Semantic Segmentation of Satellite Images for Humanitarian Missions

Serkan Karakulak, SK7685

Marina Zavalina, MZ2476

Peeyush Jain, PJ891

I. Introduction

Mapping remains a challenge in many parts of the world, particularly in dynamic scenarios such as natural disasters when timely updates are crucial. Location is critical to disaster response: first responders need fast, reliable information to reach affected areas and prioritize aid efforts.

Many disasters occur in regions that are missing from the maps and the first responders lack information to make valuable decisions regarding their relief efforts. There is a collaborative project called Missing Maps [1] which supports OpenStreetMap community [2] in developing technologies, skills, work-flows, and communities. Missing Maps volunteers manually label features like roads, buildings, rivers, highways and other classes of importance on satellite images. The mapped labels are then validated manually by another set of volunteers. These efforts are critical for humanitarian missions and there is a large volunteer community that coordinates and works on supplying timely and accurate maps of such regions. Currently, updating maps is a highly manual process requiring a large number of human labelers to either create features or rigorously validate generated outputs. As there have been several successful implementations of the image segmentation algorithms in the last few years, our project aims to utilize those advancements to make pixel-wise class predictions for highways and buildings. Our vision is to contribute to this worldwide mapping efforts by giving tools that can be used either to generate initial labels or to help identify possible mislabeling using satellite images.

With the help of the mapping community, who would have machine learning tools available at their disposal, the first responders can obtain critical information for unmapped regions in a timely manner. The resulting data then can be fed to the available open-source platforms, which can be used to print maps, for analysis, or for navigation on the ground.

This project also aim to address the broader issue of semantic segmentation of satellite images by aiming to classify the objects as buildings, highways, or marked for generic land-use. We implemented a Convolutional Neural Network suitable for this task, inspired by the U-net architecture [3].

Related Work

The U-net is a specific type of Fully Convolutional Network, that has a encoding-decoding network architecture with skip connections. It has raw images as its inputs and generates pixel-wise label predictions as its output. This particular image segmentation algorithm has received a lot of interest by generating competitive results in variety of domains. It was first proposed for the segmentation of biomedical images [3] but since has proven to be efficient for the pixel-wise classification of satellite images [4]. For instance, U-net architecture used for satellite images segmentation achieved 63% on Intersection over Union (IoU) metric for buildings and 56% for roads [6] where in addition to RGB values, the input data also has non-visible waves such as infrared and thermal wavebands.

II. Source Imagery and Labels

In order to generate pixel-wise predictions, we needed data where satellite images and their pixel-wise labels are available. We researched and found an open source API, "Label Maker" developed by DevSeed [7]. Label Maker is a python library which helps in extracting insight from satellite imagery. It pulls data from OpenStreetMap and combines that with imagery sources like Mapbox to generate images which can be used in training machine learning algorithms. Also, the list of all the regions that mapping community have manually mapped and validated in OpenStreetMap is available via a separate API [8]. This RESTful API provides information in JSON format on the mapped regions indicating which regions are validated, the bounding box

(latitude & longitude) of each project and zoom level for which the images are available.

Label Maker

Label Maker generates training data for ML algorithms focused on overhead imagery (e.g., from satellites or drones). It downloads OpenStreetMap Tile information and overhead imagery tiles. All tiles are of size 256x256x3.

Figure 1: *The left pair shows the tile with corresponding highway label and the right pair shows the tile with labeled buildings.*

OSM Task Manager API

The OSM Tasking Manager is a mapping tool designed and built to support the Humanitarian OpenStreetMap Team's collaborative mapping process. The purpose of the tool is to divide a mapping project into smaller tasks that can be completed rapidly with many people working on the same overall area.

The tool provided two relevant functionalities which we used in this project.

- *Project Grouping by requesting organization and campaigns.* One can easily group projects by different organizations and campaigns. This allowed us to easily see the overall progress of all the projects. The metadata information that we extracted from JSON response was project id, total number of projects.
- *Finding Project-specific details.* The metadata information that is relevant to us is - project id & project coordinates. We computed additional features required by Label Maker API. These were country and zoom level.

III. Data Collection

The intent was to download verified satellite images from OpenStreetMap. For this purpose, we built custom data pipeline. Below summarizes the steps involved in downloading the images -

- Make API call using OSM task manager to obtain the list of available projects.
- Iterate through each project to filter the list of validated projects. Validated projects signify that all labels within the project have been mapped and verified by the volunteers manually.
- Multiple API calls were made for each project to determine which of the projects has a 100% validation status. This information is only available at the project level.
- Extract features from project metadata - project id, project status (computed based on the status of each tile), project coordinates (latitude & longitude).
- Make use of python library geopy.geocoders to retrieve the country name based on the project coordinates.
- Align the names obtained from geopy to match with the Label Maker country list. Label Maker publishes their own set of country names which were different from what geopy returns. The country name correction is essential as Label Maker API will not accept any mismatches.
- Use python's template library to generate multiple configuration files that are in accordance with Label Maker guidance and can be used by the bash script to download the content.
- As downloading and re-tiling the satellite imagery for all validated projects is an audacious exercise, we identified 21 projects spread across different continents and wrote bash script to download the images only for those regions.
- Modify the script to download multiple zoom level of the selected projects (we downloaded 15 out of 21 projects at zoom level of 17 and 18).
- The bash script produces labels and tiles for all the selected projects. The downloaded data is then passed to the next stage where data cleaning is performed.

IV. Data Preparation and Analysis

Data Cleaning

Some of the downloaded images turned out to be empty because of the mismatch in zoom level. So, we wrote a script to check whether the image is empty or not. If it was, the tile was deleted together with the corresponding label. In the end, we got 36,595 images (and the same amount of corresponding labels); each image was of size 256x256 pixels.

We also noticed that a non-negligible proportion of satellite images were not up-to-date with the most recent masks, as new buildings had already been added. Additionally, some images were blurry or contained mostly clouds. Overall, this likely decreased the performance of our model – as it had partially learned on mislabeled and/or blurry training images – and led to a sub-optimal performance at test time.

One advantage of our dataset is that the images are labeled by humans which should result in a good accuracy. However, we discovered examples where the masks are wrongly defined by the volunteers or where the satellite images are not up to date (shown in the diagram). This may be caused by various factors, including dense urban area, dark imagery, different offsets between imagery plus often very imprecise building tracing from new or untrained volunteers.

All this points to an interesting application of our model: we can use it to detect errors on open-source mapping platforms to correct the layers, or update satellite images where it is more needed.

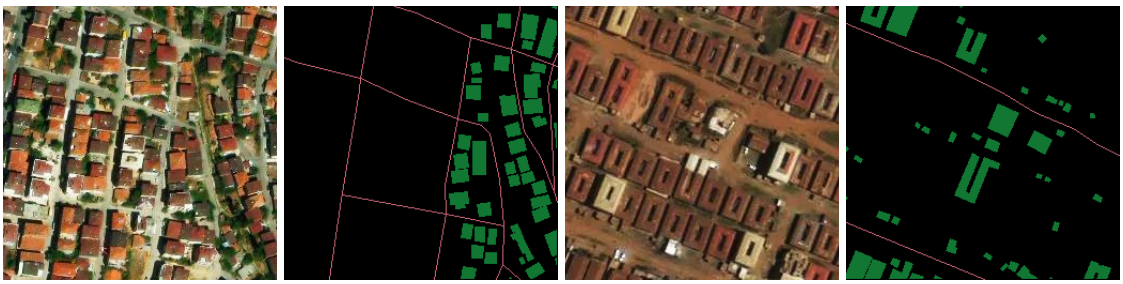

Figure 2: *Both the pair shows the buildings are missing in the labels.*

Split Train/Validation/Test Set

We wanted to evaluate both how model performs on the entire new project and how it performs on random images from different parts of the world. Thus, we created two different types of validation and test sets. First, we took random sample of entire projects of different sizes for validation (11.59% of all tiles) and test (6.12% of all tiles). Then the rest of the projects were divided in to train/validation/test (80%/10%/10%), after shuffling all of the tiles within a project. The diagram 3 shows the structure and split sizes.

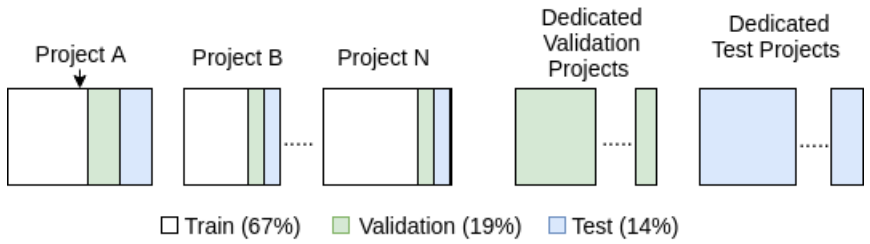

Figure 3: *Train,Validation & Test Set*

Data analysis

Out of the 194 projects that have 100% validation status, we selected 21 projects and downloaded the images & corresponding labels. We further chose 15 projects at random and downloaded multiple zoom levels(17 & 18). The zoom levels of 17 & 18 are chosen as they are the highest zoom level for which images are available. The total number of tiles downloaded were 36595.

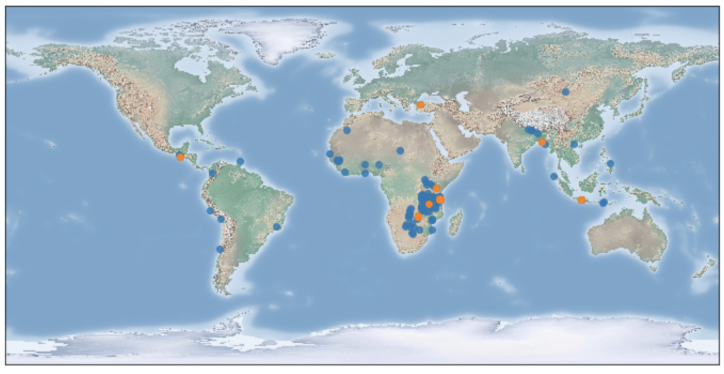

Figure 4: *Red dots show projects selected for modeling, Blue dots show all available validated projects.*

Another critical feature of the satellite imagery is an imbalanced class set: there are many more negative examples (background) than positive examples (buildings & highways). We did pixel-wise analysis of all 36595 images (each image having 256 x 256 pixels) and obtained that the classes are distributed as

- % of pixels comprising background = 87%
- % of pixels comprising buildings = 12.3%
- % of pixels comprising highways = 0.7%

V. Modeling & Evaluation

Choice of Model

Main ideas

Convolutional Neural Networks (CNN) offers an efficient way to use image data in various settings and it produced state of the art result in last several years for a large range of computer vision problems including semantic segmentation tasks. These networks can produce a segmentation mask for an entire image in a single forward pass in an end-to-end architecture. One of the most successful deep learning methods is based on Fully Convolutional Networks (FCN) [9]. The main idea is to rely on CNN as a powerful feature extractor, switching from fully connected layers to convolution ones to output spatial feature maps. Those maps are then upsampled to produce dense pixel-wise output. This approach has been further improved and transformed by U-Net neural network.

In general, a U-Net like architecture consists of a contracting path to capture context and of a symmetrically expanding path that enables precise segmentation. The contracting path consists of a convolutional network with alternating convolution and pooling operations. Each step in the expansion path consists of an up-sampling of the feature map and then applying convolution. Thus, the expansive path increases the resolution of the output. To localize upsampled features, the expansive path mixes them with high-resolution features from the

contracting path [9]. The output of the model is a pixel-wise mask that identifies the class of each pixel.

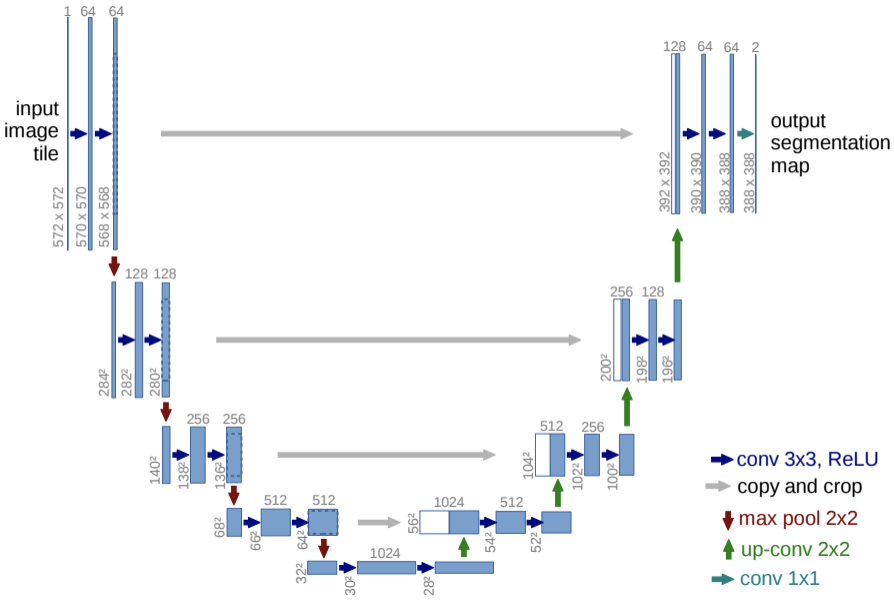

Figure 5: U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations. Source [3]

From a data science perspective, segmenting satellite images presents a class imbalance problem where most of the pixels belong to the label 'background'. Considering the whole regions that those images cover, only small percentage of the pixels are actually buildings. Roads are much rarer and much harder to classify, as they are denoted as one-pixel line width. Class imbalance problem is further amplified by the fact that making distinction between buildings and estimating their number is critical for readers of the maps, however close these buildings may be to each other. Simply defining class-wise loss weights does not address the class imbalance problem as it causes the model to predict the label 'building' when the odds of that pixel belonging to 'background' and 'building' are similar.

To address this problem we have used Focal Loss[5] function, which is an extension of the standard cross entropy criterion that 'focuses' more on difficult misclassified instances. As the majority class

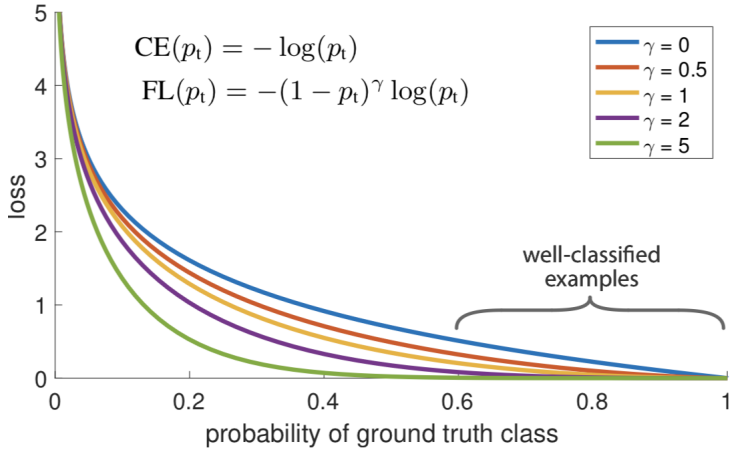

Figure 6: Focal Loss

become comparatively easier to predict, the rare class maintains a loss of non-trivial magnitude and this approach is reported to perform well in problems where classes demonstrate extreme imbalances. It would also mean that separating close buildings would potentially have higher focus as well, since that would also be an instance of non-trivial and hard-to-classify predictions. The focal loss has two hyperparameters, γ paremeter is defines the curve of the loss function. In addition there is $alpha$ parameter which is a constant multiplier to the classwise losses, which is reported to slightly improve the results. In this project, we used cross-entropy loss, focal loss without class weights α, and focal loss with class weights.

Data Augmentation

The U-net was shown in [3] to work well with a limited number of training examples, provided one made heavy use of data augmentation. Accordingly, in order to maintain a reasonable amount of images, and above all to avoid overfitting by ensuring a sufficient invariance and robustness of the network, we followed [3] and applied real-time data augmentation techniques to our training set. The satellite images were flipped and rotated by 90 degrees, which allowed us to train our model on a considerably larger set of images. This task was done using the TensorFlow framework. Additionally, we augmented the images by adding noise using Numpy framework to avoid overfitting of our model.

Models

Combining the ideas described above, we build U-net model from scratch. We have three different specifications: with cross entropy loss, with focal loss, with focal loss with weights (according to distribution of classes in train).
Also we try dynamic U-net model from fastai library [11] which takes pretrained resnet34 as the encoder and generates the decoder part as first forward pass is completed (work in progress – March 2019). Considering data preprocessing, we improved it; we cleaned all empty labels, "widened" the roads, i. e. put more pixels around each road pixel. Binary cross-entropy loss with weights (according to the distribution of classes) was used to train the model. We unfreeze the whole net and train for 5 epochs (this specification shows best results so far).

Evaluation Metrics

We adopted a metric based upon the scale invariant intersection over union (IoU) metric, also known as the Jaccard index, which provides a measure of the overlap between two objects by dividing the area of the intersection by the area of the union.

$$IoU(A, B) = Area(A \cap B)/Area(A \cup B)$$

For an object detection algorithm, the performance of algorithm depends on how many objects the algorithm detects (true positives) how many objects it fails to detect (false negatives), and how many non-objects it detects (false positives). We summed the true and false positives and negatives to compute total precision and recall scores; the F1 score is given by the harmonic mean of precision and recall:

$$F_1 = 2 \cdot (Precision \cdot Recall)/(Precision + Recall)(2)$$

Results

The images show the predictions of fastai dynamic U-net model and U-net model with focal loss with weights for randomly selected tiles.
Both models learned to recognize buildings. Our implemented U-net model achieved 38% IoU in

Figure 7: *From left to right: predictions of fastai dynamic U-net, predictions of implemented U-net model, satellite images, corresponding labels*

its prediction, fastai dynamical U-net – 39% IoU. Considering the first model, the results in the table show focal loss performs slightly better than cross entropy loss. While fastai model seems to predict buildings better based on the pictures while IoU is almost the same for both models. One reason could be the poor quality of labels. It may happen that dynamic U-net learned to predict missed / shifted labels and is penalized for it.

The result from a similar study achieved 63% IoU for buildings[6]. It appears that one of the problem in our case is poorly labeled data – sometimes the labels are missing or shifted.

Table 1: **Metrics for Satellite Image Segmentation - Buildings**

	Randomized Test Set				Dedicated Test Set			
	IoU	*Precision*	*Recall*	F_1	*IoU*	*Precision*	*Recall*	F_1
Fastai dynamic U-net	0.389	0.654	0.494	0.552	0.343	0.627	0.435	0.505
Focal Loss with Weights	0.381	0.639	0.485	0.552	0.379	0.640	0.483	0.550
Focal Loss w/o Weights	0.341	0.662	0.413	0.500	0.341	0.662	0.413	0.500
Cross Entropy (Baseline)	0.344	0.603	0.445	0.512	0.344	0.603	0.445	0.512

Another weakness of our implemented model (and partly of fastai model) is that it failed at detecting building boundaries with precision. It has a propensity to give us soft edges, instead of sharp and well defined ones. This is especially an issue in densely populated areas where buildings are close.

One way to address the problem of detecting exact building shape is to train our model to extract building boundaries, instead of entire buildings. However this has lower interest in terms of potential applications, and the output may not be desired, as it might end up detecting all straight lines in a given image. For these reasons, a more sophisticated strategy would be needed to overcome the problem of "blobby" predictions and particularly, the one presented in [10] would be suitable for that.

The implemented model did not learn to spot highways, which can be noticed by observing the metrics for highway prediction, the score is zero for most of the specifications.
One of the reasons for failing to predict could be that highways have one-pixel width, no matter how wide the highway is in the image. We address this issue in recent part of our work (fastai model) by putting more weight around each pixel of the highway (to increase the width from 1 pixel to, say, 5 pixels). Thus, fastai model learns to predict roads.

Table 2: **Metrics for Satellite Image Segmentation - Highways**

	Randomized Test Set				Dedicated Test Set			
	IoU	*Precision*	*Recall*	F_1	*IoU*	*Precision*	*Recall*	F_1
Fastai dynamic U-net	0.094	0.176	0.187	–	0.099	0.171	0.201	0.178
Focal Loss with Weights	0.000	0.076	0.000	0.000	0.000	0.075	0.000	0.000
Focal Loss w/o Weights	0.000	0.090	0.000	0.000	0.000	0.090	0.000	0.000
Cross Entropy (Baseline)	0.000	0.008	0.000	0.000	0.000	0.008	0.000	0.000

As expected the models performed best for the background pixels since they represents the majority class with 87% distribution.

Table 3: **Metrics for Satellite Image Segmentation - Background**

	Randomized Test Set				Dedicated Test Set			
	IoU	*Precision*	*Recall*	F_1	*IoU*	*Precision*	*Recall*	F_1
Fastai dynamic U-net	0.843	0.892	0.937	0.913	0.794	0.854	0.916	0.883
Focal Loss with Weights	0.861	0.901	0.950	0.924	0.859	0.899	0.950	0.925
Focal Loss w/o Weights	0.857	0.888	0.960	0.923	0.857	0.889	0.960	0.923
Cross Entropy (Baseline)	0.848	0.891	0.945	0.917	0.848	0.891	0.945	0.917

VI. Deployment

While developing our solution, we reached out to various Mapping communities including Missing Maps, Humanitarian OpenStreetMap Team (HOT), DevSeed and MapSwipe, each

of which has very active and vibrant communities. We got encouraging feedback from them regarding the utility of this project. After making necessary improvements and improving the metrics, we are aiming to integrate our model with some of the existing tools to help these high impact organizations.

We coordinated with MapSwipe group, who are in process of extending their mobile application to do pixel-wise labeling of the tiles. The tagged tiles will help mappers to detect buildings in sparsely populated regions. Our segmentation model can provide useful functionality and help the users to see computer generated labels which can be easily marked as valid or invalid. Validated labels would then be transmitted to the OpenStreetMap system and will get integrate into their maps. These would greatly decrease the effort that is required to map an area, since tagging a building on map takes significant time as mappers currently need to mark vertices of a building one by one.

Another use of the algorithm would be validating the results of the mappers. HOT is consisting of thousands of volunteer mappers, whose skill levels greatly vary. In addition, most of the labels are gathered in Mapathlon events, where the participants are doing mapping for the first time. This results in an unreliable initial labeling. Our algorithm could be deployed in the validation process to detect inaccurate labels. This would help the validator as there are thousands of labels which need to be validated in each project.

VII. Individual Contribution

During the course of this project all three of us worked together to discuss data acquisition, data pipeline, choice of architecture and model implementation. Serkan focused specifically on the use of TensorFlow to implement the model. Peeyush investigated the possibility of downloading data using different APIs and on finding good sources of satellite images. Marina concentrated on finding the most suitable U-net model to work with and on implementing different functionalities in Python.

VIII. Git Hub Repository Url

The source code for the project is available at Git hub URL - `https://github.com/peeyushster/SatelliteImagery`. The code contains the model written in Tensor Flow, fastai (pytorch) and complete data collection, cleaning & analysis modules.

References

[1] Missing Maps official website `http://www.missingmaps.org/`

[2] OpenStreetMap Wikipedia page `https://en.wikipedia.org/wiki/OpenStreetMap`

[3] O. Ronneberger, P. Fischer, T. Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation, *Computer Science Department and BIOSS Centre for Biological Signalling Studies, University of Freiburg* Germany, 2015

[4] Alexander V. Buslaev, Selim S. Seferbekov, Vladimir I. Iglovikov, Alexey A. Shvets, Fully Convolutional Network for Automatic Road Extraction from Satellite Imagery, 2018 - `https://arxiv.org/pdf/1806.05182.pdf`

[5] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr. Dollar, Focal Loss for Dense Object Detection *Facebook AI Research (FAIR)* - `https://arxiv.org/pdf/1708.02002.pdf`

[6] Vladimir Iglovikov, Sergey Mushinskiy, Vladimir Osin, Satellite Imagery Feature Detection using Deep Convolutional Neural Network, A Kaggle Competition, 2018 - `https://arxiv.org/pdf/1706.06169.pdf`

[7] Label Maker API - `https://github.com/developmentseed/label-maker/`

[8] OSM Tasking Manager API - `https://github.com/hotosm/tasking-manager/tree/develop/server/api`

[9] J. Long, E. Shelhamer, T. Darrell, Fully convolutional networks for semantic segmentation. *In Proceedings of the IEEE conference on computer vision and pattern recognition, pages* 3431–3440, 2015 - `https://arxiv.org/pdf/1605.06211.pdf`

[10] Benjamin Bischke,Patrick Helber,Joachim Folz,Damian Borth,Andreas Dengel, Multi-Task Learning for Segmentation of Building Footprints with Deep Neural Networks, 2017 - https://arxiv.org/pdf/1709.05932.pdf

[11] Fastai library https://docs.fast.ai/