Bsc. Nikolaus Heran

Semantic Drone Dataset - Creation and Evaluation

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Telematics

submitted to

Graz University of Technology

Supervisor

Prof. Dr. Friedrich Fraundorfer
Institute for Computer Graphics and Vision

Dipl.-Ing. Michael Maurer
Institute for Computer Graphics and Vision

Graz, Austria, Sep. 2018

Abstract

Datasets to train and evaluate semantic segmentation methods for images taken in nadir perspective during low altitude Unmanned Areal Vehicle (UAV) flight are to the best of our knowledge unknown. At the same time such datasets are a necessary prerequisite to enable autonomous landing of drones in unknown environments when dense semantic segmentation algorithms should be part of the decision making process. To bridge this gap we introduce a novel dataset with 400 densely annotated high resolution (6000x4000) images taken in nadir perspective from low altitude flight. We show by which criteria we selected an established tool and how we modified it to allow the creation of the dense annotations. Further we explain how we used the tool together with Amazon Mechanical Turk to obtain the annotations with a low budged. Finally the quality of the created dataset is shown by training and testing the two state of the art algorithms FC-DenseNet103 [13] and DeepLabV3+ [7] on it.

Keywords. dense, semantic, segmentation, nadir, UAV, drone, autonomous, tool, tooling, crowdsourcing, dataset

Kurzfassung

Nach unserem derzeitigen Wissensstand sind Datensätze zum Training und der Evaluierung semantischer Segmentierungsmethoden für Bilder aus der Nadir Perspective von niedrig fliegenden *UAV*s unbekannt. Gleichzeitig sind solche Datensätze eine notwendige Voraussetzung um das autonome Landen von Drohnen in unbekannter Umgebung zu ermöglichen, wenn dichte semantische Segmentierungsalgorithmen Teil der Entscheidungsprozesses sein sollen. Um diese Lücke zu schließen stellen wir einen neuartigen Datensatz mit 400 dicht annotierten hochauflösenden (6000x4000) Bildern in Nadir Perspective aus Niedrighöhenflug vor. Wir zeigen nach welchen Kriterien wir ein etabliertes Werkzeug ausgewählt, und wie wir es modifiziert haben um die Erstellung dichter Annotationen zu ermöglichen. Desweiteren erklären wir, wie das Werkzeug gemeinsam mit Amazon Mechanical Turk verwendet wurde, um die Annotationen mit einem niedrigen Budged zu erhalten. Abschließend wird die Qualität der erzeugten Datensatzes gezeigt, indem zwei Algorithmen vom neuesten Stand der Technik, FC-DenseNet103 [13] und DeepLabV3+ [7], darauf trainiert und getestet werden.

Schlüsselwörter. dicht, semantisch, Segmentierung, Nadir, UAV, Drohne, autonom, Werkzeug, Crowdsourcing, Datensatz

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

Date	Signature

Acknowledgments

First and foremost I would like to thank my supervisors Prof. Fraundorfer and Dipl.-Ing. Michael Maurer. Further I want to express my gratitude to the whole areal vision team of the ICG who contributed the raw image data which is the basis for the semantic drone dataset. I would like to thank Kathryn Hughes, Manuela Vasconcelos and all other Amazon Mechanical Turk workers wo created the vast majority of the annotations for the dataset. Last but not least I want to thank my girlfriend, my friends and my family for their patience and emotional support without which I would have not finished this thesis.

Contents

List of Figures

List of Tables

1 Introduction

Contents

1.1 Motivation

Since their introduction [16] deep fully Convolutional Neural Network (CNN) methods have been dominating computer vision in general and the field of semantic segmentation in particular. Thanks to sensor fusion and the recent advancements in computer vision autonomous cars are on the horizon. We follow this development with great interest since building the decision making processes for self driving cars and Unmanned Areal Vehicles (UAVs) in low altitude flight based on image input appear to have great parallels. Therefore we wanted to harness this synergy and explore if dense semantic segmentation methods developed for self driving cars also work for our usecase. We want to do so by using state of the art methods on images taken in nadir perspective from a drone. Finally we want to know if the predictions are sufficiently accurate to be used for decision making. To determine this we introduce a novel data set that to the best of our knowledge is the first with dense semantic segmentations on high resolution images taken from nadir perspective in low altitude flight of *UAVs*.

1.2 Outline

The success of heuristic models has made the field of semantic image segmentation research highly dependent on the data used to train them. In Chapter 2 we list some state of the art algorithms and discuss popular and task specific semantic segmentation datasets used to train and evaluate said algorithms. The focus of this chapter is laid on semantic

segmentation for high resolution images and outdoor scenes from Unmanned Areal Vehicle (UAV) perspective. Especially for high resolution images the creation of dense semantic segmentation ground truth is a time intense and cumbersome process. In this work we show what properties an annotation tool needs to exhibit in order to create useful dense semantic segmentations on a whole dataset. We conduct a survey on existing tools and compare them to the aforementioned properties.

In Chapter 3 we show how and why we modified the annotation tool used to create LabelMe [28, 32] to fit those properties. Further we show how to use that tool in conjunction with a crowdsourcing platform to create our dense semantic segmentation ground truth.

In Chapter 4 we train the two state of the art algorithms FC-DenseNet103 [13] and DeepLabV3+ [7] on our data set. The performance of these methods shows not only that the created labels are consistent and dense enough to train state of the art networks on it, but also that the novel perspective poses different challenges than previous data sets did.

Finally we summarize our findings and possible future work in Chapter 5.

2 State of the Art

Contents

2.1 Datasets and Algorithms

The research of machine learning algorithms and the creation of datasets to evaluate them on have been strongly intertwined since the computational capabilities exist to execute the algorithms on meaningful data. Most publications of new algorithms include their evaluation on existing datasets in order to increase objectivity in the analysis of the benefits and downsides of the published approach. The reliance on datasets is especially pronounced for all approaches using deep neural networks since they rely on a substantial number of annotated images to reach state of the art performance.

Data sets like PASCAL VOC 2012 [1] [Everingham et al.], ADE 20K [2] [37, 38], ImageNet [3] [27] and MS COCO [4] [15] attempt to capture the whole or at least a large variety of classes. Large datasets of this kind often contain many thousand images with class numbers around the second order of magnitude. Although most datasets at this scale try to represent a large variety of scenes there are some as SUN RGB-D [5] [30] that are specialized on a specific sub set of scenes.

The creation of datasets with this magnitude is still a very resource and time intense process. Therefore many smaller datasets have been created with a specific focus or task

[1] http://host.robots.ox.ac.uk/pascal/VOC/voc2012/
[2] http://groups.csail.mit.edu/vision/datasets/ADE20K/
[3] http://www.image-net.org/
[4] http://cocodataset.org/
[5] http://rgbd.cs.princeton.edu/

in mind. Even if these smaller datasets are not sufficiently large to train networks from scratch they can be used to fine tune a network to the desired task if it was pre-trained on a bigger dataset. To name just one example for a specific motivation the Graz02 dataset [6] [18, 21] has the goal of highlighting the difficulty of segmenting the class of bicycles. Sometimes a smaller dataset like NYU Depth dataset V2 [7] [19] and evaluation of algorithms on it can motivate the creation of a larger task specific dataset like SUN RGB-D [30].

Another area that has gained attention recently is the semantic segmentation of street scenes driven by the desire to enable autonomous cars. GaTech [8] [23], CamVid [9] [3, 4] and CityScapes [10] [8] have been created with this task in mind. A closely related task is the development of autonomous Unmanned Areal Vehicles (UAVs). Especially for low altitude flight and landing procedures the scenery is very similar to the one encountered by autonomous cars. The segmentation of images taken from *UAVs* is however a slightly more difficult task since the camera orientation is unusual and not as fixed as it is for cars. Further the scale and viewpoint of objects has a higher variety for drone imagery. Lastly the fast movements caused by the agility of many *UAVs* increases the probability to encounter motion blur. The unusual camera orientation makes it harder to find relevant datasets available to the public. Most datasets as the Inria Areal Image Labeling dataset [11] [17] and Vaihingen UAV [12] [26] feature images taken from higher altitude. Some contain images taken from a relevant height as the Stanford Drone Dataset [13] [25], the MMSPG Mini-drone video dataset [14] [1] or VIRAT Video Dataset [15] [20]. The annotations are either restricted to bounding boxes [1, 25] or single class segmentations [17]. The only exception being [20] which features polygons for multiple instances. To the best of our knowledge there are no datasets available that feature dense image segmentations of images taken from nadir view point in low altitude flight. Therefore we decided to create our own.

Previous state of the art publications in semantic image segmentation such as the one hundred layers tiramisu [13] used lower resolution datasets such as GaTech [23] or CamVid [3, 4] for training and testing purposes. With current state of the art approaches as DeepLabV3 [6], DeepLabV3+ [7] and the Pyramid Scene Parsing Network [36] a trend to higher resolution datasets such as PASCAL VOC 2012 [Everingham et al.] and CityScapes [8] has set in. This development seems to be driven by new stronger hardware as well as better software to distribute the calculations on multiple GPUs. This motivated

[6] http://lear.inrialpes.fr/people/marszalek/data/ig02/
[7] https://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html
[8] http://www.cc.gatech.edu/cpl/projects/videogeometriccontext/
[9] http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/
[10] https://www.cityscapes-dataset.com/
[11] https://project.inria.fr/aerialimagelabeling/
[12] http://www2.isprs.org/commissions/comm3/wg4/detection-and-reconstruction.html
[13] http://cvgl.stanford.edu/projects/uav_data/
[14] https://mmspg.epfl.ch/mini-drone
[15] http://www.viratdata.org/

us to also create our dataset with high resolution images of 6000x4000 pixels.

2.2 Dataset Creation

The main goal of many datasets is to provide a meaningful ground truth for others to compare their work against. Often a comparison to the human ability is considered the best benchmark. Consequently the ground truth is often obtained though annotation of images by hand. Annotating natural images for dense semantic segmentation by hand is a cumbersome and most of all very time consuming task. The time needed to perform the same quality segmentation increases with the image resolution and is inversely proportional to the scale and number of the objects on the image.

One way to address the time requirements and still get man made annotations in a reasonable time frame, is distributing the work among multiple people. Using a crowdsourcing platform gives access to a large workforce quickly, reduces the organizational overhead and simplifies financial transactions. The platform provider collects tasks from multiple sources which allows to keep a steady workforce even if individual requesters have currently no tasks available. The research group only transfers money to the platform provider who handles paying the individual workers for their labor. Using such a crowdsourcing platform usually incurs also additional costs namely the cut the platform providers retain for their services. There are many crowdsourcing platforms available to choose from. Among them Amazon Mechanical Turk [16], CloudFactory [17] and CrowdSource [18] to name just a few.

Among these Amazon Mechanical Turk has been used most often and in [15, 24, 31, 35] it has been shown that useful datasets can be created with the crowdsourcing platform. Therefore we also decided to obtain our annotations via Mechanical Turk and the people behind it.

Amazon uses the following noteworthy nomenclature and standards on their platform. The person or group that desires a distributed workforce is called a requester. Requesters can publish work in bundles called Human Intelligence Task (HIT). A *HIT* consists among other parameters of a description, the number of wanted assignment submissions, reward per assignment, an objective to be completed and the maximal allowed duration for a single attempt on it. Simpler variants for objectives are a one of many selection or a free text questionnaire. The requester can specify how many times a *HIT* may be performed thereby limiting the number of so called assignments offered by the *HIT*. Further it is possible to define qualifications that workers must have before they may accept a *HIT*. What happens when a worker submits an assignment can be defined in two ways. Either the assignment is automatically accepted or the requester is informed that a new assignment is ready for review. Has the assignment not been performed according to the instructions it can be

[16] `https://www.mturk.com/`
[17] `https://www.cloudfactory.com/`
[18] `https://www.crowdsource.com/workforce/`

rejected which increases the counter for the open assignments again. If it is accepted the reward defined in the *HIT* is transferred to the worker account and a fee of 20% of the worker reward is collected by Amazon. The process to grant qualifications is very similar to the one for assignments only that no money is transferred and the worker can receive the qualification as a reward. Should an assignment submission need changes then requesters can also choose to message the responsible worker instead of rejecting it outright. In the same way workers can message requesters if they have questions regarding the instructions or problems with the required tools.

The quality and consistency of annotations can be significantly improved by selecting and educating workers via a qualification process. A qualification test can have multiple forms. Most crowdsourcing platforms offer basic qualifications tests such as the number of assignments done, percent of successful assignments (i.e. not rejected), country, etc. While these metrics can be useful to guess the general work ethic and some basic skill (e.g. language proficiency) they do not provide any insight on the more task specific requirements. Custom qualification tests that can be created on most crowdsourcing platforms allow to remedy this problem. These custom tests are usually similar to the real tasks with the difference that the desired ground truth is already known. In general there are two extremes how the test results are evaluated by hand or by an algorithm. Evaluation by hand can be done for any kind of task and has the advantage that little implementation effort is necessary. While this approach does not easily scale to many requests - unless distributed again - it allows to individually educate the workers until they reach the desired proficiency. Using automated comparison to the ground truth with an algorithm to evaluate the requests will scale to large number requests easily, but the workers need to acquire the required proficiency without human guidance.

Combining image segmentations performed by different workers is another way to increase the quality. Worker activity can be combined either implicitly or explicitly. A simple way to combine work implicitly is to let multiple workers label the same image separately and then only use the labels where the different segmentations agree. The implementation effort for this approach is low, since a labeling tool needs to exist anyway and the consensus is easily calculated. The downside of this approach is that difficult image regions often do not reach consensus which leads to many unlabeled regions. This problem can be addressed by increasing the number of combined segmentations by different workers.

The explicit way is to have workers collaborate actively, so that they can see the labels other(s) create. As shown in [33] this has the benefit that a consensus for difficult regions can be found with fewer participants. This form of combining worker results requires more implementation effort compared to the former one. Regardless of the chosen approach combining segmentations from multiple workers increases the price per label since more man hours are required. A creative way to lower the price is to make the process enjoyable by building a game around the task. This works well for pure classification tasks as shown in [24, 33, 34]. Since almost any task can be made into a game, this way of collaboration

can also be used for semantic segmentation. Even though it reduces the worker wage per label, it still incurs an increased price per segmented image, since the game needs to be designed and implemented.

2.3 Annotation Tools

In order to obtain the reference labels for our dataset we required a tool to annotate the images. Ideally this tool should:

1. be intuitive and easy to use
2. require no user interaction during installation
3. allow many different label classes
4. allow fine grained annotation
5. allow for multiple annotations to occlude each other
6. be compatible with the crowdsourcing platform of our choice
7. work on most common operating systems
8. be available for free
9. use a permissive license to allow for extensions

Currently Mechanical Turk itself offers tools for image tagging and classification but none for segmentation. Thus we needed to find or create a tool capable of building semantic segmentations with many label classes in a dense manner for high resolution images.

2.3.1 Annotation Tool Survey

Upon examining our requirements for the tool it became evident that using a web based solution would immediately take care of points 2, 6 and 7. Especially if this web application is built on a widely used language such as Python, JavaScript or PHP, most users already have the required dependencies installed. Either they are already built into their webbrowser or are likely to be installed because other websites use them too. Widely used frameworks for these languages also support the most common operating systems. Therefore we decided to narrow the search to web based tools. It should be noted however that there are many good offline tools available, with a large spectrum of features that are not listed below. Our search showed that there are many web tools available for annotating rectangular bounding boxes, such as

- JavaScript for Object Detection Annotation [19]

[19] `https://github.com/wiany11/jsoda/`

- the Image Annotation Programme [20]
- the Fast Image Data Annotation Tool (FIAT) [21]
- the Simple Image Annotator [22]

The tools listed above only offer rectangular annotations. Since object boundaries can have any form these tool are insufficient for dense semantic labeling. Below we list a selection of tools that provide the option to draw object outlines using polygons and are therefore better suited to create semantic segmentations.

2.3.1.1 VGG Image Annotator

The VGG Image annotator [23] [9] offers a clean and well organized user interface. Additionally to the already mentioned rectangular and polygonal shapes also spherical and ellipsoid annotations can be made. Completed annotations remain visible on the image and can also be browsed in a separate table as can be seen in Figure 2.1. New columns can be added to the table to store more information about each annotation. Once created, the annotations with the meta information can be downloaded from the website. It is possible to load more than one image at the same time. This allows for an easy workflow when there are many images to label. First all of them are loaded, then one after another is annotated. Showing multible images at once is not possible however. There is no built in way to preserve occlusion information. Thus it is unable to provide semantic segmentations for more than two object classes.

Development and maintenance of VGG Image Annotator (VIA) is supported by EPSRC programme grant Seebibyte: Visual Search for the Era of Big Data (EP/M013774/1).

2.3.1.2 Annotorious

Annotorious [24] is as the other listed tools an open source project. It does offer bounding box and polygonal annotations. An optional comment can be saved for each annotation. This comment can be used to store the label class. The tool offers a JavaScript Application Programming Interface (API) that allows to extend it as desired. The mentioned polygonal annotations are still in the beta phase of development. Therefore the reliability is unknown. Beyond displaying the annotations on the images no user interface for managing the annotation is provided as can be seen in Figure 2.2. Multiple images can be shown to and annotated by a single user on the same webpage. In contrast to the other tools Annotorious does not provide a zoom function. The size of the image on the

[20] https://github.com/frederictost/images_annotation_programme
[21] https://github.com/christopher5106/FastAnnotationTool
[22] https://github.com/sgp715/simple_image_annotator
[23] http://www.robots.ox.ac.uk/~vgg/software/via/
[24] https://annotorious.github.io/about.html

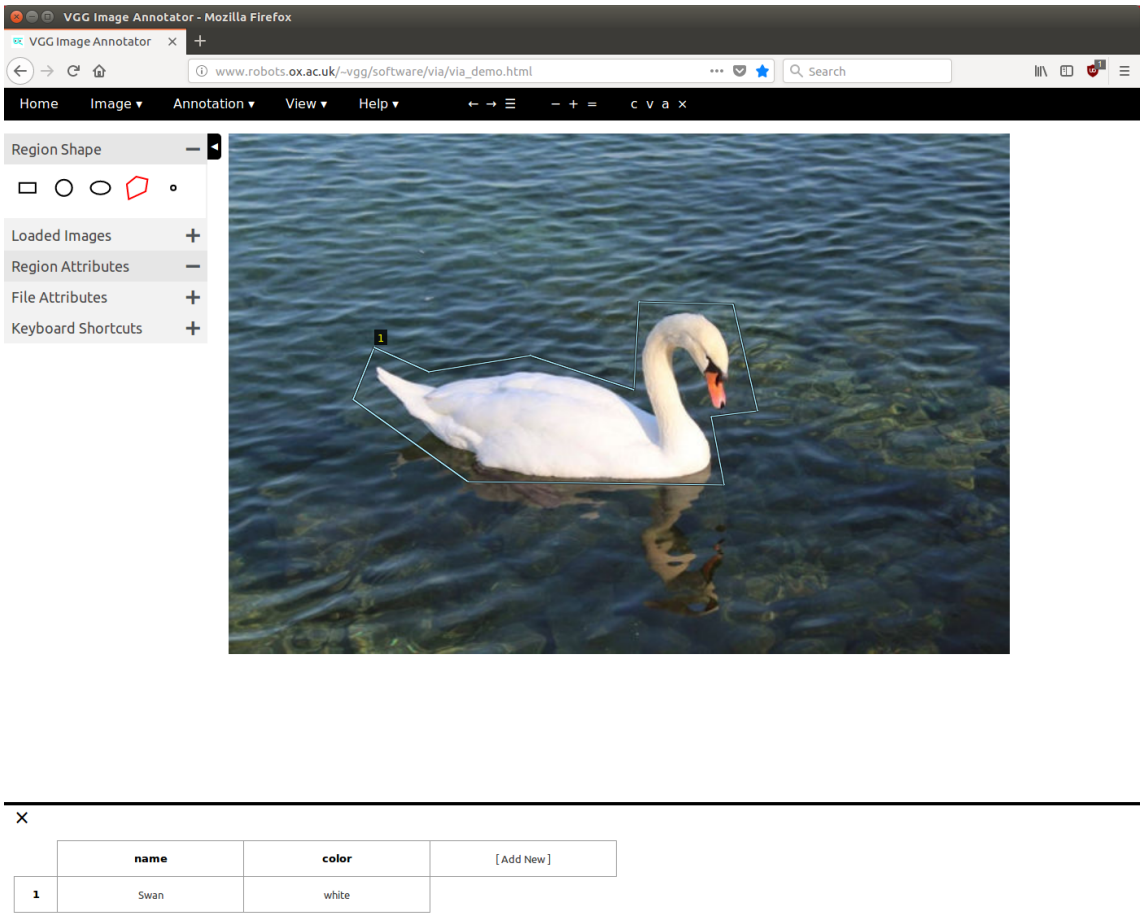


Figure 2.1: The VGG Image Annotator as seen in Mozilla Firefox 59.0.2 (64-bit) on Ubuntu 16.04.4 LTS. In the middle we see the image to be annotated. To the left the form of the annotation can be chosen. At the bottom of the screen a table containing the annotations and additional attributes is visible.

screen can only be changed using the browsers zoom function. As with the VGG Image annotator there is no built in way to handle occlusions which hinders the creation semantic segmentation.

2.3.1.3 Annotation of Image Data by Assignments (AIDA)

The Annotation of Image Data by Assignments (AIDA) [25], by Rittscher, J. and Alham, N. K. and Aberdeen of the Quantitative biological imaging team from the University of Oxford, also offers a well organized interface that seems to be inspired by or even built with material design. It is evident that it was created to annotate images of fluids or tissue photographed through a microscope. Multiple registered images called channels can be

[25] `https://github.com/alanaberdeen/AIDA`

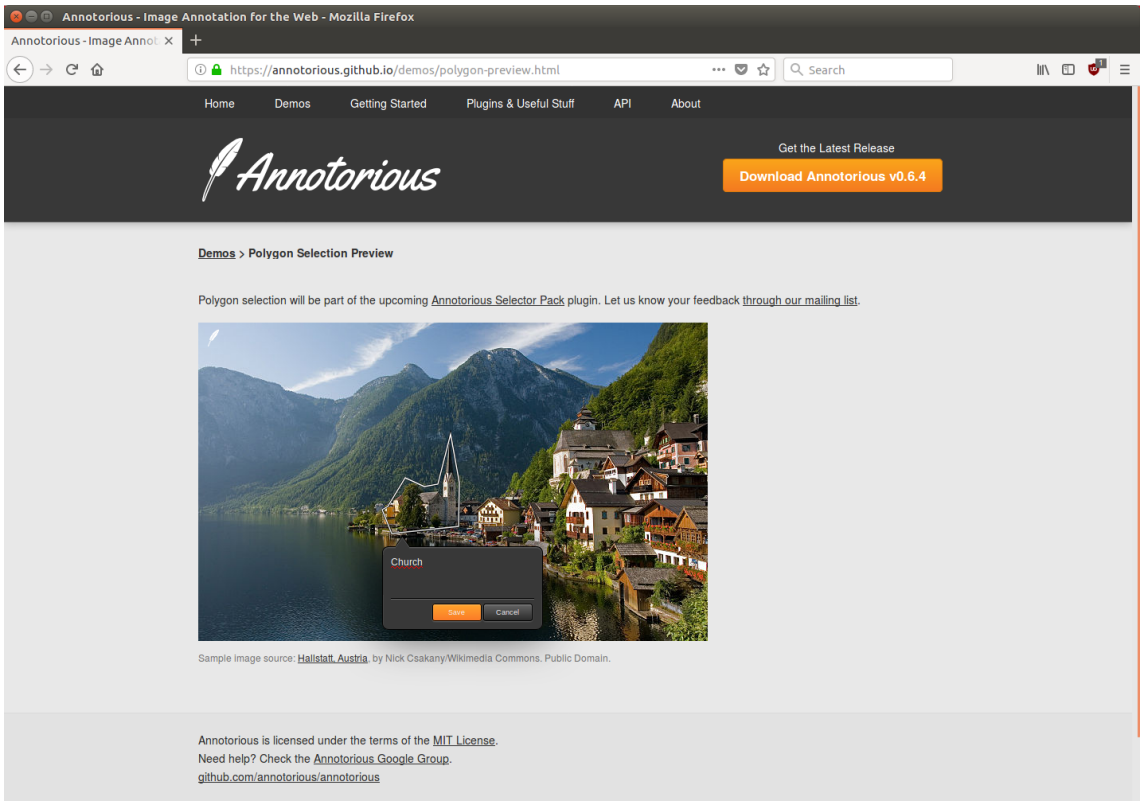


Figure 2.2: Annotorious demo as seen in Mozilla Firefox 59.0.2 (64-bit) on Ubuntu 16.04.4 LTS.

loaded. These can then be displayed together by combining them using a different opacity for each image. It is possible to draw circles, polygons and paths. Further it is possible to count annotations inside a rectangular region and create layers. Figure 2.3 shows the tool demo page with some annotations made. In contrast to most other labeling tools the annotations are not named directly. Instead each annotation is part of the named layer that was selected during its creation. This layer then defines the label class of all contained objects. Using the layer to define the label class is sufficient for classes that have a natural hierarchy, such as cell boundaries and cell nuclei. If occlusion information is however independent of the class label this approach fails to provide satisfactory segmentation results.

2.3.1.4 LabelMe Annotation Tool

The LabelMe Annotation Tool [28, 32] is part of the LabelMe framework and website [26] developed by the Computer Science and Artificial Intelligence Laboratory at MIT. Annotations can be made by drawing bounding boxes, polygons and using a similarity selection brush. The similarity selection applies a min cut / max flow approach [2] to the pixel

[26] `http://labelme.csail.mit.edu/Release3.0/`

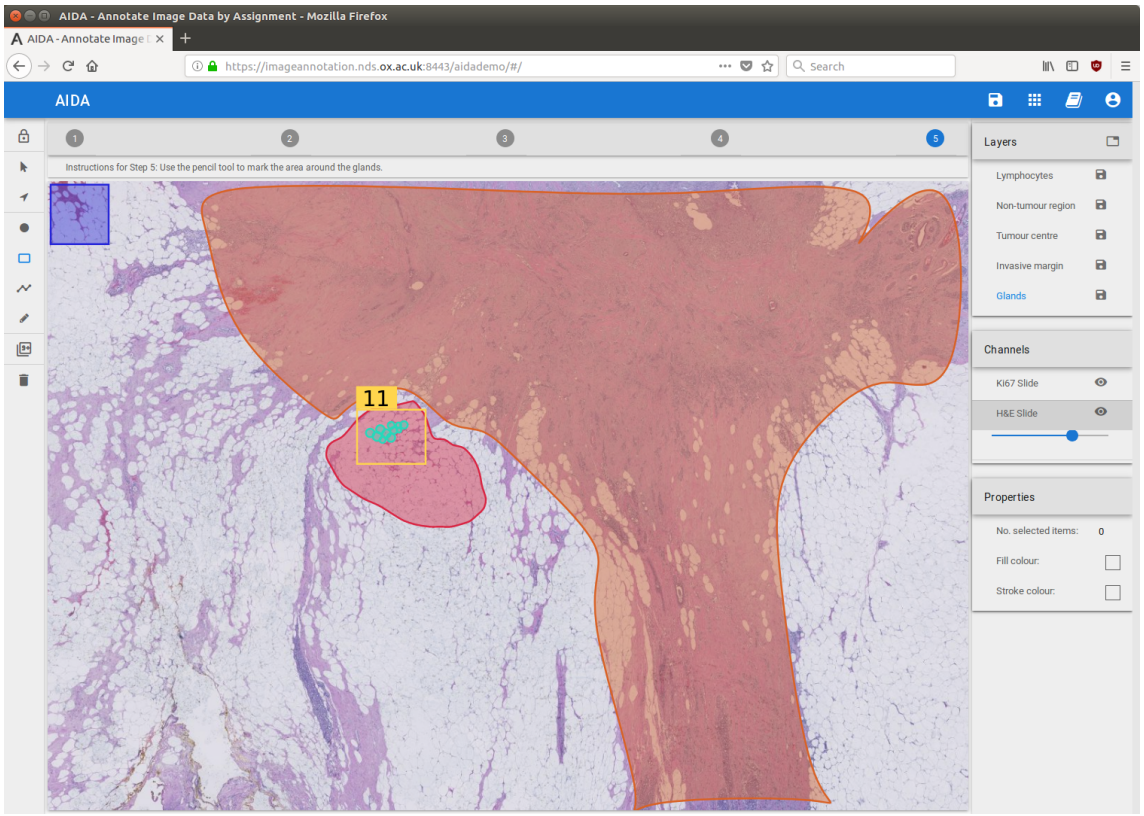


Figure 2.3: AIDA demo as seen in Mozilla Firefox 59.0.2 (64-bit) on Ubuntu 16.04.4 LTS. The annotation tools can be selected to the left. On the right side the annotation layers and image channels can be seen.

colors to determine the boundary of the polygon. Each annotation must be given a name, that can be chosen freely. To guide the naming a list of suggestions can be parametrized. Additional attributes of the annotation can be saved in a text field. Further it is possible to mark the object as occluded and the annotations can be ordered in a part whole hierarchy. The image can be zoomed in distinct steps. The step size can be configured but regardless of the choice the maximum possible zoom is 1000% of the original image size. Up to a point the browsers zoom function can be used as a workaround for this limitation. Objects and their parts can be highlighted by hovering over them in the object list. Lastly it is possible to move parts between objects via drag and drop in the object list. Creating semantic segmentations with multiple classes is possible out of the box with this tool, though some drawbacks should be mentioned. If three or more objects from different classes occlude each other then labeling becomes cumbersome. At least one boundary has to be traced twice since the object order cannot be determined beyond a binary split.

The tool was already used to create a database [28] with more than 1000 fully annotated and roughly 2000 partially annotated images. Figure 2.4 shows an example image from the data base [28] in the tool. Matlab and bash scripts using the Command Line Interface

(CLI) of Amazon Mechanical Turk to access their workforce are provided as well. The annotation tool [27] and the aforementioned scripts [28] were kindly made available. The tool is published under the very permissive MIT License. The scripts for Mechanical Turk use are published in a separate repository under the more restrictive Amazon Software License [29].

The tool itself is mainly written in javascript using the jquery library [30] but also relies on php, python, perl and some bash scripts as well as a combination of html and css documents.

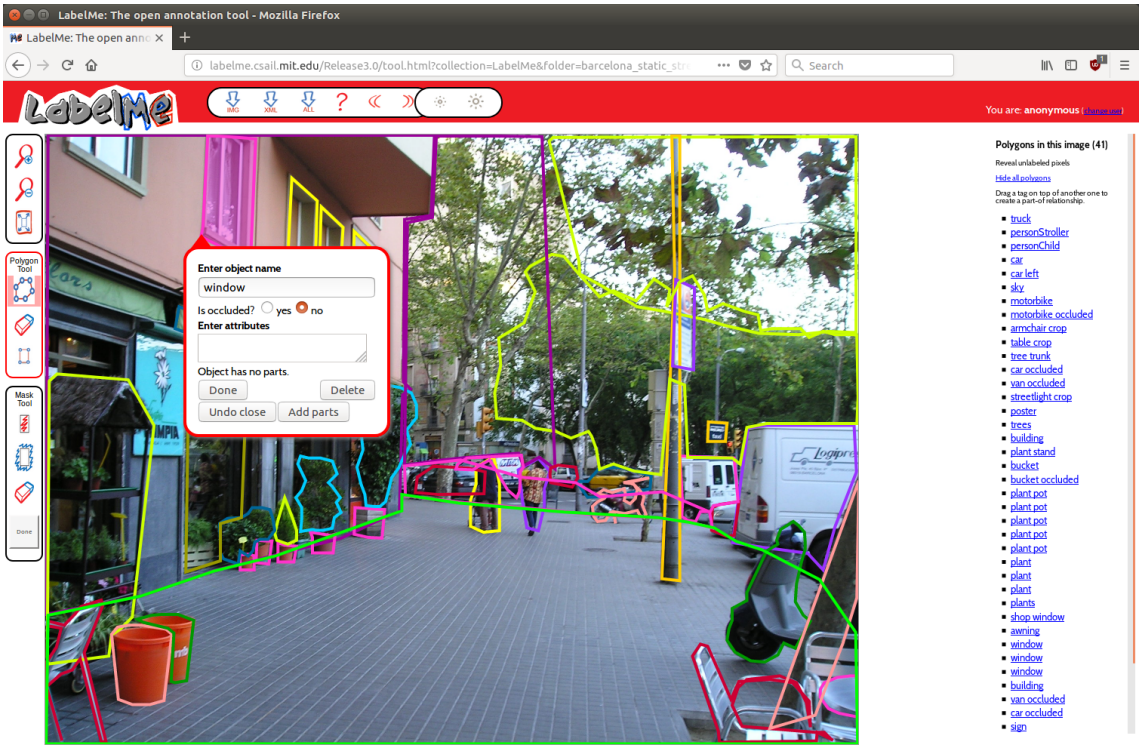


Figure 2.4: The original LabelMe Annotation Tool as seen in Mozilla Firefox 59.0.2 (64-bit) on Ubuntu 16.04.4 LTS. In the center we see the image and on top the annotations in different colors. The annotation modes can be selected to the left. The already created annotations are listed on the right.

[27] https://github.com/CSAILVision/LabelMeAnnotationTool
[28] https://github.com/CSAILVision/LabelMeMechanicalTurk
[29] http://aws.amazon.com/asl
[30] https://jquery.com/

3 Tool Modification and Usage

Contents

None of the surveyed tools seemed to be ideal for the purpose of creating dense semantic annotations. Since creating our own tool from scratch was out of the scope of for this thesis we decided to choose the best suited tool and modify it to our needs. From all surveyed tools the LabelMe Annotation tool fit the requirements mentioned on page 7 best. Additionally it appeared that it would be simple to use with Amazon Mechanical Turk since interfacing scripts were already provided. Before we set up the tool together with the crowdsourcing platform we performed some manual trials ourselves to gauge which improvements we should make before deployment. The fuzzy selection tool worked quite well but introduced an unwanted bias, thus we performed our annotations mainly by drawing polygons and boxes.

3.1 General Modifications

The label tool allows arbitrary names for the annotations. This is useful if the set of labels is not fixed. However for our case the set of labels is fixed. Therefore we changed the tool to only accept names that match one of our labels. Further we fixed the color of the shown polygons and boxes to match the label colors in our colored label images. We introduced these modifications to reduce and ease finding of annotation errors as well as to aide evaluation.

We also decided to remove the fuzzy selection tool because it would introduce a systematic bias into the annotations. Further the download buttons at the top were removed.

3.2 Addressing Gaps and Occlusions

The first mayor point we needed to address was the fact that the tool stores the annotations only in a xml format and does not provide any modality to create a semantic label image from it. Therefore we created a python script to parse the xml generated by the tool and convert it to a label image. Each pixel in the label image represents the label of the segment it belongs to in the color image. This early version of the script first read in all annotations and split them into two groups by occlusion status. Then it drew the annotations in two batches. The first batch to be drawn only contained the polygons that were marked as occluded. The second batch was drawn on top of the first and contained the remaining polygons. Inside each batch the polygons were drawn in the same order they were created in. While this process worked well to create a label image from the annotations it had a significant downside. When polygons with the same occlusion status overlap then the final label of the overlapping area depends on the order of creation.

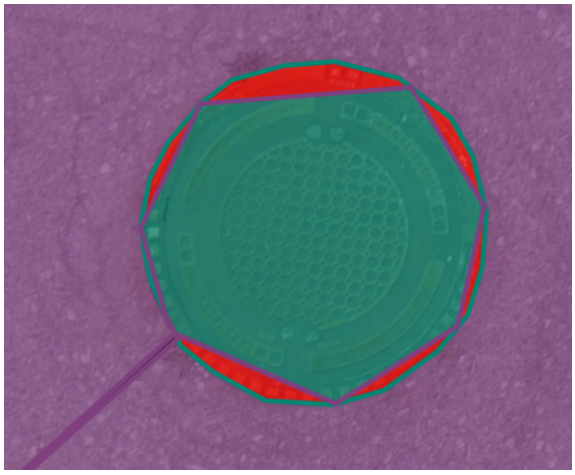


Figure 3.1: An intentionally exaggerated example of the disadvantage of the snap tool. The incorrectly labeled areas are marked in bright red. Also note the double edge of the purple polygon for the paved-area in the lower left, another source for errors in this approach.

To address this problem we first introduced a new mode to polygon drawing. While drawing in this new snap mode, a click does not create a control point exactly where the cursor is at but rather on the closest edge of an preexisting polygon. This way we expected to be able to create seamless boundaries between polygons. After a first trial run with this new snap mode we realized that this was not the case. The snapped control points can lie on the edge of the preexisting polygon between control points. This results in gaps between the two polygons. Figure 3.1 shows an exaggerated example of this problem.

We could have changed the snap mode to only allow polygons to share exact control points. While addressing one problem this would introduce another. Since boundaries of polygons do not necessarily meet where both have control points. Further we have to acknowledge that the labeling time is unnecessarily increased using this processes. Each

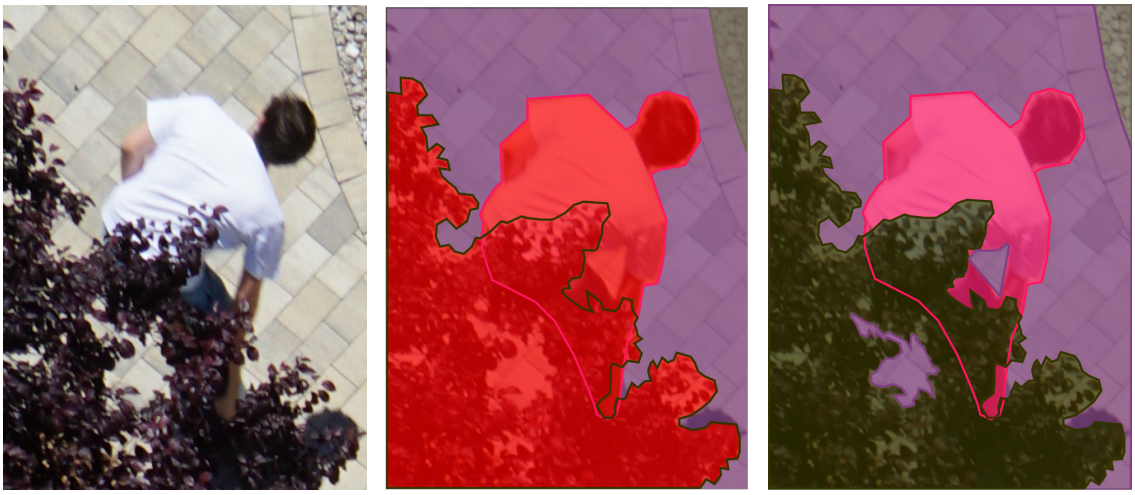
boundary needs to be drawn twice when two polygons share the same occlusion status. A problem occurs often, namely whenever more than two objects occlude each other. Finally the polygons loose expressiveness since the polygon boundaries cannot follow the actual shape of the occluded object. Thus we abandoned this approach in favor of a better suited one.

We took inspiration from conventional image editing software and AIDA and introduced layers. To do so we modified the already built in part-whole construct of the tool. The new annotation process starts by tracing the outline of each object with a polygon. Then the polygon is given the appropriate label and assigned to a layer. Objects in front are assigned to a numerically numerically higher layer, objects behind to a lower layer. The final label image is then created by drawing the layers on top of each other in ascending order. Where polygons that are not assigned to any layer are considered to be hidden behind all layers. Since the number of layers is not restricted this process allows to create seamless annotations for arbitrary complex occlusions with a single polygon for every object. Figure 3.2 shows the improvement in results when layers are used during labeling.

New layers can be created using a newly introduced button at the top of the object list called "Create new layer". In the annotation xml each layer stores its position represented by the layer number. Further we modified the display of the object list to show the layer number next to the layer and always order the layers with their parts at the top and all other objects below. The layer architecture reduced the labeling time significantly since object boundaries now only need to be drawn once. To further ease the labeling process we also added the option to insert a new layer below an existing one. Finally we added a button to show the combined labeling to the user of the tool. When the "Show combined labeling" button is clicked then the current label image is calculated and superimposed on the color image. The combined labeling is calculated on the server by a cgi script written in python using the numpy [1] package. This has the downside of increasing computational load on the server but ensures that all users see the same result and reduces the resource needs on the workers computers. The appearance of the modified tool can be seen in Figures 3.3a and 3.3b.

Although we did not directly use this feature we would like to note that these modifications allow to create 2 1/2 dimensional sketches of the scene from the annotations. Therefore it can be used to create datasets for scene understanding.

[1] `http://www.numpy.org/`



(a) Unlabeled **(b)** Without layers **(c)** With layers

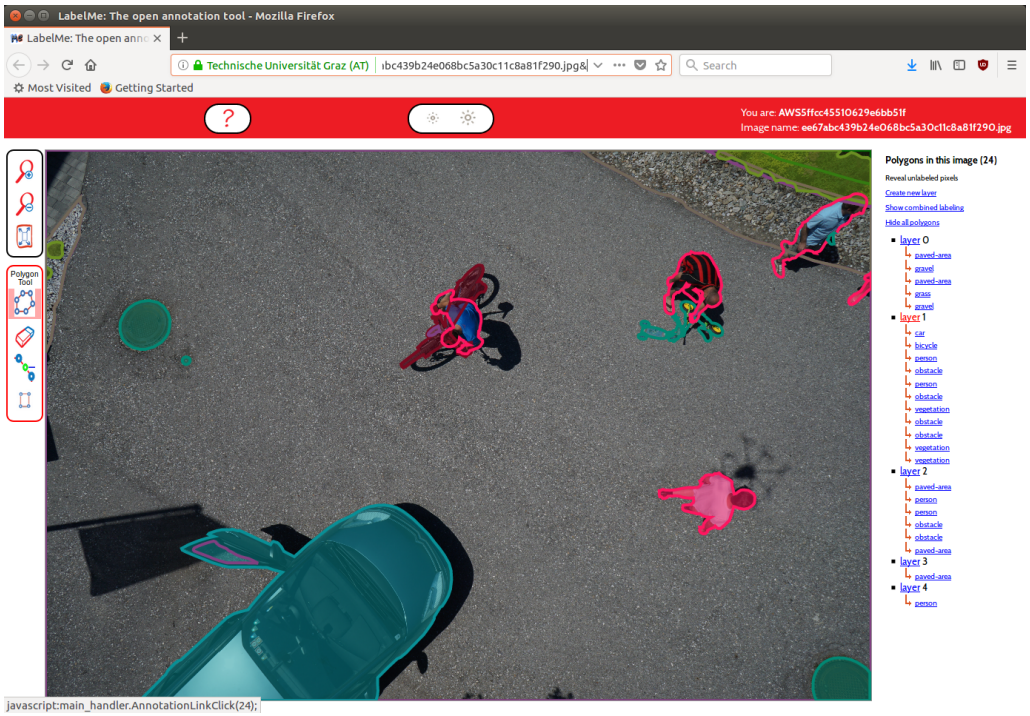
Figure 3.2: Annotation of an example image using layers. To the left the blank RGB image can be seen. The middle figure shows superimposed annotations with correct object outlines but no layer assignment. Red overlay marks conflicting areas. To the right the annotations are assigned to three different layers. The big paved area to layer 0, the person to layer 1, the tree to layer 2 and the small paved area patch inside the tree to layer 3.

3.3 Tool Integration

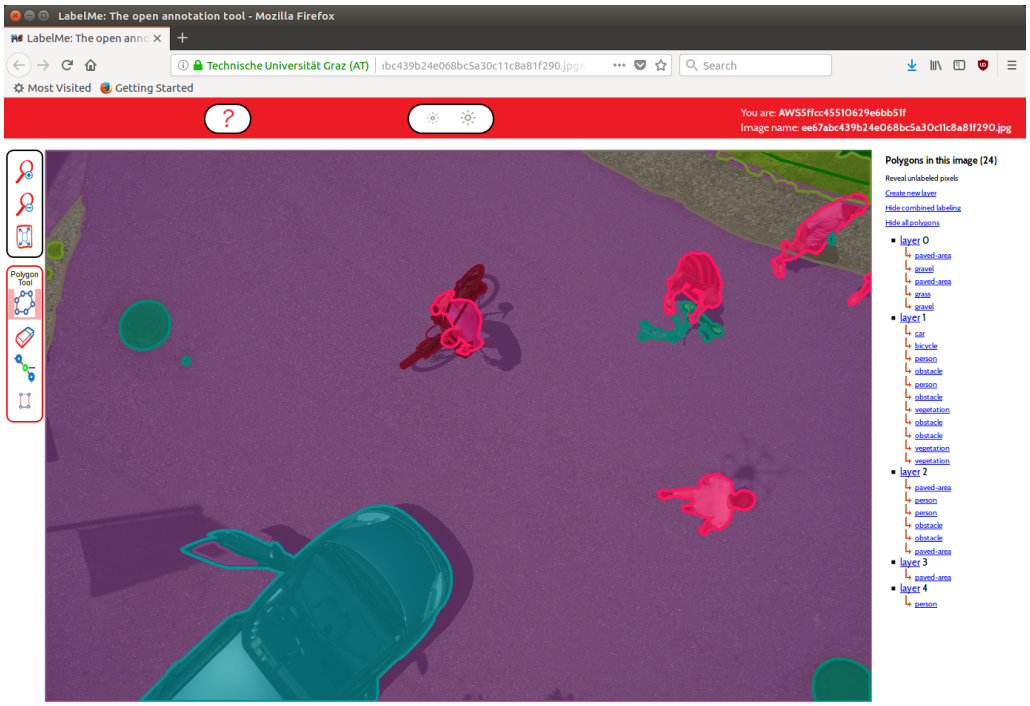
As mentioned in the tool survey 2.3.1.4 there are preexisting tools [2] to create and retrieve Human Intelligence Tasks (HITs) from Amazon Mechanical Turk. Sadly these tools have not been maintained since 2013 and the Amazon Mechanical Turk Command Line Interface (CLI) that is used internally is no longer compatible with it. Since we had to create at least 600 *HITs* - one per image - we refrained from a manual creation on the requester homepage. Consequently we had to develop our own scripts to interface with the crowdsourcing platform.

We chose to develop python scripts using the boto3 Application Programming Interface (API)[3]. We wrote scripts to create new qualification types and publish *HITs*. Further we wrote a simple *CLI* to aid the maintenance, evaluation and storage of qualification requests, assignment submissions and related messages. The latter decision was partially forced on us since the *HITs* created via the *API* may only be maintained through it. In order to stay compatible with other crowdsourcing platforms we decided to redirect workers to our own server where the annotation tool is hosted. To ensure that we can uniquely connect each worker to their work we automatically assign every connected browser a unique user identification string and store it in a cookie. Further we added a small Common Gateway Interface (CGI) script to the tool. That python script creates an empty

[2] `https://github.com/CSAILVision/LabelMeMechanicalTurk`
[3] `https://boto3.readthedocs.io/en/latest/`



(a) Default mode: In the middle we can see the image with the polygons drawn on top. The polygons edge color matches the chosen label. To the right a list of layer objects and their sub elements can be seen. All polygons that belong to layer 1 are shaded because the cursor is hovering over it in the object list.



(b) Overlay mode: Note that some areas are covered by multiple polygons but are assigned the correct label despite overlap because of the layer assignments.

Figure 3.3: The modified LabelMe Annotation Tool in default and overlay mode as seen in Mozilla Firefox 59.0.2 (64-bit) on Ubuntu 16.04.4 LTS.

annotation every time the page corresponding to the Human Intelligence Task (HIT) is visited. This structure allows both forms of worker collaboration on the same image.

3.4 Compiling the Dataset

3.4.1 Image Acquisition and Label Definitions

The images used in this dataset were kindly made available by the areal vision team from the Institute of Computer Graphics and Vision of the Graz University of Technology. A total of 3876 images from nadir perspective were taken with the institutes octocopter at two locations. The first shooting location is a house in the country side, where 1330 images were taken in the season of fall. The second location is a local suburban area composed of showcase houses and gardens, where 2546 images were taken in summer.

Figure 3.4: The mapping of labels in this dataset and the colors used to visualize them. Conflicting and unlabeled are not explicit labels. Conflicting was only used to visualize errors during the labeling process. Mainly caused by overlapping polygons that share the same layer. While training the networks conflicting was treated as unlabeled area.

The images were selected in three stages. In the first stage we removed any pictures that had too pronounced motion blur artifacts. In a second stage we only kept a selected subset of images for each sub location. In last step 600 of the remaining 1022 were selected in a uniformly random subset. The following 22 labels were chosen after the first selection step.

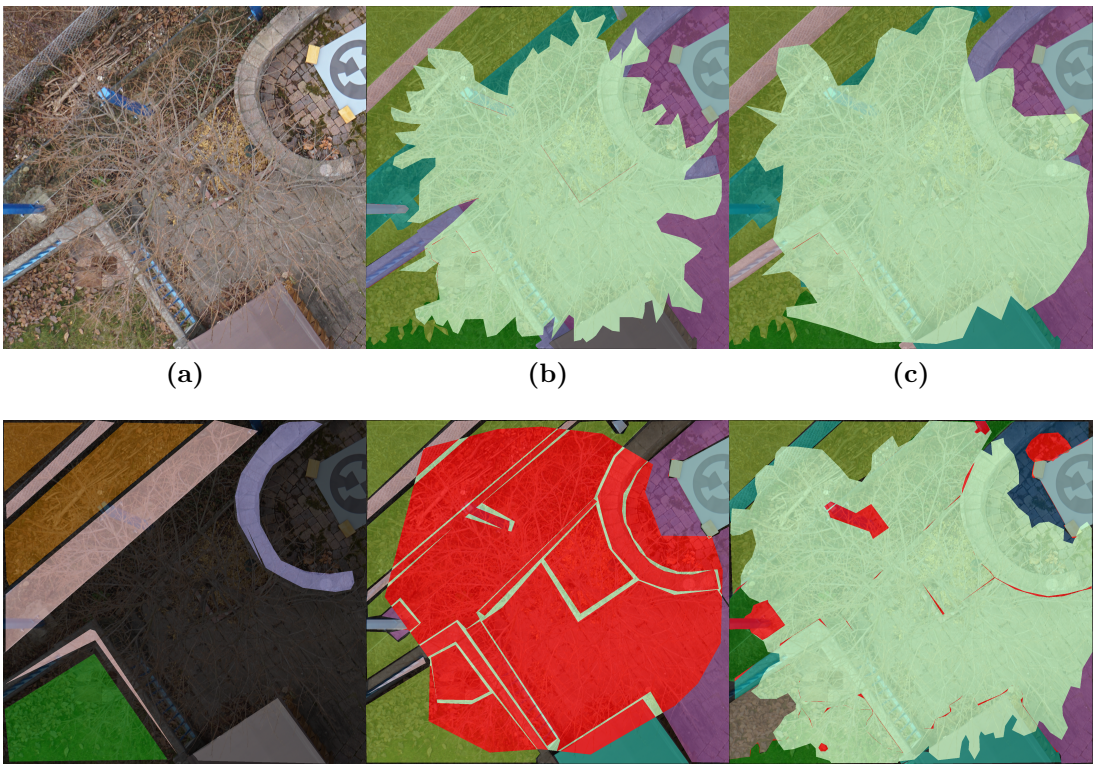
- **tree**: Living trees and big shrubs with a crown of leaves or needles.
- **bald-tree**: Trees or big shrubs missing leaves and needles. The outline does not need to follow every branch and twig. Only a rough outline needs to be given.

- **grass**: Lawn and small greens.
- **vegetation**: All plants dead or alive except trees and grass. e.g. hedges, scrubs, large weeds, small bushes as well as piles of dead leaves or branches
- **dirt**: Visible ground surfaces that are not rock, gravel, grass, covered by dead leaves or vegetation nor paved over.
- **gravel**: Ground surface that is covered by small stones. e.g. some walkways, some plant beds
- **rock**: Large stones e.g. decorative boulders
- **water**: Surface that is covered by water. e.g. ponds, lakes, creeks, pools if filled
- **paved-area**: Ground that is covered by hard man made surfaces like asphalt, flat stone, concrete, wooden floors. e.g. most walkways, drive ways, streets
- **pool**: Swimming pools even if empty. The contained water if any is not considered part of this class.
- **person**: People that can be seen in the image. Including clothing but without tools, toys, bags, backpacks, bikes or scooters.
- **dog**: Only the animals. Toys or drawings are considered obstacles.
- **car**: Only full sized cars. Toys or drawings are considered obstacles.
- **bicycle**: Two wheeled vehicles to sit on. The rider is not considered part of this class. Scooters and other bike like vehicles are considered obstacles.
- **roof**: The top covering of buildings if not filled with gravel, dirt or a paved area.
- **wall**: The vertical parts of structures made of concrete, bricks or wood.
- **fence**: All forms of fences, wooden, wire, etc. Except glass panes which are considered obstacles.
- **fence-pole**: Clearly visible fence poles (e.g. wire fence).
- **window**: Windows in buildings. Car windows are considered part of the car.
- **door**: Entrances and exits to buildings. Car doors are considered part of the car. Fence doors are considered part of the fence.
- **ar-marker**: Printed images used for augmented reality applications.
- **obstacle**: Anything that is not part of the above labels should be marked as obstacle. e.g. toys, umbrellas, chairs, tables, cords, glass roofs, acrylic roofs, ...

The label bald-tree was intentionally chosen to contain more than the exact pixels covered by the twigs and branches for two reasons. First the drone should keep greater distance from such objects making it difficult to detect the exact appearance of the finer structures without optical zoom. Secondly it is vastly faster to label the rough area containing the bald-tree than labeling every branch and twig.

3.4.2 Custom Qualification Test

To overcome the limitations of the predefined qualification metrics we decided to create our own qualification test. Figure 3.5a shows the image that should be annotated for the qualification test. The qualification tests were carried out using the same tool as the actual assignments and had the same instructions. This has the benefit that we can also evaluate if the worker understands how our annotation tool can and should be used.



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Figure 3.5: 3.5a shows the qualification image without any overlay. It contains 9 out of the 22 defined labels. Special care was taken to include the difficult labels bald-tree, vegetation, armarker and fence, as well as glass panes that should be marked as obstacle. 3.5b and 3.5c show segmentations for which a qualification was granted. 3.5d, 3.5e and 3.5f show submissions for which the qualification request was rejected.

At first we intended to evaluate the requests automatically by calculating a label agree-

ment to a self annotated reference. After receiving a few qualification requests we realized that the responses of the automated evaluation could not be used to efficiently educate workers about the desired changes. Therefore we decided to evaluate the qualification requests in a semi-automated manner. The algorithmic comparison was still performed but man made notes were added to the response. The possibility to educate workers was deemed very important because a low price per assignment, a new tool and the lack of other qualifications lead to few skilled and many unskilled workers applying.

Of the over 400 received qualification requests the overwhelming majority were rejected. Only 30 lead to a granted qualification. It should be noted the number of rejected requests is different from the number of workers who attempted the test. Each worker could send multiple requests until the qualification was granted. Most rejected requests had large unlabeled parts suggesting that the workers submitting them either lost interest while doing them or did not understand the tool usage. The next most frequent rejection reason was the lack of or incorrect layer usage. We stopped processing new qualifications once we had a stable workforce since new educating workers seemed to have diminishing returns on the time to finish the dataset. Figure 3.6 shows the number of qualified workers over time.

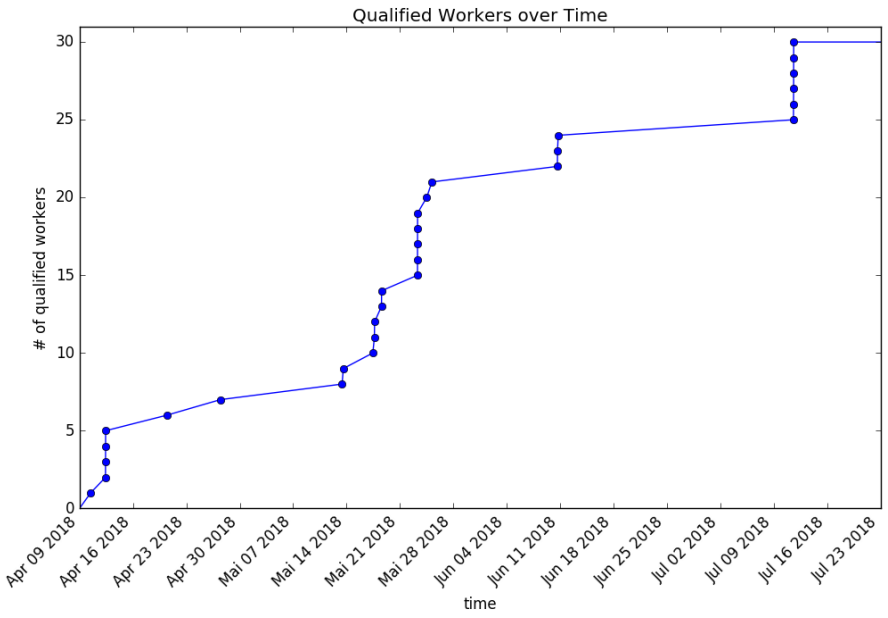


Figure 3.6: The development of workers who had the needed qualification over time. Each dot represents a granted qualification. The time of the grants is accurate to the date.

3.4.3 Processing *HITs*

Once the mandatory qualification test was put in place we started to publish *HITs* on the platform.

As the aim of our project was to create high quality full semantic image labels at a low cost we decided to only use annotations by one worker per image. The workflow for a new worker was as follows:

1. Find one of our *HITs*.
2. Click "Qualify" next to the *HIT* and be redirected to the qualification test page on Amazon Mechanical Turk.
3. Read the short instructions there and open the link to a more detailed instruction page hosted on our server in a new browser tab.
4. Click on "Start qualification" to be redirected to our tool where a new uniquely named copy of the qualification image is shown without any annotations.
5. Perform the segmentation by drawing polygons, assigning labels to them, creating layers and building a occlusion hierarchy with them.
6. Copy the unique image name and the automatically assigned user id visible in the top right of our page to the corresponding answer fields on the qualification test page on Amazon Mechanical Turk.
7. Submit the filled in qualification test and wait until the qualification request was processed by us.
8. If the qualification request was rejected we would include the link to the tool page that would allow to refine the submission. Thus the workers had two options. Go back to step 1 to start on a blank image or go back to step 5 to refine the submission. If the qualification was granted then work on a real *HIT* could be started as described in the next step.
9. Find an open *HIT* and read the short instructions on the Amazon Mechanical Turk page.
10. Open the link to the detailed instructions page hosted on our server and read them.
11. Click on "Start labeling the image for this HIT" to be redirected to the tool with a new uniquely named copy of the image for this *HIT* is shown without any annotations.
12. Perform the segmentation by drawing polygons, assigning labels to them, creating layers and building a occlusion hierarchy with them.
13. Copy the unique image name and the automatically assigned user id visible in the top right of our page to the corresponding answer fields on the *HIT* page on Amazon Mechanical Turk.

14. Submit the *HIT* answer and wait until the assignment submission is processed by us.

15. If the assignment submission did not live up to the instructions we would message the worker and include the link to the tool page that would allow to refine the submission. Thus the workers needed to go back to step 12 and message us afterwards.

16. If the assignment submission was approved the money would be transferred and work on another HIT could be started leading back to step 9.

We gauged the time to perform a full image segmentation by labeling a few images by hand ourselves. For us it took between 45 and 90 minutes to create an acceptable segmentation depending on the image content although we intentionally chose difficult images. We performed a few experiments with different rewards per assignment. For the first test we posted a few *HITs* with 1 United States Dollar (USD) reward per assignment. We restricted some of them to Indian workers and other to workers from the United States of America. After over a week we had a few people attempt the qualification, none of which we could grant due the low quality of the annotations. Thus we did not receive any useful *HIT* submissions. Therefore we increased the reward significantly to 12 *USD* and posted three more *HITs*. The higher reward resulted in many qualification requests. Although many still were of low quality, we also received some high quality submissions for which we granted a qualification. The performed annotations also proved very useful. Sadly we realized that this was too expensive for the budged we aimed for.

Thus we reduced the reward to 3 *USD* which is according to [12] slightly below the mean hourly wage of Amazon Mechanical Turk workers. To test our assumption that this reward was sufficient to motivate workers to perform segmentations for us we again published a few *HITs*. In this last test run we still received enough qualification requests and assignment submissions. Although the response time and quality of the average submission dropped significantly. Thus all further *HITs* were published using 3 *USD* as reward per assignment and one assignment per *HIT*. With this setup the total price per full image segmentation transferred to Amazon came to 3.6 *USD* since a Amazon collects a fee of 20% on each worker reward.

We decided to publish the *HITs* in multiple batches. This allowed us to respond to all submissions within a week with one work days effort per week. The batch size was increased multiple times until we used up all the allotted processing time per week. The maximum batch size was 100 *HITs* being published at once. Figure 3.7 shows the number of accepted assignments by worker and date. The spikes in the plot are a result of waiting until one batch is mostly processed before posting the next.

The time to manually process an assignment submission varied greatly depending on the quality of the submission. Good submissions could mostly be accepted without manual changes or change request. Submissions of very low quality were handled quickly as well since they either resulted in a outright rejection or a standard response. The most work

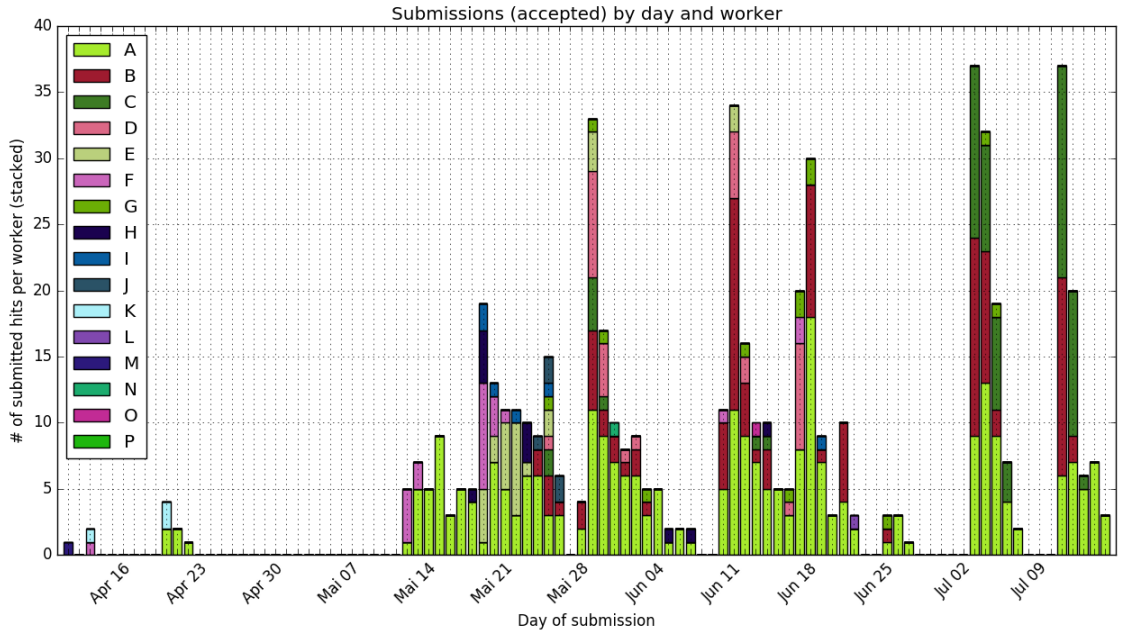


Figure 3.7: The accepted assignments by date. Each color corresponds to an individual worker account. The worker ids were anonymized.

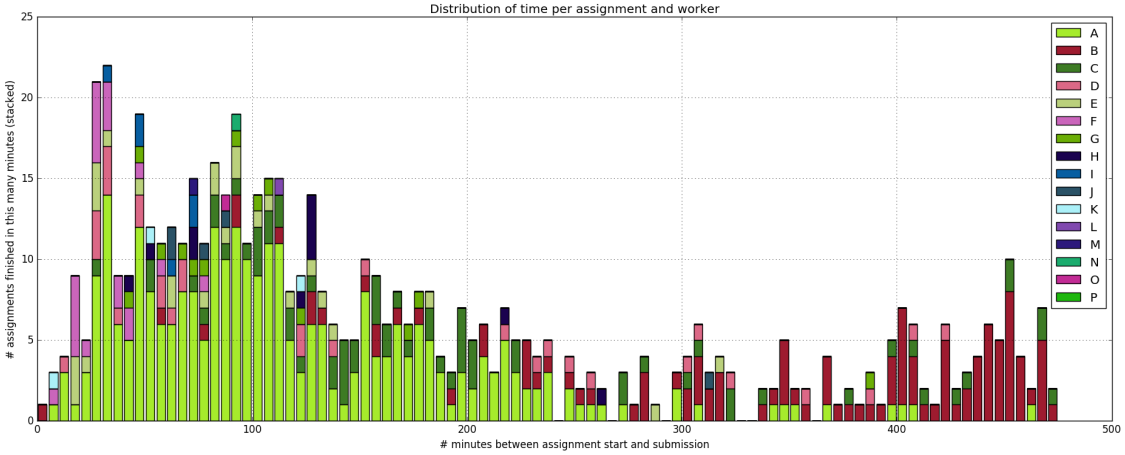


Figure 3.8: Distribution of time between assignment start and submission. Each color represents a different worker account. The distribution has a mean of 170 and a median of 123 minutes per assignment. It should be noted that this time includes breaks if any were taken.

was caused by submissions of medium quality. A rejection right away would discourage the worker. Even with a note explaining what should be done differently, an acceptance right away would result in more submissions of the same-sub standard quality. The best course of action for medium quality submissions was to message the worker what should be improved and why. While time consuming on our part this method improved the current and all further submissions by that worker. Additionally it allowed building a personal relationship with some of the workers leading to improvement suggestions to the tool and the instructions. We did not measure the time to process an assignment for all assignments. For those where we did measure it the mean comes to about 6 minutes per evaluation. Although not ideal it still reduces the researchers time effort per full image segmentation by an order of magnitude. Note that the development time for the required tools is not included in this calculation.

For all *HITs* posted before the 18$^{\text{th}}$ of May we used four hours as the maximum time allowed between assignment start and submission. We received requests by multiple workers to increase this maximum. Since we did not see any obvious downside the maximum time was increased to eight hours. The quality of responses did not suffer in a noticeable way. The time taken by the workers to complete the assignments can be seen in Figure 3.8.

We could reproduce the findings of [31] that a few workers were responsible for most submissions. Figure 3.9 illustrates this relation for our data. Once we had one segmentation for each image in the training set we performed a final manual revision. In this last polishing phase we made sure that the same objects were labeled in the same way in all images as best we could.

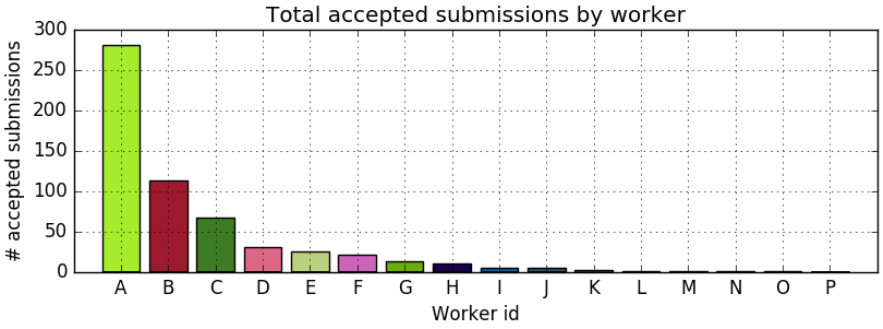


Figure 3.9: The total number of accepted *HIT* assignments by worker. The worker ids were anonymized.

4 Experimental Dataset Evaluation

Contents

4.1 Experimental Setup

To prove that the dataset introduced with this work can be used to benchmark algorithms we trained state of the art semantic segmentation algorithms on it. We evaluate our dataset using the FC-DenseNet103 [13] and the DeepLabV3+ [7]. Since we desire others to be able to replicate our work we only used images and annotations from the set which we will make publicly available online.[1] We split the 400 public images into three sets. A training set consisting of 240 images, a validation set of 10 images and a test set composed of the remaining 150 images. Table 4.1 lists the exact images that make up the training, validation and test sets.

During training only images and ground truth from the training set are used to calculate gradients and update the network. Every ten epochs the progress is validated on the validation set. In the training phase the algorithm is only exposed to the training and validation set. This ensures that the images and ground truth of the test set are unknown to the network until the test phase.

Each algorithm was trained on a single Nvidia™ Tesla K80 Graphics Processing Unit (GPU) with 11441 MebiByte (MiB) of graphics memory. As training framework we used the Semantic-Segmentation-Suite [2], but also used our own code for calculating the metrics for validation and test.

[1] `dronedataset.icg.tugraz.at`
[2] `https://github.com/GeorgeSeif/Semantic-Segmentation-Suite`

training images	000 001 002 003 004 005 006 008 011 013 014 015 016 018 019 021 022 023 026 028 031 035 038 040 041 042 043 044 045 047 049 051 052 053 055 056 057 058 059 060 062 063 065 068 070 071 073 074 075 077 078 079 080 081 083 086 088 089 092 095 098 099 100 101 102 103 104 106 107 109 110 111 112 113 116 117 118 119 120 121 122 123 124 126 128 130 133 134 135 136 137 138 139 140 141 145 146 147 148 149 150 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 170 171 172 173 174 175 176 177 178 179 180 181 182 185 186 188 190 192 193 194 195 198 199 200 202 204 206 207 208 209 213 214 215 216 217 219 220 221 222 223 225 226 228 229 230 232 233 234 235 236 237 238 239 240 243 244 246 248 250 251 252 255 257 258 259 260 261 262 263 265 266 271 272 273 275 276 277 281 283 287 288 289 290 292 294 295 296 298 299 301 302 303 304 305 306 309 310 311 312 313 314 316 318 320 321 322 323 324 325 326 329 330 331 332 334 335 338 339 341 342 344 345 346 347
validation images	349 366 386 410 424 425 428 435 443 452
test images	348 351 355 356 361 363 367 372 373 375 376 378 380 381 382 383 385 388 389 390 391 393 397 398 403 406 408 409 411 412 413 414 416 419 420 421 423 426 427 429 430 431 433 434 437 438 439 440 442 444 445 446 447 451 454 455 457 458 460 461 462 463 464 465 467 470 472 473 474 475 476 478 479 480 484 485 488 489 491 493 494 497 498 499 500 501 502 507 508 509 510 512 513 514 515 517 518 521 524 525 526 529 530 531 532 533 535 536 537 538 540 543 544 545 549 551 554 556 558 559 560 561 563 564 565 566 567 568 569 570 572 573 574 576 579 580 582 583 584 585 586 587 588 590 591 592 593 594 596 598

Table 4.1: Images used for training, validation and test of our semantic drone dataset

4.1.1 Dealing with Memory Constraints

At the original resolution of 6000x4000 pixels not all network layers fit into the *GPU* memory at once, since the networks memory consumption scales with the input size. Therefore even for a batch size of one either only parts of the network may be loaded or the input resolution has to be lowered. Training the network in parts is possible but increases training time significantly.

The popular choice [5] to address this problem is to downsample the input and upsample the output. Downsampling has the benefit of preserving the all of the context. The downsides of this approach include that higher frequency details are hidden from the network and that are predictions worse for fine label boundaries since the output boundaries are fixed to the lower resolution.

The greatest benefit of higher resolution images is the preservation of high frequency details of the scene. In order not to loose these details we decided to refrain from downsampling the images. To reduce the input size we use square cut-outs with a fixed size

l. For each epoch the network is presented with a cut-out of each training image. The position of the cut-out is chosen in a uniformly random manner for every image and every epoch. With each epoch the likelihood that the network has seen all parts of the training images increases. Conversely the likelihood of exposing the network to unseen material decreases with every epoch.

During validation and test we perform full image predictions at the original resolution. We achieve this by partitioning the image into generally non overlapping squares with side length l and performing a prediction for each square. The full image prediction is then stitched from the predictions of the parts. When the image width or height is not a multiple of the square side length l we allow an overlap at the lower and right edges of the full sized image. In case of overlapping squares we still only use the labels of one square namely the one closest to lower right image corner.

4.1.2 Metrics

The main metric we use to evaluate the performance of the algorithms is the Intersection over Union (IoU). The IoU is calculated in the following way. For each label find the intersection or true positives by determining the number of pixels in prediction that match the ground truth. Then determine the union for each label by finding all pixels that have the current label either in the prediction or the ground truth. The IoU for a label is then given by dividing the intersection by the union. The IoU is a standard metric used to compare semantic segmentations in the field [7, 13, 16, 22, 29].

Other well known metrics are

1. $\text{accuracy} = \frac{\#\text{ correct pixels}}{\#\text{ all pixels}}$
2. $\text{precision} = \frac{\#\text{ true positives}}{\#\text{ true positives} + \#\text{ false positives}}$
3. $\text{recall} = \frac{\#\text{ true positives}}{\#\text{ true positives} + \#\text{false negatives}}$
4. $\text{F1-score} = 2\frac{\text{precision}\cdot\text{recall}}{\text{precision} + \text{recall}}$

which we report for the results on test set as well. All the described metrics are so far only defined for a single image. To expand them to multiple images we calculate each metric on each image and then form the uniform and a weighted mean over multiple images. We report the mean of the metrics weighted by the label prevalence to remove the bias from differing amount of input data by label from the model evaluation. This shifts the focus from the sparsity robustness to the label dependent performance of the architecture. If the ground truth of an image does not contain a specific label it is ignored for the calculation of that labels metrics. This is necessary to get results that represent the achieved performance accurately. The human ability on average reaches an accuracy of 82% [38], a precision of above 0.95 and a recall of up to 0.85 [15].

4.2 Training Protocol

4.2.1 FC-DenseNet

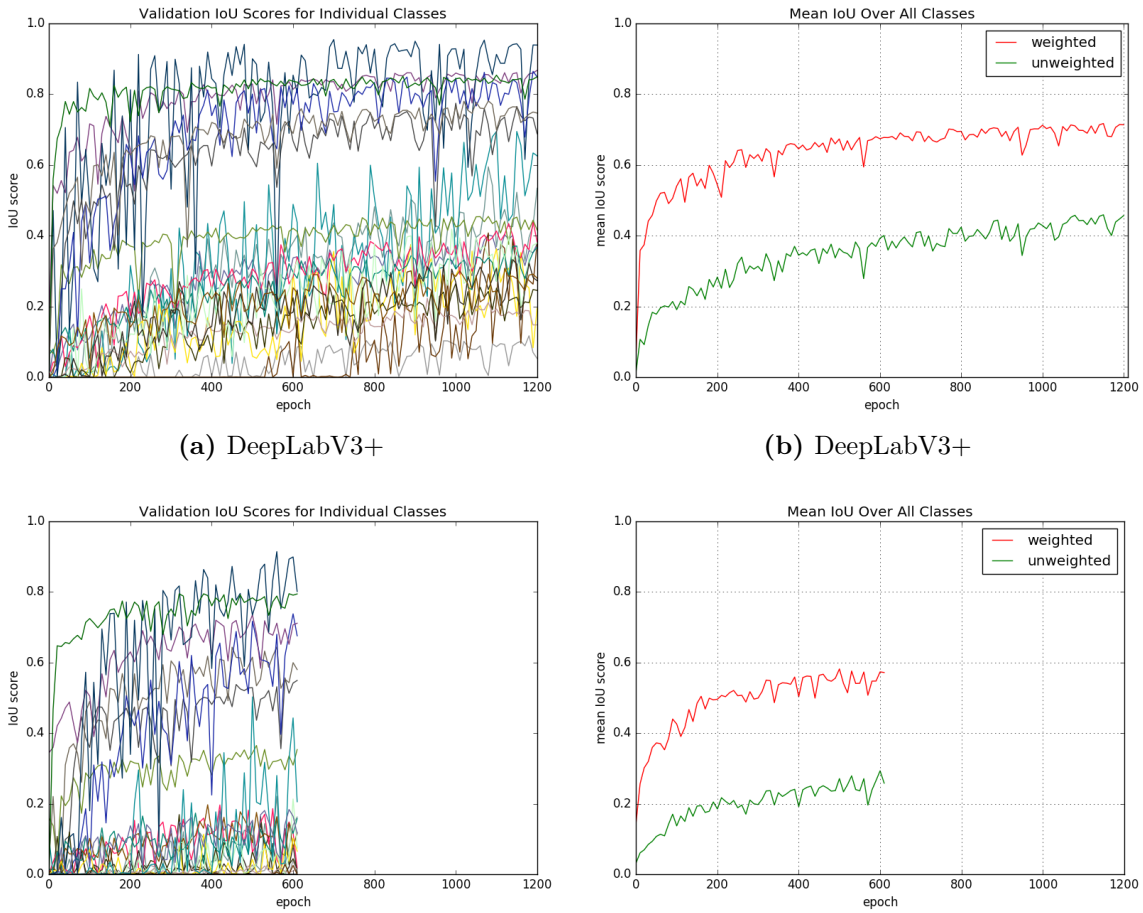
We trained the network using the Adam stochastic optimizer introduced in [14] which implements a stochastic gradient decent with decaying first and second order momentum. As parameters we chose an initial learning rate of 10^{-3}, first decay moment $\beta_1 = 0.9$, second decay moment $\beta_2 = 0.999$ and a numerical stability factor $\hat{\epsilon} = 10^{-5}$. The network was trained for 620 epochs. The network weights were initialized using the modified uniform sampling process introduced with [11]. Figures 4.1c and 4.1d show the progress of the *IoU* during the training phase. The training took about 160 hours in total or 16 minutes per epoch including calculation of the validation metrics.

It is evident that some labels are easier to learn than others. The classes grass, pool, paved-area, roof and water reach good performance with a few epochs while others as person, obstacle, window, dirt, bald-tree and obstacle remain difficult even after 620 epochs. Although the classes rocks, fence-pole, door and dog remain practically impossible to detect for the network the overall progress proves that our data can be used to train deep Convolutional Neural Network (CNN) architectures from scratch.

4.2.2 DeepLabV3+

For this network we employed transfer learning using the pre-trained ResNet-152-V2 as a backbone. The ResNet-152-V2 was pre-trained on ImageNet [27] at Google and the weights were kindly made available online [3] [4]. Weights outside of the ResNet-152-V2 were initialized with the Glorot initializer [11]. The Adam optimizer [14] had an initial learning rate of 10^{-2}, first decay moment $\beta_1 = 0.9$, second decay moment $\beta_2 = 0.999$ and a numerical stability factor $\hat{\epsilon} = 1$. The network was trained for 1200 epochs. The training time including the validation calculations amounts to about 161.5 hours in total or 8 minutes per epoch. We decided to train this network for more epochs than FC-DenseNet103 to evaluate if longer training time brings significant performance improvements. Figures 4.1a and 4.1b show the change in *IoU* during training. While an improvement is still visible from epoch 600 to 1200 the gains on most labels are diminishing, especially compared to the first 300 epochs. Only the label car receives a significant improvement in the last 300 epochs. Although the relative gains in performance to previous epochs reduce over time in absolute terms some progress is still made. This suggests that only a very large number of training epochs like 10^3 or even $2 \cdot 10^3$ as done in [5] would make a significant difference in performance. As with the FC-DenseNet103 the DeepLabV3+ also learns some labels easier than others. Again the labels grass, pool, paved-area, roof and water reach good performance quickly while others such as person, window, rocks, dirt and wall take longer and stay at a lower performance.

[3] Overview: `https://github.com/tensorflow/models/tree/master/research/slim`
[4] Direct download: `http://download.tensorflow.org/models/resnet_v2_152_2017_04_14.tar.gz`



(a) DeepLabV3+ **(b)** DeepLabV3+

(c) FC-DenseNet **(d)** FC-DenseNet

Figure 4.1: Evolution of the *IoU* during training with an input size of 512x512. Subfigures 4.1a and 4.1c show the for the individual labels during training. Subfigures 4.1b and 4.1d show the difference between unweighted mean and a weighting of each class by its prevalence in the whole set of 400 images. The line colors match the label colors in Figure 3.4. The high fluctuations are due to the low number of validation images used.

4.3 Test Set Results

4.3.1 Analysis with Performance Measures

After training we use the networks to perform predictions on the unseen 150 test set images. The full resolution predictions are created with the procedure described in Section 4.1.1. Using the predictions and the ground truth created by the methods described in Chapter 3 we calculate the metrics from Section 4.1.2. Table 4.2 lists the metrics for both networks for each label class as well as the weighted and unweighted mean. The second column of Table 4.2 shows the prevalence of each label in the public part of the dataset.

	prev.	DeepLabV3+					FC-DenseNet103				
label	[%]	acc	prec	rec	F1	IoU	acc	prec	rec	F1	IoU
mean	4.348	0.50	0.38	0.40	0.38	0.42	0.32	0.28	0.29	0.25	0.24
weighted mean		0.82	0.68	0.66	0.66	0.72	0.70	0.57	0.58	0.55	0.58
paved-area	38.000	0.94	0.94	0.90	0.92	0.86	0.88	0.87	0.83	0.85	0.76
grass	20.000	0.93	0.84	0.81	0.82	0.85	0.78	0.66	0.76	0.69	0.72
roof	7.400	0.79	0.24	0.24	0.23	0.70	0.72	0.16	0.13	0.13	0.54
gravel	7.300	0.82	0.57	0.57	0.57	0.70	0.84	0.58	0.42	0.46	0.54
vegetation	7.100	0.66	0.24	0.26	0.24	0.52	0.72	0.12	0.16	0.11	0.43
obstacle	3.500	0.52	0.18	0.29	0.20	0.31	0.31	0.07	0.10	0.07	0.16
dirt	3.200	0.60	0.55	0.54	0.51	0.43	0.15	0.14	0.49	0.20	0.14
wall	2.700	0.53	0.39	0.37	0.37	0.35	0.11	0.36	0.27	0.29	0.09
water	2.200	0.93	0.77	0.73	0.75	0.81	0.62	0.62	0.64	0.60	0.56
tree	2.100	0.45	0.21	0.27	0.21	0.35	0.10	0.10	0.10	0.09	0.10
bald-tree	1.300	0.49	0.13	0.21	0.12	0.34	0.17	0.08	0.04	0.04	0.13
person	1.100	0.49	0.33	0.40	0.34	0.39	0.03	0.26	0.25	0.23	0.02
fence	0.960	0.18	0.40	0.43	0.39	0.15	0.13	0.28	0.29	0.25	0.05
car	0.790	0.66	0.29	0.33	0.28	0.62	0.32	0.17	0.16	0.14	0.29
rocks	0.720	0.30	0.33	0.31	0.31	0.19	0.00	0.27	0.26	0.24	0.00
pool	0.640	0.95	0.37	0.39	0.38	0.91	0.87	0.26	0.35	0.29	0.64
window	0.560	0.24	0.60	0.62	0.60	0.21	0.24	0.52	0.46	0.46	0.10
ar-marker	0.230	0.62	0.16	0.22	0.16	0.52	0.26	0.05	0.09	0.05	0.21
bicycle	0.220	0.32	0.33	0.36	0.32	0.23	0.02	0.15	0.13	0.11	0.02
fence-pole	0.053	0.03	0.29	0.30	0.27	0.03	0.00	0.17	0.19	0.16	0.00
door	0.031	0.00	0.41	0.39	0.39	0.00	0.00	0.33	0.30	0.27	0.00
dog	0.014	0.14	0.25	0.33	0.27	0.13	0.00	0.17	0.17	0.13	0.00
unlabeled	0.220										

Table 4.2: The metrics shown here are calculated for predictions of the test set performed by DeepLabV3+ and FC-DenseNet103 after training on our 240 image strong training set for 1200 and 620 epochs respectively. The values are rounded to the second decimal point. The column labeled "prev." marks the percent of pixels in the dataset with that label.

Unsurprisingly for statistical models the highly prevalent labels which mostly cover large connected regions such as paved-area, grass and roof are easier to recognize for both networks. Similarly labels such as dog, door, window, rocks and fence-pole which are underrepresented in the dataset are more difficult to predict correctly for the learned models. A stark exception to this rule is exhibited by the label pool which only makes up about half a percent of all input pixels but both networks achieve a very high *IoU* for it. This is very likely explained by the fact that only one physical pool instance is part of the training set and it has a very unique shade of blue. The networks likely learn to associate that single color with the label.

Predictions for some other labels also exhibit good results despite low prevalence. The results for the label ar-marker can probably again be explained with its quite unique color. Further the distinct shapes of the ar-markers are hand crafted to be easily recognized. Regions belonging to cars are also recognized by both networks quite good. At least for FC-DenseNet103 this is unexpected since no pre-training is performed and multiple car instances are contained in the data. Dogs are hard to detect for both networks but the pre-training of DeepLabV3+ shows a significant benefit. While water is not extremely rare in the dataset it only makes up 2.2% of all pixels and is still recognized with a high accuracy as well as a good *IoU* score. The half frozen water in the pool is probably easily detected by its neighborhood to the pool. However this cannot alone be the cause for the good performance since multiple other ponds and creeks are contained in the summer images which border rocks, gravel, walls and paved-area.

Conversely some labels remain difficult to predict for both networks although making up large parts of the data. Vegetation for example is significantly harder to determine for both networks than similarly prevalent labels like gravel or roof. Large intra class variability for vegetation is likely the cause for this difference. The dataset contains images taken during summer and fall. While roof and gravel look the same in both seasons the appearance of vegetation changes drastically. Even without the season related changes vegetation exhibits far more different textures and instances than roof or gravel could.

Overall DeepLabV3+ reaches better performance than FC-DenseNet103 for all classes and metrics after the same learning time. Examining the Figures 4.1a and 4.1c indicates that this ranking also holds when both networks are trained for the same number of epochs, suggesting a clear advantage of the pre-training and possibly the network architecture of DeepLabV3+.

4.3.2 Analysis by Visual Inspection

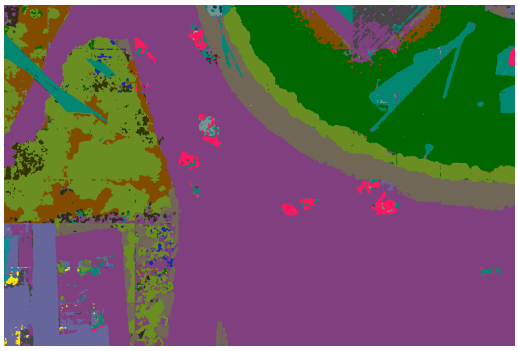
Figure 4.2 illustrates the performance of the networks on two example images from the test set. In the first row Figures 4.2a and 4.2b show the unlabeled images as taken by the camera. Figures 4.2c and 4.2d the corresponding ground truth can be seen. The remaining images show the predictions of both networks. For image 517 the performance of both networks is rather good. All bigger regions are classified correctly and segmented

(a) Image 517

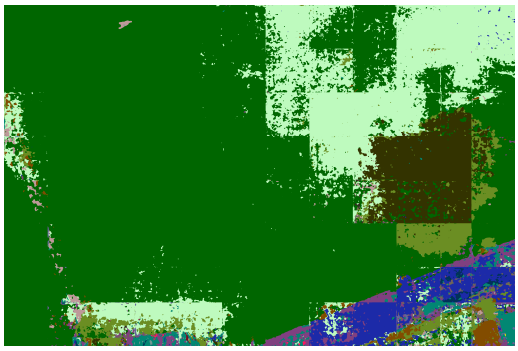
(b) Image 563

(c) gt for image 517

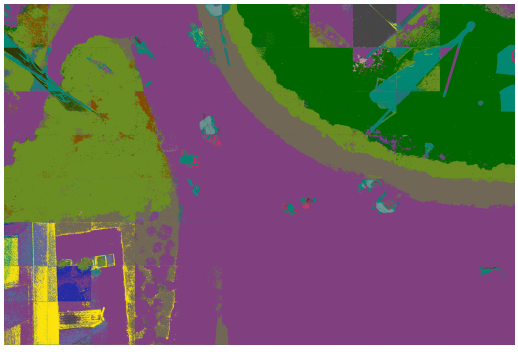
(d) gt for image 563



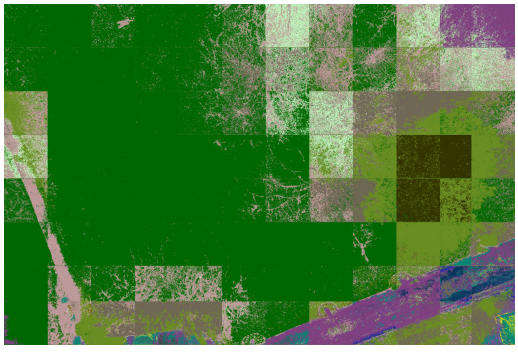
(e) DeepLabV3+ prediction 517



(f) DeepLabV3+ prediction 563



(g) FC-DenseNet103 prediction 517



(h) FC-DenseNet103 prediction 563

Figure 4.2: Examples for good and bad performance on test set images for both networks.

quite well. In contrast the predictions for image 563 are wrong for most labels except grass. This image is difficult for multiple reasons. Large parts are covered by the low prevalence and difficult label bald-tree, the paved-area which makes up the drive way appears quite inhomogen and seems to be partly made out of concrete and the image contains multiple labels with little context at the lower image border. The assembly of the full image prediction from smaller predictions can be discerned in Figures 4.2f and 4.2h. To some extent the effect can also be seen in the predictions in Figures 4.2e and 4.2g. Similar behavior is exhibited on many of the predictions. The effect highlights that deep architectures derive much of their power from the context surrounding the object. A comparatively small cut out also removes large parts of the context that is otherwise used to detect the labels. Human performance also has a strong dependency on context so it makes sense that algorithms such as deep *CNN*s which are modeled after the human brain have similar downsides.

Looking at Table 4.2 it appears that FC-DenseNet103 is almost completely unable to detect people correctly. By inspecting predictions as seen in Figure 4.2g however it becomes clear that the network learns to discern areas showing people from labels such as paved-area, grass and dirt. People are often confused with the class obstacle, a difficulty also seen less pronounced in predictions from DeepLabV3+. The novel viewpoint from above could be at least in part responsible for the poor performance of both networks.

The bad performance for people is likely also a result from noise in the data set regarding human confusion of the labels person and obstacle. Even for workers it is often not clear when an object carried by a person stops being part of the label class person. Despite being clearly part of the obstacle class backpacks, hats, helmets and held or very loosely worn clothing are often considered part of the class person by annotators as well. The only consistent decision seen in the man made annotations is that these objects are considered separate if no person is close to them. Especially for the heavily pre-trained DeepLabV3+ this results is surprising despite the noise because the pre-training should give a significant advantage as it does for the difficult classes dog and bicycle.

Another interesting observation is the way in which the predictions for DeepLabV3+ and FC-DenseNet103 differ when they are wrong. Wrong predictions of DenseNetV3+ are mostly convex blobs even when incorrect. Labels in general tend to be large connected regions. This is hinting that most of the prediction is based on context and the high level combination of many high frequency feature detections. In contrast wrong predictions of FC-DenseNet103 are fine grained high frequency dots and lines. Likely this means that small high frequency features cause the label selection without considering the larger context, a claim that is supported by the fact that bald-tree and fence are often confused for each other by FC-DenseNet103.

This correlates well with findings that deep architectures tend to learn small features first while later layers that make decisions based on larger context and field of view tend to be learned later in training.

5 Conclusion

Contents

5.1 Conclusion

We have shown that it is possible to create a useful semantic segmentation dataset with a small budged by distributing the annotation work among many workers on a crowd sourcing platform. As much as 3.6 United States Dollar (USD) and some preparation are sufficient to obtain a single dense semantic image labeling given that our tool is used on a crowd sourcing platform with qualified workers.

The available tool landscape was surveyed and the benefits and downsides of multiple tools were analyzed. The well known LabelMe Annotation Tool [28, 32] was reverse engineered and modified to enable web based dense semantic image segmentation. As a side effect we introduce a tool that can create annotations needed for 2 $^1/_2$ dimensional sketches.

We introduced a dataset that to the best of our knowledge is the first to include many dense semantic segmentations for low altitude drone nadir images. By doing so we help pave the way to fully autonomous Unmanned Areal Vehicle (UAV) landing in unknown suburban areas.

The possibility to train statistical models for semantic segmentation based on deep Convolutional Neural Network (CNN) architectures on the introduced dataset was proven by the results obtained from FC-DenseNet103 [13]. Further an overall and a class specific baseline for dense label prediction on our dataset were established by fine-tuning DeepLabV3+ [7] to our dataset in 1200 epochs of training.

Finally we showed that the our dataset contains a variety of new challenges for state

of the art models due to the unusual perspective and the combination of images from different seasons.

5.2 Future Work

For the creation of future datasets we suggest using multiple annotations per image especially when exact label borders are of interst. Even with great effort on the worker side the combination of two or more annotations is vastly superior to a single image segmentation. If the described process shall be scaled to thousands of images then it is necessary to automate the process of qualification request and assignment submission evaluation. A comparison of the created tool to the OpenSurfaces tool used to create MS COCO [1] [15] would be of interest.

For future work in the development of autonomous drone landing the analysis of additional algorithms on the introduced dataset will be necessary. The focus should be on architectures capable of fast predictions once trained, since *UAV* flight control imposes at least soft real time constraints.

[1] `http://cocodataset.org/`

A

List of Acronyms

API	Application Programming Interface
CGI	Common Gateway Interface
CLI	Command Line Interface
CNN	Convolutional Neural Network
GPU	Graphics Processing Unit
HIT	Human Intelligence Task
HITs	Human Intelligence Tasks
IoU	Intersection over Union
MiB	MebiByte
UAV	Unmanned Areal Vehicle
UAVs	Unmanned Areal Vehicles
USD	United States Dollar

Bibliography

[1] Bonetto, M., Korshunov, P., Ramponi, G., and Ebrahimi, T. (2015). Privacy in mini-drone based video surveillance. In *Automatic Face and Gesture Recognition (FG), 2015 11th IEEE International Conference and Workshops on*, volume 4, pages 1–6. IEEE. (page)

[2] Boykov, Y. and Kolmogorov, V. (2004). An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE transactions on pattern analysis and machine intelligence*, 26(9):1124–1137. (page)

[3] Brostow, G. J., Fauqueur, J., and Cipolla, R. (2008a). Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, xx(x):xx–xx. (page)

[4] Brostow, G. J., Shotton, J., Fauqueur, J., and Cipolla, R. (2008b). Segmentation and recognition using structure from motion point clouds. In *ECCV (1)*, pages 44–57. (page)

[5] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2018a). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848. (page)

[6] Chen, L.-C., Papandreou, G., Schroff, F., and Adam, H. (2017). Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*. (page)

[7] Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., and Adam, H. (2018b). Encoder-decoder with atrous separable convolution for semantic image segmentation. *arXiv preprint arXiv:1802.02611*. (page)

[8] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. (page)

[9] Dutta, A., Gupta, A., and Zissermann, A. (2016). VGG image annotator (VIA). `http://www.robots.ox.ac.uk/~vgg/software/via/`. Accessed: 2017-11-01. (page)

[Everingham et al.] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. The pascal visual object classes challenge 2012 (voc2012) results". http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html. (page)

[11] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. (page)

[12] Hara, K., Adams, A., Milland, K., Savage, S., Callison-Burch, C., and Bigham, J. P. (2018). A data-driven analysis of workers' earnings on amazon mechanical turk. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 449. ACM. (page)

[13] Jégou, S., Drozdzal, M., Vazquez, D., Romero, A., and Bengio, Y. (2017). The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*, pages 1175–1183. IEEE. (page)

[14] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. (page)

[15] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer. (page)

[16] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440. (page)

[17] Maggiori, E., Tarabalka, Y., Charpiat, G., and Alliez, P. (2017). Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark. In *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE. (page)

[18] Marszalek, M. and Schmid, C. (2007). Accurate object localization with shape masks. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE. (page)

[19] Nathan Silberman, Derek Hoiem, P. K. and Fergus, R. (2012). Indoor segmentation and support inference from rgbd images. In *ECCV*. (page)

[20] Oh, S., Hoogs, A., Perera, A., Cuntoor, N., Chen, C.-C., Lee, J. T., Mukherjee, S., Aggarwal, J., Lee, H., Davis, L., et al. (2011). A large-scale benchmark dataset for event recognition in surveillance video. In *Computer vision and pattern recognition (CVPR), 2011 IEEE conference on*, pages 3153–3160. IEEE. (page)

[21] Opelt, A., Pinz, A., Fussenegger, M., and Auer, P. (2006). Generic object recognition with boosting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(3):416–431. (page)

[22] Peng, C., Zhang, X., Yu, G., Luo, G., and Sun, J. (2017). Large kernel matters¿improve semantic segmentation by global convolutional network. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 1743–1751. IEEE. (page)

[23] Raza, S. H., Grundmann, M., and Essa, I. (2013). Geometric context from video. *IEEE CVPR*. (page)

[24] Richter, S. R., Vineet, V., Roth, S., and Koltun, V. (2016). Playing for data: Ground truth from computer games. In *European Conference on Computer Vision*, pages 102–118. Springer. (page)

[25] Robicquet, A., Sadeghian, A., Alahi, A., and Savarese, S. (2016). Learning social etiquette: Human trajectory understanding in crowded scenes. In *European conference on computer vision*, pages 549–565. Springer. (page)

[26] Rottensteiner, F., Sohn, G., Gerke, M., and Wegner, J. D. (2013). Isprs test project on urban classification and 3d building reconstruction. *Commission III-Photogrammetric Computer Vision and Image Analysis, Working Group III/4-3D Scene Analysis*, pages 1–17. (page)

[27] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252. (page)

[28] Russell, B. C., Torralba, A., Murphy, K. P., and Freeman, W. T. (2008). Labelme: a database and web-based tool for image annotation. *International journal of computer vision*, 77(1-3):157–173. (page)

[29] Shen, T., Lin, G., Shen, C., and Reid, I. (2017). Learning multi-level region consistency with dense multi-label networks for semantic segmentation. *arXiv preprint arXiv:1701.07122*. (page)

[30] Song, S., Lichtenberg, S. P., and Xiao, J. (2015). Sun rgb-d: A rgb-d scene understanding benchmark suite. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 567–576. (page)

[31] Sorokin, A. and Forsyth, D. (2008). Utility data annotation with amazon mechanical turk. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, pages 1–8. IEEE. (page)

[32] Torralba, A., Russell, B. C., and Yuen, J. (2010). Labelme: Online image annotation and applications. *Proceedings of the IEEE*, 98(8):1467–1484. (page)

[33] Upchurch, P., Sedra, D., Mullen, A., Hirsh, H., and Bala, K. (2016). Interactive consensus agreement games for labeling images. In *Proceedings of the 4th AAAI Conference on Human Computation and Crowdsourcing*. (page)

[34] Von Ahn, L. and Dabbish, L. (2004). Labeling images with a computer game. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 319–326. ACM. (page)

[35] Wah, C., Branson, S., Welinder, P., Perona, P., and Belongie, S. (2011). The caltech-ucsd birds-200-2011 dataset. (page)

[36] Zhao, H., Shi, J., Qi, X., Wang, X., and Jia, J. (2017). Pyramid scene parsing network. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2881–2890. (page)

[37] Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., and Torralba, A. (2016). Semantic understanding of scenes through the ade20k dataset. *arXiv preprint arXiv:1608.05442*. (page)

[38] Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., and Torralba, A. (2017). Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. (page)