DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Interdisciplinary Project

Bootstrapping Dataset Labeling for Architectural Façade Segmentation using Deep Learning

Dávid Komorowicz

Contents

1 Introduction

City-wide 3D models are available and used in every day life (e.g. Google Maps). In addition to a surface mesh, it would be useful for city planners to know the usage of each building/facade.

The goal of this IDP is to investigate ways to train a neural network that can segment building facades using available labelled and unlabelled data.

The results of this project can later be used for building usage classification (downstream task).

2 Datasets

There are a number of datasets for facade segmentation, however they are rather small. These datasets can be categorized into rectified and non-rectified.

Rectification means that the images are transformed such that parallels in real life are parallel in the images.

1. Rectified
    - ECP
    - Graz
    - CMP
    - Paris Art Deco
2. Non-rectified
    - LabelMeFacade
    - eTrims
    - Varcity
    - ZuBuD (unlabeled)

Table 1 shows the number of images in each dataset in each split. Some datasets don't have separate train/val/test splits, in this case everything is listed in the training column and there are datasets without labels which will be used for bootstrapping.

The images in rectified datasets have more fine grained labels, including things like: balcony, blind, pillar, etc.

On the other hand, non-rectified images have additional information about other objects on the street such as cars, vegetation and the types of roads.

Among these datasets there is a large variety of the granularity of the segmentation classes. If we wanted to combine multiple datasets to increase the overall number of images, we would have to select the lowest common denominator. With the problem statement in mind, the most important classes are windows, doors and balconies, along with walls and roofs. Walls and roofs don't pose a problem because they are already contained in the base model trained on the cityscapes dataset.

	Number of Training Images	Number of Validation Images	Number of Test Images	Unlabeled
ECP	104	-	-	5
Graz	30	20	-	-
CMP	374+228	-	-	-
Paris Art Deco	80	-	-	-
LabelMeFacade	660	189	96	-
eTrims	60	-	-	-
Varcity	113	202	-	113
ZuBuD	-	-	-	1000

Table 1: Number of images in datasets in each split

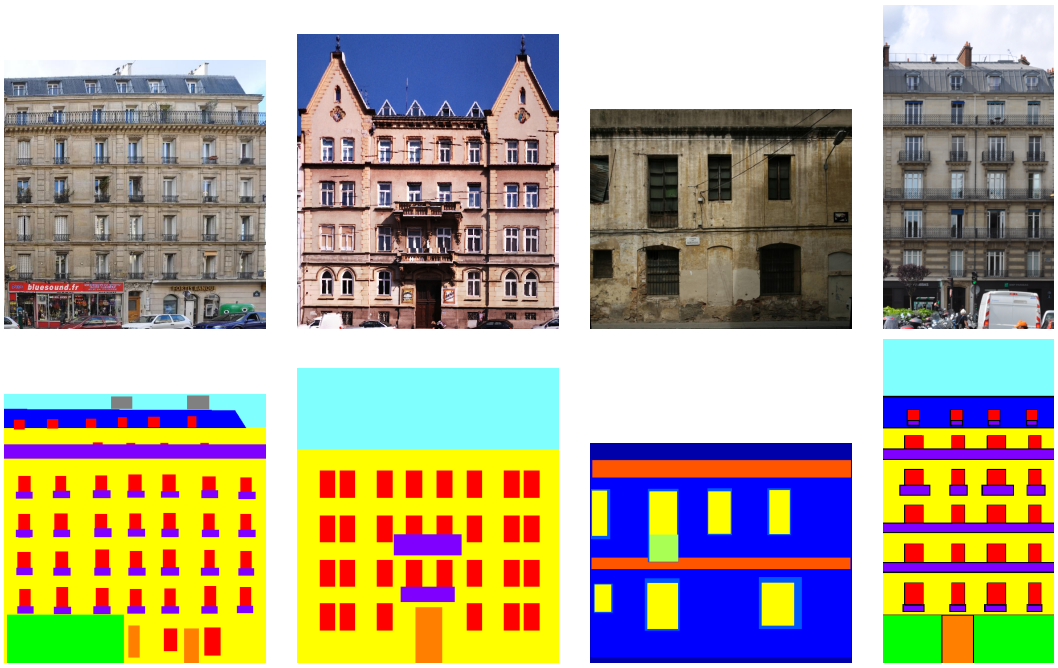

Figure 1: Sample images from rectified datasets: ECP, Graz, CMP, Paris Art Deco

3 Method Description

3.1 Transfer Learning

The number of labelled images is rather low. In these cases transfer learning is often used. The task of facade segmentation is very similar to street view semantic segmentation for self driving cars. More precisely, facade segmentation is a subset of segmentation used for self driving since the labels already contain the facades.

Due to the interest in the topic, large datasets were created and pre-trained models are available.

I've chosen a DeepLab v3+ model with a ResNet101 backbone due to previous experience with the architecture, available high performance checkpoint on cityscapes and easy extend-ability of the Pytorch code.

Figure 3 shows an example image from the LabelMeFacade dataset, output of

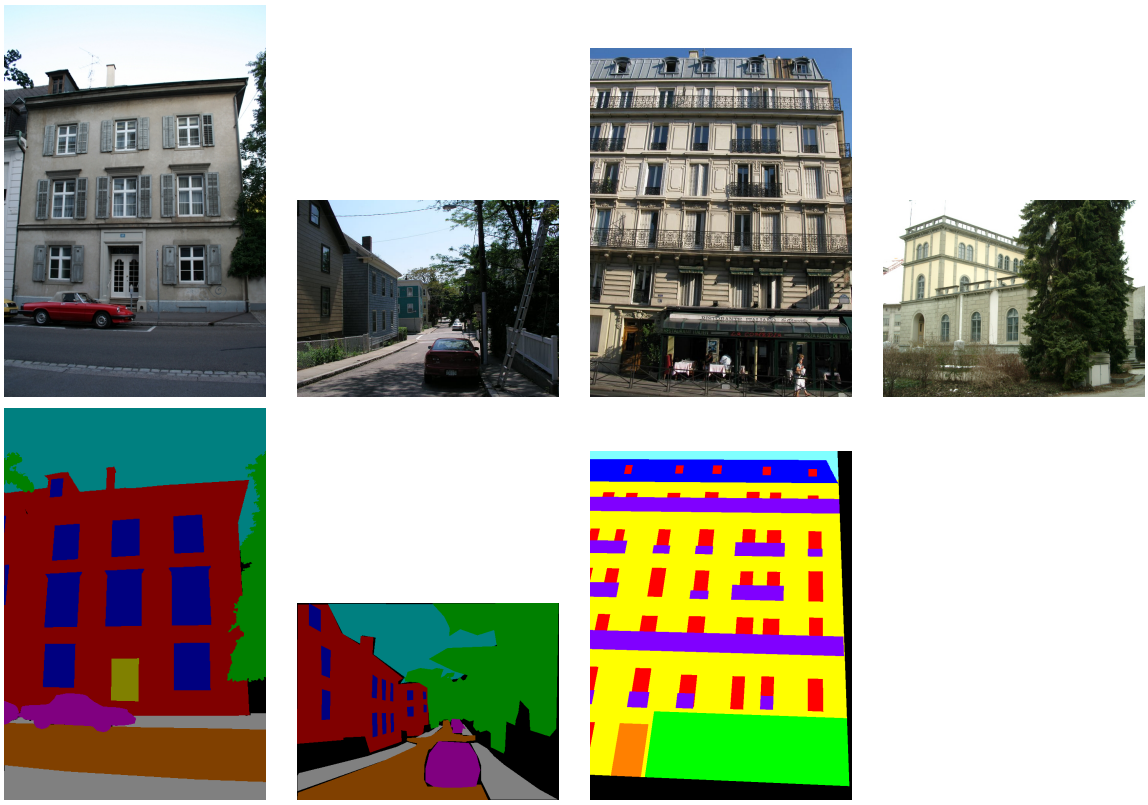

Figure 2: Sample images from non-rectified datasets: LabelMeFacade, eTrims, Varcity, ZuBuD

inference run on the chosen model and the ground truth label for comparison. As expected, the self driving car segmentation is very close to building facade segmentation.

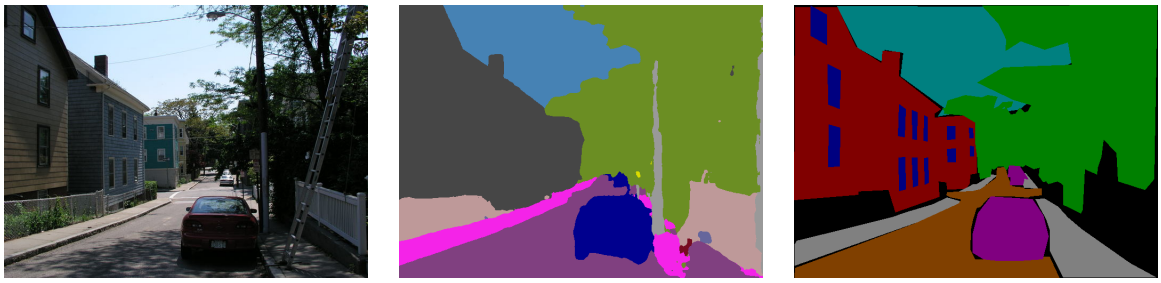

Figure 3: Image from labelme dataset (left), inference using pre-trained model (middle), facade segmentation label (right)

Fine-tuning a semantic segmentation model for a new task requires a new classification head. The size of the classification head corresponds to the number of classes, 19 for cityscapes and 9 for the LabelMeFacade dataset.

First, the model is initialized with the pretrained ckeckpoint, then the whole model is frozen by disabling the gradients, then the classification head is replaced and finally the training starts using the new dataset.

Because most of the weights in this large model are fixed, it is possible to use a

batch size of even 32. In semantic segmentation tasks, training on consumer GPUs, the required batch size is usually 1-2 per GPU. To see whether the largest available batch size achieves the best performance, I run a hyperparameter search along with other parameters such as learning rate. For details refer to subsection 4.2.

3.2 Bootstrapping

Bootstrapping means that, using a limited labelled data, we propagate this information to new unlabelled data assuming that the model will improve if trained on this new, larger dataset. The assumption is that the unlabelled data is similar enough to the training data so that the model can correctly infer the segmentation classes but different enough that including them in subsequent training iterations add new information to the model.

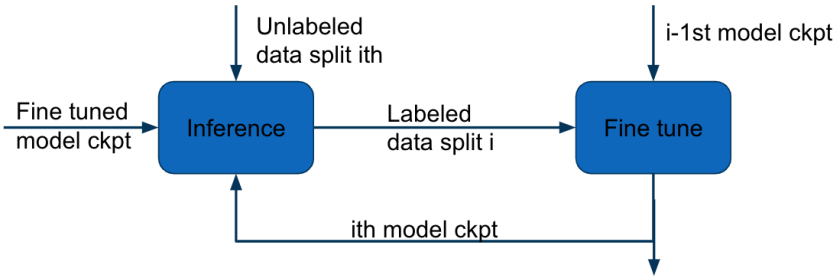

Figure 4: Naive Bootstrapping Loop

Bootstrapping can be realized in several different ways. The simplest model, shown in Figure 4 starts out from the fine-tuned model detailed in subsection 3.1. Given an unlabelled dataset split, labels are inferred and then the split is used for further fine-tuning of the currently used model checkpoint. This loop can be executed multiple times.

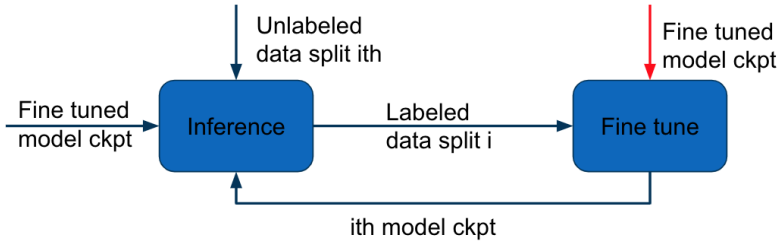

Figure 5: Bootstrapping from the Original Checkpoint

The second variant differs in the way checkpoints are fine tuned. Instead of using the previous checkpoint as a starting point, it always starts fine tuning from the original checkpoint. The first iteration of this method is identical to the first variation but subsequent runs retain the original information in the checkpoint and add information through inference on a new unlabelled data split.

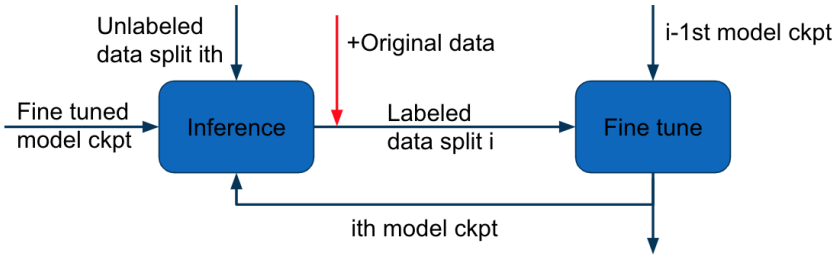

Figure 6: Bootstrapping Loop

The third variant differs from the first one in that, in addition to the inferred data split, the original dataset is concatenated.

The second and third variant aim to retain information present in the ground truth labels and minimize the effects of segmentation class shrinking, as detailed in subsection 4.4.

The second and third variants are independent additions to the first model, therefore they can be combined.

4 Experiments

4.1 Data Augmentation

Another way to improve model robustness beside collecting more training data is to augment the existing dataset.

For images and semantic segmentation there are a wide range of transformations. The transformation has to be performed together on the input image and the label which might require slight modifications, see Figure 7.

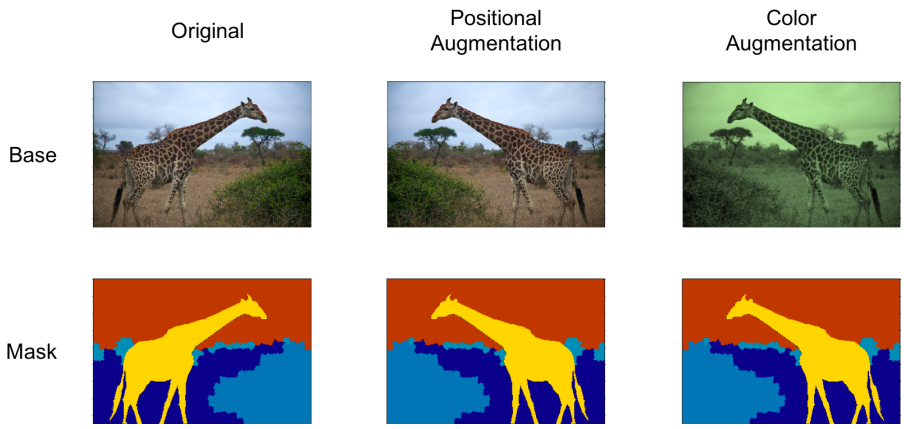

Figure 7: Types of Data Augmentation
Source: `https://divamgupta.com/image-segmentation/2019/06/06/deep-learning-semantic-segmentation-keras.html`

Some transformations are the same for both images: Random Flipping, Random Crop. Some transformations have slight differences: Rotation, Scale. The mask has

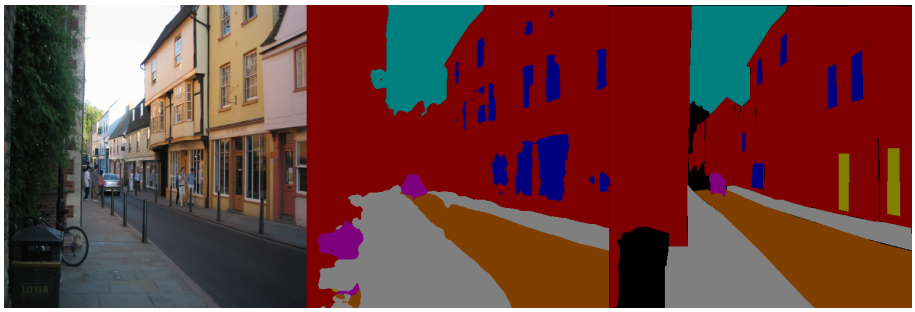

Figure 8: "Overfit" model finds windows not present in labels

to be pixel perfect, therefore the filtering method cannot blend together neighboring pixel categories. In this case Nearest Neighbor filtering is used. In the last category, only the input image is transformed: Color Jitter.

4.2 Hyperparameter Search

After doing a successful fine tuning it is important to find the best hyperparameters. I've run a grid search on the learning rate and batch size using the optuna package. The search space is as follows: 'lr': [1e-6, 1e-5, 1.7e-5, 1e-4, 1e-3], 'batch_size': [2, 4, 8, 16, 32]

It would have been interesting to see the result with a batch size of 1, however it wasn't possible due to the network architecture limitations.

The trained models were evaluated on the validation set for generalizability to avoid overfitting on the training set. The CE loss and a manual evaluation method were used, the latter of which is detailed in subsection 4.3. The hyperparameters and loss metrics for the best models are shown in the following table:

	Cross Entropy Loss	Manual evaluation
learning rate	1e-05	1e-03
batch size	8	32

Table 2: Best Hypterparameters

4.3 Manual Evaluation

The Cross-Entropy loss based on the dataset labels is not necessarily the best evaluation metric. Especially when the labels are missing, but also when the label contains categories unrelated to the downstream task. In our case, the label for windows is often missing and the distinction between vegetation, sky and pavement is not interesting.

This caused the phenomenon (shown in Figure 8) that a trained model found more windows than existing in the label. This shows that the model can generalize but the used metric incorrectly shows it as overfitting. This is a promising finding, suggesting that the model can generalize and correct missing labels and possibly generate additional training data on unseen images.

Therefore I established a manual evaluation metric based on the number of detected windows and doors. On 4 selected validation images I calculated the ground truth windows and doors in each image and each label. The final scores are made up by calculating the number of objects available in the labels and the rest which are not available and taking the ratio out of the number of ground truth objects. Finally these are summed up for all 4 selected validation images. The formula is:

$$\frac{1}{4}\sum_{i=1}^{4}\frac{n_i}{N_i}+\frac{m_i}{M_i}$$

, where i is the index of validation image, n_i is the number of detected windows and door in the prediction present in the validation image, m_i is the number of detected objects not present in the validation image and N_i and M_i are the ground truth for these number respectively. The maximum number for this value is therefore 2.

The advantage of this method is that it directly incorporates the goal of the downstream task. The downside is that false positives are not taken into account, meaning that incorrectly detected windows or weird shaped blobs are not penalized. It also introduces subjectivity regarding the minimum size and shape of an acceptable object, however, this could be automated by correcting labels and setting size thresholds.

4.4 Bootstrapping

In the experiment all possible combinations were run for the following options:

Starting model	Amount of data used	Next checkpoint
ce	all	best
hand	new	last
	last	original

Table 3: Possible bootstrapping configurations

Totalling 12 different configurations with 3 bootstrapping iterations.

Each iteration was run for 5 epochs.

The bootstrapping dataset used is the ZuBuD dataset containing 1000 images 300 for each iteration.

Fine tuning in the bootstrapping iteration takes at most 3 minutes depending on the amount of data in the configuration. However, the inference within the iteration takes a significantly longer time, 7 minutes, making the whole bootstrapping 3x10 minutes long on a GTX 1080Ti.

The model performance is evaluated on the validation set of the fine tuning dataset. All of the runs in the bottom half of the graph, corresponding to the best performing models, use all available training data in each iteration (original training data + all previously labelled data). Therefore these are further investigated in Figure 10.

The bootstrapping runs are clearly divided into 3 groups which correspond to the 3 iterations starting from the bottom up. This means that in each iteration the

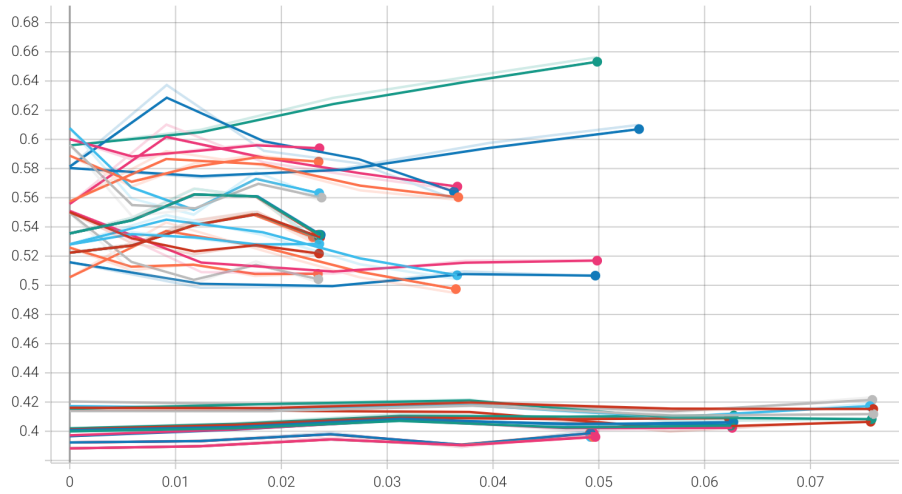

Figure 9: Validation error for all bootstrapping types

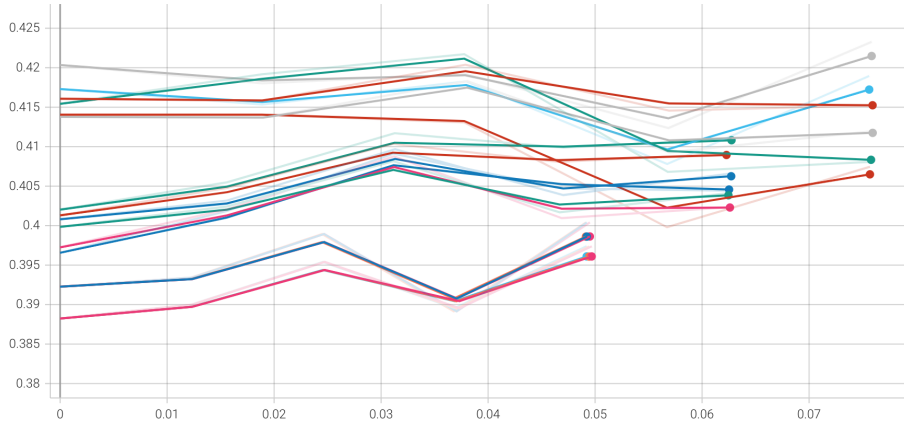

Figure 10: Validation error for the best performing bootstrapping types

model gets worse results on the original dataset but the model is not "forgetting" as fast as the other variants with less data (upper part of Figure 9).

The choice of initial model has little influence (ce vs hand), both perform best according to a different metric.

The last variable determines which checkpoint to use in subsequent iterations. the last one, the best one (early stopping) or always the original (see Figure 5). In order the best one is using the original checkpoint, then the best one and the last checkpoint. This is also expected because the model forgets the least from the fine tuning step.

	Transfer Learning Hand	Transfer Learning CE	hand_original	hand_last	hand_best	ce_original	ce_last	ce_best
eTrims	0.483	0.481	0.543	0.590	0.544	0.553	**0.591**	0.55
labelmefacade	0.450	0.429	0.454	**0.483**	0.465	0.447	0.469	0.450

Table 4: Evaluation on test set

5 Evaluation

The best 6 bootstrapped models are evaluated and compared to the 2 best models from transfer learning.

Quantitatively, the models are evaluated on the whole eTrims dataset and the labelmefacade test set.

The best bootstrapping models are CE_last and hand_last respectively. And both of them are better than the initial models after transfer learning.

Qualitatively the 6 best performing models are evaluated on the labelmefacade validation set Figure 11, the labelmefacade test set Figure 12 and eTrims dataset Figure 13 as described above.

6 Conclusion

During this Interdisciplinary Project I investigated the possibility to improve semantic segmentation models for facade segmentation using unlabeled datasets.

These results show that bootstrapping can indeed learn new information from the unlabeled dataset and generalize to unseen data.

The extent of this generalization is not the outside of the scope of this work but additional steps might be necessary to be competitive to large scale labeled training data.

7 Future Work

Here is a list of potential directions to further improve the model performance.

Large portion of the images contain lower granularity labels such as buildings, road as opposed to windows and doors that are most important for the downstream task. This class imbalance could be remedied using a method such as the Focal loss.

During transfer learning and bootstrapping, only the classification head was able learn. To increase the model capacity the top convolution layers might need to be unfrozen.

To further alleviate the model degradation around windows and doors during the bootstrapping iterations. It might be necessary to add a post-processing step where the shape of the detections are better matched to the image, making them follow the edges (rectengular, oval).

Figure 11: Qualitative results on validation images

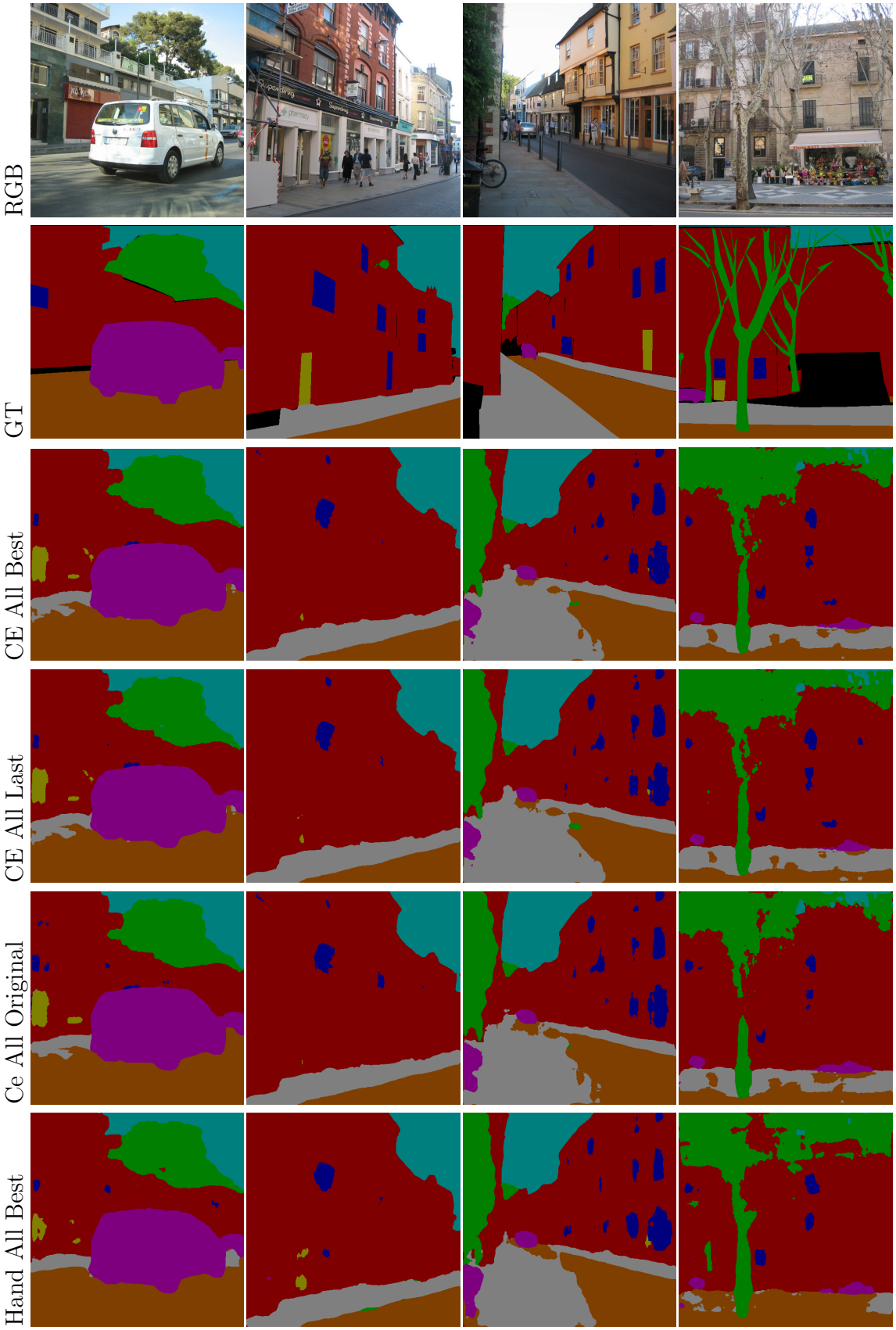

Figure 11: Qualitative results on validation images

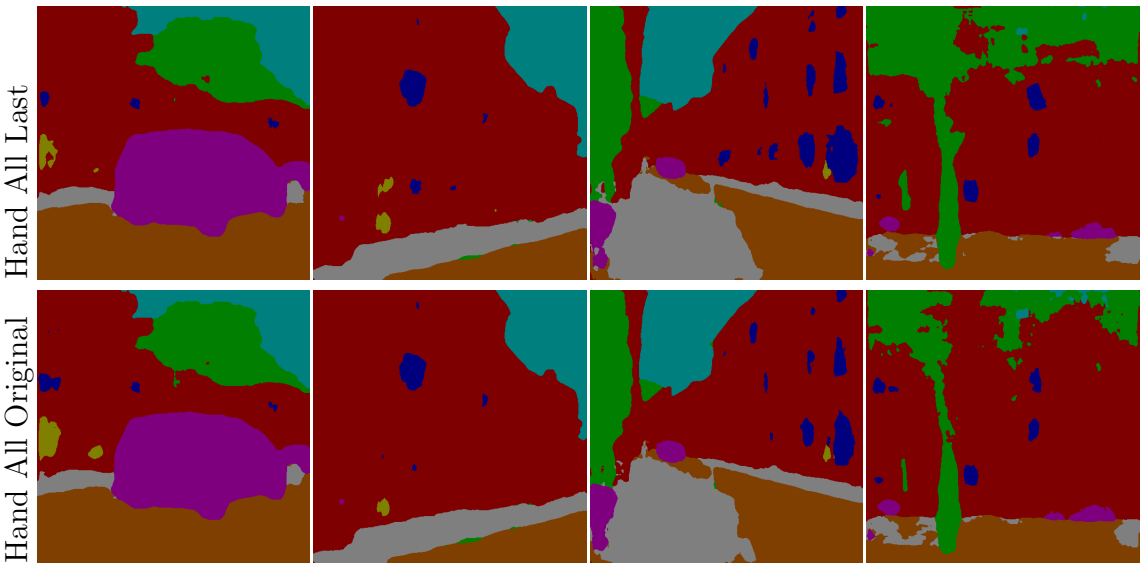

Figure 12: Qualitative results on test images labelmefacade

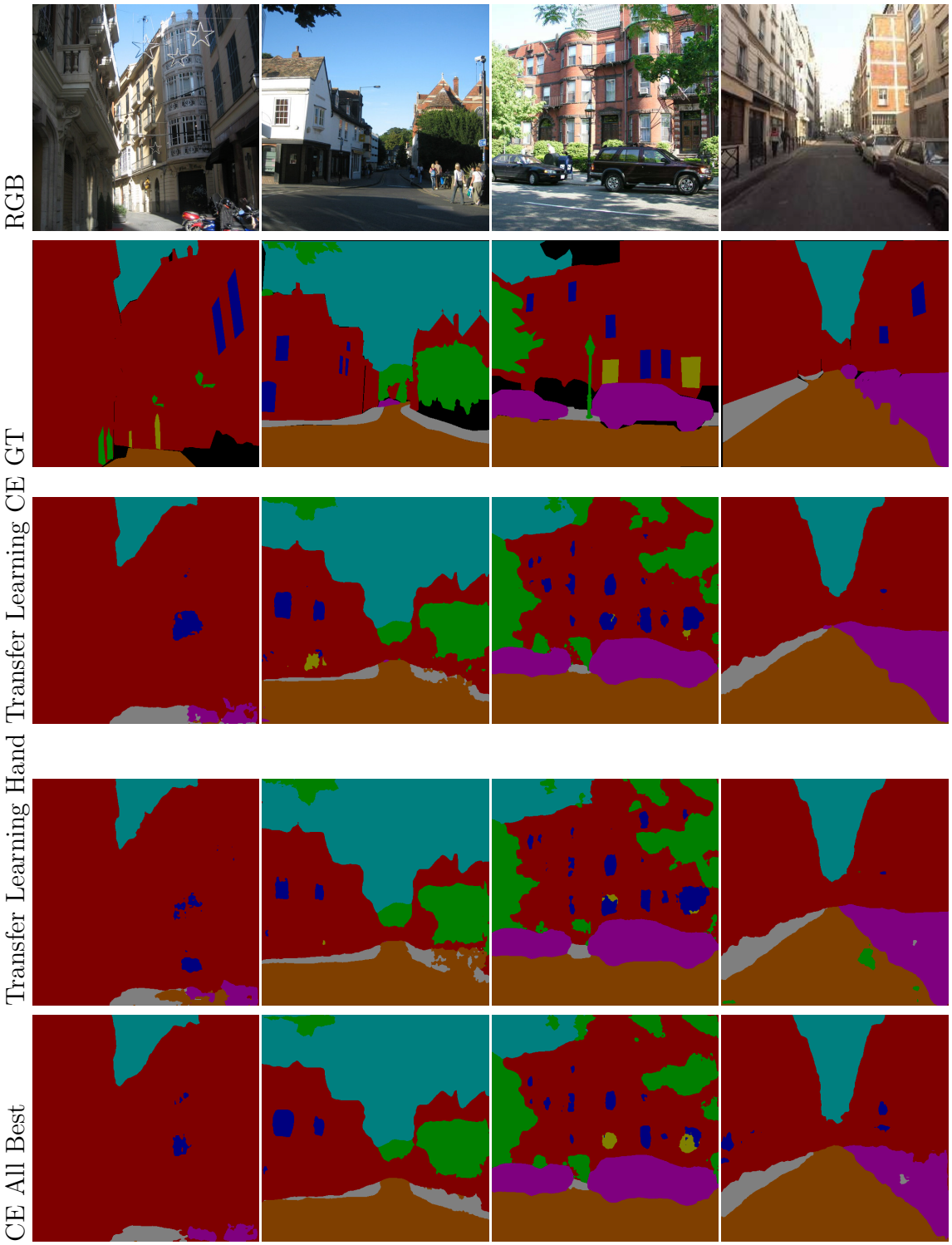

Figure 12: Qualitative results on test images labelmefacade

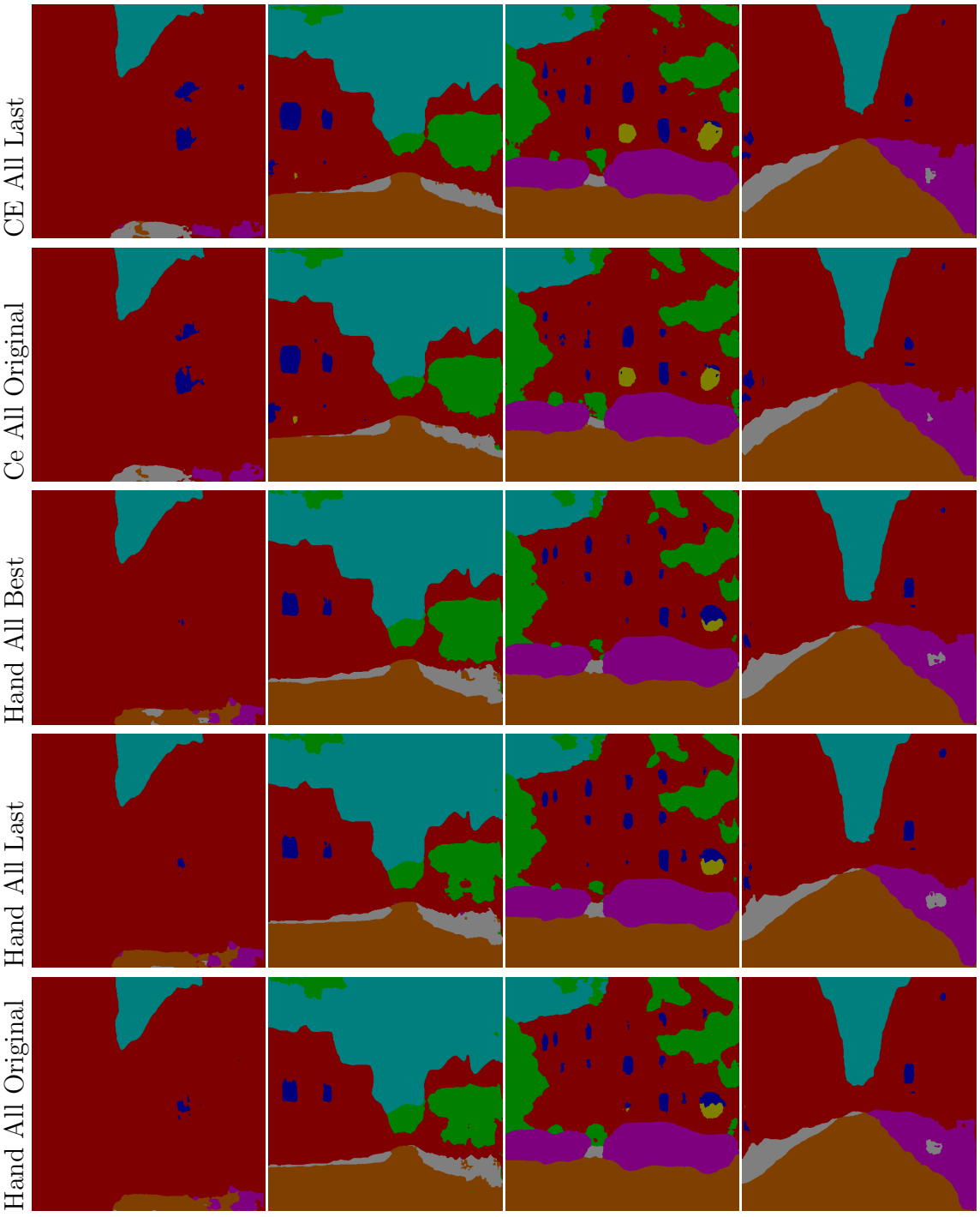

Figure 13: Qualitative results eTrims dataset

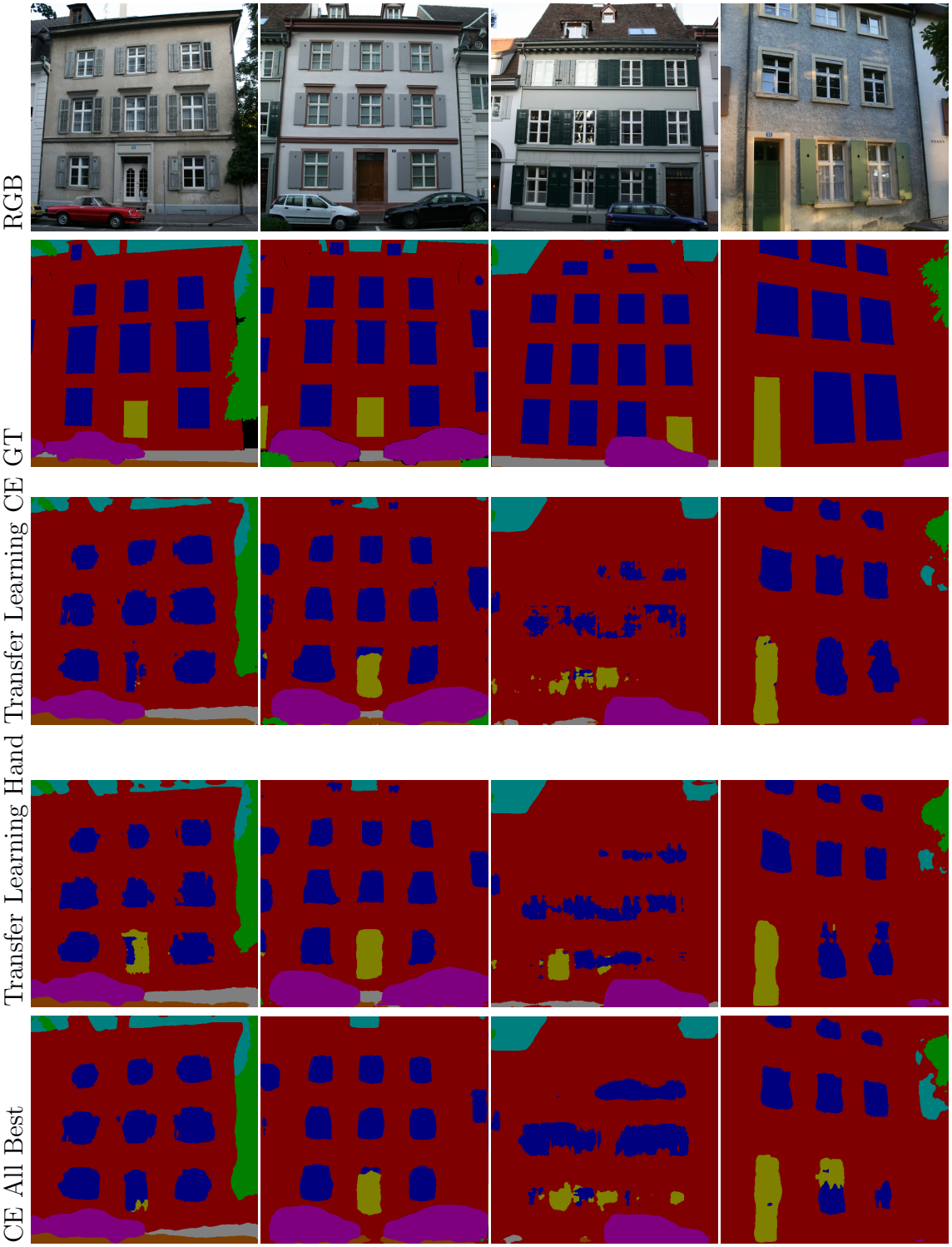

Figure 13: Qualitative results eTrims dataset