Is RGB all you need? The usefulness of depth in semantic segmentation: Final Report

G114 (s1762992, s1766172)

Abstract

Conventional datasets for computer vision tasks include images with solely RGB features. Nevertheless, in particular domains, it is possible to gather per pixel information from other sensors. Especially, the DroneDeploy Segmentation Dataset includes images of large aerial scenes with additional information about the elevation of the spatial region. We try to determine whether utilizing the elevation features improves the performance of the semantic segmentation in that dataset by exploring different approaches of incorporating the elevation data into a DeepLabV3 model. Additionally, we question the usefulness of the depth information itself. Looking at the quality of the dataset we conclude that issues with annotation can overshadow any of the benefits of the depth.

1. Introduction

The performance of novel machine learning approaches on computer vision tasks is typically evaluated on a very narrow set of image collections. Examples of such datasets include CIFAR-10, CIFAR-100, (Krizhevsky et al., 2009) or ImageNet (Deng et al., 2009) for object recognition as well as Cityscapes (Cordts et al., 2016), COCO (Lin et al., 2014), or PASCAL VOC (Everingham et al., 2012) for semantic segmentation.

The images in all of the previously mentioned datasets are represented with RGB features only. Nevertheless, in some particular domains, the images can also contain other characteristics than the colour intensities. For example, KITTI (Geiger et al., 2013) is a dataset that, apart from the RGB features, contains a depth map, which contains information about the distance between the captured object and the camera. Similarly, the images of large aerial scenes captured by satellites as in SpaceNet 6 (Shermeyer et al., 2020) or by drones as in DroneDeploy Segmentation Dataset (Nicholas Pilkington & Holmes, 2019) can also contain *multispectral* features.

When multimodal characteristics are available, restricting the inputs of machine learning tasks solely to RGB features seems unreasonable. The data from other sensors can convey valuable information that can lead to better discrimination of particular classes in tasks such as semantic segmentation, object detection or image classification.

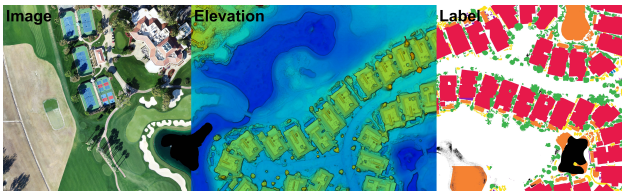

Figure 1. Example image representing the data from the DroneDeploy dataset, taken from the dataset's repository

Notwithstanding, simply adding non-RGB features may not necessarily lead to performance improvements either, as the noisiness of such information sources might overshadow their discriminative power.

What has particularly inspired us to tackle the problem of depth in Computer Vision is the following quote by Elon Musk, the CEO of Tesla, Inc.:

> *LiDAR is a fool's errand... and anyone relying on LiDAR is doomed* - **Elon Musk**

Uber, Waymo, Toyota and almost every other company working on an autonomous car rely on the depth data captured by the expensive LiDAR sensor in their Computer Vision systems. LiDAR is a method of targeting an object with a laser and measuring the time for the reflected light to return to the receiver, calculating the depth in the process. This is currently the most advanced and reliable way of capturing depth. LiDAR isn't used, however, by the most popular self-driving car manufacturer[1], Tesla. Elon Musk claims that, if humans can do it without explicit depth sensors, cars should be able to do it too.

Is it worth investing in elevation measuring sensors? Such sensors can be expensive and if they do not lead to substantial improvements in performance, then what is the point of investing in them? Maybe it is better to improve performance by gathering more data rather than buying another sensor? If a single good quality LiDAR device on a car can cost up to $10,000[2] and up to $100,000[3] if we want to fly it on a drone, perhaps it is better to invest in a higher quality dataset?

[1] https://www.forbes.com/sites/greatspeculations/2020/07/03/tesla-king-of-self-driving-cars/

[2] https://towardsdatascience.com/why-tesla-wont-use-lidar-57c325ae2ed5

[3] https://wingtra.com/drone-photogrammetry-vs-lidar/

To address the question of the usefulness of depth we have decided to study the difficulty of incorporation of the depth data into a semantic segmentation model. We have chosen the DroneDeploy dataset (Figure 1), a relatively unknown dataset containing large, high-quality aerial scenes with an elevation map attached to each image. We have found only 2 related papers (Heffels & Vanschoren, 2020; Parmar et al., 2020), that use this dataset, none of which use the depth data. Our goal was to improve the baseline result of DeepLabV3 (Chen et al., 2017b), a popular DeepLabV3 semantic segmentation model and determine whether more datasets should start capturing the depth data or rather focus on better quality of RGB images by verifying the usefulness of the depth in the dataset.

We start with adding the depth as a fourth channel in the input, following with more sophisticated methods of incorporating depth using the ESANet (Seichter et al., 2020) architecture with a separate ResNet-based depth encoder and Fusion blocks and lastly, a Depth-aware convolution (Wang & Neumann, 2018) operation that can be used in place of standard convolution layers.

We evaluate our proposed models and show that the incorporation of depth does not result in obvious improvements in the accuracy of our baseline RGB model. Moreover, we conduct experiments on the dataset itself, showing that the addition of depth cannot overcome underlying issues in the data. More specifications are provided in Section 3 and 4.

2. Data set and task

2.1. DroneDeploy Dataset

DroneDeploy dataset consists of large, aerial scenes captured from drones. Each scene has a ground resolution of 10 cm per pixel. For each scene, there is a corresponding RGB image, elevation map and an annotated label image.

The dataset contains 55 very large RGB images: 35 for training, 8 for validation and 12 for testing purposes, totalling 9.1 GB with the corresponding elevation and label data. Each image is split into non-overlapping 300x300 chunks, yielding 6888 chunks for training and validation in total. Similarly, elevation and label maps have been split in the same manner.

The images are stored as RGB TIF files. The elevation maps are single-channel TIFs, where each pixel has a corresponding pixel in the RGB image and represents an elevation in meters. Finally, the labels are PNG files with seven colours representing the seven classes, namely Building, Car, Vegetation, Clutter, Ground, Water and Ignore, where the Ignore class refers to a mask area with a missing label or outside of the image boundary. When the original image is split into 300x300 chunks, any chunk that contains a pixel with the Ignore label is rejected. Thus, in the final dataset divided into chunks, all of the pixels fall into one of six and not seven of the previously mentioned classes.

Table 1 shows the unbalanced distribution of pixels among the classes in the dataset. Furthermore, the class distributions vary significantly between the training and validation set. Figure 7 shows the distribution of per-pixel elevation in each image in the datasets. We also notice a slight imbalance between the datasets.

Class	Training Set	Validation Set
Building	3.84	6.85
Clutter	2.31	3.74
Vegetation	26.27	16.66
Water	2.95	2.31
Ground	64.29	70.08
Car	0.33	0.36

Table 1. Distribution of pixels among classes in the training and validation sets of the DroneDeploy dataset. The table displays the percentage of pixels with a given label in the particular set.

2.2. Preprocessing

Although no pixel with the Ignore label was present in the dataset, some of the pixels were still missing the elevation data. The missing elevation was encoded with a -32,767 value when the remainder of elevation was in the $[-40, 505]$ range. Fortunately, only less than 0.0002% of pixels in the dataset were missing elevation. We have decided to impute such missing values with k-NN as suggested in (Butcher & Smith, 2020). We have used $k = 8$, as it is reasonable that a typical pixel shares edges with four adjacent pixels and four vertices with another four diagonal pixels. Euclidean distance measure based on the difference between pixel locations in the 2D pixel grid was utilized to find the nearest neighbours. Nearest neighbours with missing elevation were not taken into account. We believe that the use of k-NN imputation was justified in our scenario as it preserves the local elevation distributions which should aid the classifier in the segmentation task.

After the imputations, all of the RGB and elevation values were normalized to the $[0, 1]$ range. Then we have further standardized the resultant values with the per channel mean and standard deviation computed on the training set *(the elevation was treated as a fourth channel during the preprocessing)*. We have applied a mean of $[0.5220, 0.5120, 0.4516, 0.0937]$ and std of $[0.1983, 0.1882, 0.1934, 0.0588]$.

2.3. Image segmentation

Image segmentation is a pixel-level classification task where each pixel in the image has to be classified into one of the available classes. This can be described as partitioning the image into multiple segments, where each segment contains a collection of neighbouring pixels belonging to the same class. Segmentation is typically split into two types: *semantic segmentation* and *instance segmentation*. The definition interpreted from (Arnab et al., 2018) describes them as follows:

Semantic segmentation is the process of assigning a label to every pixel in the image, where multiple objects of the same class are treated as the same entity

Instance segmentation is the process of assigning a label to every pixel in the image, but here multiple objects of the same class are treated as distinct individual objects (or instances).

Due to the additional difficulty in separating the segmented mask into distinct objects, instance segmentation is considered to be a much harder task than semantic segmentation. We have concluded that the trade-off of difficulty for the ability to find individual instances of the object is not favourable for aerial imagery and is typically not as useful. Hence, we have decided to focus on semantic segmentation, seeing more resources and related work for this task.

2.4. Intersection over Union

The typical accuracy benchmark that is used in the research community for image segmentation is the Intersection over Union averaged across all classes or **mIoU** (Equation 1), where $0 \leq mIoU \leq 1$.

$$mIoU = \frac{\sum_{i=1}^{C} \frac{A_i \cap B_i}{A_i \cup B_i}}{C} \tag{1}$$

The intersection $A_i \cap B_i$ for each class i represents the pixels found both in the prediction mask and ground truth mask. The union $A_i \cup B_i$ has all of the pixels found in either of the masks. The ratio of the above shows how well the predicted mask overlaps the ground truth - a too small prediction mask will cause the intersection to be small, a too-large prediction mask will make the union large, in both cases making the mIoU smaller.

The calculated score over each class is then averaged out over the number of classes C. Although the score can be balanced on the number of examples for each class, an unbalanced version is typically used (Heffels & Vanschoren, 2020), as seen in Equation 1.

2.5. "Depth" Neural Networks

In standard semantic segmentation, the model extracts the features from an image using a backbone that is common to other tasks like image classification or object detection. However, the main difference between the classification and segmentation models is in their classifiers. Since for segmentation we expect an output of the same resolution as the image, with the classes localised and their boundaries annotated, the features cannot undergo such an intense compression. What's more, the level of detail in segmentation has to be passed and maintained through all of the layers to ensure good accuracy.

Another issue can be the distinction of separate classes during segmentation, which might be difficult to distinguish due to similar colours (grass vs tree) or poor lighting conditions (shaded area).

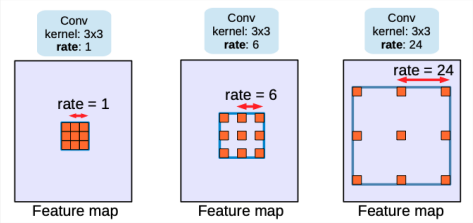

Figure 2. Example 3x3 atrous convolutions with 3 rates. Image taken from (Chen et al., 2017b).

Depth information that has recently become more widely used thanks to the addition of stereo depth estimation and the better accessibility to LiDAR sensors, in theory, could help in solving both of these issues. This single-channel data, as an addition to the RGB data, could theoretically help extract additional geometric cues that are immensely important for segmentation. In particular, (Wang & Neumann, 2018) claims that the extracted geometry from the depth image should help distinguishing correlations not captured by the RGB image, like more concrete contours of the distinctive classes.

In both of the aforementioned issues, precise depth information could help disambiguate similar-looking classes when the RGB information can be lost due to a similar signal frequency. RGB-D models, when incorporating the depth information correctly, should have a better ability to be resistant to unfavourable perturbations of the input.

3. Methodology

In this section, we describe the baseline segmentation model and review how it extracts dense features from the RGB image for semantic segmentation. We then propose two models that incorporate the additional depth information into their feature output and discuss how these differ from the traditional RGB approach.

3.1. Baseline model

For this project, we have decided to use DeepLabV3 (Chen et al., 2017b) as the base semantic segmentation model. The family of DeepLab models for semantic segmentation has been one of the most popular architectures thanks to its fairly straightforward architecture, based on a pre-trained backbone model like VGG (Simonyan & Zisserman, 2015) or ResNet (He et al., 2015) and atrous convolution. Base DeepLabV3 model uses ResNet50 as the encoder to extract visual features that later get decoded with an Atrous Spatial Pyramid Pooling (ASPP) (Chen et al., 2017a) module. Original ASPP module contains four parallel convolutions, with one 1x1 convolution and three 3x3 atrous convolutions with different atrous rates that enable effectively capturing multi-scale information from the encoder. To incorporate additional global-level features that are crucial in an accurate semantic segmentation, DeepLabV3 introduces image-level features using global average pooling on the last layer

of the backbone. The output of the pooling is then passed through a 1x1 convolution to reduce the number of channels to 256 and bilinearly upsampled to the desired output resolution. These 5 outputs in the ASPP module are then concatenated, passed through a final classifier module that is then upscaled to final image resolution.

DeepLab introduces the notion of output stride, which signifies how much smaller is the output resolution of the model. A typical ResNet model reduces the output resolution by 32x before it performs final classification. By using atrous convolution in the last layer of the ResNet encoder, DeepLabV3 can preserve the resolution of the second last backbone feature map which is 16x smaller than the resolution of the image, hence the *output_stride=16*. Atrous convolution or dilated convolution (Figure 2) allows the model to effectively enlarge the field of view of filters to incorporate multi-scale context (Chen et al., 2017b), without reducing the resolution as done by pooling layers or striding in convolution.

Learning rate policy: We use a learning rate of 0.01 (Chen et al., 2017b; 2018; Heffels & Vanschoren, 2020). However, as implemented by (Chen et al., 2017b), we decrease the learning rate to 0.001 for the backbone when using pre-trained weights from ResNet. The lower learning rate for the encoder part ensures the model relies on the visual features extracted from the image and prevents its overfitting, thus promoting healthy learning of the classifier.

We use a standard multiplicative *"poly"* learning rate scheduler for segmentation tasks as in (Chen et al., 2017b; Wang & Neumann, 2018). In this scheme, the learning rate is updated every number of iterations via equation 2, where η_{init} stands for the learning rate at the beginning of training. To be more specific we follow the approach from (Wang & Neumann, 2018) more closely by updating the learning rate every ten iterations.

$$\eta_{iter} = \eta_{init} * \left(1 - \frac{iter}{max\_iter}\right)^{0.9} \tag{2}$$

$$max\_iter = \frac{\#epochs * \#training\_chunks}{batch\_size} \tag{3}$$

Furthermore, in every experiment, we use a batch size of 8.

Data augmentation: Data augmentation is an industry standard that allows the models to train on more coherent yet diverse data.

For datasets like ImageNet or PASCAL VOC, it is common to add a horizontal flip of the image but there is no added value if that variant doesn't exist in real life, e.g. vertically flipped portrait. However, (Heffels & Vanschoren, 2020) claims that aerial imagery differs in this regard from ground-level images since the top-down perspective can be flipped along both axes without distortion, hence we have decided to use both x-axis and y-axis random flip.

Additionally, we have added random scaling of the input images by a factor in the range [0.5, 2,25]. For the crop size, we have decided to use the maximum size of each chip - 300x300, following the findings of (Chen et al., 2017b) which showed that the large receptive field of the atrous convolution required a non-padded image, hence a smaller crop size would result in atrous convolution layers being applied to zero-padded regions.

3.2. RGB-D input

The most trivial way to incorporate the depth or elevation into a model is to simply treat the depth as a 4^{th} input channel of the image. This alteration results in a mild change of network structure as solely the input layer has to accommodate more parameters to handle the new channel. Intuitively, the modest adjustment should lead to improvements in performance as the model will have more information on which it will be able to base its discriminative decisions.

However, what is not trivial is how to incorporate the 4^{th} input channel into a pre-trained model on the RGB-only dataset. Simply initializing the parameters that handle the elevation data as if the model was trained from scratch is not a good solution. In such a scenario, the model simply learns to neglect the depth input as initially, the randomly initialized weights make such input too noisy when compared to the already finetuned colour data. A better approach to incorporating the depth data into a model pre-trained on RGB-only dataset was defined in (Seichter et al., 2020) where the weights handling depth in the input are initialized to the sum of the standard RGB handling parameters along the input channel dimension. Here we use this method when incorporating the elevation as 4^{th} channel into the pre-trained model.

3.3. Efficient Scene Analysis Network (ESANet)

As the treatment of depth as simply the 4^{th} input channel seems quite naive, we have decided to look for more sophisticated methods of incorporating depth into our baseline model via architectural augmentation. One such approach was proposed in (Seichter et al., 2020). The ESANet architecture utilizes two encoders, one for RGB and one for depth. Each encoder is a ResNet and the information from the depth encoder is fused into the RGB encoder via RGB-D Fusion blocks that are applied at every resolution stage. The overall architectural structure of an ESANet is demonstrated in Figure 3.

The core distinctive feature of an ESANet encoder is its RGB-D fusion block. The RGB-D fusion block demonstrated in Figure 4 applies a separate Squeeze and Excitation (SE) (Hu et al., 2018) operation to depth and colour data and then fuses the two outputs via summation.

Precisely, SE has a form of global average pooling followed by 1x1 convolution that reduces the number of channels by a factor of 16. Then ReLU activation is applied, the initial number of channels is restored via another 1x1 convolution followed by sigmoid. Finally, the initial input is rescaled

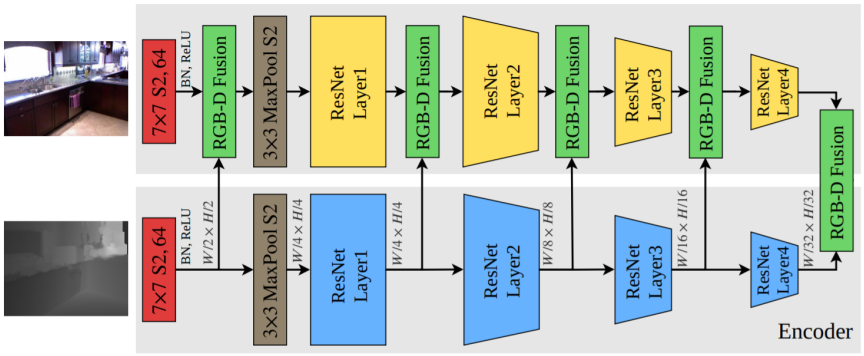

Figure 3. The encoder architecture in ESANet. 7x7 relates to the kernel size. S2 indicates a stride of 2. 64 is the number of output channels in the first convolution. Note that in our implementation of ESANets the last ResNet layer applies atrous convolution to preserve the image resolution. Hence, our final feature maps are 16 and not 32 times smaller than the resolution of the original image, unlike the original ESANet in the Figure above. Image taken from (Seichter et al., 2020)

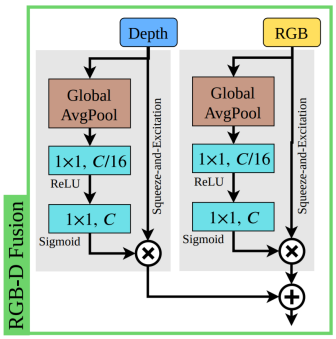

Figure 4. The structure of the RGB-D Fusion block in ESANet. C is the number of input channels into the block, so $C/16$ or C in a cyan block refers to the number of output channel after respective 1x1 convolution. Image taken from (Seichter et al., 2020)

by elementwise multiplication of the results of the previous operations to attain SE's output.

The initial global average pooling termed *"squeezing"* aggregates global spatial information into a channel descriptor. Then the following two 1x1 convolutions employ the *"excitation"* operation that extracts the inter-channel dependencies in the data. Consequently, SE is an attention-based mechanism that exploits the global inter-channel dependencies in a signal of a particular type. This lets the model learn how to reinforce certain features key for discrimination and suppress others in the face of a given input.

As we have reviewed different sources it became apparent that fusion of depth into the RGB signal can occur at different stages in the architecture. Our initial naive model with 4^{th} depth input channel fused depth at the earliest possible level. (Seichter et al., 2020) fuses the depth at multiple stages in the backbone while (Gupta et al., 2014) is a proponent of late fusion. As there are so many possibilities, we have decided to construct four models that fuse depth into RGB at different stages to empirically determine the most successful one. All but the last ASPP fusion model incorporate the RGB-D fusion unit to fuse depth with colour.

All of the models can use depth encoder pre-trained on RGB-only data. The pretraining on the first singular depth input channel is incorporated as in section 3.2.

Early Fusion applies the first 7x7 convolution on the depth input map and fuses its output with RGB. No more separate depth processing is undertaken.

ESANet approach fuses the depth signal with colour exactly as in (Seichter et al., 2020) and Figure 3.

Late fusion follows the approach from () where the full separate depth encoder is utilized but the signals are fused only after the final ResNet layer.

ASPP fusion is our custom fusion model that does not use the RGB-D fusion units. This model has a full depth ResNet encoder, its output is passed to the Deeplab's classifier which processes it in parallel exactly as an RGB signal. Encoded depth goes through the 1x1 convolution, three 3x3 dilated convolutions and one global average pooling. Therefore, before the final ASPP projection, the 10 colour and depth feature maps are fused via concatenation, compared to the 5 in the standard ASPP module as described in Section 3.1.

3.4. Depth-aware convolutions

A compelling way of incorporating depth information into the model called depth-aware CNN was presented in (Wang & Neumann, 2018). What is the most appealing about the depth-aware CNN is that unlike the preceding methods it neither alters the network's architecture nor increases its number of parameters. Depth-aware CNN succinctly incorporates the depth into the model by simply augmenting the classical convolution operation to take advantage of the availability of non-RGB information. In a classical convolution, a kernel is applied to a local grid of pixels by multiplying each pixel-value by the corresponding weight in the kernel. However, in the depth-aware convolution, the kernel weight multiplies a value of its corresponding

pixel that was first scaled by the similarity of that pixel's elevation to the elevation of the central pixel in the local grid. *(This assumes that the kernel is a square and that its side sizes are odd).*

The depth similarity of two pixels is defined by equation 4, whereas the overall operation of depth-aware convolution can be summarized with equation 5.

$$F_D(p_i, p_j) = e^{-\alpha * |D(p_i) - D(p_j)|} \tag{4}$$

$$y(p_0) = \sum_{p_n \in R} w(p_n) * F_D(p_0, p_0 + p_n) * x(p_0 + p_n) \tag{5}$$

$D(p_i)$ is the depth of pixel p_i. α is a hyper-parameter that can be adjusted to increase or decrease the effect of scaling via depth. Smaller α increases the influence of depth, while larger one does the opposite. When $\alpha = 0$ the depth information is not taken into consideration. (Wang & Neumann, 2018) suggests that $\alpha = 8.3$ is the optimal value of this hyper-parameter. Thus, in our experiments with depth-aware CNN we have used this value.

In equation 5 R defines the set of pixels in local grid around the central pixel p_0. $w(p_n)$ is the kernel weight corresponding to pixel p_n. $x(p_0 + p_n)$ is the value of pixel p_n.

The depth-similarity scaling in the depth-aware convolution ensures that pixels with similar elevations to the central pixel in the local grid have significantly higher activations. In effect, the depth-awareness smoothly transforms the receptive field of the output pixel, so that it focuses on the regions of similar elevations. The usefulness of the depth-aware convolutions relies on the degree to which the assumption that the pixels in proximity with similar elevations should have the same label is true.

Implementation details: Unfortunately, the sole PyTorch implementation of depth-aware convolution that we have found was outdated. Thus, we had to implement such a module from scratch.[4] As we are not pro-efficient with writing C++ extensions for PyTorch we have limited ourselves to an implementation that uses basic PyTorch components only. Especially, we have utilized the PyTorch `unfold` function which turned the images into `im2col` representation that allowed us to optimize the code via vectorization. The scaling was implemented via element-wise tensor multiplication of pixel values and their corresponding depth similarities within a given window. On the other hand, the kernel was applied to the scaled pixel-values via tensor multiplication.

As our implementation is not optimized for CUDA, a ResNet50 that uses depth-aware convolutions is approximately 5 times slower than one that uses standard PyTorch convolutions.

[4] *The custom implementation can be found at:* https://github.com/Marti242/DConv

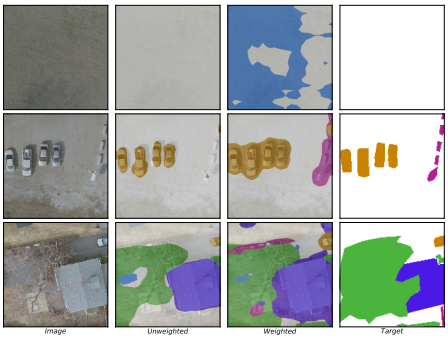

Figure 5. Differences in segmentation among baseline models optimized with weighted and unweighted cross-entropy loss.

Architecture: The paper that introduced the depth-aware CNNs utilized a VGG architecture. Precisely, (Wang & Neumann, 2018) replaced only the first standard convolution within each VGG block with a depth-aware one. As we are using a different architecture we had to make certain adjustments. Especially as ResNet50 has Bottleneck blocks with 2 1x1 convolutions and a 3x3 convolution in between them. For a 1x1 convolution, there is no between pixel depth-similarity to measure. Thus, it only makes sense to replace the middle 3x3 convolution with a depth-aware one. Furthermore, as the 4 layers are distinguished within the ResNet architecture, we have decided to check whether it is better to insert the depth-aware convolution into each Bottleneck in the backbone or to mimic (Wang & Neumann, 2018) and apply depth-aware convolution only in the first Bottleneck block of each ResNet layer. We further explore this topic in section 4.

The feature maps become smaller as they get closer to the output layer of the ResNet. To apply the depth-aware convolution the depth map has to be of the same size as the feature maps. Thus, we were applying average pooling with a 3x3 kernel, a stride of 2 and padding of 1 to the depth map whenever the resolution of the RGB feature map is reduced.

4. Experiments

All experiments have been performed on RTX 2080Ti GPUs, available on the Informatics Research cluster, due to the excessive size of the DeepLabV3 model. The longest experiments took about 13h to finish with only 30 epochs.

4.1. Loss Function

$$loss = \frac{\sum_{i=1}^{C} w_i * (-x_i + log(\sum_j e^{x_j}))}{\sum_{i=1}^{C} w_i} \tag{6}$$

As a loss function, we have decided to use a cross-entropy defined by equation 6, where C stands for the number of classes. However, after analyzing the dataset we were concerned whether the choice of unweighted cross-entropy *(one in which $w_i = 1$ for each class i)* is appropriate for

the dataset at hand in which classes are greatly unbalanced. Thus, we have verified whether a weighted version of the cross-entropy would yield better results. We have trained two baseline models for 30 epochs, but one was optimizing weighted cross-entropy while the other was optimizing its regular unweighted version. The weight for a given class was computed as an inverse of the fraction of pixels in the training set assigned to that class. As a result, the baseline model that optimized weighted cross-entropy attained 80.6% accuracy on the validation set, compared to 85.9% of its unweighted counterpart.

Interestingly, the difference in outputs of the two models is visible. Figure 5 demonstrates examples of such differences. Primarily, the unweighted cross-entropy leads to the prediction of sharper and more realistic class boundaries. The model is more conservative at assigning the pixels to other class than the most common "Ground" label. On the other hand, the model with weighted loss is penalized the least for assigning a wrong class to the "Ground" class, while the reverse is true for missing less frequent classes. Consequently, the model with unweighted loss prefers to assign more than necessary number of pixels to the less frequent classes to avoid costly errors.

This is especially visible in how differently the models classify the class of cars. Initially, we were concerned that the model with unweighted loss will completely neglect the less prevalent "Car" class but it distinguishes cars as accurately as its counterpart. Interestingly, the model with weighted loss tends to make some random predictions that do not make sense like the one on the first image.

After the experiment, we have decided that the unbalanced version of the loss is better for our task as it leads to higher accuracy and more reasonable segmentation. We attribute part of the weighted loss imperfection to the significant difference among the class distributions in the training and validation set. In the remainder of the experiments, we use the unbalanced version of the mIoU for the same reasons why the unbalanced loss function was chosen.

Moreover, the inspection of the predictions has showed us a number of mistakes in the ground truth labels of the images. For example, some trees have been labelled as a car or vice versa. We believe that these errors may prevent any model from acquiring good performance on this dataset.

4.2. Regularization

When have trained our models initially with weight decay of $\lambda = 10^{-4}$, we have realized that although the models demonstrated healthy learning curves on the training set the same cannot be said about the performance on the validation set. This phenomenon is demonstrated in Figure 9. The high variability in the validation performance exemplifies the overfitting of our models. Thus, we have undertaken a parameter search of the weight decay, seen in Table 2. Due to computational constraints, we have utilized solely the best among our models in these experiments.

Model	From scratch	Pre-trained
ESANet	0.228	0.495
Late Fusion	0.269	0.570
RGB-D	0.302	0.504
Early Fusion	0.313	0.576
Baseline	0.328	**0.598**
ASPP Fusion	0.332	0.582
Depth-aware CNN^1	0.348	0.526
Depth-aware CNN^2	**0.348**	0.591

Table 3. Experiment results. The values are mean validation IoU.

Consequently, we have determined that for models incorporating depth a 50 times greater weight decay than the one suggested in (Chen et al., 2017b) was leading to the best results. On the other hand, $\lambda = 10^{-3}$ was leading to the best baseline performance. In the remainder of the experiments, we use $\lambda = 5 * 10^{-3}$ in models that incorporate depth and $\lambda = 10^{-3}$ for the baseline.

Decay	Baseline	ESANet	Depth-aware CNN
$1 * 10^{-4}$	0.586	0.576	0.570
$1 * 10^{-3}$	**0.598**	0.557	0.562
$2 * 10^{-3}$	0.590	0.576	-
$5 * 10^{-3}$	0.590	**0.582**	**0.591**
$1 * 10^{-2}$	-	0.556	0.586

Table 2. Results of the weight decay parameter search. The values are mean validation IoU.

4.3. Final experiment

Finally, we have trained each of the models defined in Section 3 for 30 epochs with the previously specified parameter values. Additionally, for each model, we have assessed its performance both when trained from scratch or pre-trained on ImageNet. The results of the experiment are demonstrated in Table 2 where the mIoU refers to the best mean IoU ever recorded for such a model. Depth-aware CNN^1 refers to a model in which the Depth-aware convolutions are applied in every Bottleneck block of the encoder. Meanwhile, depth-aware CNN^2 mimics the approach from (Wang & Neumann, 2018) by applying the depth-aware convolution only in the first Bottleneck of each ResNet layer.

As we have initially predicted, the simplest RGB-D model is too naive to lead to any performance improvements over the baseline. The quite arbitrary way of adjusting the first layer of the pre-trained model to accommodate the depth channel in reality only adds noise. Surprisingly, the RGB-D model is also worse than the baseline when trained from scratch.

Counterintuitively, when it comes to the fusion-based models, the most sophisticated one of them, ESANet, consistently underperforms. It is not surprising in the pre-trained case as the core RGB-D fusion units must be trained from scratch even when the remainder of the encoder is pre-

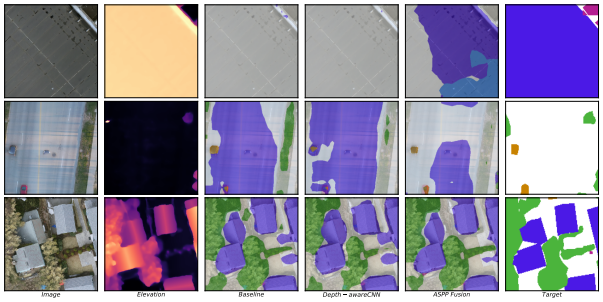

Figure 6. Differences in segmentation between best pre-trained baseline, depth-aware and ASPP fusion models.

trained, which inevitably hinders the performance. The underperformance when training entirely from scratch can probably be attributed to the difference in the classifier that we have used and the one in the original ESANet.

Surprisingly, our ASPP fusion method outperforms its sibling fusion-based models. Notably, one of the factors that may contribute to this phenomenon is the fact that the ASPP fusion model does not utilize the RGB-D Fusion unit. This would explain why ASPP is significantly better than the very similar late fusion model.

The depth-aware CNNs were the most successful among all of the models in incorporating the depth information. This especially highlights the parameter efficiency of this approach as it achieves better results than the fusion models that contain 60% more parameters (Table 4). Albeit, there was not much difference in the performance among the two depth-aware CNN versions when trained from scratch, the two models are visibly distinct when utilized in conjunction with a pre-trained backbone. In our opinion, the architecture from (Wang & Neumann, 2018) is better than its sibling that uses more depth-aware convolutions because it simply is less invasive in its accommodation of depth signal.

Model	number of parameters
Baseline	39,635,016
Depth-aware CNN	39,635,016
RGB-D	39,638,152
Early Fusion	39,639,440
ASPP Fusion	63,464,456
Late Fusion	64,189,704
ESANet	64,538,736

Table 4. The total number of parameters in the model when ResNet50 is used as the encoder and DeepLabV3's head as a classifier.

Unfortunately, we were not able to outperform the pre-trained baseline model. Solely, when we compare the models trained from scratch, the Depth-aware CNNs and our ASPP fusion emerge superior. Notably, the commonality among the best depth using models is that they augment the architecture of the baseline in a less invasive manner. This finding highlights the difference in the nature of the RGB and the depth information that makes the naive fusion of these signals counterproductive. We were able to outperform the baseline when using elevation and training our models from scratch, so multispectral input data can be useful. Nevertheless, the performance boosts acquired via data of non-RGB nature is modest and, in our opinion, does not justify the costs of attaining it. The models adjusted to non-RGB modality input are especially cumbersome in taking advantage of the ready pre-trained components. This further increases the cost of using depth as acquiring a performance level of a pre-trained model requires hundreds of training epochs and substantial computational resources.

5. Related work

Although we have mentioned related work when describing our contributions, we think that there are other recent papers worth mentioning as well. (Gupta et al., 2014) suggests an HHA transformation, or *horizontal disparity, height above ground, and the angle of the local surface normal with the inferred gravity direction*. (Long et al., 2015) compares different ways of merging the HHA features with the RGB features and claims that a *late fusion* approach, where the features are combined at the final layer, yields better performance than merging the features in earlier layers. *GLPNet* (Chen et al., 2021b) shows that, by using two depth fusion modules can give substantial advantage over simple RGB data in NYU-Depth v2 (Nathan Silberman & Fergus, 2012) dataset. (Chen et al., 2021a) has proposed *S-Conv*, a convolution block similar to Depth-aware implemented by us, that attempts to integrate the 3D information from the depth during the convolution. Similarly, (Xing et al., 2020) presents *Malleable 2.5D convolution* that is supposed to learn the receptive field along the depth-axis during convolution. Lastly, the paper by (Wang et al., 2020) shows *CEN*, a similar to ESANet parameter-free multimodal fusion framework that dynamically exchanges channels between sub-networks of different modalities.

6. Conclusions

We present a novel work on the DroneDeploy dataset by incorporating the depth data to a DeepLabV3 baseline model in three distinctive ways: a naive 4^{th} channel implementation, with an ESANet's RGB-D Fusion module and a Depth-aware convolution. We conclude that, despite our initial assumptions, introducing the **elevation does not result in obvious improvements** and can usually lead to worse performance due to the superb optimisation of pure RGB segmentation models and the difficulty in obtaining pre-trained RGB-D models. The benefits of the additional depth information could be simply surpassed by a larger, better quality dataset (Sun et al., 2017). **RGB data typically carries enough information** to properly extract necessary features, hence it limits the usefulness of capturing additional depth maps. In the future we would like to study the relative usefulness of depth in relation to obtaining a larger datasets. Perhaps Elon Musk is right, RGB could be all you need.

References

Arnab, Anurag, Zheng, Shuai, Jayasumana, Sadeep, Romera-Paredes, Bernardino, Larsson, Mans, Kirillov, Alexander, Savchynskyy, Bogdan, Rother, Carsten, Kahl, Fredrik, and Torr, Philip. Conditional random fields meet deep neural networks for semantic segmentation: Combining probabilistic graphical models with deep learning for structured prediction. 2018.

Butcher, Brandon and Smith, Brian J. Feature engineering and selection: A practical approach for predictive models. *The American Statistician*, 74(3):308–309, 2020.

Chen, Liang-Chieh, Papandreou, George, Kokkinos, Iasonas, Murphy, Kevin, and Yuille, Alan L. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs, 2017a.

Chen, Liang-Chieh, Papandreou, George, Schroff, Florian, and Adam, Hartwig. Rethinking atrous convolution for semantic image segmentation, 2017b.

Chen, Liang-Chieh, Zhu, Yukun, Papandreou, George, Schroff, Florian, and Adam, Hartwig. Encoder-decoder with atrous separable convolution for semantic image segmentation, 2018.

Chen, Lin-Zhuo, Lin, Zheng, Wang, Ziqin, Yang, Yong-Liang, and Cheng, Ming-Ming. Spatial information guided convolution for real-time rgbd semantic segmentation. *IEEE Transactions on Image Processing*, 30: 2313–2324, 2021a. ISSN 1941-0042. doi: 10.1109/tip.2021.3049332. URL http://dx.doi.org/10.1109/TIP.2021.3049332.

Chen, Sihan, Zhu, Xinxin, Liu, Wei, He, Xingjian, and Liu, Jing. Global-local propagation network for rgb-d semantic segmentation, 2021b.

Cordts, Marius, Omran, Mohamed, Ramos, Sebastian, Rehfeld, Timo, Enzweiler, Markus, Benenson, Rodrigo, Franke, Uwe, Roth, Stefan, and Schiele, Bernt. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results, 2012.

Geiger, Andreas, Lenz, Philip, Stiller, Christoph, and Urtasun, Raquel. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

Gupta, Saurabh, Girshick, Ross, Arbeláez, Pablo, and Malik, Jitendra. Learning rich features from rgb-d images for object detection and segmentation, 2014.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition, 2015.

Heffels, Michael R. and Vanschoren, Joaquin. Aerial imagery pixel-level segmentation, 2020.

Hu, Jie, Shen, Li, and Sun, Gang. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Krizhevsky, Alex, Nair, Vinod, and Hinton, Geoffrey. Cifar-100 (canadian institute for advanced research). 2009. URL http://www.cs.toronto.edu/~kriz/cifar.html.

Lin, Tsung-Yi, Maire, Michael, Belongie, Serge, Bourdev, Lubomir, Girshick, Ross, Hays, James, Perona, Pietro, Ramanan, Deva, Zitnick, C. Lawrence, and Dollár, Piotr. Microsoft coco: Common objects in context, 2014. URL http://arxiv.org/abs/1405.0312.

Long, Jonathan, Shelhamer, Evan, and Darrell, Trevor. Fully convolutional networks for semantic segmentation, 2015.

Nathan Silberman, Derek Hoiem, Pushmeet Kohli and Fergus, Rob. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

Nicholas Pilkington, Stacey Svetlichnaya and Holmes, Tom. Github - dronedeploy/ddml-segmentation-benchmark: Dronedeploy machine learning segmentation benchmark, 2019. URL https://github.com/dronedeploy/dd-ml-segmentation-benchmark.

Parmar, Vivek, Bhatia, Narayani, Negi, Shubham, and Suri, Manan. Exploration of optimized semantic segmentation architectures for edge-deployment on drones, 2020.

Seichter, Daniel, Köhler, Mona, Lewandowski, Benjamin, Wengefeld, Tim, and Gross, Horst-Michael. Efficient rgb-d semantic segmentation for indoor scene analysis, 2020.

Shermeyer, Jacob, Hogan, Daniel, Brown, Jason, Etten, Adam Van, Weir, Nicholas, Pacifici, Fabio, Haensch, Ronny, Bastidas, Alexei, Soenen, Scott, Bacastow, Todd, and Lewis, Ryan. Spacenet 6: Multi-sensor all weather mapping dataset, 2020.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition, 2015.

Sun, Chen, Shrivastava, Abhinav, Singh, Saurabh, and Gupta, Abhinav. Revisiting unreasonable effectiveness of data in deep learning era, 2017.

Wang, Weiyue and Neumann, Ulrich. Depth-aware CNN for RGB-D segmentation. *CoRR*, abs/1803.06791, 2018. URL http://arxiv.org/abs/1803.06791.

Wang, Yikai, Huang, Wenbing, Sun, Fuchun, Xu, Tingyang, Rong, Yu, and Huang, Junzhou. Deep multimodal fusion by channel exchanging, 2020.

Xing, Yajie, Wang, Jingbo, and Zeng, Gang. Malleable 2.5d convolution: Learning receptive fields along the depth-axis for rgb-d scene parsing, 2020.

A. Additional tables and figures

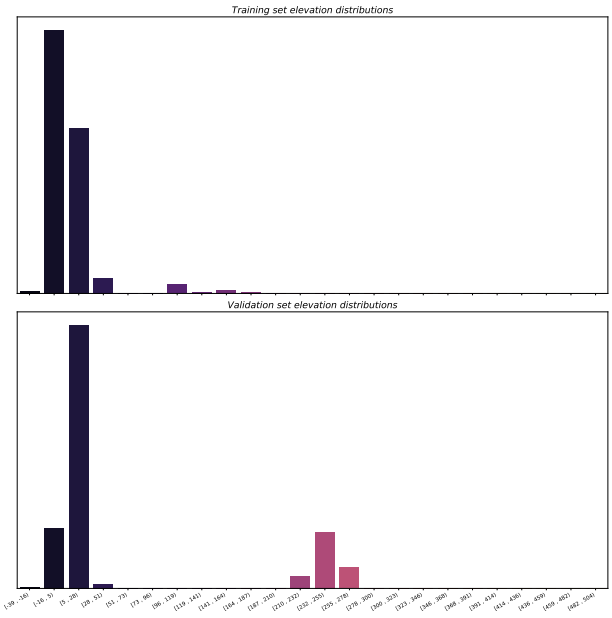

Figure 7. Distribution of elevation in the training and validation sets of the DroneDeploy dataset. Zoom in to see more details.

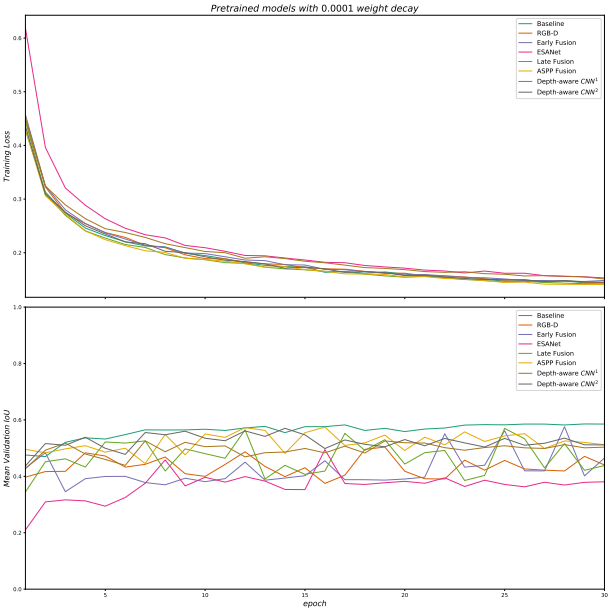

Figure 8. The learning curves of initial pre-trained models. Zoom in to see more details.

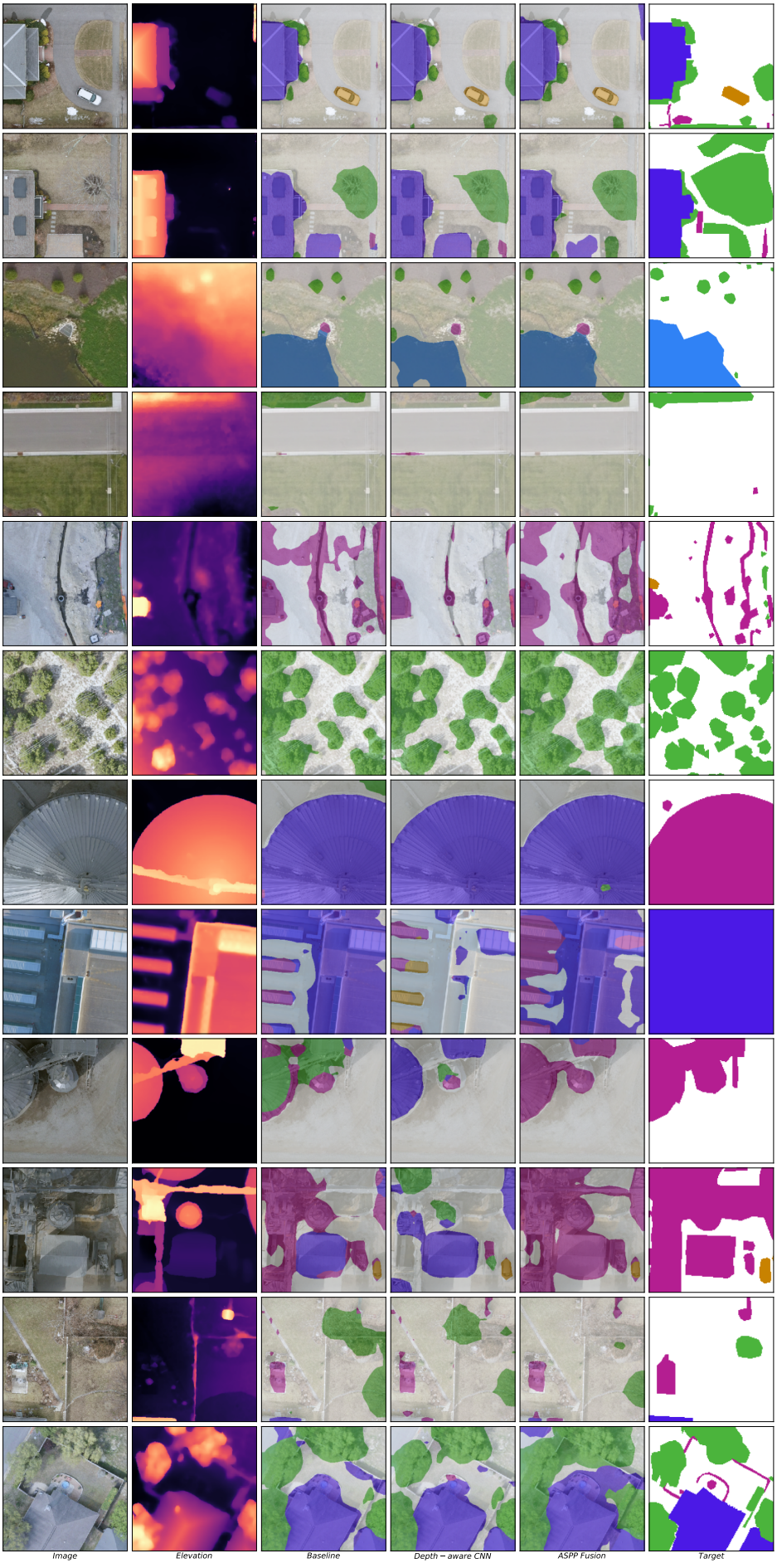

Figure 9. Further differences in segmentation between best pre-trained baseline, depth-aware and ASPP fusion models.