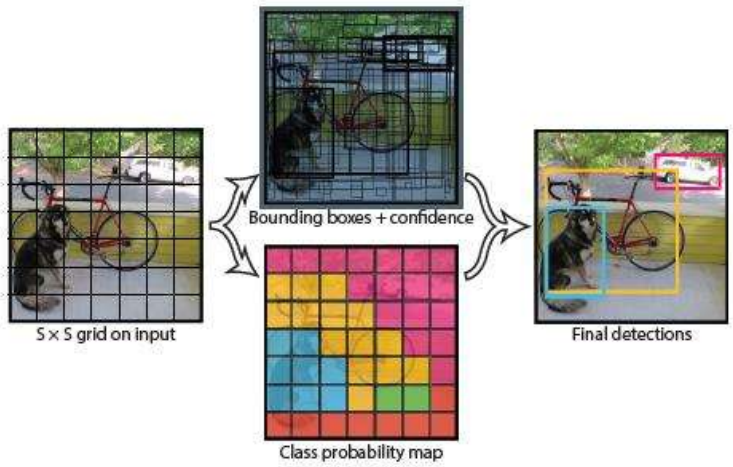

Figure 17: Yolo algorithm class probability map and bounding boxes

As we know current detection systems transforms image classifiers to perform object detection. Yolo acts different. IT reframes object detection like a regression problem so it's fast and don't have complex pipeline. YOLO base network can run 45 frame per second. It means that yolo can be used for real time object detection. [27][28]

7. SSD: Single Shot Detector

In 2016, Lui, W. et al. developed single shot detector model. It has similarities to YOLO. But main difference is SSD have feature pyramid and its decision making process similar to Faster-RCNN. [29]

SSD does, regions of interest and region classification in "single shot". It predicts bounding box and class at the same time.

SSD steps:

- Passes input images a series of convolutional layers in different scales. For example, 10 x 10 then 6 x 6
- Like Faster R-CNN model uses 3x3 convolutional filter to create small bounding boxes

- Model predicts bounding box and class simultaneously
- Predicted boxes base on Jaccard index. Best predicted box marked a "positive"

SSD steps illustrated in Figure 18:

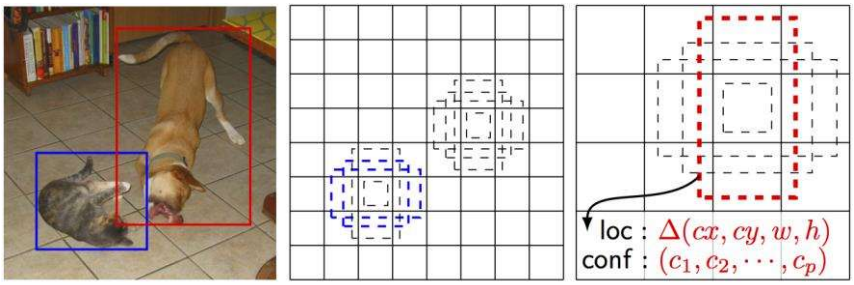

Figure 18: single shot detector algorithm steps

2. ABOUT DATA

SpaceNet is a commercial satellite imagery and labeled training data to foster innovation in the development of computer vision algorithms. DigitalGlobe, CosmiQ Works, and NVIDIA have partnered to release the SpaceNet data set to the public to enable developers and data scientists.

2.1. Data Access

SpaceNet dataset hosted as an Amazon Web Services (AWS) Public Dataset. The AWS Command Line Interface (CLI) must be installed with an active AWS account. A sample of directory tree of dataset is below.

```
── AOI_3_Paris_Train.tar.gz
        │    ├── geojson
        │    │   └── buildings  # Contains GeoJson labels of buildings for each tile
        │    ├── MUL             # Contains Tiles of 8-Band Multi-Spectral raster data from WorldView-3
        │    ├── MUL-PanSharpen # Contains Tiles of 8-Band Multi-Spectral raster data pansharpened to 0.3m
        │    ├── PAN          # Contains Tiles of Panchromatic raster data from Worldview-3
```

│ ├── RGB-PanSharpen # Contains Tiles of RGB raster data from Worldview-3
│ └── summaryData # Contains CSV with pixel based labels for each building in the Tile Set.

Example of dataset training directory illustrated in figure 19:

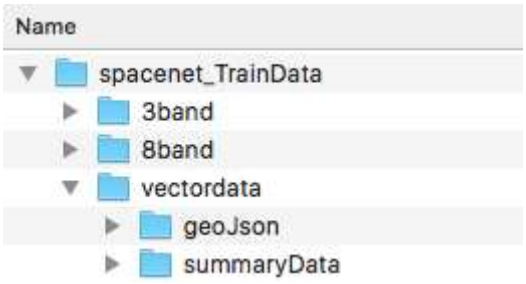

Figure 19: SpaceNet Sample Directory

2.2. Details About Data

3-band imageries are pan-sharpened and they are 438–439 pixels in width, and 406–407 pixels in height. 8-band images have not been pan-sharpened and so have 1/4 the resolution of the 3-band imagery at 110 x 102 pixels. Each image has unique Id and we can find related label record in vectordata/geoJson directory. geoJSON file has polygon vertices in latitude and longitude. Computer vision algorithms operate in pixel spaces. Because of that, the label data must be converted to a matrix of pixel positions. We will explain three methods of converting GeoJSON label files into pixel coordinates. [31][33][35]

Example of JSON label file for img950:

```
{
    "type": "FeatureCollection",
    "crs": {
        "type": "name",
        "properties": {
            "name": "urn:ogc:def:crs:OGC:1.3:CRS84"
        }
    },
    "features": [{
                "type": "Feature",
                "properties": {
                    "timestamp": "2016-06-22T21:27:50Z",
                    "version": "1",
                    "changeset": "5404",
                    "user": "Derick",
                    "uid": "43",
                    "HGIS_OID": "1182422.0",
```

```
                    "building": "yes",
                    "type": "None",
                    "id": "way\/75286",
                    "area": "None",
                    "QAStatus": "Original_Building",
                    "HGISOID": 1182422.000000,
                    "TaskArea": "West",
                    "Revision1": "No",
                    "Shape_Leng": 0.000500,
                    "Shape_Area": 0.000000,
                    "partialBuilding": 0.000000,
                    "partialDec": 1.000000
                },
                "geometry": {
                    "type": "Polygon",
                    "coordinates": [
                        [
                            [-43.715911899999981, -22.894361599999968, 0.0],
                            [-43.715955299999962, -22.894427499999949, 0.0],
                            [-43.716095199999984, -22.894322599999953, 0.0],
                            [-43.716051799999946, -22.894261799999981, 0.0],
                            [-43.715911899999981, -22.894361599999968, 0.0]
                        ]
                    ]
                }
            }
```

1. Building Outline Coordinates

As I mention before JSON files has building polygon vertices in latitude and longitude. To be able to use these coordinates, we have to convert them into pixel coordinates. This kind of process could be done with the GDAL library. [31][33][35]

Example of code: *Outline Coordinates:*

Now we have pixel coordinates and we can add another layer our input images to reveal polygons that we have pixel coordinates.

Example of code: *Plot Coordinates:*

Input image illustrated in Figure 19:

Figure 19: Input image for Ground Truth Polygons demonstration

Ground Truth Polygons image illustrated in Figure 20:

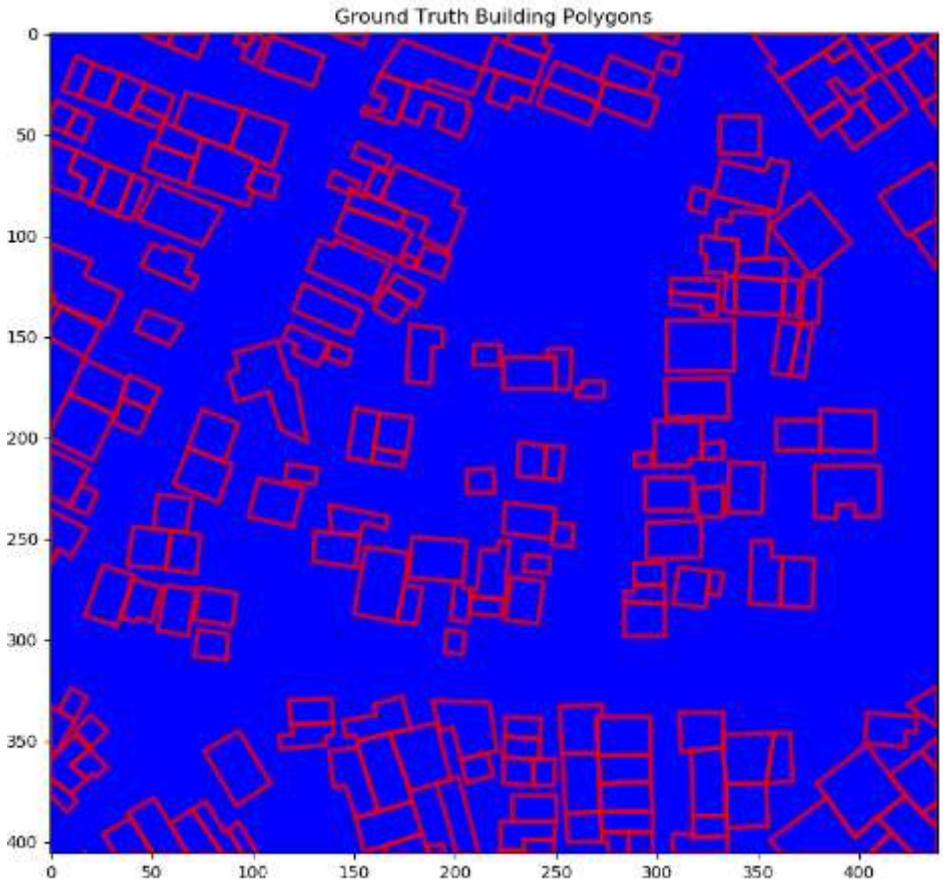

Figure 20: Ground Truth of Building Polygons example

2. Building Mask

Image masks are useful for neural network segmentation algorithms. Building masks illustrated below: [31][33][35]

Example of code: *Building Mask:*

Input Image for building mask example illustrated in Figure 21:

Figure 21: Input image for building mask example

Ground Truth Building Mask image example illustrated in figure 22:

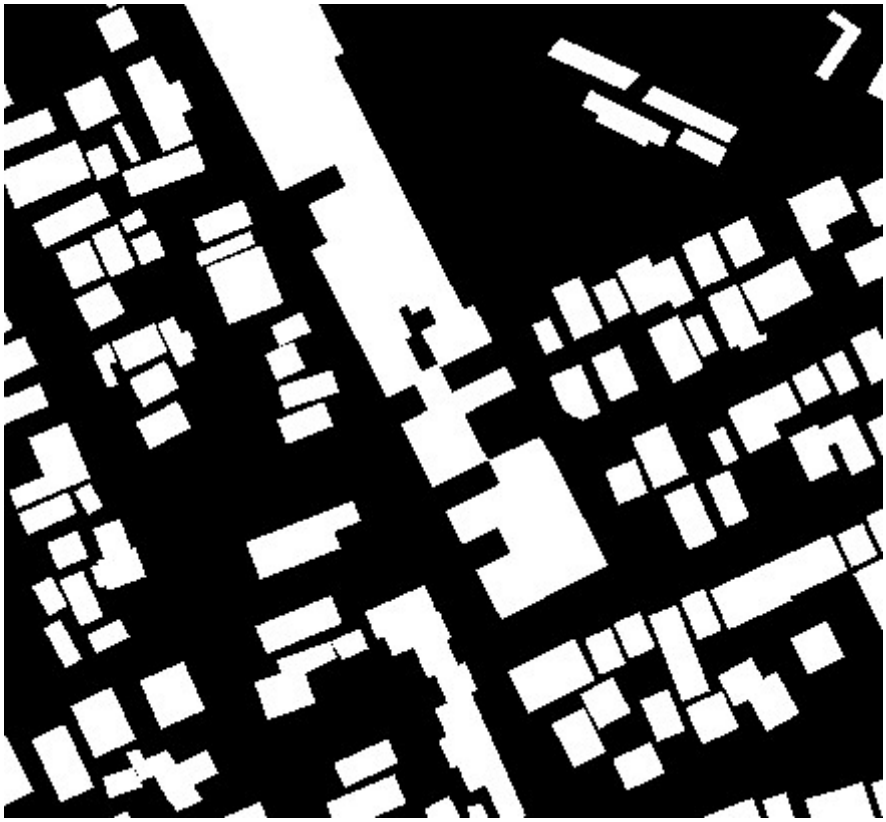

Figure 22: Output image of building mask

3. Signed Distance Transform

In 2016, Juangye Yuan published a paper which is about building extraction and signed distance transform. This is another method for labeling ground truth.

Example of Code: *Signed Distance Transform:*

Input Image for signed distance transform example illustrated in Figure 23:

Figure 23: Input image for Signed Distance Transform example

Signed Distance Transformed Image illustrated in Figure 24:

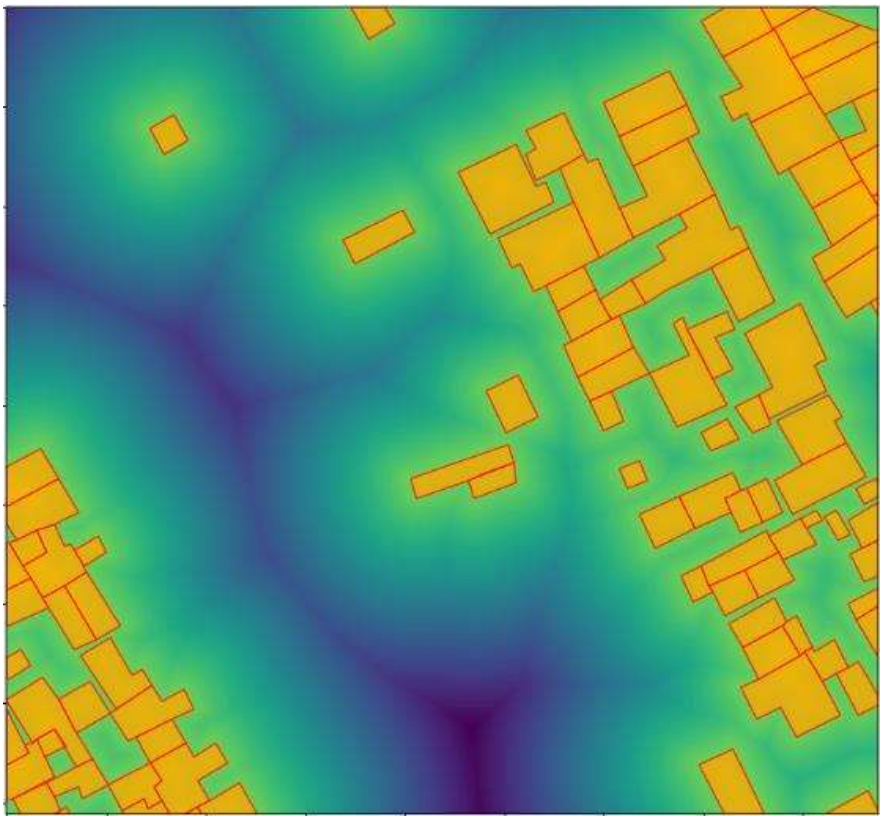

Figure 24: Signed Distance Transformed Image

3. PROJECT DEFINITION

In this section we will discuss the objective and scope of project.

3.1. Problem Statement

Geospatial industry getting bigger and bigger these days in addition makes massive amount of data. There is not automated solution that extracts map features such as roads, building footprints from satellite imagery.

3.2. Project Objective

Finding automated method for extracting building footprints from satellite imagery. I will visualize it by creating polygonal areas which represents buildings.

3.3. Project Scope

Scope of project covers only building footprint extraction.

4. METHODOLOGY

To solve this problem, we used Convolutional Neural Network. It's based on Semantic Segmentation of Small Objects and Modeling of Uncertainty in Urban paper [34]. Due to the lack of computer power. We decided to use 1 image with shredded in pieces. Modeling process has 2 steps. These steps are especially data preparation section are implemented from SpaceNet Challenge 1 – Rio de Janeiro Building Footprint Extraction solution. [35]

4.1. Data Preparation

In previous sections, we explain how we can use geojson files. In this section, we used GDAL, shapely, CV2 Python packages to create input and target numpy arrays. In general, we create 2 numpy array. Input and target for training. For more training data we imputed images.

Firstly, we read 3Band image file, applied polygons on that file using the geojson file and convert it to numpy array. This was our target. To input image, we used 3Band and 8Band images and created 4d dimension numpy array.

Here is some detail about process, we read 3band image and get geo data using GetGeoTransform() method from GDAL library. To draw a polygon, we converted geo transformation point list and feed them to CV2 package drawContours function. After that

process we create target numpy array. For input image we normalized and create numpy array for training.

The second step is about image imputing. So we could have more training data. Each 512*512 images are imputed into 128*128 images with %50 overlap. So we have 49 numpy array file for training model.

Example code: *Data Preparation*

4.2. Model Creation

We used the approach that used by Tensorflow in Build a Convolutional Neural Network using Estimators tutorial. [36]

Our input layer has 4 parameters. Batch size, image height, image width, and channels. Our image size is 128 x 128, we have 3 channel (red, green blue) we don't use 1 because our image is not monochrome.

CNN architecture:

1- Convolutional layer #1: Applies 32 3x3 filters. With ReLU activation function
2- Pooling Layer #1: Performs max pooling using 2x2 filter
3- Convolutional layer #2: Applies 64 3x3 filters. With ReLU activation function
4- Pooling Layer #2: Performs max pooling using 2x2 filter.
5- Convolutional layer #3: Applies 128 3x3 filters. With ReLU activation function
6- Pooling Layer #3: Performs max pooling using 2x2 filter.
7- Convolutional layer #4: Applies 1024 5x5 filters. With ReLU activation function
8- Convolutional layer #5: Applies 128 1x1 filters. With ReLU activation function
9- Deconvolution layer #1: 2 class (building or not building) 16 x 16 with 8 x 8 subsample
10- Softmax activation.

CNN architecture illustrated in Figure 28:

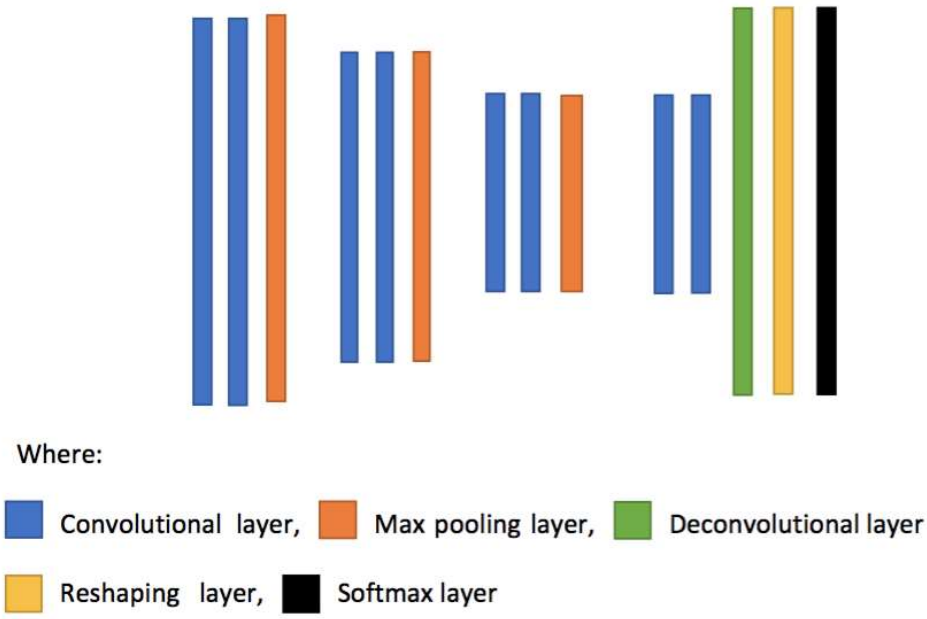

Figure 28: CNN Architecture

In this figure, blue layers represent convolutional layers, plus these layers have Relu and batch-normalization layers. In this network we have 8 convolutional layers. Red layers represent pooling layers. Green layer represents the fractional-strided convolution layer. We can call it deconvolution layer too. Final layer which is black colored represents softmax layer, it classifiers pixel into building or not building.

Model Diagram illustrated Figure 25:

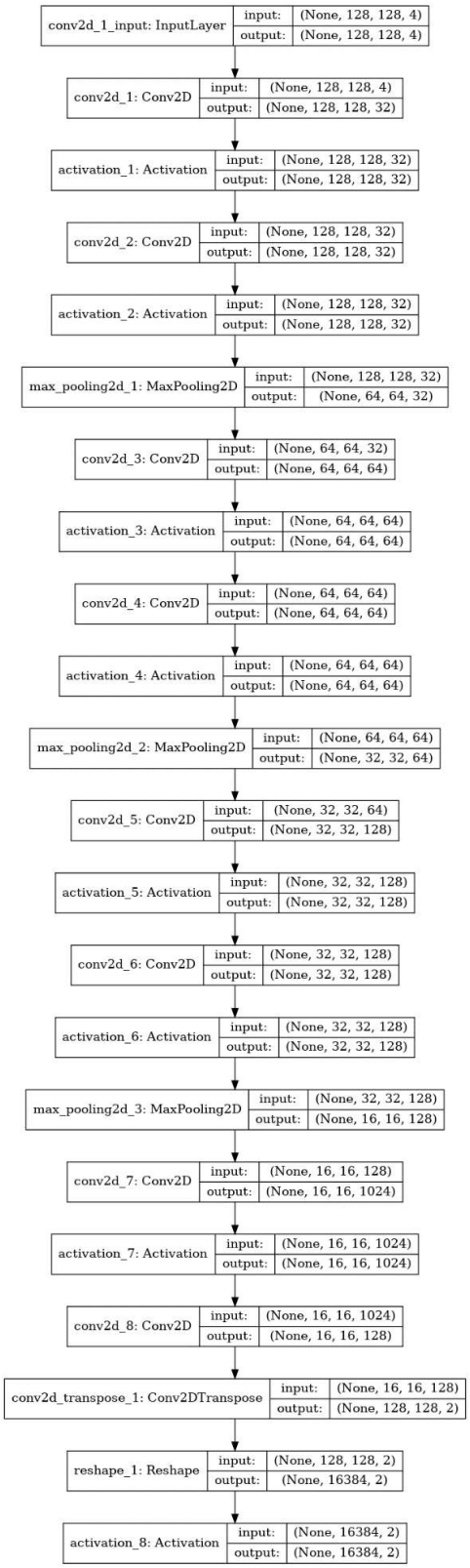

Figure 25: Model Diagram

Example Model Code: *Create Model*

5. RESULTS

5.1. Model output

As we mentioned before we used 1 image for training model because of lack of computing power and time. Training process with epoch parameter 200 took 2 hours with NVIDIA GTX 970 GPU. Final model weights ~ 200MB (from 1 image training) and the final model accuracy score is approximately 0.16.

Here are the parameters of final CNN network:

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 128, 128, 32)	1184
activation_9 (Activation)	(None, 128, 128, 32)	0
conv2d_10 (Conv2D)	(None, 128, 128, 32)	9248
activation_10 (Activation)	(None, 128, 128, 32)	0
max_pooling2d_4 (MaxPooling2	(None, 64, 64, 32)	0
conv2d_11 (Conv2D)	(None, 64, 64, 64)	18496
activation_11 (Activation)	(None, 64, 64, 64)	0
conv2d_12 (Conv2D)	(None, 64, 64, 64)	36928
activation_12 (Activation)	(None, 64, 64, 64)	0
max_pooling2d_5 (MaxPooling2	(None, 32, 32, 64)	0
conv2d_13 (Conv2D)	(None, 32, 32, 128)	73856
activation_13 (Activation)	(None, 32, 32, 128)	0
conv2d_14 (Conv2D)	(None, 32, 32, 128)	147584

```
_________________________________________________________________
activation_14 (Activation)   (None, 32, 32, 128)       0
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 16, 16, 1024)      3277824
_________________________________________________________________
activation_15 (Activation)   (None, 16, 16, 1024)      0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 16, 16, 128)       131200
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 128, 128, 2)       65538
_________________________________________________________________
reshape_2 (Reshape)          (None, 16384, 2)          0
_________________________________________________________________
activation_16 (Activation)   (None, 16384, 2)          0
=================================================================
==
Total params: 3,761,858
Trainable params: 3,761,858
Non-trainable params: 0
_________________________________________________________________
```

First convolutional block has 32 filters with 2x2 pooling. Second block has 64 filters; third block has 128 filters. Forth layer has 1024 filters, fifth layer has 128 filters. [34][35]

Predicted building footprints illustrated below:

Input image illustrated in Figure 26:

Figure 26: Input image for model building footprint prediction demonstration

Building footprints as an output of model illustrated in Figure 27:

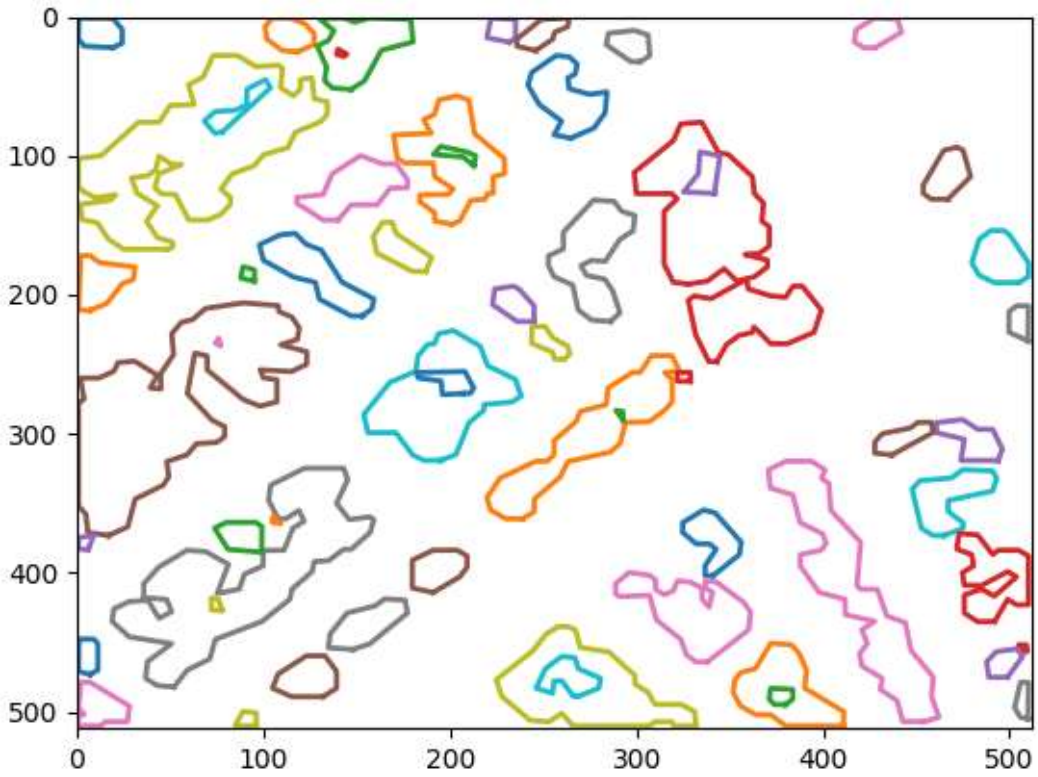

Figure 27: Building Footprint model predicted output

5.2. Conclusion

We can see that our accuracy score is too low 0.16 but the model prediction is not bad for standalone buildings. In addition, prediction comes from the model which is trained from 1 image so this is not bad. Results are promising.

In this paper, we deep dive into object recognition and deep learning algorithms. In addition, Deep learning in Geographic industry. Deep learning applications in geographic industry are too new. The oldest paper on this topic is 2 years old and new ideas show up. Even the preparation process of this paper new object detection algorithm released. (You Only Look Twice). Moreover, lead companies like NVIDIA, Digital Globe are supporting and investing the satellite imagery industry.

Creating new neural network especially imagery subject is too expensive. Training model consumes time and computing power. For example, the size of the area of interest training dataset

for Vegas is 30 Gb. So as another approach to the problem instead of creating a new network from scratch. Pre-Trained models can be used to solve problems.

The challenging part of using pre-trained models is data preparation. For example, for machine learning techniques most of the time your data will be CSV file. Let's say that we use python. To create a new model, we will read data with pandas, clear data and feed it to some of the scikit-learn algorithms. On the other hand, deep learning is different most of the pre-trained models uses public datasets for the training process. So to use that model you have to convert your data to that formats.

Dataset types:

1. PASCAL VOC2012
2. DARKNET
3. Segmentation Boundaries Dataset (SBD)

In addition, for most of the models this transformation is not enough. For example, to use tensorflow pre-trained models like inceptionv3 or mobilenet we have to create generate tenseorflow records types of data (TFRecord type).

6. REFERENCES

1. Dai, K., Li, Y., He, K., Sun, J. (2016). R-FCN: Object Detection via Region-based Fully Convolutional Networks. https://arxiv.org/abs/1605.06409
2. Ren, S., He, K., Girshick, R., Sum, J. (2016). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. https://arxiv.org/abs/1506.01497
3. Girshick, R. (2015). Fast R-CNN. https://arxiv.org/abs/1504.08083
4. Uijlings, J.R.R, van de Sande, K.E.A., Smeulders. A.W.M. (2012). Selective Search for Object Recognition. http://www.huppelen.nl/publications/selectiveSearchDraft.pdf
5. Dalal, H., Triggs, Bill. (2005). Histograms of Oriented Gradients for Human Detection. https://hal.inria.fr/inria-00548512

6. Alom, Z., Taha, T., Yakopcic, C., Westberg, S. Hasan, M., Van Esesn, B., Awwal, A., Asari, V. (2018). The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches. https://arxiv.org/abs/1803.01164
7. Van Etten, A. (2018). You Only Look Twice: Rapid Multi-Scale Object Detection In Satellite Imagery. https://arxiv.org/abs/1805.09512
8. Shantaiya, S. Verma, K., Mehta, Kamal. (2013). A Survey on Approaches of Object Detection. https://www.semanticscholar.org/paper/A-Survey-on-Approaches-of-Object-Detection-Shantaiya-Verma/40f0394ac879d54c4d4f4fad31e24d33e3aca155
9. Tang, S., Yuan, Y. (2017). *Object Detection based on Convolutional Neural Network.* http://cs231n.stanford.edu/reports/2015/pdfs/CS231n_final_writeup_sjtang.pdf
10. Krizhevsky, Alex., Sutskever, I., Hinton, G. (2012). *ImageNet Classification with Deep Convolutional.*http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
11. Ouaknine, Arthur. (2018, Feburary 5). Review of Deep Learning Algorithms for Object Detection. Retrieved from https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852
12. Sachan, A. (2018). Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN,YOLO,SSD Retrived from http://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/
13. Xu, J. (2017, August 15). An Intuitive Guide to Deep Network Architectures. Retrieved from: https://towardsdatascience.com/an-intuitive-guide-to-deep-network-architectures-65fdc477db41
14. Olah, C. (2014, July 8). Conv Nets: A Modular Perspective Retrieved from https://colah.github.io/posts/2014-07-Conv-Nets-Modular/
15. Deshpande, A. (2016, July 20). A Beginner's Guide to Understanding Convolutional Neural Networks Part-1. Retrieved from: https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/
16. Deshpande, A. (2016, July 29). A Beginner's Guide to Understanding Convolutional Neural Networks Part-2. Retrieved from

https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

17. Deshpande, A. (2016, August 24). The 9 Deep Learning Papers You Need To Know About. Retrieved from https://adeshpande3.github.io/adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html
18. Dar, P., (2018, May). Don't miss out on these awesome GitHub Repositories & Reddit Threads for Data Science & Machine Learning. Retrieved from: https://www.analyticsvidhya.com/blog/2018/06/top-5-github-reddit-data-science-machine-learning-may-2018/
19. Gupta, V. (2017, September 25). Deep learning using KERAS – The Basics. Retrieved from https://www.learnopencv.com/deep-learning-using-keras-the-basics/
20. Shaikh, F., (2018, June 28). Understanding and Building an Object Detection Model from Scratch in Python. Retrieved from https://www.analyticsvidhya.com/blog/2018/06/understanding-building-object-detection-model-python/
21. Gao, H. (2017, August 8). A Walk-through of AlexNet. Retrieved from https://medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637
22. HOG Features. (2017). Retrieved from http://www.vlfeat.org/overview/hog.html
23. Image Net. (2016). Retrieved from http://image-net.org/about-overview
24. AlexNet. (2018, June 8). Retrieved from https://en.wikipedia.org/wiki/AlexNet
25. ImageNet. (2018, June 12). Retrieved from: https://en.wikipedia.org/wiki/ImageNet#ImageNet_Challenge
26. Histogram of oriented gradients. (2018, February 27). Retrieved from https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients
27. Redmon, J., Divvala, S., Girshick, R., Farhadi, A. (2016, May 9). You Only Look Once: Unified, Real-Time Object Detection. Retrieved from https://arxiv.org/abs/1506.02640
28. Redmon, J. (2018). YOLO: Real-Time Object Detection. Retrieved from https://pjreddie.com/darknet/yolo/

29. Lui, W., Anguelow, D., Erhan D., Szegedy, C., Reed, S., Fu, C., Berg, A. (2016, December 16). SSD: Single Shot MultiBox Detector. Retrieved from https://arxiv.org/abs/1512.02325
30. Jaccard index (2018, June 7). Retrieved from https://en.wikipedia.org/wiki/Jaccard_index
31. Etten, A. (2017, January 5). Getting Started with SpaceNet Data. https://medium.com/the-downlinq/getting-started-with-spacenet-data-827fd2ec9f53
32. Noh, H., Hong, S., Han, B. (2015, May 17). Learning Deconvolution Network for Semantic Segmentation. https://arxiv.org/abs/1505.04366
33. Yuan, J., (2016, February 21). Automatic Building Extraction in Aerial Scenes Using Convolutional Networks. https://arxiv.org/abs/1602.06564
34. Kampffmeyer, M., Salberg, A., Jenssen, R., (2016, July 1). Semantic Segmentation of Small Objects and Modeling of Uncertainty in Urban. https://ieeexplore.ieee.org/document/7789580/
35. SpaceNet Challenge 1 – Rio de Janeiro Building Footprint Extraction. (2017). https://github.com/SpaceNetChallenge/BuildingDetectors/blob/master/bic-user/src/crop_feats.py
36. Build a Convolutional Neural Network using Estimators. (2018, July 30). https://www.tensorflow.org/tutorials/estimators/cnn

7. APPENDIX

Outline Coordinates:

```
from matplotlib.collections import PatchCollection
from osgeo import gdal, osr, ogr, gdalnumeric
from matplotlib.patches import Polygon
import matplotlib.pyplot as plt
import numpy as np
import shutil
import json
import glob
import sys
import os


# Spacenet utilities library:
https://github.com/SpaceNetChallenge/utilities

path_to_spacenet_utils =
'/home/obi/gitProjects/utilities/python/spaceNetUtilities'
spacenet_data_dir = '/home/obi/deeplearning/processedBuildingLabels'
spacenet_explore_dir = '/home/obi/deeplearning/LabelingFiles'



# import packages
sys.path.extend([path_to_spacenet_utils])
import geoTools as gT


# This function takes 2 argument and transforms geojson file to point of
pixels
# raster_file input is .tff file, geojson_file is location of json label
file.



def geojson_to_pixel_arr(raster_file, geojson_file):


    # load geojson file
    with open(geojson_file) as f:
        geojson_data = json.load(f)

    # load raster file and get geo transforms
    src_raster = gdal.Open(raster_file)
    targetsr = osr.SpatialReference()
    targetsr.ImportFromWkt(src_raster.GetProjectionRef())

    geom_transform = src_raster.GetGeoTransform()

    # get latlon coords
    latlons = []
    types = []
```

```
    for feature in geojson_data['features']:
        coords_tmp = feature['geometry']['coordinates'][0]
        type_tmp = feature['geometry']['type']

        print "feature['geometry']['coordinates'][0]", z
        latlons.append(coords_tmp)
        types.append(type_tmp)
        print feature['geometry']['type']

    # convert latlons to pixel coords
    pixel_coords = []
    latlon_coords = []
    for i, (poly_type, poly0) in enumerate(zip(types, latlons)):

        if poly_type.upper() == 'MULTIPOLYGON':
            #print "oops, multipolygon"
            for poly in poly0:
                poly=np.array(poly)
                print ("poly.shape:", poly.shape)

                # account for nested arrays
                if len(poly.shape) == 3 and poly.shape[0] == 1:
                    poly = poly[0]

                poly_list_pix = []
                poly_list_latlon = []
                print ("poly", poly)
                for coord in poly:
                    print ("coord:", coord)
                    lon, lat, z = coord
                    px, py = gT.latlon2pixel(lat, lon,
input_raster=src_raster,
                                             targetsr=targetsr,
                                             geom_transform=geom_transform)
                    poly_list_pix.append([px, py])
                    print ("px, py", px, py)
                    poly_list_latlon.append([lat, lon])

                    ptmp = np.rint(poly_list_pix).astype(int)

        elif poly_type.upper() == 'POLYGON':
            poly=np.array(poly0)
            print ("poly.shape:", poly.shape)

            # account for nested arrays
            if len(poly.shape) == 3 and poly.shape[0] == 1:
                poly = poly[0]

            poly_list_pix = []
            poly_list_latlon = []
            print ("poly", poly)
            for coord in poly:
                print ("coord:", coord)
                lon, lat, z = coord
                px, py = gT.latlon2pixel(lat, lon,
input_raster=src_raster,
                                         targetsr=targetsr,
```

```
                                        geom_transform=geom_transform)
                poly_list_pix.append([px, py])
                print ("px, py", px, py)
                poly_list_latlon.append([lat, lon])

                ptmp = np.rint(poly_list_pix).astype(int)

            pixel_coords.append(ptmp)
            latlon_coords.append(poly_list_latlon)

        elif poly_type.upper() == 'POINT':
            print ("Skipping shape type: POINT in
geojson_to_pixel_arr()")
            continue

        else:
            print ("Unknown shape type:", poly_type, " in
geojson_to_pixel_arr()")
            return

    return pixel_coords, latlon_coords
```

Plot Coordinates:

```
from matplotlib.collections import PatchCollection
from matplotlib.patches import Polygon
import matplotlib.pyplot as plt
import numpy as np


#This function takes input images, pixel coordinates. and returns ground
truth buildings

def plot_truth_coords(input_image, pixel_coords,
                  figsize=(8,8), plot_name='',
                  add_title=False, poly_face_color='orange',
                  poly_edge_color='red', poly_nofill_color='blue',
cmap='bwr'):

    if add_title:
        suptitle = fig.suptitle(plot_name.split('/')[-1],
fontsize='large')

    # create patches
    patches = []
    patches_nofill = []
    if len(pixel_coords) > 0:
        # get patches
        for coord in pixel_coords:
            patches_nofill.append(Polygon(coord,
facecolor=poly_nofill_color,
                                          edgecolor=poly_edge_color,
lw=3))
```

```
            patches.append(Polygon(coord, edgecolor=poly_edge_color,
fill=True,
                                   facecolor=poly_face_color))
        p0 = PatchCollection(patches, alpha=0.25, match_original=True)
        p2 = PatchCollection(patches_nofill, alpha=0.75,
match_original=True)


    # truth polygons
    zero_arr = np.zeros(input_image.shape[:2])

    ax1.imshow(zero_arr, cmap=cmap)
    if len(patches) > 0:
        ax1.add_collection(p2)
    ax1.set_title('Ground Truth Building Polygons')


    plt.tight_layout()
    if add_title:
        suptitle.set_y(0.95)
        fig.subplots_adjust(top=0.96)
    plt.show()
```

Building Mask:

```
def create_building_mask(rasterSrc, vectorSrc, npDistFileName='',
                             noDataValue=0, burn_values=1):


    ## open source vector file that truth data
    source_ds = ogr.Open(vectorSrc)
    source_layer = source_ds.GetLayer()

    ## extract data from src Raster File to be emulated
    ## open raster file that is to be emulated
    srcRas_ds = gdal.Open(rasterSrc)
    cols = srcRas_ds.RasterXSize
    rows = srcRas_ds.RasterYSize

    ## create First raster memory layer, units are pixels
    # Change output to geotiff instead of memory
    memdrv = gdal.GetDriverByName('GTiff')
    dst_ds = memdrv.Create(npDistFileName, cols, rows, 1, gdal.GDT_Byte,
                           options=['COMPRESS=LZW'])
    dst_ds.SetGeoTransform(srcRas_ds.GetGeoTransform())
    dst_ds.SetProjection(srcRas_ds.GetProjection())
    band = dst_ds.GetRasterBand(1)
    band.SetNoDataValue(noDataValue)
    gdal.RasterizeLayer(dst_ds, [1], source_layer,
burn_values=[burn_values])
    dst_ds = 0

    return
```

Signed Distance Transform:

```
def create_dist_map(rasterSrc, vectorSrc, npDistFileName='',
                            noDataValue=0, burn_values=1,
                            dist_mult=1, vmax_dist=64):


    source_ds = ogr.Open(vectorSrc)
    source_layer = source_ds.GetLayer()


    srcRas_ds = gdal.Open(rasterSrc)
    cols = srcRas_ds.RasterXSize
    rows = srcRas_ds.RasterYSize

    geoTrans, poly, ulX, ulY, lrX, lrY = gT.getRasterExtent(srcRas_ds)
    transform_WGS84_To_UTM, transform_UTM_To_WGS84, utm_cs \
                                                    =
gT.createUTMTransform(poly)
    line = ogr.Geometry(ogr.wkbLineString)
    line.AddPoint(geoTrans[0], geoTrans[3])
    line.AddPoint(geoTrans[0]+geoTrans[1], geoTrans[3])

    line.Transform(transform_WGS84_To_UTM)
    metersIndex = line.Length()

    memdrv = gdal.GetDriverByName('MEM')
    dst_ds = memdrv.Create('', cols, rows, 1, gdal.GDT_Byte)
    dst_ds.SetGeoTransform(srcRas_ds.GetGeoTransform())
    dst_ds.SetProjection(srcRas_ds.GetProjection())
    band = dst_ds.GetRasterBand(1)
    band.SetNoDataValue(noDataValue)

    gdal.RasterizeLayer(dst_ds, [1], source_layer,
burn_values=[burn_values])
    srcBand = dst_ds.GetRasterBand(1)

    memdrv2 = gdal.GetDriverByName('MEM')
    prox_ds = memdrv2.Create('', cols, rows, 1, gdal.GDT_Int16)
    prox_ds.SetGeoTransform(srcRas_ds.GetGeoTransform())
    prox_ds.SetProjection(srcRas_ds.GetProjection())
    proxBand = prox_ds.GetRasterBand(1)
    proxBand.SetNoDataValue(noDataValue)

    opt_string = 'NODATA='+str(noDataValue)
    options = [opt_string]

    gdal.ComputeProximity(srcBand, proxBand, options)

    memdrv3 = gdal.GetDriverByName('MEM')
    proxIn_ds = memdrv3.Create('', cols, rows, 1, gdal.GDT_Int16)
    proxIn_ds.SetGeoTransform(srcRas_ds.GetGeoTransform())
    proxIn_ds.SetProjection(srcRas_ds.GetProjection())
    proxInBand = proxIn_ds.GetRasterBand(1)
    proxInBand.SetNoDataValue(noDataValue)
    opt_string2 = 'VALUES='+str(noDataValue)
```

```
    options = [opt_string, opt_string2]
    #options = ['NODATA=0', 'VALUES=0']

    gdal.ComputeProximity(srcBand, proxInBand, options)

    proxIn = gdalnumeric.BandReadAsArray(proxInBand)
    proxOut = gdalnumeric.BandReadAsArray(proxBand)

    proxTotal = proxIn.astype(float) - proxOut.astype(float)
    proxTotal = proxTotal*metersIndex
    proxTotal *= dist_mult

    # clip array
    proxTotal = np.clip(proxTotal, -1*vmax_dist, 1*vmax_dist)

    if npDistFileName != '':
        # save as numpy file since some values will be negative
        np.save(npDistFileName, proxTotal)
        #cv2.imwrite(npDistFileName, proxTotal)

    #return proxTotal
    Return
```

Data Preparation

```
import json
from shapely.geometry import Polygon, shape, Point
import gdal
import numpy as np
import os
import cv2

# Set parameters
MAX_UINT8 = 255.0
MAX_UINT16 = 65535.0

# Original Size
ORIG_XDIM = 438
ORIG_YDIM = 406

# Expected Dimension in this case 512*512
EXPECTED_DIM = 512

XFACTOR = EXPECTED_DIM / float(ORIG_XDIM)
YFACTOR = EXPECTED_DIM / float(ORIG_YDIM)


ds3 = gdal.Open('3band_AOI_1_RIO_img6931.resized.tif')
ds8 = gdal.Open('8band_AOI_1_RIO_img6931.resized.tif')

geo_trans = ds3.GetGeoTransform()

borders = np.zeros([EXPECTED_DIM, EXPECTED_DIM])
building = np.zeros([EXPECTED_DIM, EXPECTED_DIM])

# Creates points list
def convert_points(points, geo_trans):
```

```
    converted_points = []
    for p in points:
        cp = Point(world_2_pixel(geo_trans, p[0], p[1]))
        converted_points.append([cp.x, cp.y])
    return converted_points


def draw_polygon(polygon, geo_trans, buildings, borders):
    points = polygon.exterior.coords[:]
    converted_points = convert_points(points, geo_trans)
    cv2.drawContours(buildings,
                     [np.array(converted_points, dtype=np.int32)],
                     -1, 1, thickness=-1)
    cv2.drawContours(borders,
                     [np.array(converted_points, dtype=np.int32)],
                     -1, 1, thickness=2)
    if polygon.interiors:
        for inner_polygon in polygon.interiors:
            points = inner_polygon.coords[:]
            converted_points = convert_points(points, geo_trans)
            cv2.drawContours(buildings,
                             [np.array(converted_points,
dtype=np.int32)],
                             -1, 0, thickness=-1)
            cv2.drawContours(borders,
                             [np.array(converted_points,
dtype=np.int32)],
                             -1, 1, thickness=2)

def convert_points(points, geo_trans):
    converted_points = []
    for p in points:
        ul_x = geo_trans[0]
        ul_y = geo_trans[3]
        x_dist = geo_trans[1]
        y_dist = geo_trans[5]
        x_pix = (p[0] - ul_x) / x_dist
        y_pix = (p[1] - ul_y) / y_dist
        cp = Point(round(x_pix), round(y_pix))
        converted_points.append([cp.x, cp.y])
    return converted_points


with open('Geo_AOI_1_RIO_img6931.geojson', 'r') as f:
    js = json.load(f)
    for feature in js['features']:
        polygon = shape(feature['geometry'])
        if polygon.type == 'Polygon':
            draw_polygon(polygon, geo_trans, building, borders)
        elif polygon.type == 'MultiPolygon':
            draw_polygon(polygon[0], geo_trans, building, borders)

        # save target files
building -= borders
building = np.clip(building, 0, 1)
street = np.zeros([EXPECTED_DIM, EXPECTED_DIM])
street.fill(1)
```

```
street -= building
target = np.zeros([EXPECTED_DIM, EXPECTED_DIM, 2])
target[:, :, 0] = street
target[:, :, 1] = building
target = np.reshape(target, (EXPECTED_DIM * EXPECTED_DIM, 2))
np.save('target_AOI_1_RIO_img6931.resized', target)

inputs = np.zeros([EXPECTED_DIM, EXPECTED_DIM, 3 + 8])
input_idx = 0
for i in range(1, 4):
    channel = np.array(ds3.GetRasterBand(i).ReadAsArray()) / MAX_UINT8
    inputs[:, :, input_idx] = channel
    input_idx += 1
for i in range(1, 9):
    channel = np.array(ds8.GetRasterBand(i).ReadAsArray()) / MAX_UINT16
    inputs[:, :, input_idx] = channel
    input_idx += 1

inputs = inputs[:, :, [0, 1, 2, 10]]

np.save('in_AOI_1_RIO_img6931.resized',inputs)

# Converts geo data to pixel
def world_2_pixel(geo_trans, i, j):
    ul_x = geo_trans[0]
    ul_y = geo_trans[3]
    x_dist = geo_trans[1]
    y_dist = geo_trans[5]
    x_pix = (i - ul_x) / x_dist
    y_pix = (j - ul_y) / y_dist
    return[round(x_pix), round(y_pix)]



#Image Imputing.

in_image = np.load('in_AOI_1_RIO_img6931.resized.npy')
in_dimensions = in_image.shape



EXPECTED_DIM = 512
CROP_DIM = 128
OVERLAP = 0.5

pieces = []
start_pts = np.linspace(0, EXPECTED_DIM,EXPECTED_DIM / (OVERLAP *
CROP_DIM) + 1)
start_pts = start_pts[:-2]
start_pts = [int(x) for x in start_pts]
for i in start_pts:
    for j in start_pts:
        croped_image = in_image[i:i+CROP_DIM, j:j+CROP_DIM, :]
        pieces.append(croped_image)
```

```
name =
os.path.splitext(os.path.basename('in_AOI_1_RIO_img6931.resized.npy'))[0
]

for idx in range(len(pieces)):
    piece = pieces[idx]
    np.save(os.path.join('%s_%d' % (name, idx)), piece)


#To target


tg_image = np.load('target_AOI_1_RIO_img6931.resized.npy')
tg_dimensions = tg_image.shape
tg_image = np.reshape(tg_image, (EXPECTED_DIM, EXPECTED_DIM,
tg_dimensions[1]))



EXPECTED_DIM = 512
CROP_DIM = 128
OVERLAP = 0.5

pieces = []
start_pts = np.linspace(0, EXPECTED_DIM,EXPECTED_DIM / (OVERLAP *
CROP_DIM) + 1)
start_pts = start_pts[:-2]
start_pts = [int(x) for x in start_pts]
for i in start_pts:
    for j in start_pts:
        croped_image = tg_image[i:i+CROP_DIM, j:j+CROP_DIM, :]
        pieces.append(croped_image)

name =
os.path.splitext(os.path.basename('target_AOI_1_RIO_img6931.resized.npy'
))[0]

for idx in range(len(pieces)):
    piece = pieces[idx]
    np.save(os.path.join('%s_%d' % (name, idx)), piece)
```

Create Model

```
    model = Sequential()
    # first convolutional block
    model.add(Convolution2D(32, 3, 3, input_shape=(DIM, DIM, CHANNELS),
                            border_mode='same'))
    model.add(Activation('relu'))
    model.add(Convolution2D(32, 3, 3, border_mode='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # second convolutional block
    model.add(Convolution2D(64, 3, 3, border_mode='same'))
    model.add(Activation('relu'))
```

```
    model.add(Convolution2D(64, 3, 3, border_mode='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # third convolutional block
    model.add(Convolution2D(128, 3, 3, border_mode='same'))
    model.add(Activation('relu'))
    model.add(Convolution2D(128, 3, 3, border_mode='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    # back to the original dim
    model.add(Convolution2D(1024, 5, 5,
                            border_mode='same'))
    model.add(Activation('relu'))
    model.add(Convolution2D(128, 1, 1,
                            border_mode='same'))
    model.add(Deconvolution2D(CLASSES, 16, 16,
                              output_shape=(BATCHSIZE, DIM, DIM,
CLASSES),
                              subsample=(8, 8),  border_mode='same'))
    model.add(Reshape((DIM * DIM, CLASSES)))
    model.add(Activation('softmax'))
    optmzr = adam(lr=LEARNING_RATE)
    model.compile(loss='categorical_crossentropy', optimizer=optmzr,
                  metrics=['accuracy'])
    print(model.summary())
```