Image Segmentation algorithms based on "transformer" neural networks

Ioannis Kaziales - csd4267

Thesis submitted in partial fulfillment of the requirements for the

Bachelor's of Science degree in Computer Science and Engineering

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Assistant Professor *Nikos Komodakis*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

University of Crete
Computer Science Department

Image Segmentation algorithms based on "transformer" neural networks

Thesis submitted by
Ioannis Kaziales
in partial fulfillment of the requirements for the
Bachelor of Science degree in Computer Science

THESIS APPROVAL

Author: Ioannis Kaziales

Committee approvals:

Nikos Komodakis
Assistant Professor, Thesis Supervisor

Yannis Stylianou
Professor, Committee Member

Departmental approval:

Antonios Argyros
Professor, Director of Graduate Studies

Heraklion, October 2022

IMAGE SEGMENTATION ALGORITHMS BASED ON "TRANSFORMER" NEURAL NETWORKS

Abstract

Semantic segmentation of images is a foundational task in the field of computer vision, but classifying each pixel is an obscure and ambiguous task. In this work we introduce MeanShifter, a transformer based model for semantic image segmentation which extends the recent Segmenter model architecture. Our approach is inspired (as the name suggests) by mean shift [3], an older pattern recognition algorithm used for clustering, which assigns each datapoint to the local mode of the density distribution iteratively by shifting points towards the direction of the highest density. Specifically, we introduce the *Mean Shift (MS)* Block, which is a simple, easy to interpret and computationally efficient way to group together semantically similar data for easier classification. MeanShifter achieves competitive results for semantic segmentation on ADE20K.

Keywords: computer vision, semantic image segmentation, deep learning, encoder-decoder model, vision transformer, segmenter, self-attention, mean shift clustering

1 Introduction

Semantic image segmentation is a fundamental topic in computer vision, with the goal to assign a semantic label to each pixel of an image based on the category of the object depicted in the pixel. The result of this process is a segmentation mask overlaying the image, which provides a high-level representation of the image by identifying and separating the classes of the depicted objects. This representation is simpler, more meaningful and makes further analysis easier. The applications of image segmentation are numerous and include medical diagnosis, scene understanding, autonomous driving and augmented reality. Challenges such as object occlusions, deformations, background clutter and high degree of intra-class variations are among the reasons why image segmentation is still considered a difficult (and often ambiguous) task.

A wide variety of algorithms and neural network architectures have been proposed for image segmentation over the time. Up until recently, models based on convolutions were considered the norm in various tasks in computer vision, including image segmentation, because convolutions can capture useful spatial information. Even though individual convolutions capture local spatial information by nature and their receptive field is small, by stacking many of them, their receptive field gradually increases and they can capture more global and semantically meaningful information. Encoder-decoder models based on convolutions were a common occurrence in image segmentation. In these models, an encoder would try to extract and compress the low-level semantic information of the input image into a latent space representation. Based on that representation, the decoder would try to upsample the underlying information and make pixel-level predictions. Fully Convolutional Networks (FCN) were another popular approach. These architectures use solely locally connected layers, like convolutions, pooling and upsampling layers, removing dense layers completely. Ever since their introduction in [6], transformer neural networks have been used extensively and have made a great impact in the field of natural language processing (NLP), achieving state of the art results in tasks that deal with sequences of input, output, or both. Transformers are encoder-decoder models based exclusively on self-attention mechanisms, eliminating recurrence and convolutions entirely. Taking inspiration from the model's tremendous success, computer vision researchers have recently started employing transformer-like architectures for tasks such as image classification and segmentation and the results are very promising. Initial attempts used some form of the self-attention mechanism in conjunction with convolutions. More recent approaches use architectures that rely solely on self-attention and remove convolutions completely, staying more faithful to the transformer model architecture. The advantage of transformers is that they lack the inductive bias that convolutional networks suffer from, due to the local nature of convolutions. Therefore, they can utilize global interactions and useful contextual information at every stage of the model. Additionally, due to their parallelism, transformers enable faster and easier training on large datasets. Unfortunately, treating a large image as a sequence of raw pixels is infeasible, because calculating the self-attention between the sequence elements has quadratic complexity. To cope with that, model architectures such as Vision Transformer (ViT) [2], Data-efficient image Transformer (DeiT) [5] and Segmenter [4] split the image to a sequence of small 2D patches, reformulate the tasks of image classification and segmentation as sequence tasks and apply a variation of the transformer model. These models have attained impressive performance gain over multiple benchmarks compared to modern ConvNets.

Following the recent work on Segmenter [4], we reshape the image into a sequence of patches, linearly project them to patch embeddings and use those embeddings as input tokens for the transformer encoder. The contextualized sequence of semantically meaningful encodings produced by the encoder is then shifted using a mean shift block, in order to cluster related encodings together and make decoding easier. Those encodings are finally upsampled by a simple linear decoder to per-pixel class scores. MeanShifter attains competitive results while remaining simple, flexible, and fast.

2 Our approach: Meanshifter

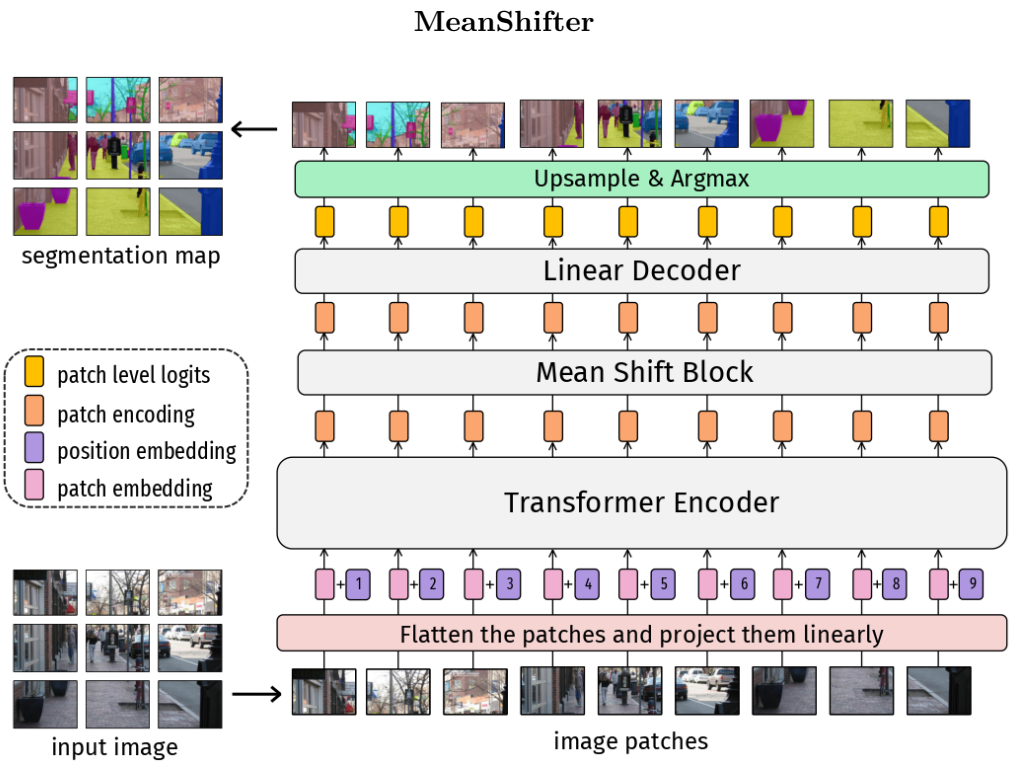

Fig. 1: MeanShifter model overview. The image is split into patches, which are projected to a sequence of embeddings. Position encoding is added to the embeddings and then they are encoded with a transformer encoder. The resulting encodings pass through the mean shift layers, which try to group similar encodings together. Finally, the linear decoder takes as input the encodings and predicts the segmentation mask.

MeanShifter is a convolution-free encoder-decoder architecture based on transformer neural networks, that maps a sequence of patch embeddings to pixel-level class annotations. An overview of the model is depicted in Figure 1. To the best of our ability, we adhere to the

model architecture of Segmenter [4] with a linear decoder and the addition of the mean shift block between the transformer encoder and the decoder.

We reshape the given image $\mathrm{x} \in \mathbb{R}^{H \times W \times C}$ into a sequence of 2D patches $\mathbf{x_p} = [x_1, x_2, \ldots, x_N] \in \mathbb{R}^{N \times (P^2 \times C)}$, where (H, W) is the resolution of the image, (P, P) is the patch size and C is the number of the image's channels. The number of the resulting patches (length of input sequence for the Transformer encoder) is $N = HW/P^2$. Since transformers use a constant latent vector size D, each patch is flattened and linearly mapped to D dimensions. This way, the sequence of patches becomes a sequence of patch embeddings $\mathbf{x_0} = [x_1\mathbf{E}, x_2\mathbf{E}, \ldots, x_N\mathbf{E}] \in \mathbb{R}^{N \times D}$, where $\mathbf{E} \in \mathbb{R}^{(P^2 \times C) \times D}$. Because the transformer operates on the input embeddings concurrently (in parallel), their positional information is neglected. Thus, to make use of the order of the sequence, we add extra learnable positional embeddings $\mathbf{E}_{pos} = [pos_1, pos_2, \ldots, pos_N] \in \mathbb{R}^{N \times D}$ to the patch embeddings. The resulting embeddings $\mathbf{z}_0 = \mathbf{x_p}\mathbf{E} + \mathbf{E_{pos}} = [x_1\mathbf{E} + pos_1, x_2\mathbf{E} + pos_2, \ldots, x_N\mathbf{E} + pos_N]$ are considered the input sequence of tokens.

We pass the sequence of tokens $\mathbf{z}_0$ through a transformer [6] encoder consisted of L layers. The encoder maps the input sequence $\mathbf{z}_0 = \left[\mathrm{z}_1^0, \mathrm{z}_2^0, \ldots, \mathrm{z}_N^0\right]$ of patch embeddings to $\mathbf{z}_L = \left[\mathrm{z}_1^L, \mathrm{z}_2^L, \ldots, \mathrm{z}_N^L\right]$, a contextualized encoding sequence containing rich semantic information. Each transformer layer contains a multiheaded self-attention (MSA) block, succeeded by a Multilayer perceptron (MLP) block composed of two layers with a GELU non-linearity. Layer normalization (LN) is applied before every block of the transformer layer and residual connections are added after each block. Each transformer layer ℓ relies on the results of the previous layers and generates a sequence of contextualized encodings $\mathbf{z}_\ell \in \mathbb{R}^{N \times D}$ as follows:

$$\begin{aligned} \mathbf{z'}_\ell &= \mathrm{MSA}(\mathrm{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, && \ell = 1, \ldots, L \\ \mathbf{z}_\ell &= \mathrm{MLP}(\mathrm{LN}(\mathbf{z'}_\ell)) + \mathbf{z'}_\ell, && \ell = 1, \ldots, L \end{aligned}$$

The self-attention mechanism contains three learnable point-wise linear layers, which are applied to the tokens and produce three intermediate representations: *queries* $\boldsymbol{Q} \in \mathbb{R}^{N \times d}$, *keys* $\boldsymbol{K} \in \mathbb{R}^{N \times d}$ and *values* $\boldsymbol{V} \in \mathbb{R}^{N \times d}$. Essentially, self-attention (SA) for a token i is computed as the weighted sum over the value representations of all tokens in the sequence. The weight W_{ij} between tokens i, j is based on the dot-product similarity of the query $\boldsymbol{Q}_i$ and the key $\boldsymbol{K}_j$ representations, scaled by $\sqrt{d}$ to achieve more stable gradients. Multihead self-attention (MSA) with k *heads* is an extension of self-attention, in which k self-attention operations are computed in parallel (with different $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ representations per head, since they try to capture different interactions between the tokens). The head size, d, of each SA block is set to $d = D/k$, in order to keep the number of parameters constant when modifying the numbers of heads, k. The outputs of the SA blocks are concatenated and then linearly projected to $\mathbb{R}^{N \times D}$.

$$
\begin{aligned}
[\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}] &= \mathbf{z}\boldsymbol{E}_{qkv}, & \boldsymbol{E}_{qkv} &\in \mathbb{R}^{D\times 3d} \\
W &= \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{\text{d}}}\right), & W &\in \mathbb{R}^{N\times N} \\
\text{SA}(\mathbf{z}) &= W \cdot \boldsymbol{V} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{\text{d}}}\right)\boldsymbol{V} & & \\
\text{MSA}(\mathbf{z}) &= [\text{SA}_1(\mathbf{z}), \text{SA}_2(\mathbf{z}), \ldots, \text{SA}_k(\mathbf{z})] \cdot \boldsymbol{E}_{msa}, & \boldsymbol{E}_{msa} &\in \mathbb{R}^{k\cdot d\times D}
\end{aligned}
$$

To facilitate the patch-level classification of the resulting contextualized encoding sequence $\mathbf{z}_L$ by the decoder, we would like to group semantically similar encodings together. To achieve that, we pass the encoding sequence $\mathbf{z}_L$ through the Mean Shift (MS) block. The Mean Shift block is similar to the self-attention mechanism, albeit simpler in structure. It consists of L_{ms} layers and includes a learnable point-wise linear layer with shared weights across all layers of the block, which is applied to the patch encodings and produces the intermediate representation $\boldsymbol{R} \in \mathbb{R}^{N\times d_{ms}}$. $\boldsymbol{R}$ takes the role of both the $\boldsymbol{K}$ and $\boldsymbol{Q}$ representations of the SA blocks. The weight S_{ij} between encodings i, j is based on the dot-product similarity of $\boldsymbol{R}_i$ and $\boldsymbol{R}_j$ representations, scaled by $\sqrt{d_{ms}}$ to achieve more stable gradients. The weight matrix S is symmetric ($S = S^\top$), therefore the similarity between encodings i, j is the same as the similarity between encodings j, i ($S_{ij} = S_{ji}$). After ℓ layers of the mean shift block, the original patch encodings $\mathbf{z}_L^{(0)}$ have been shifted ℓ times towards the local modes of the encodings density distribution, thus becoming $\mathbf{z}_L^{(\ell)}$.

$$
\begin{aligned}
\boldsymbol{R} &= \mathbf{z}_L^{(\ell-1)}\boldsymbol{E}_r, & \boldsymbol{E}_r &\in \mathbb{R}^{D\times d_{ms}} \\
S &= \text{softmax}\left(\frac{\boldsymbol{R}\boldsymbol{R}^\top}{\sqrt{\text{d}_\text{ms}}}\right), & S &\in \mathbb{R}^{N\times N} \\
\mathbf{z}_L^{(\ell)} &= \text{MS}\left(\mathbf{z}_\text{L}^{(\ell-1)}\right) = S \cdot \mathbf{z}_L^{(\ell-1)} = \text{softmax}\left(\frac{\boldsymbol{R}\boldsymbol{R}^\top}{\sqrt{\text{d}_\text{ms}}}\right)\mathbf{z}_L^{(\ell-1)} & &
\end{aligned}
$$

As a decoder, we apply a point-wise linear layer to the shifted patch encodings $\mathbf{z}_L^{(L_{ms})} \in \mathbb{R}^{N\times D}$ to construct patch-level class logits $\mathbf{z}_{lin} = \mathbf{z}_L^{(L_{ms})}\boldsymbol{E}_{dec}$, where $\boldsymbol{E}_{dec} \in \mathbb{R}^{D\times K}$, $\mathbf{z}_{lin} \in \mathbb{R}^{N\times K}$ and K is the number of classes. The sequence $\mathbf{z}_{lin}$ is reshaped into a 2D feature map $\mathbf{s}_{lin} \in \mathbb{R}^{H/P\times W/P\times K}$ and upsampled by bilinear interpolation to the original image resolution $\boldsymbol{s} \in \mathbb{R}^{H\times W\times K}$. The final segmentation mask $\boldsymbol{s} \in \mathbb{R}^{H\times W}$ is generated by applying softmax to the result along the class dimension.

3 Experimental Results

3.1 Setup and Implementation Details

Hardware & Software Configuration
All collected results and experiments were performed using a single NVIDIA GeForce RTX 3060 GPU with cuda 11. We used Python 3.7.6 and Pytorch 1.7.0. For the implementation of the MeanShifter model, we relied on MMSegmentation [1] v0.29, which is an open source semantic segmentation toolbox based on PyTorch and a part of the OpenMMLab project. More specifically, in order to implement MeanShifter, we combined the ViT backbone already used in the segmenter model along with our newly implemented decode head. Our code and models are available at `https://github.com/JohnnyKaz/mmsegmentation`. We also implemented our model by modifying the official Segmenter github repository at `https://github.com/JohnnyKaz/meanshifter`, but we rely on the MMSegmentation implementation for our experiments.

Datasets & Metrics
We evaluated the proposed models on the ADE20K semantic segmentation dataset [7], which contains 150 semantic classes. The training set consists of 20,210 scene-centric images annotated with pixel-level object labels, while the validation and test sets include 2,000 and 3,352 images respectively. We report the percent of pixels that are classified correctly (all pixel accuracy - aAcc), the mean accuracy per class (mAcc) and the mean Intersection over Union (mIoU) over all the classes (also known as the Jaccard index).

Model Configurations
MeanShifter configurations are based on those used for Segmenter [4]. For the transformer encoder, we rely on the "Tiny" variant of the Vision Transformer (ViT) [2], due to hardware constraints (limited computational resources). In particular, we use input patch size $P \times P = 16 \times 16$, $L = 12$ layers and token size $D = 192$. Since the head size of a multiheaded self-attention (MSA) block is fixed to $d = 64$, the number of self-attention heads is given as $k = D/d = 192/64 = 3$. Also, the size of the hidden layer of the encoder's MLP block is set to $4 \times D = 768$.

Training
We follow the same pre-training, data augmentation and training protocols as the ones proposed in the Segmenter architecture [4] and refer the interested readers to [4] for a detailed description. In short, the MeanShifter models are pretrained on ImageNet-21k and adhere to the standard pipeline of the MMSegmentation toolbox [1]. We use pixel-wise cross-entropy loss, "poly" learning rate policy and stochastic gradient descent (SGD) optimizer. We train each model for 160K iterations. During training, images must have a fixed size of 512×512, so smaller images are padded, while larger images are randomly cropped to match this size. During inference we use a sliding window with resolution 512×512, to be able to treat images of varying sizes.

3.2 Quantitative Results

In this section, we investigate the behaviour and performance of different variants of MeanShifter on the ADE20K validation dataset and compare our approach with the Segmenter model.

As a first experiment, we wanted to evaluate the behaviour of our approach when trained with different numbers of mean shift layers L_{ms} and channels d_{ms} for the mean shift block. The exhaustive results are documented in Table 1. We observe that the more mean shift layers we add to our approach, the worse it performs. The performance of the model when $L_{ms} = 1$ is adequate and comparable to that of the Segmenter (even if it is slightly lower). However, even with 2 layers, our model's performance can no longer compete with the results of the Segmenter. As for the mean shift channel size, d_{ms}, we observe that small or large values perform worse than values closer to the dimension of the input encodings (D).

To further study the impact of the number of mean shift layers, we train MeanShifter with a specific number of layers, L_{ms}, but at inference time we use a different number of layers, L'_{ms}. Table 2 describes the results extensively. We are led to believe that using a small number of layers works best. On many occasions, we can even observe that using fewer layers than the ones the model was trained with produces better results. On the other hand, using more layers at inference tends to reduce the model accuracy of the model.

Model	L_{ms}	d_{ms}	aAcc	mAcc	mIoU	Params
Segmenter-Ti/16*	-	-	75.77	41.13	31.44	5.71 M
MeanShifter-Ti/16	1	64	75.19	38.72	29.49	5.72 M
MeanShifter-Ti/16*	1	128	75.2	38.91	29.8	5.74 M
MeanShifter-Ti/16*	1	192	75.89	40.96	31.05	5.75 M
MeanShifter-Ti/16	1	256	74.77	37.15	28.46	5.76 M
MeanShifter-Ti/16	1	768	74.5	38.06	28.65	5.86 M
MeanShifter-Ti/16	2	64	65.48	18.2	13.0	5.72 M
MeanShifter-Ti/16	2	128	70.09	26.16	19.31	5.74 M
MeanShifter-Ti/16	2	192	69.7	26.39	19.37	5.75 M
MeanShifter-Ti/16	2	256	72.49	31.67	23.83	5.76 M
MeanShifter-Ti/16	2	768	70.86	29.23	21.47	5.86 M
MeanShifter-Ti/16	3	64	45.27	4.64	2.86	5.72 M
MeanShifter-Ti/16	3	128	66.84	21.53	15.93	5.74 M
MeanShifter-Ti/16	3	192	37.98	6.53	4.07	5.75 M
MeanShifter-Ti/16	3	256	71.91	32.4	23.5	5.76 M
MeanShifter-Ti/16	3	768	59.72	16.66	11.32	5.86 M

Tab. 1: Performance comparison of Segmenter (as a reference) and different MeanShifter models trained with varying numbers of mean shift layers L_{ms} and channels d_{ms} on the ADE20K validation set. Models with * are derived as median over 5 runs.

3.3 Qualitative Results

In this section, we present qualitative results in Figures 2, 3, 4. In figure 2 we present some cases where MeanShifter manages to group semantically similar objects together and produces better segmentation maps than Segmenter. In figure 3, we present some cases where MeanShifter struggles and produces worse segmentation maps than Segmenter. MeanShifter can sometimes be more prone to cluster different objects with similar appearance to the same class, due to the shift of the encodings (e.g. it categorizes a statue as an armchair and a pillow as a bed). Another thing to note is that in some cases Segmenter handles boundaries between different objects better (e.g. in the case of the tombstones). Both methods struggle to segment small (or thin) object instances, such as tree branches, lamps and people. In figure 4, we have trained a MeanShifter using 1 mean shift layer, but use a different number of layers during inference. As we can see, the more layers we use, the more objects are clustered together. This can pose a problem, since unrelated objects are more likely to be grouped together, thus increasing the segmentation error (in the extreme case, all pixels of the image would be labeled as a single class). On the other hand, on some cases adding more layers can improve the prediction (e.g. in the penultimate image the 'green' class that was incorrectly predicted for $L'_{ms} = 1$ is almost completely eliminated by adding more layers and in the last image the boundaries of the images generated with more layers are more faithful to the ground truth).

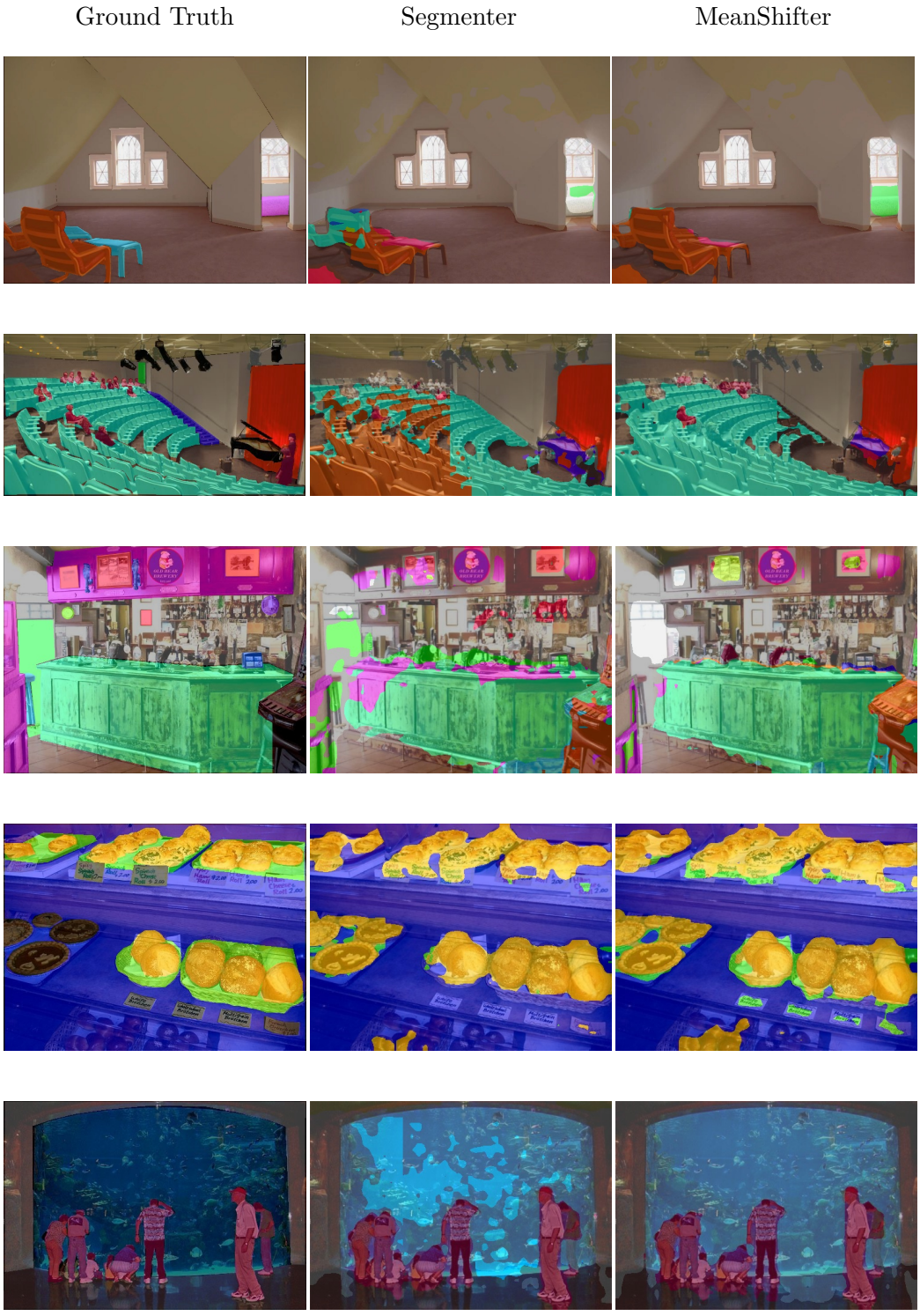

Fig. 2: Images where MeanShifter (trained with 1 MS layer and MS dimension 192) produces more coherent segmentation maps than Segmenter

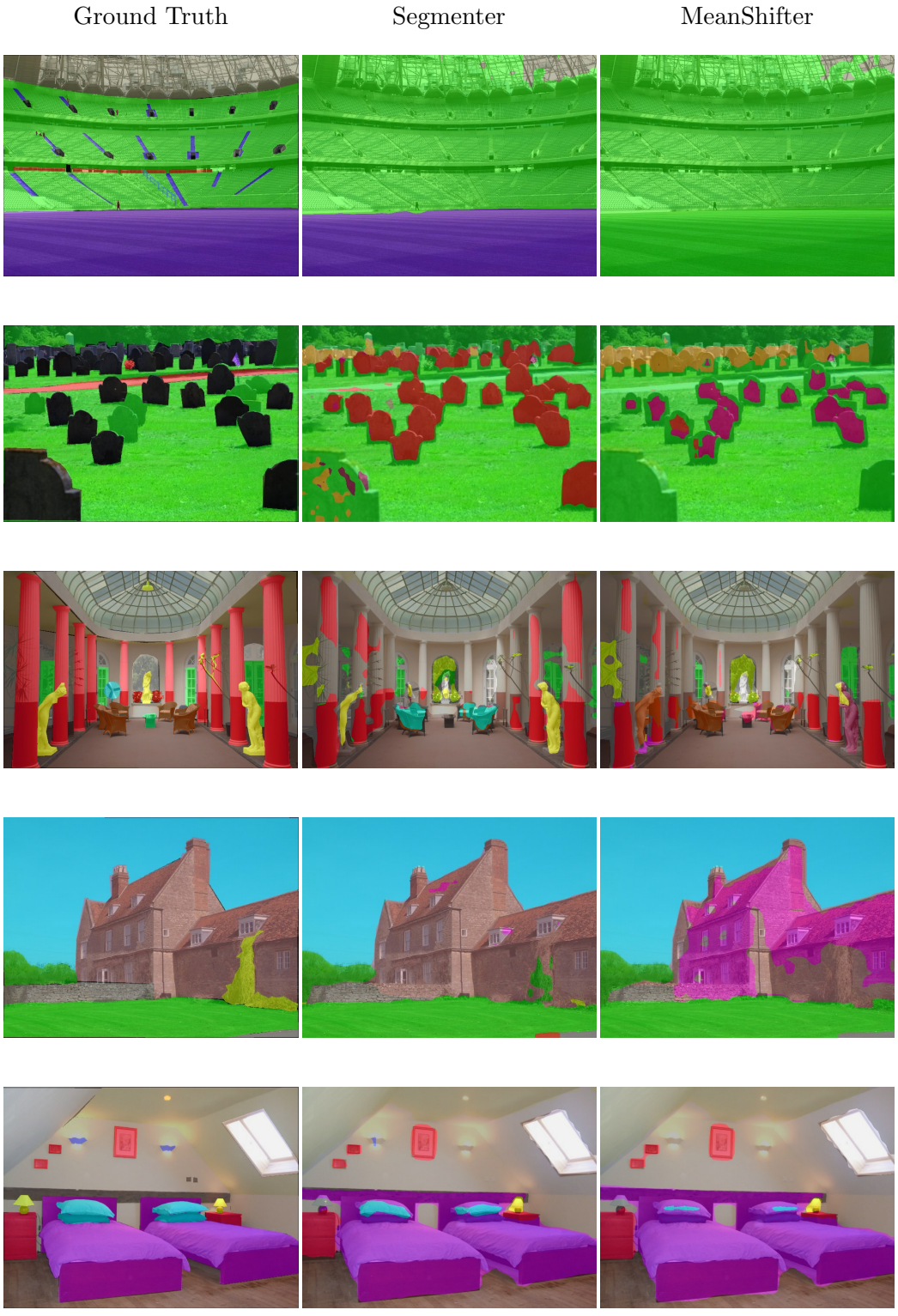

Fig. 3: Images where MeanShifter performs worse compared to Segmenter.

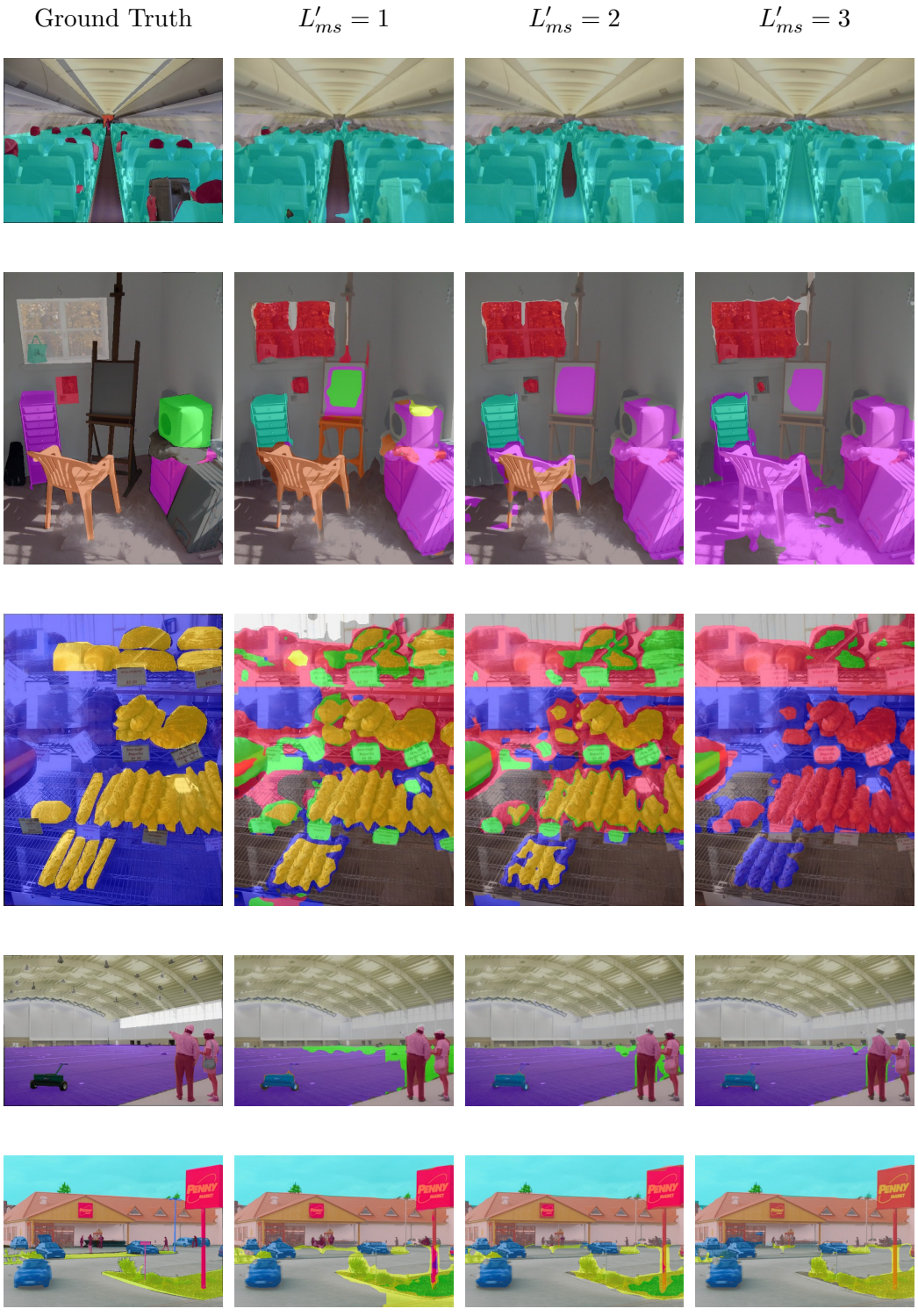

Fig. 4: Using different number of mean shift layers L'_{ms} at inference time. MeanShifter was trained using $L_{ms} = 1$ and $d_{ms} = 192$.

4 Conclusion & Future Work

This work builds upon the recent Segmenter model for semantic image segmentation, in the sense that uses the same transformer encoder and linear decoder. Our novel contribution is that we add a mean shift layer between the encoder and the decoder, that shifts the encodings produced by the encoder in a way that groups them together based on semantic similarity, thus helping the pixel-level classification of the decoder. MeanShifter produces competitive results. In the future, we would like to perform an extensive ablation study and study the possibility of extending the mean shift block to also work for transformer decoders, instead of simple point-wise linear decoders.

References

[1] MMSegmentation Contributors. MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. `https://github.com/open-mmlab/mmsegmentation`, 2020.

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[3] Keinosuke Fukunaga and Larry Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on information theory*, 21(1):32–40, 1975.

[4] Robin Strudel, Ricardo Garcia, Ivan Laptev, and Cordelia Schmid. Segmenter: Transformer for semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7262–7272, 2021.

[5] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[7] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, 127(3):302–321, 2019.

$\mathbf{d_{ms}}$	$\mathbf{L'_{ms}}$	**aAcc**	**mAcc**	**mIoU**
64	1	75.19	38.72	29.49
64	2	74.06	38.21	28.71
64	3	71.55	36.96	26.94
64	4	68.93	35.39	24.98
128	1	75.2	38.91	29.8
128	2	73.79	37.66	28.4
128	3	70.56	35.85	26.24
128	4	67.29	33.69	23.72
192	1	75.89	40.96	31.05
192	2	74.2	38.77	29.26
192	3	70.44	36.25	26.57
192	4	66.59	33.69	23.75
256	1	74.77	37.15	28.46
256	2	73.14	35.39	26.85
256	3	69.15	33.07	24.1
256	4	64.57	30.57	21.22

(a) models trained with $L_{ms} = 1$.

$\mathbf{d_{ms}}$	$\mathbf{L'_{ms}}$	**aAcc**	**mAcc**	**mIoU**
64	1	65.42	17.8	12.77
64	2	65.48	18.2	13.0
64	3	65.18	18.26	12.94
64	4	64.74	18.19	12.81
128	1	70.08	25.96	19.16
128	2	70.09	26.16	19.31
128	3	69.72	26.16	19.25
128	4	69.25	26.1	19.12
192	1	69.39	26.1	19.01
192	2	69.7	26.39	19.37
192	3	69.08	26.08	19.04
192	4	68.1	25.61	18.49
256	1	72.26	31.72	23.74
256	2	72.49	31.67	23.83
256	3	71.94	31.25	23.34
256	4	70.86	30.59	22.6

(b) models trained with $L_{ms} = 2$.

$\mathbf{d_{ms}}$	$\mathbf{L'_{ms}}$	**aAcc**	**mAcc**	**mIoU**
64	1	45.52	4.75	2.9
64	2	45.5	4.69	2.88
64	3	45.27	4.64	2.86
64	4	44.95	4.56	2.79
128	1	67.48	22.28	16.44
128	2	67.24	21.98	16.3
128	3	66.84	21.53	15.93
128	4	66.5	21.22	15.66
192	1	40.8	7.19	4.63
192	2	38.01	6.54	4.09
192	3	37.98	6.53	4.07
192	4	37.98	6.53	4.07
256	1	71.92	32.41	23.48
256	2	72.03	32.55	23.64
256	3	71.91	32.4	23.5
256	4	71.71	32.28	23.37

(c) models trained with $L_{ms} = 3$.

Tab. 2: Evaluation of MeanShifter models trained with a specific number of mean shift layers (L_{ms}), when a different number of mean shift layers (L'_{ms}) is used at inference time.