Open Cities AI Challenge

Segmenting Buildings for Disaster Resilience

Autonomous Agents course (CS513)

Michalis Galanis (2016030036)

Konstantinos Vlachos (2016030042)

Professor Michail Lagoudakis

COMPETITION OVERVIEW

Open Cities AI Challenge: Segmenting Buildings for Disaster Resilience is a competition hosted by DRIVENDATA which aims to accelerate the development of more accurate, relevant and usable open-source AI models to support mapping for disaster risk management in African cities.

The objective of this challenge is to map building footprints only by having access to drone imagery and thus simulating an event of a natural hazard. **Participants are tasked to build an AI model to classify the presence or absence of buildings on a pixel by pixel basis.**

This competition was completed on March 16th and our team ***"La Geo Stormers"*** was ranked **108th** out of **1106** participants.

A link to the competition's page can be accessed here:

COURSE OVERVIEW

This competition was selected as our **CS513 autonomous agents** class project. As image segmentation was revolutionized by the use of deep convolutional neural networks and is one of the most complex tasks in computer vision, this project was also a decent fit for this class.

INTRODUCTION & PREREQUISITES

In this section, we are going to introduce several terms that are essential to this project.

AI, Machine Learning & Deep Learning

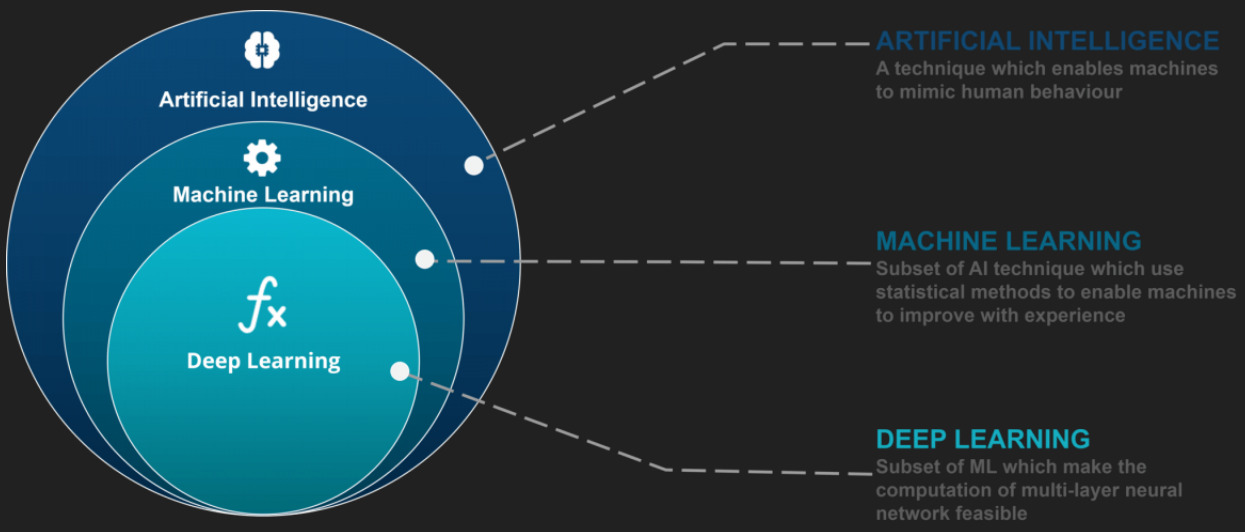
These terms are often used interchangeably but it's important to understand how these can be applied differently.

Artificial intelligence is a broader concept in which the use of computers mimics the cognitive functions of humans, basically when machines complete tasks based on algorithms in an "intelligent" manner.

Machine learning is a subset of AI and focuses on the ability of machines to learn for themselves and make statistical predictions based on the information they are processing.

Deep Learning which is a subset of machine learning takes things to the next level by using multi-layer neural networks. It can automatically discover the features to be used for classification whereas ML requires these features to be provided manually.

Neural networks are a set of algorithms that are designed to recognize patterns and are inspired by how a human brain works. Just as the brain can recognize patterns and help us categorize and classify information, neural networks do the same for computers. Neural networks and deep learning currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing.



Computer Vision tasks

There are various levels of details

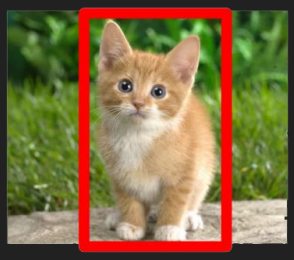
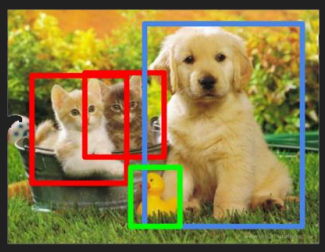
in which computers can gain an understanding of images. The input for all of the described tasks below is a high-resolution image.

The most fundamental task in computer vision is the **Image Classification** problem where given an image, the task is to output a discrete label of the main object in the image.

Classification with Localization is the next step where along with the discrete label we also expect the localization of where exactly the object is present in the image.

Object Detection takes the localization to the next level. The image can now contain multiple objects and the task is to classify and localize all the objects in the image.

Finally, there is the **semantic segmentation** task (which is the one we are going to focus on) in which the goal is to label each pixel of an image with a corresponding class of what is being represented. The expected output is itself a high-resolution image where each pixel corresponds to a particular class.

Classification	Classification + Localization	Object Detection	Image Segmentation
			

Working Environment

Python was used for this project, as it is considered the default option when working on artificial intelligence. Its simplicity, community support and the existence of thousands of libraries available makes it a compelling choice.

Jupyter Notebook was our preferred code environment. It's a document that contains both computer code and rich text elements. Notebook documents are both human-readable documents containing the analysis description and the results as well as executable documents that can be run to perform data analysis.

Google Colab was our environment of choice. It's basically a Jupyter notebook that can be executed on Google's cloud servers, meaning we can leverage the power of

Google hardware including GPUs and TPUs. This is great when working on hardware-demanding applications like machine & deep learning.

Keras was the framework we used to manage our model and network. It is a minimalist, highly modular neural networks library written in Python and capable of running on top of Tensorflow. It allows for easy and fast prototyping and supports both convolutional and recurrent networks.

Dataset

Let's take a look at the provided dataset. The features are the images themselves which are stored as large **Cloud Optimized GeoTiffs (COG)** whereas the labels containing the building footprints are stored as **GeoJSONs**. All images include 4 bands (RGBA). The labels provided are not always perfect, so the training data have been divided up into tier 1 and tier 2 subsets. The first subset contains higher quality (more accurate) labels than the latter. All training images have been reprojected to the appropriate **UTM zone projection** for the region that they represent. Meanwhile, the test set contains 11.481 1024x1024 COG fragments derived from scenes that are not present in the training set. The metadata for the competition datasets are stored in **Spatiotemporal Asset Catalogs (STACs)**. We can work with this format using **PySTAC**, a simple Python library for manipulating STAC objects.

Workflow Overview

Our deep learning pipeline can be defined in 5 stages:

1. **Gathering data**: This step is simple in our case as the competition has already provided us with all the training data required for our project.
2. **Data pre-processing**: One of the most important steps in deep learning. This is where we "clean" the raw data, meaning we are converting it to a format appropriate for our model to be trained. This includes visualizing the labels (building masks) and linking them with their corresponding images and then slicing them into lower-resolution tiles so they can become our model's input.
3. **Researching the most appropriate model**: We are dealing with a supervised learning model. Although image segmentation can be performed using a number of different architectures, we picked U-NET as our model. The U-net (a specific type of FCN) has received a lot of interest in the segmentation of biomedical images but has proven to be also very efficient for pixel-wise classification of satellite images.
4. **Training and testing the model on data**: For training the classifier, only the training dataset is available so we can later test the performance of our classifier on an unseen test dataset.

5. **Data Post-processing:** After acquiring the output of our predicted results, further post-processing is required in order to meet the submission format.

All of these steps are going to be analyzed in the following sections.

IMPLEMENTATION

Pre-processing

Before we begin processing our data, we first need to download the required files.

Data set	Link
train_tier_1	
train_tier_2	
test	

We start by installing and importing all the required libraries and APIs needed. These include **geopandas**, **solaris**, **rio-tiler**, **rasterio**, **pandas**, **descartes** and **pystac**. We then proceed to define some directories for our datasets to be extracted. The next step is to access each remote catalog and create a dictionary of its containing areas and scenes. For each scene (TIF file), the corresponding geojson label file is loaded which provides geometric information about present buildings. This file then gets visualized (converted to a mask) to match the image file. We then begin to slice each scene into many lower resolution tiles (512x512 resolution png's) so we can input them into our model. As the last step, we renamed all files to a 0-based numerical format.

Data augmentation

To ensure sufficient robustness and invariance of the network, data augmentation techniques were applied to our training set. The images went through random transforms such as rotations, width/height shifts, shear & zoom range and horizontal flips. This allowed us to train our model on a significantly larger set of images. We did not have enough time to try out different values when augmenting our data. The following table shows the selected value randomness of the augmentation process:

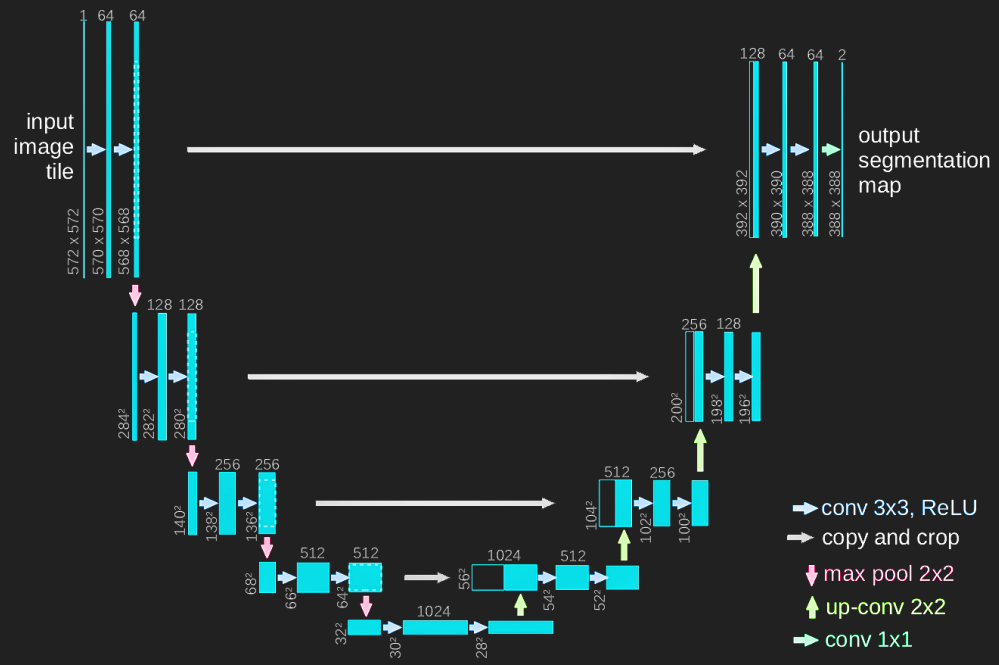
Rotation range	Width shift range	Height shift range	Shear range	Zoom range	Horizontal flip
0.5	0.1	0.1	0.1	0.1	Enabled

Training

Regarding our model, instead of developing a model from scratch, we decided to use an existing model of Convolutional Neural Network for image segmentation, the U-net. The U-net is a convolutional network architecture for fast and high-accuracy segmentation of images. Basically, it is an upgrade over the Fully Convolutional Network.

It's made of two paths, the first one being a contraction path (encoder) which extracts features of different levels through a series of convolutions, ReLU activations, and max poolings.

Then a symmetric expanding path (decoder) upsamples the result to increase its resolution of the detected features. Skip connections are added to allow context and precise localization. The same path also contains a sequence of up-convolutions and concatenations with the corresponding map. We also added batch normalization after each layer to improve speed, performance, and stability. A figure of the U-net architecture is presented below.



The loss function used to optimize our model is the **Binary Crossentropy** and it's used on true/false decisions. The loss is an indication of how wrong the model's predictions are. This loss function is represented by the following equation:

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=0}^{N}(y \cdot \log(\widehat{y_i}) + (1-y) \cdot \log(1-\widehat{y_i}))$$

where y is the true value and $\hat{y}$ is the predicted value. The Binary Crossentropy loss function measures the deviance of the true label from the predicted label. In general, this is done for each of the classes but in our case, we only have two classes: “Building” and “No Building”. After the calculations are completed, it then averages these class-wise errors to obtain the final loss. The loss function is used to optimize the model which means it will get minimized by the optimizer.

The optimizer used in our implementation was **Adam**. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iteratively based on training data. It is computationally efficient and achieves decent results fast. Adam is also an adaptive learning rate method which means it computes individual learning rates for different parameters.

The training was carried out by Google’s own GPU (known to use Tesla K80’s). Considering the memory available on the GPU we did training of mini-batches of 16 images. We went through with a cumulative training time of about 14 hours. Training was stopped after 7 epochs to avoid overfitting as no significant progress was seen.

Metrics

After the implementation of deep learning model, we have to evaluate how effective our model is based on metrics. A metric is a function that is used to judge the performance of our model. The metric used for our training was the **Average Accuracy** metric. Average Accuracy is often used as a choice of evaluation for classification problems which are well balanced which means it’s ideal for our binary classification problem. Accuracy is the proportion of true results among the total number of cases examined as shown in the following equation.

$$\boldsymbol{Accuracy} = \frac{Correct\ Predictions}{Total\ Predictions}$$

The competition also makes use of the **Jaccard Index** metric as it’s the main evaluation score for its participants. Jaccard is a similarity measure between two label sets and is defined as the intersection divided by the union. It is an accuracy metric so a higher value is better. It can be calculated as follows:

$$\boldsymbol{J(A,B)} = \frac{|\mathrm{A} \cap \mathrm{B}|}{|\mathrm{A} \cup \mathrm{B}|} = \frac{|\mathrm{A} \cap \mathrm{B}|}{|\mathrm{A}| + |\mathrm{B}| - |\mathrm{A} \cap \mathrm{B}|}$$

where A is the set of true labels and B is the set of the predicted labels. Let’s remember that the competition has access to the set of true labels (but they are unavailable to its participants) and that’s why a Jaccard metric can be calculated.

Testing

When training is completed, an hdf5 weights file is generated containing all the modified (according to our training) weight values of our network. We then can load it and either continue training on the augmented data or make some predictions on our test data. All images are outputted as 512x512 triple-band png files on a given folder with a "`_predict`" suffix at the end of the file name.

Post-processing

The competition states that the submission format should consist of a zipped file containing all 11.481 single-band 1024x1024 TIFF files named after its corresponding imagery chip in the test set.

Our output files are 512x512 binary masks with a 0-based format name. Having stored their real file names before entering our prediction model, we proceed to restore them. We also resize the final images back to 1024x1024 resolution and we finally construct an archive of the submission folder so we can upload our submission.

RESULTS

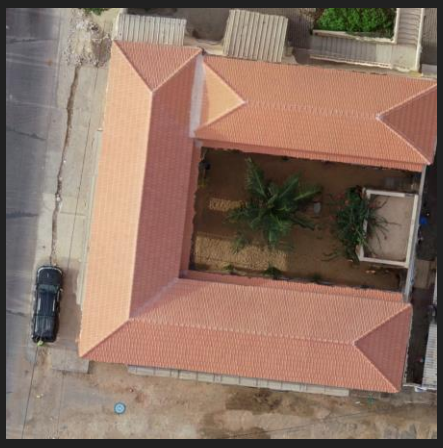
Images

We present below the predictions outputted by our model on some testing data:

Original image (0ab0d6.tif)

Prediction (0ab0d6_predict.tif)

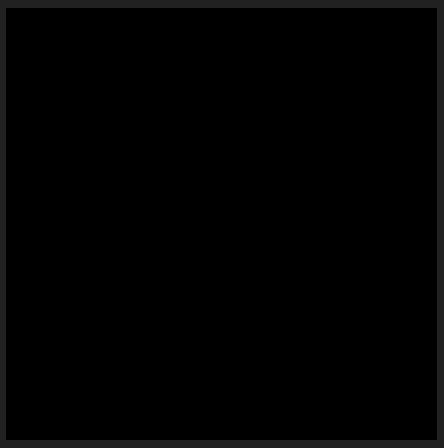
Original image (9df909.tif)



Prediction (9df909 _predict.tif)

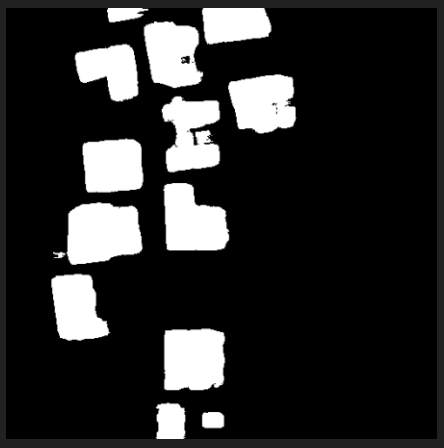
Original image (036484.tif)

Prediction (036484_predict.tif)



Original image (9b7841.tif)

Prediction (9b7841_predict.tif)



We can see from the results above that we achieved a decent precision across different scenes. Our training accuracy reached over 96% and we obtained a loss coefficient of 0.07.

Individual Contributions

During the course of this project we mainly worked together to discuss data pipeline, choice of architecture, implementation and problem-solving. In terms of code, both members of the team contributed equally to the creation of the preprocessing stage of the training dataset. Michalis focused on the training phase of the project (network selection, Keras implementation, training parameters, weight generation, etc.) meanwhile Konstantinos focused on the testing phase including the pre-processing required of the testing dataset as well as the post-processing of the predicted results in order to satisfy the submission format. Konstantinos also performed a final code cleanup.

Presentation-wise, the report and the website were mainly developed by Michalis while the presentation was mainly created by Kostas.

Conclusion

In this project, we made a comprehensive analysis for semantic segmentation of satellite images for building detection. The datasets were provided by the competition itself, resulting in more than 60GB of training data and 8GB of testing data. We implemented a UNET architecture and used the Keras API to implement it and augment our data in order to improve the robustness of the model.

REFERENCES

1. A Keras Pipeline for Image Segmentation
2. Keras Image Preprocessing
3. Keras Metrics
4. Keras Optimizers
5. Keras Loss Functions
6. Step-by-step introduction to Image Segmentation Techniques
7. Competition's community forum (Getting Started)
8. UNet — Line by Line Explanation