Oracle Report

Shai Guendelman Elad Amar

1 Introduction

In recent years there is a growing interest in Unmanned Aerial Vehicles (UAVs). With time drones are becoming more accessible, more advanced and with enhanced abilities. This make UAVs ideal for wide number of applications such as: tracking, deliveries, ground assessment, etc. In order to deploy a fully autonomous UAV system, the system has to have certain basic capabilities, like the ability to autonomously land. This problem can be split into sub-tasks, some of them are:

1. finding and evaluating possible landing sites
2. finding an optimal approach path to the landing site
3. avoiding obstacles and moving objects during flight

In this work we focus on 1. Another issue is that the UAV must determine its landing site in real-time using on board limited computational resources. The above calls for an efficient(in terms of memory and compute time) method that can recognize many different types of landing sites. We chose to use image semantic segmentation as the core of our method, solely using on board monocular camera.

2 Related Work

2.1 Landing Site Detection(LSD)

Different approaches to the subjects differ in some key categories:

1. Definition of a landing site
2. Sensors used
3. Using prior knowledge

Landing Site Definition Some studies of the subject focus on searching for a specific pattern to land on, for example a Helipad [35]. The more challenging of finding a landing site "In The Wild", i.e. landing on a site that is not built specifically for that purpose, is also considered in many papers, for example in [19] the authors filter out from high altitude the places that are not suitable for landing using features extracted via classical Computer Vision techniques and an SVM classifier. In [11, 16, 18] The authors focus on the geometric properties of the terrain, preferring sites that are flat, smooth and clear of obstacles. In our work we use the fact that a dense urban environment is highly structured, and if detect certain structures (like an empty parking lot) we can be quite certain that it is a good landing site.

Sensors Most of studies mainly use vision for the task, some like ours use vision only - [16, 22], some combined with inertial measurements [11] to preform geometric analysis of the terrain, and some combine inputs from depth cameras [18]. Another popular sensor that can be used is Li-DAR [17, 3]. Each sensor has it's trade-offs - more sensors means heavier payload and higher power consumption. Depth cameras usually have a limited depth range around 30m. Li-DAR sensor give better range and higher accuracy, but are more complicated to use. Using visual input combined with deep learning techniques seems highly promising since the success it has had in other tasks.

Prior Knowledge Some revisited studies also utilize data that is known in advance about the environment, to allow offline computations and avoid some of the computations needed during the mission [7, 1]. In this work we assume that this is made in another module in the system.

2.2 Semantic Segmentation

In recent years there are significant improvements in semantic segmentation methods. An important milestone in the evolution of semantic segmentation problem was the utilization of models based on fully convolutional network(FCNs) lead by [15]. Later on, those methods outperformed by end-to-end encoder-decoder models [4, 2]. The goal of this methods is for the encoder to produce low resolution features maps, and the decoder upsamples those to full input resolution feature maps for pixel-wise classification. A novel and recent advance in semantic segmentation problem are the skip-connections that are used to copy features from the encoder's layers onto the decoder's layers. The skip-connections for semantic segmentation problem introduced in U-net[23] and being used in many state-of-the-art models [33, 21, 5]. The skip connections feeds deeper layers with activations of early layers and adds alternative paths for the gradients to backpropagate. Resulting the ability to train deeper networks.

3 Problem Description

The final goal of this work is to enable a fully autonomous UAV to safely land in a dense urban environment.

3.1 Scenario

The scenario that we are considering is of a drone hovering at high altitude of about 400 meters above a dense urban zone. The drone descends and at the same time is trying to locate a safe landing site, without knowing where to land in advance. The drone is equipped with multiple color cameras, Li-DAR and depth camera sensors, and has some partial prior knowledge of the area beneath it. The drone has limited power and payload, thus the mission is limited in time, computational resources, and movement.

3.2 The System

This work is part of a larger work on building a system that can accomplish the task above. The system is split into modules, where every module tackles a different part of the problem. Our module is called **Oracle**, and it is tasked with extracting information from sensory input for the **Decision Making** module to use. Our module is memory-less, which means that it operates independently for every input it is given, we assume that the **Decision Making** module maintains and integrates the information gained over time. To simplify the problem for now, we decided to use only color images as input to our module, also this is arguably the sensory input that provides the richest information about the environment.

3.3 Problem Formulation

The information that we want to provide to the **Decision Making** module (will be referred to as **System**) given an input image I, is a score ranging from 0 to 1 for each pixel in that image, S. Formally:

$$I \in [0,1]^{H \times W \times 3} = \mathcal{I}$$
$$S \in [0,1]^{H \times W} = \mathcal{S}$$

where the image resolution is of height H and width W, and it is **RGB** encoded with 3 channels. Entries with high values (close to 1) in S will be interpreted as pixels that are associated with a good landing site that we would like to land on, and entries with low values (close to 0) will be interpreted as pixels that are associated with places that are risky, and we would like to avoid landing on them. We can formalize this notion that the entry $S[i,j]$ is the

probability of safely landing on the place projected onto pixel $[i, j]$ in the image I.

Finally, the task of the **Oracle** is to be a mapping from I to an appropriate S:

$$S = Oracle(I) \tag{1}$$

4 Solution

4.1 High Level Description

The Oracle takes as input a RGB encoded image I, taken from a monocular camera. The Oracle outputs is a $2D$ scores map: S, each pixel in the scores map $S[i, j]$, represents how likely for that pixel to be a part of a potential landing zone. To accomplish this, We define a set of classes C that are interesting to us. We define a mapping from classes in C to a real number in $[0, 1]$ that represents how safe is that class for landing on, $Score$:

$$Score : C \to [0, 1] \tag{2}$$

assigning each class with a score, the higher the score is, the better this class as a landing site. We use a CNN model M_θ to find a pixel-wise segmentation of an input image to the classes in C. The M in that notation represents the model's architecture, and θ represents the learnable parameters of the model. The CNN model M_θ is a function from the image space, to an intermediate space, that associate with each pixel $[i, j]$ a real number value, for each one of the different classes:

$$M_\theta : \mathcal{I} \to \mathbb{R}^{H \times W \times |C|} \tag{3}$$

$$M_\theta(I) = CPL \tag{4}$$

CPL stands for Class Probability Logits. To get the model's predicted Class Probabilities CP, i.e. for each pixel $[i, j]$, a vector of $|C|$ dimensions, where the i'th entry is the probability of that pixel to belong to the i'th class in C, We take the $Softmax$ operation:

$$Softmax : \mathbb{R}^n \to [0, 1]^n$$

$$Softmax(x) = y$$

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

For each entry $CPL[i, j] \in \mathbb{R}^{|C|}$ we calculate (note that CP is of the same dimensions as CPL):

(a) (b) (c)

Figure 1: The Scheme: first the input image 1(a) is processed by the semantic segmentation model 1(b). Then using the scores for each class, the *Oracle* predicts the places that have high probability 1(c) for landing on(red) and places that we should avoid (blue)

$$CP[i,j] = Softmax(CPL[i,j]) \tag{5}$$

Once we have the class probability vector for each pixel in the image, $CP[i,j]$, we reduce it to a single value in $[0,1]$ that signifies it's score, $S[i,j]$. The reduction function $Reduce: [0,1]^{|C|} \rightarrow [0,1]$ can be defined in different ways, where we use the classes scores $Score$ defined in (2) for calculating $Reduce$. To get the final score we evaluate for each pixel $[i,j]$:

$$S[i,j] = Reduce(CP[i,j]) \tag{6}$$

.

4.2 Low Level Detailed Description

4.2.1 Segmentation Models

The CNN image segmentation model is the most computationally intensive part of our solution. Thus we preferred using a relatively small and fast networks. We experimented with 2 architectures:

BiSeNet [34] In this work, the authors identify a conflict that arises in the design of Deep Learning models for semantic segmentation. In order to correctly classify a pixel of an image, we need both **context** and **spatial** information. By **context** we mean the relation between the given pixel and the entire image. For example, say we have a task of classifying each pixel either as 'human' or 'car' as in Figure 3. A group of red pixels, can be both 'human' or 'car'. In order to correctly classify the pixels we need to look at the broader image. By **spatial** refers to the near by pixels. This is especially important in identifying patterns in an image, like a dog's ear or a wheel's rim. The problem that arises in the design of CNN's is that in order to get **context** we need a pixel's prediction to be affected by a large amount of other pixels('large receptive field'), and the

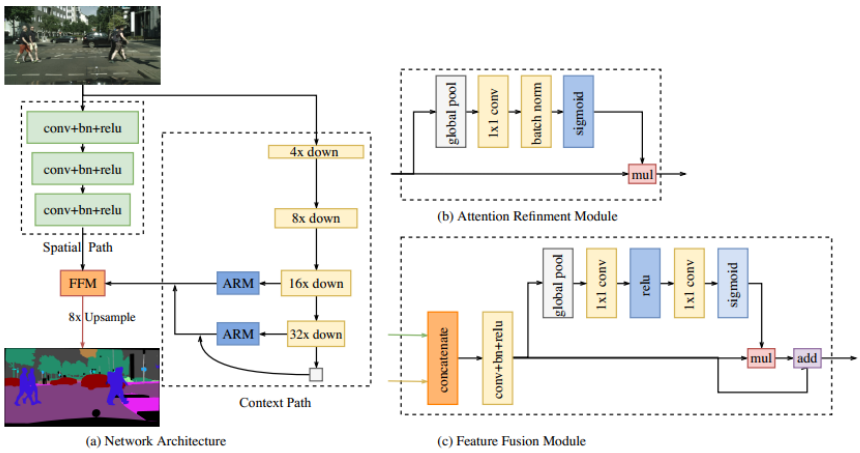


Figure 2: BiSeNet architecture scheme

Figure 3: A man wearing a red shirt, and a red car

way to do that efficiently is to rapidly down-sample the features. This rapid down-sampling prevents rich spatial features from being develops, so it comes with a price to the **spatial** information. The authors try to solve that problem by creating 2 paths - one with spatial information, where each convolution is affected by a small number of inputs, and with moderate down sampling, and a context path, with aggressive down sampling. Then they fuse the 2 paths for the prediction. In the paper they get to 68.4% $mIoU$ on the Cityscapes test set [6] a popular banchmark, and speed of 105 FPS on NVIDIA Titan XP card with input of 2048×1024 resolution images.

Fast-SCNN [21] is a semantic segmentation model consists of Down-sampling, feature extraction, feature fusion and finally a classifier as shown in Figure 4. Noting that first layers of a CNN extract low-level features, the down-sampling block has only three layers. It takes the input image and outputs 1:8 resolution feature map. Then, the output is passed through a bottleneck residual block in

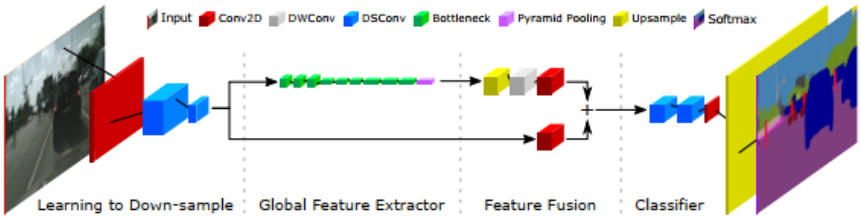


Figure 4: Fast-SCNN architecture scheme

the feature extraction part, and also passed to the feature fusion block, comprising a two-branch setup. The feature fusion use simple addition and non linear function and then passes the output for the classifier. The model leverages many features from state-of-the-art architectures in order to achieve a fast and low memory performance, yet having competitive results. A key aspects are using depth wise separable convolutions(DSConv) [27] and inverse residual blocks[24]. The DSConv consists of channel-wise spatial convolution followed by pointwise convolution. This method uses less memory and faster rather than the classic convolution which convolve over the entire volume. Moreover, the model utilizing efficiently the inverse residual blocks which expand a low-dimensional input to a higher-dimensional space. In contrast to other architecture, the low resolution features are shared in the two-branch approach, which helps the model to be shallow and keep spatiality in later layers. With all of the above the model achieve 68.0% $mIoU$ at 123.5 FPS on Cityscapes.

4.2.2 The Reduce Function

We experiment with 2 modes:

Threshold - With the **threshold** parameter $t \in (0.5, 1)$; for each pixel $[i, j]$ we take the class with the highest probability as predicted by the model (7). If the probability is greater or equal to the threshold, then we assign the score of the pixel to be the score of the predicted class, as defined by the mapping $Score$. If the condition does not hold, then we simply set it to 0(8).

$$C[i,j] = \underset{c}{argmax}(CP[i,j,c]) \tag{7}$$

$$S[i,j] = \begin{cases} Score(C[i,j]) & CP[i,j,C[i,j]] \geq t \\ 0 & otherwise \end{cases} \tag{8}$$

Weighted Average - In this mode we consider the entire probability distribution over the different classes that the pixel $[i, j]$, $CP[i, j]$. We consider the random variable $C[i, j]$ that takes values in C that represents the class of

the pixel $[i, j]$, where $p(C[i, j] = c) = CP[i, j, c]$. Then the reduce function is calculated by taking the expected value of $Score(C[i, j])$:

$$S[i, j] = \mathbb{E}(Score(C[i, j])) = \sum_{c \in C} Score(c) \cdot p(C[i, j] = c) \tag{9}$$

substituting $p(C[i, j] = c) = CP[i, j, c]$, and writing $Score$ as a vector $SV \in [0, 1]^{|C|}$ such that $SV[c] = Score(c)$, we can write the above as a dot product:

$$S[i, j] = CP[i, j] \cdot SV \tag{10}$$

This method makes sense as it also considers the richer information that the model provides, and takes into account the measure of confidence that the network has in it's prediction.

4.3 Kernel Smoothing

Another possible add-on for the reduction function, is to apply a convolution layer over S with a blurring kernel. This way each score is computed as the weighted average of neighboring scores, and the weights for closer pixels are higher. In our problem, this method helps us the penalize pixels with hazardous objects around it, and also, to get higher scores for centers of possible landing site. This can be expressed for Kernel K size $k \times k$ as:

$$S'[i, j] = K * S[i, j] = \sum_{n=-k/2}^{-k/2} \sum_{m=-k/2}^{-k/2} K[n, m] \cdot S[i + n, j + m] \tag{11}$$

for every pixel $[i, j]$. We assume that indexes outside of S are padded with zeros

5 Evaluation

5.1 Conditions and settings

Bellow are the setting on which the evaluation was preformed:

5.1.1 Data set

The data set on which the **Oracle** was evaluated was created using Unreal Engine and Airsim [25]. The data set was partitioned into 995 samples for the train set, and 438 samples for the test set.

5.1.2 Hardware

the system was tested in a single CPU-GPU setup, with the fallowing hardware: GPU: GeForce RTX 2080, CPU: Xeon(R) CPU E5-2683 v4 @ 2.10GHz,

Figure 5: an image and the segmentation ground truth from the data set

5.1.3 Software

We run on a Linux operating system (Ubuntu), python 3.8, pytorch version 1.5.0.

5.2 Correctness

The correctness is only evaluated for the semantic segmentation model, as ground truth labels are available for the evaluation.

5.2.1 Metrics Used

Bellow We describe the metrics we evaluated. All are common metrics used for evaluating semantic segmentation models. The metrics are calculated on per-image basis.

Notes and Notations

- y denotes the ground truth labels of image pixels, and $\hat{y}$ denotes the predicted labels. The predicted labels are taken based on the class that got the maximum value in the CPL produced by the model. Formally, for pixel $[i, j]$:
$$\hat{y}[i, j] = \underset{c}{argmax}\{CPL[i, j, c]\}$$
- All the "per class" metrics bellow are calculated for each class $c \in C$ for the entire set of classes.
- we will denote with $N = W \cdot H$ to be the total number of pixels in an image.
- $I_{condition}$ represents an indicator getting the value 1 if *condition* is true, otherwise gets the value 0.
- in all the "per class" metrics bellow, we use the term **Positive** (in True Positive/False Positive) to describe pixels that are of the class, and by **Negative** pixels that are not of that class(i.e., any other class). True means that the ground truth and the prediction agree on that pixel, False means that they do not agree.
- We use the shorthand TP - True Positives, TN - True Negatives, FP - False Negatives, FN - False Negative. We use $X[c]$ ($X \in \{TP, TN, FP, FN\}$) to denote that the results is with respect to class c.
- clearly $N = TP + TN + FP + FN$

1. **mean IoU(mIoU)** This is just taking the mean over the classes IoU:
$$mIoU = \frac{\sum_c IoU[c]}{|C|}$$

2. **per pixel accuracy** This is the total number of pixels that were correctly classified, divided by the total number of pixels.

$$per\ pixel\ accuracy = \frac{\sum_{i,j} I_{y[i,j]=\hat{y}[i,j]}}{N}$$

3. **per class IoU** shorthand for "Intersection over Union". This means taking the $Intersection$ of the pixels that belong to that class both in the prediction and the ground truth(= $TP[c]$), and dividing it by the size of the $Union$, the pixels that are labeled of that class in either the ground truth, prediction or both(= $TP[c] + FP[c] + FN[c]$). Formally:

$$pixels[c] = \{(i,j) :\ y[i,j] = c\}$$
$$\hat{pixels}[c] = \{(i,j) :\ \hat{y}[i,j] = c\}$$
$$Intersection[c] = \hat{pixels}[c] \cap pixels[c]$$
$$Union[c] = \hat{pixels}[c] \cup pixels[c]$$
$$IoU[c] = \frac{|Intersection[c]|}{|Union[c]|}$$

4. **per class accuracy** accuracy referrers to the ratio of the True predictions over all the predictions:

$$accuracy[c] = \frac{TP[c] + TN[c]}{N}$$

5. **per class recall** The ratio of between the number of successfully predicted pixels of the given class, over the total number of actual pixels of the class, intuitively, the probability of our model to detect a given pixel of that class given that it is of that class:

$$recall[c] = \frac{TP[c]}{TP[c] + FN[c]}$$

6. **per class precision** The ratio between the number of successfully predicted pixels of the given class, and the total number of predicted positives. Intuitively, the probability of a pixel belonging to the given class given that it was predicted of that class:

$$precision[c] = \frac{TP[c]}{TP[c] + FP[c]}$$

5.2.2 Results

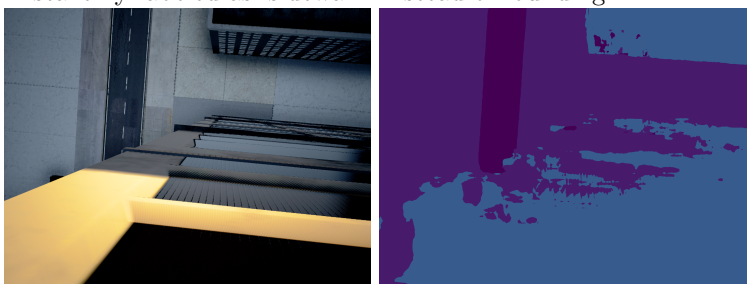
We compared the correctness based on the different segmentation model used.

- Results for class specific metric are shown in figure 8.
- Exact(up to 3 decimal points) per-class IoU is shown in figure 9.
- Results for pixel-wise accuracy and mean IoU are summarized in figure 10.

Note on accuracy: All accuracy results are biased, as they highly depend on the distribution of the labels in the dataset. For example, if jut 100 pixels in an image with a total of 10^6 pixels are of class *pole*, and non of them is predicted to be such, it will still get the accuracy of $(10^6 - 100)/10^6$ which is pretty close to 1 even though we did not detect any *pole* pixel successfully. We can say that for rare obstacles, we would like not to miss them, thus we should require high **recall** score on those classes, as this implies we have high probability for detecting them if they are present.

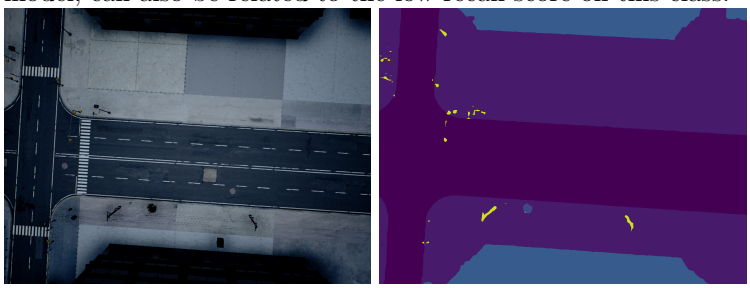
Some pitfalls: When visually inspecting the results of our model, we detected some issues:

1. Some shutters in buildings with textures similar to those of sidewalks were mistakenly labeled as 'sidewalk' instead of 'building':

   This might be solved with more image samples of those shutters, so the model will be able to better preform on them.

2. Some parts of the poles are missing in the prediction made by Fast-SCNN model, can also be related to the low recall score on this class:

   This might be to due the low frequency of this class in the dataset, and might be fixed by changing the train loss function to give extra weight to classes that are less frequent.

3. Road damage is some times incorrectly labeled, for example:

This might be solved by augmenting the data by pasting realistic road textures on to the roads in the simulation, and increasing the number of samples of this type in the dataset.

5.3 Speed and Memory

We tested our implementation runtime speed and memory usage in the fallowing ways:

CPU speed Using python's time.perf_counter recording it just before running the module, and after function ends, and counting the time differences. The results were collected over 100 different test set samples. The histogram of the results is shown in figure 13.

CUDA speed The CUDA runtime was evaluated using pytorch cuda event API. It is almost identical to the CPU time measurements, except that it requires synchronization, i.e., making sure that all threads are finished when recording the end-time, due to the asynchronous nature of GPUs. Results are shown in figure 16. All runtime speeds of the different modules are shown in figure 18.

CUDA memory it is important to track CUDA memory usage, as the the embedded device used will run multiple threads for different parts in the system, and might be a limiting factor in our system's performance. We measured peak memory used during runtime with pytorch cuda memory API. Results are shown in figure 17. One peculiar result we found is that the complete oracle module runs with less memory then the segmentation model only. This might be due to pytorch's inner workings.

6 Ideas / Notes

Using Li-DAR Integrating the Li-DAR sensory input for our model can result in increased safety and robustness. The downside is that it might increase our computational resource usage. Work done in that direction is [17].

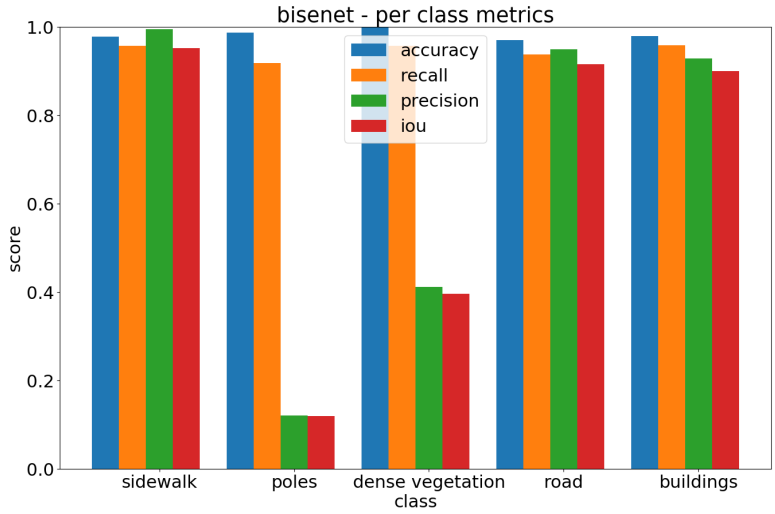


Figure 6: BiSeNet

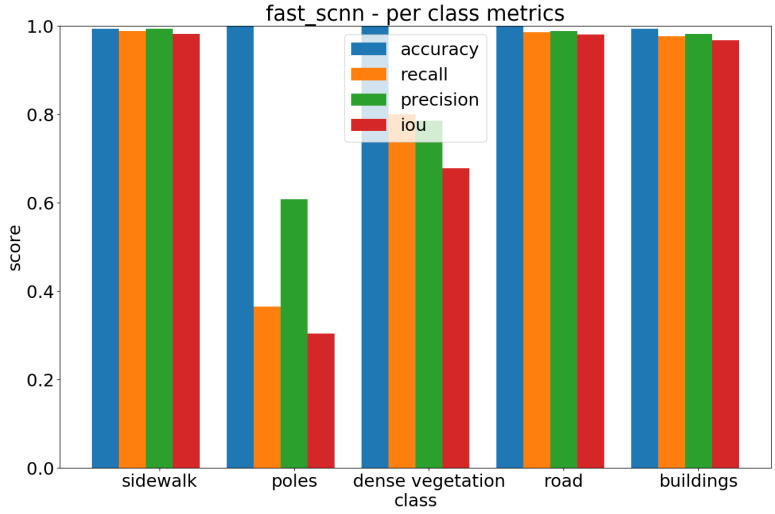


Figure 7: Fast-SCNN

Figure 8: per class metrics for our models

class name	segmentation model	mean value
buildings	bisenet	0.900
buildings	fast scnn	0.967
dense vegetation	bisenet	0.397
dense vegetation	fast scnn	0.679
poles	bisenet	0.119
poles	fast scnn	0.304
road	bisenet	0.916
road	fast scnn	0.980
sidewalk	bisenet	0.952
sidewalk	fast scnn	0.982

Figure 9: Per Class IoU

metric name	segmentation model	mean value
mean iou	bisenet	0.803
mean iou	fast scnn	0.886
pixelwise accuracy	bisenet	0.957
pixelwise accuracy	fast scnn	0.993

Figure 10: All Classes Metrics

Weak Supervision A problem we will face when transferring from the simulation to a real world scenario is acquiring training data for our segmentation model. Pixel-wise labeling by humans is not practical. Weak supervision methods [32, 14, 20] aim to train the segmentation models based on partial labeling, for example by using image level classification labels or object bounding box annotations in the images, that are much easier to generate. Thus using weak supervision can reduce the cost of labeling real images. It is also worth mentioning that many data sets with image or object level annotation exists that can be used during development, like VisDrone.

Synthetic Data Set Augmentation Using synthetic data to train models has been done many times, for example in [28]. different methods have been done on transferring models that were trained on synthetic data to predict labels of real images. One method is Domain Randomization [30, 31]. In this method we render the scene in different settings, like different textures or lighting. This forces the model to learn more meaningful features. Another interesting method is using a model that transforms the synthetic data to look more real, while preserving the correctness of the ground truth labels [26]. There are many more methods that can be tested and applied, as in [9].

Formal Verification Formal Verification is using mathematical methods for proving certain properties of a piece of software. It is relatively immature in the context of deep learning systems, and current methods can only be applied to

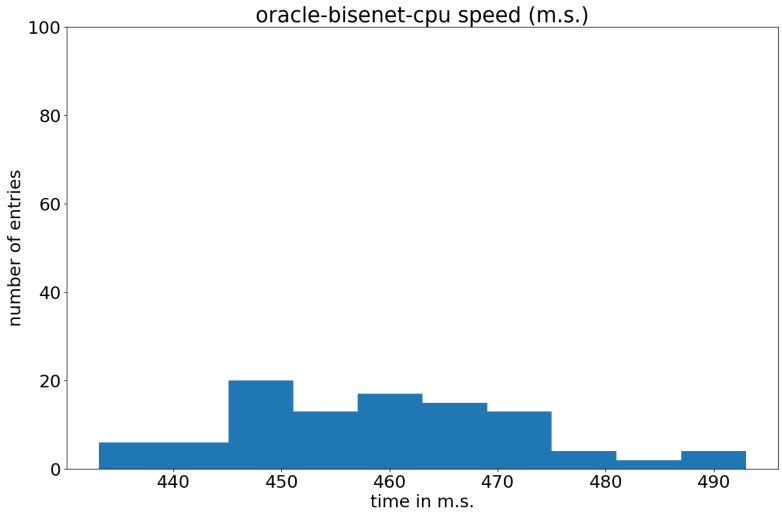


Figure 11: BiSeNet

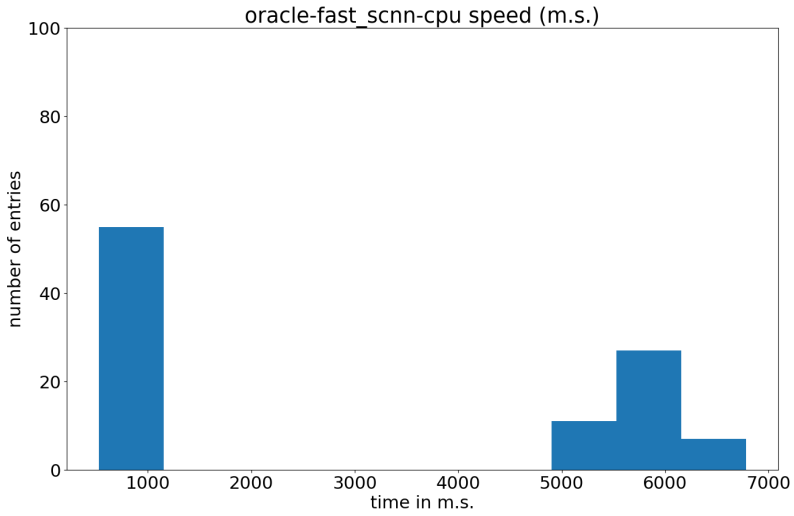


Figure 12: Fast-SCNN

Figure 13: CPU m.s. per-image histogram

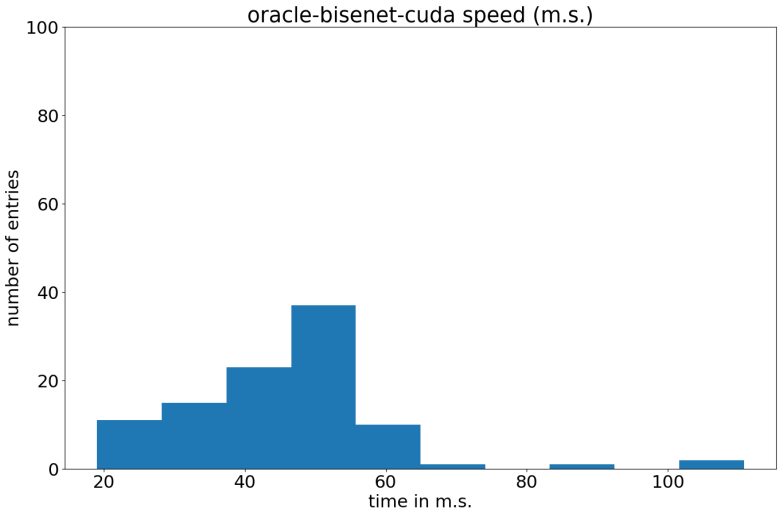


Figure 14: BiSeNet

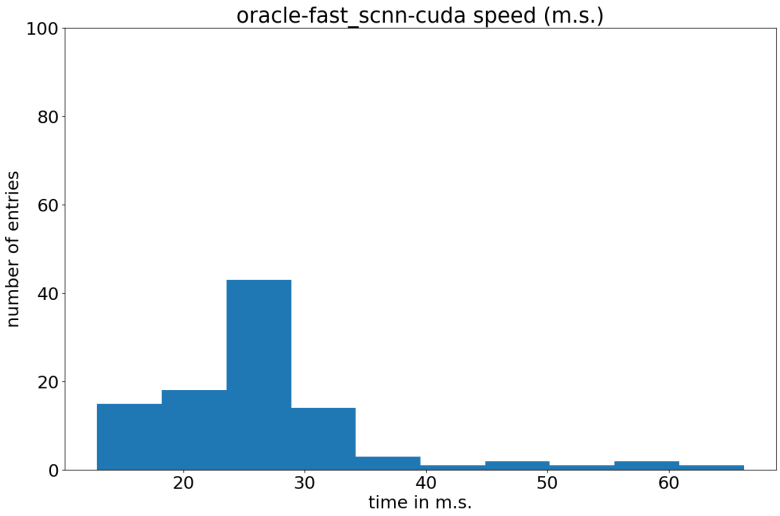


Figure 15: Fast-SCNN

Figure 16: CUDA m.s. per-image histogram

module	segmentation model	mean value
segmentation	bisenet	860709888
segmentation	fast scnn	1034959360
oracle	bisenet	239887667
oracle	fast scnn	300391936

Figure 17: Memory Usage on CUDA in Bytes

device	module	segmentation model	mean value
cpu	data conversion input transform	bisenet	50.399
cpu	data conversion input transform	fast scnn	45.864
cpu	data conversion output transform	bisenet	9.939
cpu	data conversion output transform	fast scnn	11.422
cpu	oracle	bisenet	459.432
cpu	oracle	fast scnn	2928.876
cpu	segmentation	bisenet	393.751
cpu	segmentation	fast scnn	510.711
cuda	oracle	bisenet	46.010
cuda	oracle	fast scnn	26.350
cuda	segmentation	bisenet	20.344
cuda	segmentation	fast scnn	11.182

Figure 18: Runtime in m.s. of different parts of the module

analyze only a small model. In the future this methods might enable to build much more robust models, that can be safely deployed in places where error might be very costly(like in a city environment) [12, 8, 10, 13, 29].

Tensor RT TensorRT is offered by NVIDIA for optimizing deep learning models performance by optimizing the software to the NVIDIA hardware it runs on. It is possible to convert a pytorch trained model to a TensorRT model using torch2trt. NVIDIA offers a tutorial on it.

References

[1] Nicolas Audebert, Bertrand Le Saux, and Sébastien Lefèvre. "Beyond RGB: Very high resolution urban remote sensing with multimodal deep networks". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 140 (2018), pp. 20–32.

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation". In: *IEEE transactions on pattern analysis and machine intelligence* 39.12 (2017), pp. 2481–2495.

[3] Jeremy Castagno, Yu Yao, and Ella Atkins. "Realtime Rooftop Landing Site Identification and Selection in Urban City Simulation". In: *arXiv preprint arXiv:1903.03829* (2019).

[4] Liang-Chieh Chen et al. "Rethinking atrous convolution for semantic image segmentation". In: *arXiv preprint arXiv:1706.05587* (2017).

[5] Wuyang Chen et al. "FasterSeg: Searching for Faster Real-time Semantic Segmentation". In: *arXiv preprint arXiv:1912.10917* (2019).

[6] Marius Cordts et al. "The cityscapes dataset for semantic urban scene understanding". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3213–3223.

[7] Jerry Ding et al. "Initial designs for an automatic forced landing system for safer inclusion of small unmanned air vehicles into the national airspace". In: *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE. 2016, pp. 1–12.

[8] Krishnamurthy Dvijotham et al. "A Dual Approach to Scalable Verification of Deep Networks." In: *UAI*. Vol. 1. 2018, p. 2.

[9] Debidatta Dwibedi, Ishan Misra, and Martial Hebert. "Cut, paste and learn: Surprisingly easy synthesis for instance detection". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1301–1310.

[10] Ruediger Ehlers. "Formal verification of piece-wise linear feed-forward neural networks". In: *International Symposium on Automated Technology for Verification and Analysis*. Springer. 2017, pp. 269–286.

[11] Timo Hinzmann et al. "Free LSD: prior-free visual landing site detection for autonomous planes". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 2545–2552.

[12] Xiaowei Huang et al. "Safety verification of deep neural networks". In: *International Conference on Computer Aided Verification*. Springer. 2017, pp. 3–29.

[13] Yafim Kazak et al. "Verifying deep-RL-driven systems". In: *Proceedings of the 2019 Workshop on Network Meets AI & ML*. 2019, pp. 83–89.

[14] Anna Khoreva et al. "Simple does it: Weakly supervised instance and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 876–885.

[15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[16] Alina Marcu et al. "SafeUAV: Learning to estimate depth and safe landing areas for UAVs from synthetic data". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018.

[17] Daniel Maturana and Sebastian Scherer. "3d convolutional neural networks for landing zone detection from lidar". In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 3471–3478.

[18] Mayank Mittal, Abhinav Valada, and Wolfram Burgard. "Vision-based autonomous landing in catastrophe-struck environments". In: *arXiv preprint arXiv:1809.05700* (2018).

[19] Kausar Mukadam, Aishwarya Sinh, and Ruhina Karani. "Detection of landing areas for unmanned aerial vehicles". In: *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*. IEEE. 2016, pp. 1–5.

[20] Deepak Pathak, Philipp Krahenbuhl, and Trevor Darrell. "Constrained convolutional neural networks for weakly supervised segmentation". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1796–1804.

[21] Rudra PK Poudel, Stephan Liwicki, and Roberto Cipolla. "Fast-scnn: fast semantic segmentation network". In: *arXiv preprint arXiv:1902.04502* (2019).

[22] L. Oyuki Rojas-Perez, Roberto Munguia-Silva, and José Martinez-Carranza. "Real-Time Landing Zone Detection for UAVs using Single Aerial Images". In: *10th International Micro Air Vehicle Competition and Conference*. Ed. by S. Watkins. Melbourne, Australia, Nov. 2018, pp. 243–248.

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[24] Mark Sandler et al. "Mobilenetv2: Inverted residuals and linear bottlenecks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.

[25] Shital Shah et al. "Airsim: High-fidelity visual and physical simulation for autonomous vehicles". In: *Field and service robotics*. Springer. 2018, pp. 621–635.

[26] Ashish Shrivastava et al. "Learning from simulated and unsupervised images through adversarial training". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2107–2116.

[27] Laurent Sifre and Stéphane Mallat. "Rigid-motion scattering for image classification". In: *Ph. D. thesis* (2014).

[28] Hao Su et al. "Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2686–2694.

[29] Xiaowu Sun, Haitham Khedr, and Yasser Shoukry. "Formal verification of neural network controlled autonomous systems". In: *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 2019, pp. 147–156.

[30] Josh Tobin et al. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2017, pp. 23–30.

[31] Jonathan Tremblay et al. "Training deep networks with synthetic data: Bridging the reality gap by domain randomization". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 969–977.

[32] Jia Xu, Alexander G Schwing, and Raquel Urtasun. "Learning to segment under various forms of weak supervision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3781–3790.

[33] Changqian Yu et al. "BiSeNet V2: Bilateral Network with Guided Aggregation for Real-time Semantic Segmentation". In: *arXiv preprint arXiv:2004.02147* (2020).

[34] Changqian Yu et al. "Bisenet: Bilateral segmentation network for real-time semantic segmentation". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 325–341.

[35] Leijian Yu et al. "Deep learning for vision-based micro aerial vehicle autonomous landing". In: *International Journal of Micro Air Vehicles* 10.2 (2018), pp. 171–185.