



MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BİTİRME PROJESİ II

**YAPAY SİNİR AĞLARI VE UYDU GÖRÜNTÜLERİ İLE OSM
HARİTALARIN KALİTESİNİ İYİLEŞTİRME**

Öğrenci: Kayodé Hadilou ADJE, 151180074

Danışman: Prof. Dr. M. Ali AKÇAYOL

HAZİRAN 2019

TABLE OF CONTENTS

I. Abstract	5
II. Introduction.....	6
III.. Related Work.....	6
IV. Dataset	7
1. Dataset Acquisition and Splitting	7
2. Data Preprocessing.....	7
V. Methodology	8
1. Model Architecture	8
2. Modification.....	9
3. Negative Mining	9
VI. Experiments and Results	10
1. Metrics.....	10
2. Training.....	11
3. Evaluation and Results.....	11
VII. Conclusion	13
VII. Annexes.....	14
1. First 10 epochs.....	15
2. Following 15 epochs.....	16
3. Following 25 epochs.....	17
4. Last 20 epochs.....	18
VIII. References.....	19

LIST OF FIGURES

Figure 1. From to right to left: aerial tiles with its corresponding masks. Green color corresponds to building while red color corresponds to road.....	7
Figure 2. Original UNET Biomedical for Segmentation.....	9
Figure 3. Predictions Before and After Negative Mining.....	10
Figure 4. Mean Intersection Over Union.....	11
Figure 5. Pixel Level Segmentation for building type no 309998.....	12
Figure 6. Good Prediction: From left to Right: MapBox Satellite Image, OSM and Predicted Mask.....	12
Figure 7. Bad Prediction: From left to Right: MapBox Satellite Image, OSM and Predicted Mask.....	13

I. ABSTRACT

Location based services rely on geographical data whether private or open source. The latest ones such OpenStreetMap offer developers opportunities to perform any kind of analysis, access the raw geographical data by accepting data entries from any volunteer without any control or restriction; while this is a big opportunity, it also underlines the quality of OSM data as data can be wrong or vandalized. This project aims to build use UNET artificial neural network model to perform semantic segmentation on aerial images labelled with OSM tags. The comparison between the prediction and the combination of aerial and OSM tag will help in localizing errors and vandalism in OSM data, thus improve the quality of the most used open geo-data. This example is performed using map and aerial tiles of Ankara. A model have been trained to detect building using Robosat UNET implementation and a mIOU of 0,803 (mean intersection over union) plus an MCC of 0,771 has been obtained.

II. INTRODUCTION

OpenStreetMap [2] or OSM is a free, editable, worldwide and open source geographical database. It is the Wikipedia of geographical data and has over 2M contributors. Everyone can become a contributor and add data to the project [2].

Rather than a user interface, OSM is a data set of places, there exist many other tools such as MapBox, Mapsui... used to display the OSM's geodata: the process is known as rendering. OSM data is a set of three basic data structures: nodes, ways and relations [2]. Physical features on the ground (eg: building, farms, roads...) are represented using tags attached to the data structures. A tag is a pair of two other properties: key and value, each tag is a geometric representation of the ground object shown by the nodes, ways or relations. For example, physical features such as roads or rivers can be represented by two or more nodes.

Compared to Google Maps or Bing Maps, the main advantage of OpenStreetMap is the possibility it offers to individuals, organizations or third parties to edit or add data to the project without any restriction: this goes with the aim of the project which is to provide people with geodata for free.

One of the most significant technical problems with OSM is the lack of a review model, that is for a change to the map to be staged and then reviewed before being applied. Not having this functionality caused ripples of problems throughout the system. Editing on OSM can be challenging for beginners, beginners can map incorrectly and because OSM's data model doesn't include a review stage bad edits are committed to the map and often left undiscovered [3]. This project aims to develop a deep learning application that can be used to detect errors in OSM data. The application can detect the following errors: *incorrectly Labelled Objects*: these are physical objects of a given type but wrongly labelled as another type; *Out of Date Maps*: Due to changes, maps are often not updated and then can contain wrong geodata; *incorrectly Localized Objects*: Objects in the map are localized as an ensemble of nodes. If the nodes (which are nothing but 2D coordinates or a pair of longitude and latitude coordinates [2]) aren't correctly represented, the localization of the object can be compromised.

III. RELATED WORK

The quality of OSM data is under constant criticism by the scientific community. OSM data has no specific metrics to measure the quality of each contribution made by volunteers whether they are new or experts. Despite the fact that OSM project itself does not include a quality control mechanism, many researches have been made by the community to ensure the liability of OSM geodata:

BBBike and Geofabrik developed *OSM map Compare Tool* [5], a web interface comparison tool that compares OSM tiles to others maps data such as Google Maps, Bing Maps.

Osmose [6], an acronym for OSM Search Engine Optimization is a tool used to detect errors and inconsistencies in OSM data and show them on the map so that contributor can edit them.

DeepOSM [4] is a deep learning OSM data quality assurance tool. It can detect errors in OSM's roads networks. The system downloads satellite imagery and the corresponding OSM data showing road features. The system then predicts the correctness of the map based on a deep neural network backed algorithmic computation.

Map Roulette [7] is another quality assurance tool used by the community. It is a web application that detects errors in OSM and shows them as challenges; each challenge is a set of tasks and contributors can correct errors like usual.

In addition, many other researches have been conducted on computer vision applied to satellite imagery, some articles have been published on that topic. We can cite [8], [9] and [10]. Essentially, the first two articles use SVM algorithms to extract buildings from very high resolution (VHR) satellite images. The base-case accuracy reached in each case is respectively 74% and 83%. The last paper presents a list of results on the INRIA Aerial Image dataset, using different architectures for image segmentation such as Fully Convolutional Networks (FCN) and SegNet .

The U-net – a specific type of FCN – has received a lot of interest for the segmentation of biomedical images using a reduced dataset, but has proven to be also very efficient for the pixel-wise classification of satellite images as in Robosat [11]. We mainly built upon the UNET implementation of Robosat accessible at github.com/mapbox/robosat. In the original paper, the authors developed a U-Net specifically dedicated to biomedical image segmentation.

IV. DATASET

1. Data Acquisition and splitting

Our dataset is a combination of aerial images with map tiles. It was obtained using label-maker tool developed by a team at MapBox. The tool can generate aerial images for a given some osm data. So firstly, we download OSM geo-data belonging to Ankara and for each tile, we download the corresponding aerial image from MapBox. The dataset consist of 70.000 images and labels. The labels are segmentation masks with $n+1$ number of colors where n is the number of classes. Each color is the class of the respective pixel. Each tile and label is 256*256 jpg or png format. The name of the files corresponds to the Slippy Map convention; this helps us associate map tiles with their corresponding geo-positions. The entire dataset is divided in 3 parts: 80% for training, 10% for validation and 10% for holdout.

2. Preprocessing

As the labels we extracted from OSM only consisted of the “buildings” or “road” layer, there was no need for an entire RGB channel, so we converted the label images to a binary mask of size $256 \times 256 \times 1$ with 1 for pixels labeled as” buildings”, 0 otherwise (same thing for roads). Below is one of the satellite images from our dataset, along with its corresponding mask:



Figure 1: From to right to left: aerial tiles with its corresponding masks. Green color corresponds to building while red color corresponds to road.

Some images were also blurry. Overall, this likely decreased the performance of the model as it had partially learned on mislabeled and/or blurry training images – and led to a suboptimal performance at test time.

Also, the mask were in png format of size $256*256*3$, we converted them to a 8-bit color indexed images meaning from $256*256*3$ to $256*256*1$. A conversion table or palette was stored to insure the correct reproducing or displaying to RGB format. With the new format we constantly reduce memory used to store the images as color with three RGB values is labelled with an integer, the conversion information is stored in a table called palette.

V. METHODOLOGY

In this part, we describe the proposed approach to solve the problems mentioned above and consequently contribute to improve OSM data's quality. Rather than being a well-established plan, the approach described here is a proposition and may change during the project. We divide the project in two parts: the first one, focusing on the application of a deep learning model to compare map data to aerial imagery and the second part which is a web interface used to show the differences between OSM data and the prediction made by the deep learning model.

1. Model Architecture

Instead of developing a model from scratch, we decided to use an existing model of Convolutional Neural Network for image segmentation. Namely, we turned to the U-net, originally developed for biomedical image segmentation [10].

Once trained, the network was able to output a pixelwise binary classification (building or not for now) with good accuracy. Basically, the U-net builds upon the Fully Convolutional Network [10]. A contracting path extracts features of different levels through a sequence of convolutions, ReLU activations and max poolings, allowing to capture the context of each pixel.

A symmetric expanding path then upsamples the result to increase the resolution of the detected features. In the U-net architecture, skip-connections (concatenations) are added between the contracting path and the expanding path, allowing precise localization as well as context. The expanding path therefore consists of a sequence of up-convolutions and concatenations with the corresponding feature map from the contracting path, followed by ReLU activations. The number of features is doubled at each level of down sampling.

A figure of the U-net is presented below.

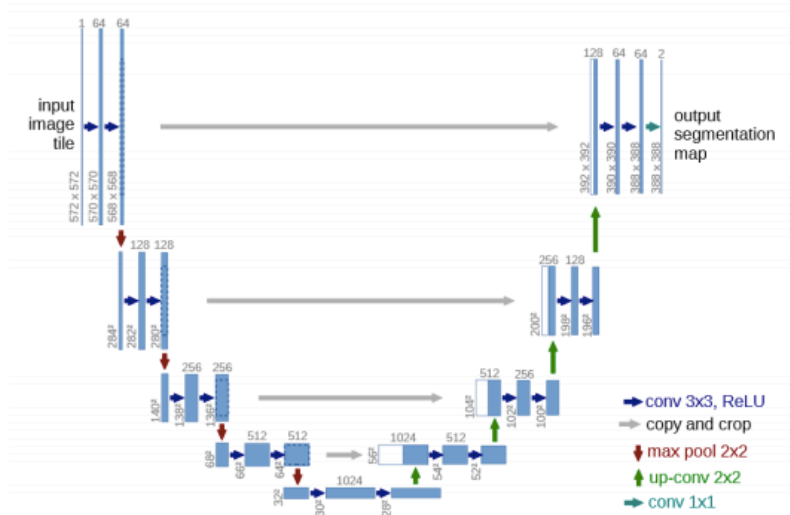


Fig. 2. Original U-net for biomedical image segmentation

2. Modification

For this building extraction problematic, we started with a version of the U-net implemented for Robosat [10]. We used the weight pretrained on ImageNet as initial weight and tuned the parameters to obtain the results mentioned above. We did not use dropout, as we did not see any overfitting while training the model partly thanks to negative hard mining that we will explain below.

3. Negative Mining

This section walks through the steps necessary to tune the model using "negative" images. In our case that means satellite images that 100% definitely do not have a building in them.

Our objective here is to add a set of negative images and their associated negative masks to the dataset and retrain. The result should be a better performing model with fewer false positives. The false positives are due to how we created the dataset: we bootstrapped a dataset based on tiles with buildings in them. Even though these tiles have some background pixels they won't contain enough background (so called negative samples) to properly learn what is not a building. If we never showed the model a single image of water it has a hard time classifying it as background.

There are two ways for us to approach this problem:

- Add many randomly sampled background tiles to the training set, re-compute class distribution weights, then train again, or
- Use the model we trained on the bootstrapped dataset and predict on tiles where we know there are no buildings; if the model tells us there is a building put these tiles into the dataset with an all-background mask, then train again

Although one may achieve better results with option 2, we used option one for simplicity reasons. We have created a new module called NotBuildingHandler which is the inverse of the building handler. Instead of extracting the geojson for buildings, it extracts the geojson for everything that is not a building. We can use this geojson file to download (what we believe will be) negative tiles. We will need to verify manually. After that, we complete the process of splitting, training as we did for buildings.

Below we show a difference between predicted masks for a same tile: one before negative mining and the other just after.



Figure3.a Before Negative Mining



Figure3-b After Negative Mining

VI. EXPERIMENTS and RESULTS

1. Metrics

We used MCC and mIOU as metrics to evaluate the performance of the model.

MCC or The Matthews correlation coefficient is used as a measure of the quality of binary (two-class). It considers true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are imbalance. The MCC is in essence a correlation coefficient between the observed and predicted binary classifications; it returns a value between -1 and $+1$. A coefficient of $+1$ represents a perfect prediction, 0 no better than random prediction and -1 indicates total disagreement between prediction and observation.

$$MCC = (TP * TN - FP * FN) / \sqrt{((TP + FP)(TP + FN)(TN + FP)(TN + FN))}.$$

mIOU or mean intersection over union is shown in the figure below:


$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Figure4: Mean Intersection Over Union mIOU

mIOU is obtained by taking the mean of IOU values.

2. Training

The prediction core is a segmentation model — a fully convolutional neural net which we train on pairs of images and masks. The aerial imagery we download from our Mapbox Maps API in all its beauty. The masks we extract from OpenStreetMap geometries and rasterize them into image tiles. These geometries might sometimes be coarsely mapped but automatically extracting masks allows us to quickly bootstrap a dataset for training.

We then have two slippy map structures with images and corresponding masks. The Slippy Map directory[10] structure helps us in preserving a tile's geo-reference which will allow us later on to go back from pixels to coordinates. It is RoboSat's main abstraction and most pipeline steps transform one Slippy Map directory into another Slippy Map directory.

The training was carried out on Microsoft Azure NC12 GPUs machines. Considering the memory available on the GPU, we did a training on mini-batches of 2 (up to 10 epochs) and 10 images (after the 10th epoch). We went through approximately 77 epochs and each epoch took like 1h and 30 min. We started from a learning rate of 0.0001 and was reduced using a decay rate. In the index you can find the changes observed in the metrics values.

3. Evaluation and results

We evaluate the model on the validation test using MCC and mIOU metrics. We present the result for each epoch below. When we use the checkpoints for prediction on a Slippy Map directory with aerial imagery we get a Slippy Map directory with probabilities for every pixel in image tiles:



Figure 5: Pixel Level Segmentation for building type no 309998 .

We then compare the obtained prediction with a real satellite image and a layer of OpenStreet map. As it can be seen below there is a huge difference between OSM data and the prediction. The prediction looks more like the real data taken directly from mapbox satellite images while the layer extracted from OSM seem to be completed.



Figure 6. Good Prediction: From left to Right: MapBox Satellite Image, OSM and Predicted Mask

This figure shows a good prediction and illustrates exactly the problem we wanted to solve with this project.

However, sometimes the predictions can be worse than the extracted data from OSM. This scenario is shown on the following figure.



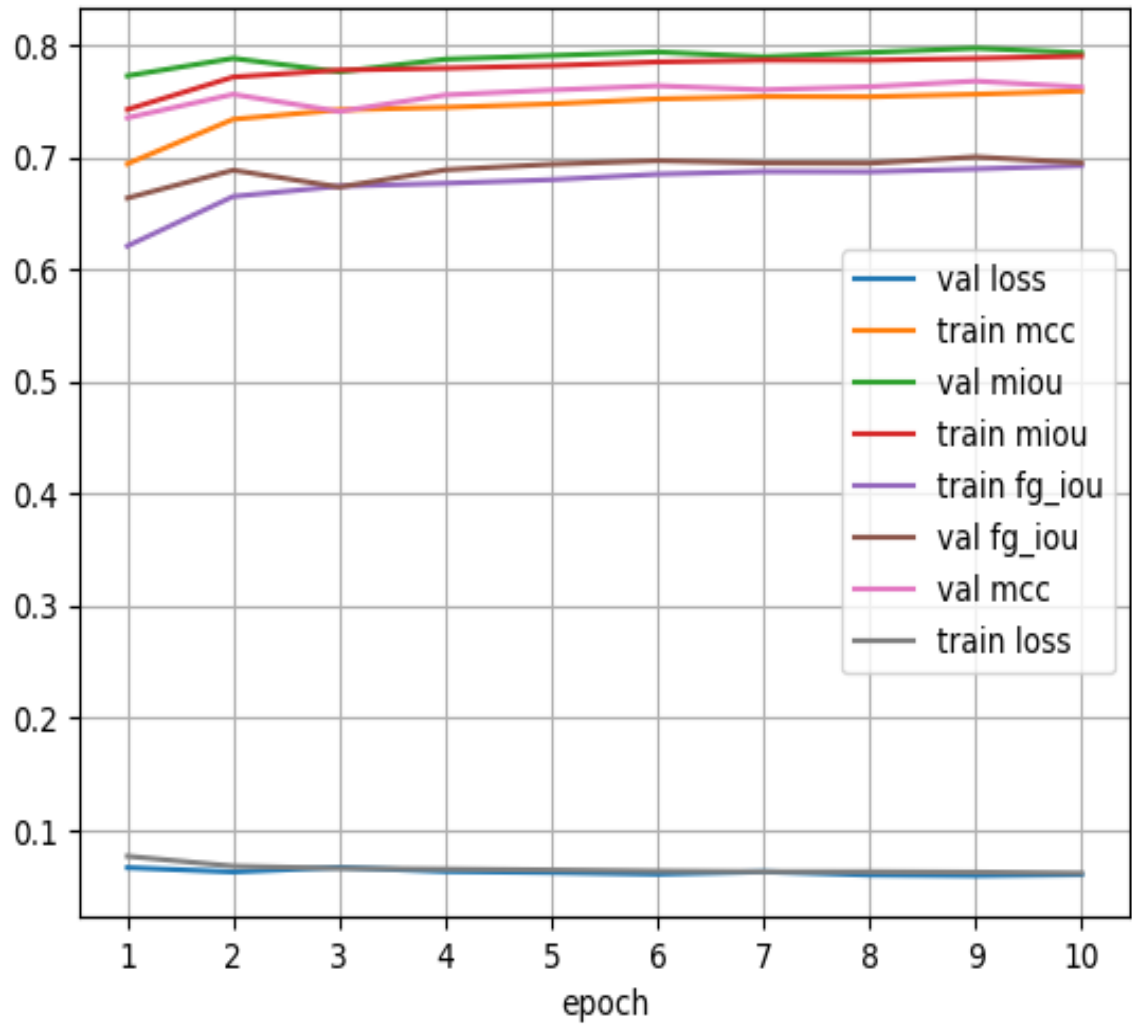
Figure 7. Bad Prediction: From left to Right: MapBox Satellite Image, OSM and Predicted Mask.

VII. CONCLUSION

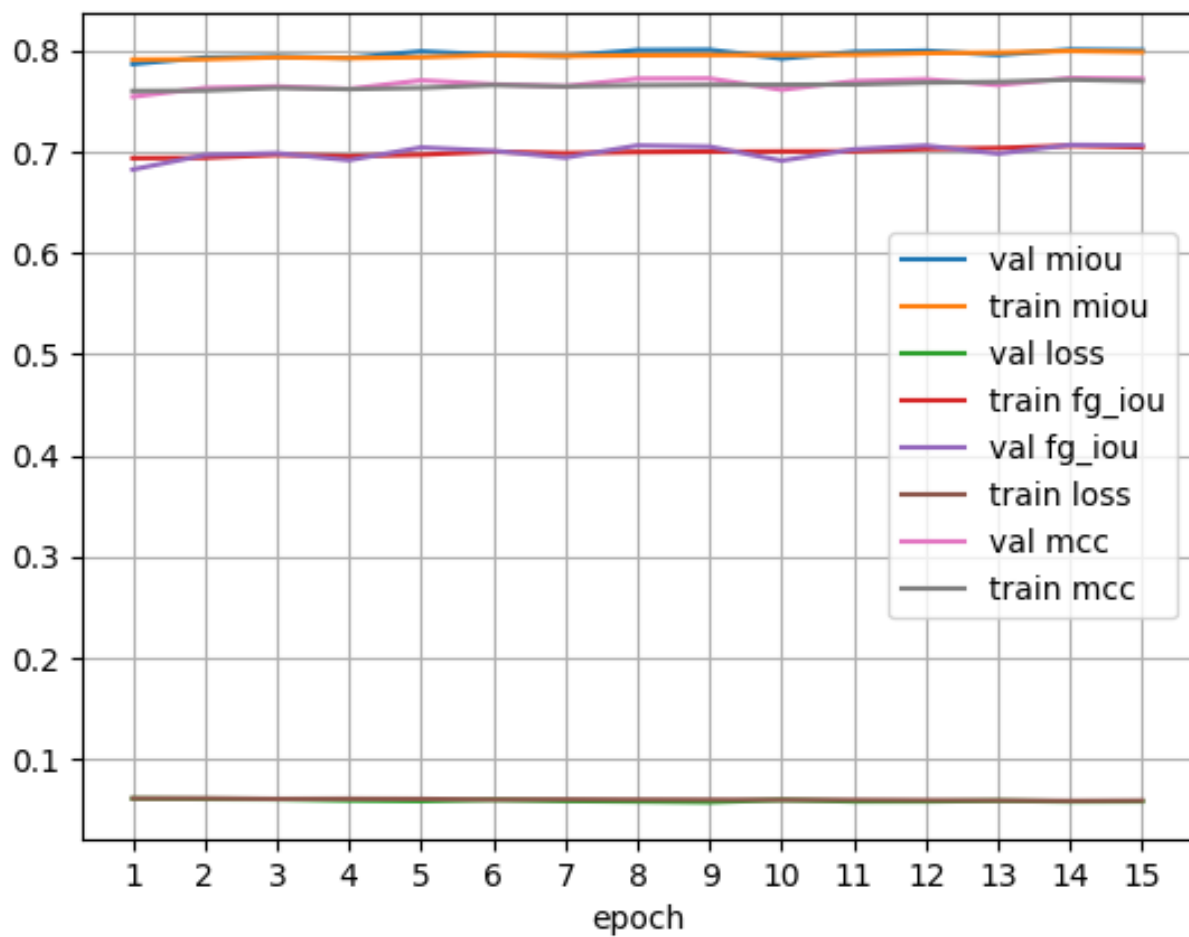
This project can be considered as a building block for multiple use-cases such as looking at every edit in OpenStreetMap in real-time to flag suspicious changesets. At the same time it can help to let good looking changesets go through without manual verification; telling how complete the map is in a specific area for a specific feature. For example: “Buildings in Sincan are 90% mapped”. And then it can show you unmapped buildings and polygon recommendations for them. This can also be integrated into imagery platforms like OSM or toolchains like OSM Drone to generate a better understanding of the area minutes after flying your drone. And while the possibilities are endless it is good to want to emphasize that the project can neither be meant for fully automated mapping nor capable of doing so because it still requires a lot of manual work and negative meaning for high predictions performances. It can only be used as a supporting tool but not for automated imports. It is a tool that can be used to improve OSM Data.

IV. Annexes

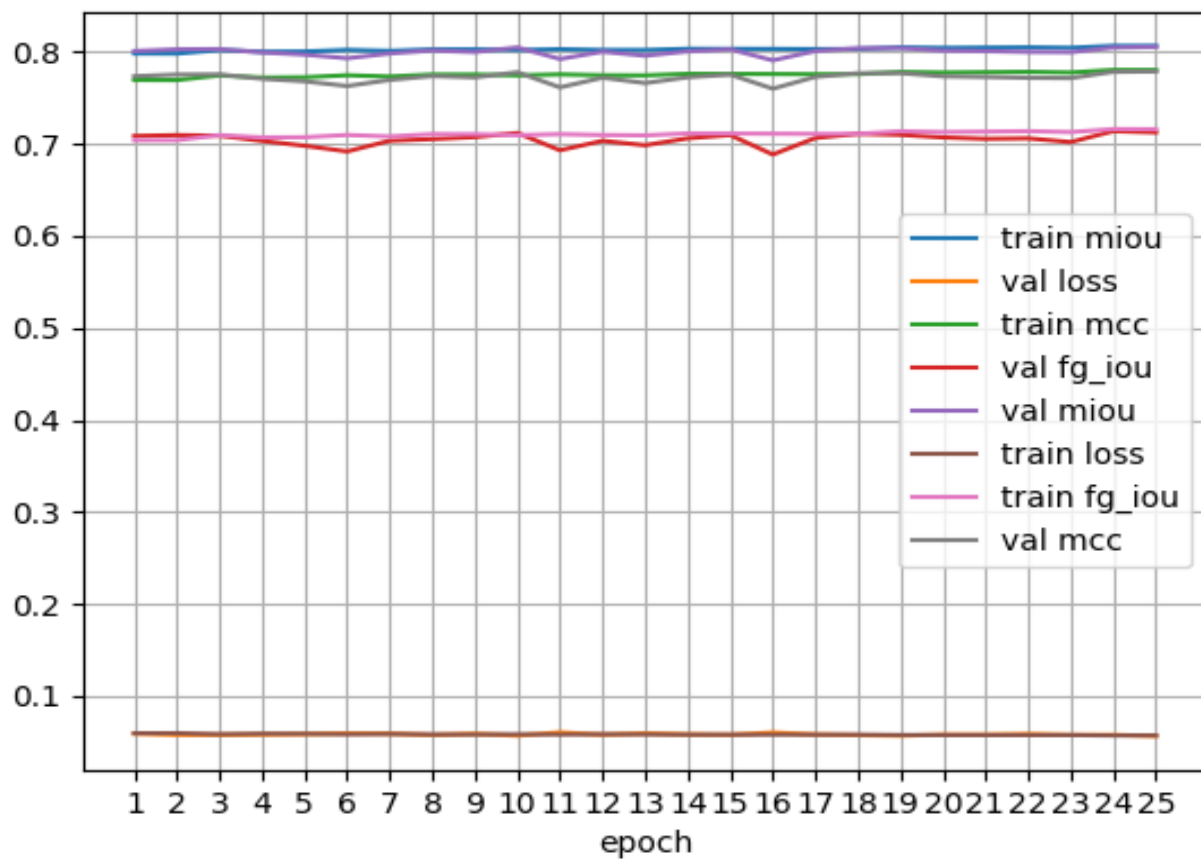
1. First 10 Epochs



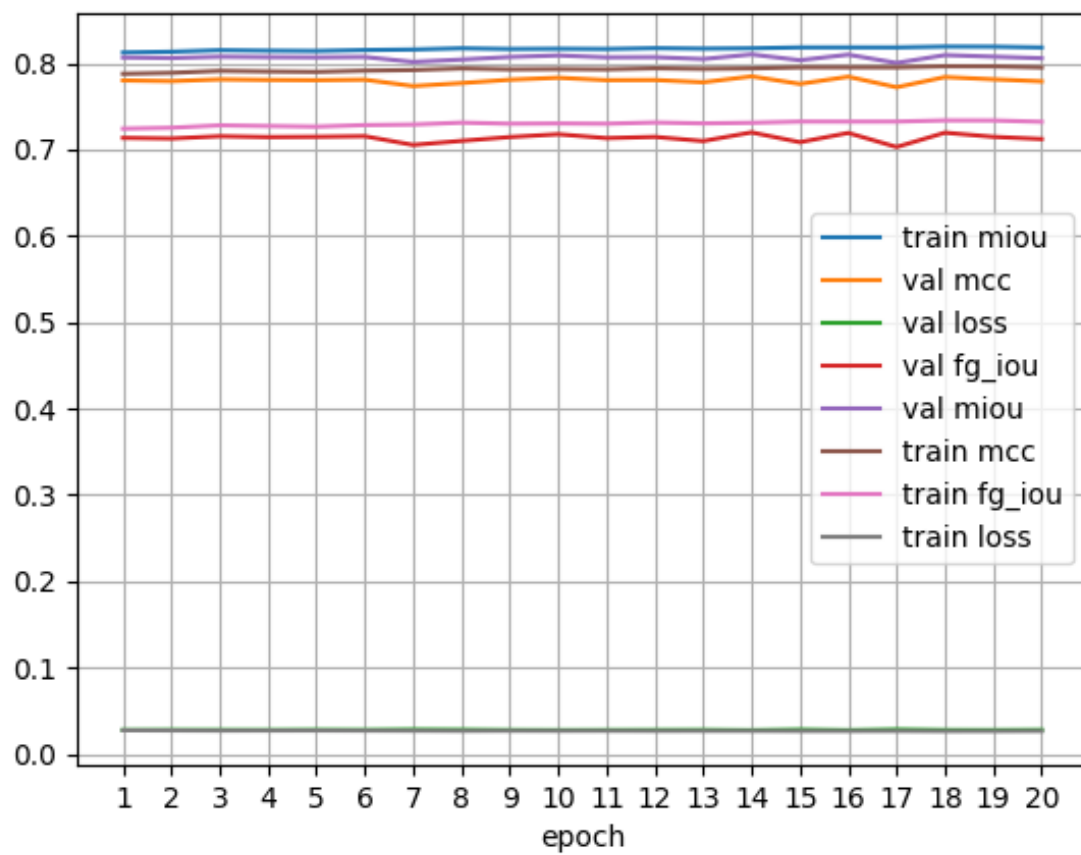
2. Following 15 epochs



3. Following 25 epochs



4. Last 20 epochs



VIII. References

1. Courtin, O. How Deep Learning could help to improve OSM Data Quality ?. State of The Map. Milano 2018
2. Mooney, P and Minghini, M. 2017. A Review of OpenStreetMap Data. In: Foody, G, See, L, Fritz, S, Mooney, P, Olteanu-Raimond, A-M, Fonte, C C and Antoniou, V. (eds.) Mapping and the Citizen Sensor. Pp. 37–59. London: Ubiquity Press. DOI: <https://doi.org/10.5334/bbf.c>. License: CC-BY 4.0
3. S. Wroclawski, "Why OpenStreetMap is in Serious Trouble", Blog.emacsen.net, 2018. [Online]. Available: <https://blog.emacsen.net/blog/2018/02/16/osm-is-in-trouble/>. [Accessed: 10- Feb- 2019].
4. DeepOSM, GitHub/DeepOSM. [Online]. Available: <https://github.com/trailbehind/DeepOSM>. [Accessed: 11- Feb- 2019].
5. "BBBike Map Compare", Mc.bbbike.org. [Online]. Available: <https://mc.bbbike.org/mc/>. [Accessed: 09- Feb- 2019].
6. Osmose - OpenStreetMap Oversight Search Engine, Osmose.openstreetmap.fr. [Online]. Available: <http://osmose.openstreetmap.fr/fr/>. [Accessed: 11- Feb- 2019].
7. Maproulette.org. [Online]. Available: <https://maproulette.org/>. [Accessed: 11- Feb- 2019].
8. M. Vakalopoulou, K. Karantzalos, N. Komodakis, N. Paragios, Building Detection in Very High Resolution Multispectral Data with Deep Learning Features, Nov 2015
9. Benjamin Bischke, Patrick Helber, Joachim Folz, Damian Borth, Andreas Dengel, Multi-Task Learning for Segmentation of Building Footprints with Deep Neural Networks, arXiv:1709.05932v1 [cs.CV], September 2017.
10. Robosat, <https://github.com/mapbox/robosat>