LayeredCNN: Segmenting Layers with Autoregressive Models

Jakob Lønborg Christensen[1], Patrick Møller Jensen[1], Morten Rieger Hannemose[1], Anders Bjorholm Dahl[1], and Vedrana Andersen Dahl[1]

[1]Department of Applied Mathematics and Computer Science, Technical University of Denmark

Abstract

We address a subclass of segmentation problems where the labels of the image are structured in layers. We propose applying autoregressive CNNs which, when given an image and a partial segmentation of layers, complete the segmentation. Initializing the model with a user-provided partial segmentation allows for choosing which layers the model should segment. Alternatively, the model can produce an automatic initialization, albeit with some performance loss. The model is trained exclusively on synthetic data from our data generation algorithm. It yields impressive performance on the synthetic data and generalizes to real data it has never seen. Our method implementation is available at `https://github.com/JakobLC/LayeredCNN`.

1 Introduction

When analysing biological tissues or manufactured components we often meet structures that are arranged in layers. Two examples are shown in Figure **1**: a μCT slice of bone growth plate, and an optical coherence tomography image of retina. Many segmentation tasks can therefore be formulated as finding layers in images. This motivates us to formulate a model that can segment layers.

Consider any of the examples in Figure **1**. Given the image and corresponding partial label[1], a non-expert should be able to complete the label by utilizing layer appearance and the template given by the partial label. With our model, which we call LayeredCNN we aim to automate this task.

The partial label always consists of the leftmost columns of the label, and can contain just a single column of pixels. Extracting the full label based on the partial label is quite challenging. Instead, consider the much easier task of labelling just the single column of pixels to the right of the partial label. A model with this ability can iteratively label subsequent columns, and we can continue until all columns of the image have been labeled.

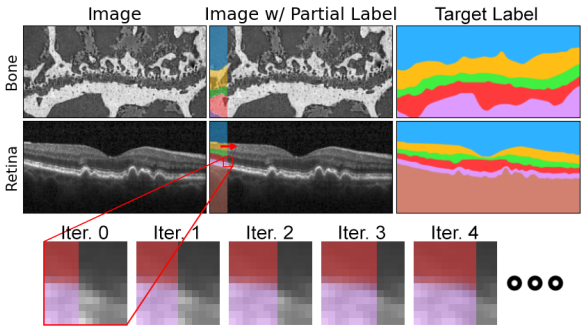

Figure 1: **Top:** Examples of layered images with a partial label and target label. **Bottom:** Zoom in showing propagation of labels.

We will use an autoregressive convolutional neural network (CNN) that conditions on label information to the left to predict the next column of labels. The network will however be able to use image information from the whole image. In our definition, layered images have labels that can be represented by ordered, non-intersecting curves placed at the boundary between two neighbouring label class regions. We call those curves *layer curves*. For each x-value, every layer curve has a uniquely associated y-value. For example, the labels of the bone image in Figure **1** can be represented by four layer curves. Our definition of layered images ensures that all label classes present in the image will appear in the partial label.

[1]The term *label* refers to label image.

https://doi.org/10.7557/18.6254

In a standard segmentation network, the k^{th} channel of the label prediction will be predetermined as a specific class, e.g. the first channel is road, the second channel is pedestrian, etc. Our network operates differently. The partial label defines the classes and not the network architecture. In a sense, our network is much more adaptive than similar networks [9] since we are also learning the label class from the input. We refer to this formulation as a class-agnostic model. A big benefit is that we are capable of segmenting layers in a wide variety of data using the same network.

The model is trained exclusively on synthetic data from our data generation algorithm, yet it learns segmentation that generalizes to real data that it has never been trained on. This makes our network useful as a tool to segment layered images without needing to train a completely new network and manually segmenting large quantities of data.

2 Related Works

Our model is inspired by the PixelCNN [14] framework and its extensions [15, 13]. PixelCNNs were suggested for data generation or image inpainting. We want to use it to expand labels instead, as in work by Leopold et al. [9]. Unlike previous works for autoregressive labelling we use a class-agnostic formulation and apply it to layered images.

Some notable extensions to the PixelCNN framework are the Gated PixelCNN [15] and PixelCNN++ [13]. The PixelCNN++ paper introduced a multi-scale approach with up and downscaling of neurons, similar to that of U-Net [12].

Graph-based methods have also been used for detecting layers in images. One example is the use of dynamic programming [1]. Here, a layer is found as the path that minimizes the accumulated cost along that path through image. This has been extended to multiple layers by formulating a so-called optimal net surface problem [16] that can be efficiently solved using *s-t* graph cut [10]. This has further been extended to multiple exclusive objects [7]. However, all these methods depend on handcrafted energy functions to separate the layers. Our approach differs in that it avoids explicitly formulating an energy function.

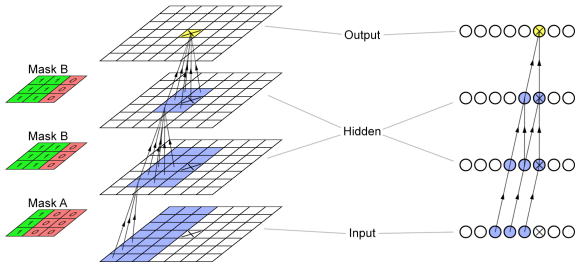

Figure 2: A 2D (left) and 1D (right) visualization of the label dependence graph with no down and upscaling. The output neuron (yellow with cross) is dependent on blue neurons.

3 Method

3.1 Model Architecture

The model always segments from left to right. We can still segment images in other directions if the image is appropriately flipped /rotated.

Our model can extend a partial label by one column of pixels per forward pass. It is able to do this because of one key aspect: a column of labels is predicted using only label information to the left, which we denote left label dependence. This dependence can be satisfied using masked convolutions. These work as normal convolutions except some weights are ignored. Masked convolutions ignore kernel information to the right of the center column. If the center column of a masked convolution is also ignored then it is denoted as a mask A convolution and if not then mask B [15]. We show how masked convolutions are used to satisfy the left label dependence in Figure **2** for a simple network. Note that the marked output neuron only depends on label information to the left.

We implemented up and downscaling of neurons by adapting the method from PixelCNN++ [13]. Figure **3** demonstrates how a naive up and downscaling would result in breaking the left label dependence. Instead, each downscaling of the label neeeds to be preceded by shifting the neurons to the right.

An elementary unit of LayeredCNN architecture is a convolutional block implemented to handle layered images and illustrated in Figure **4**. The convolutional block consists of two parallel feature stacks, image feature stack and label feature stack, each

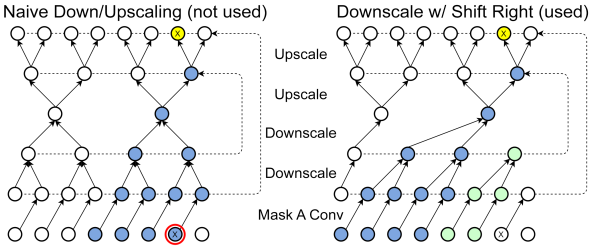

Figure 3: A 1D visualization of the label dependence graph with down and upscaling. The naive implementation contains illegal dependencies (marked red). The output neuron (yellow with cross) is dependent on the blue and green neurons. The output is only dependent on the green neurons due to the element-wise skip connections (dashed lines).

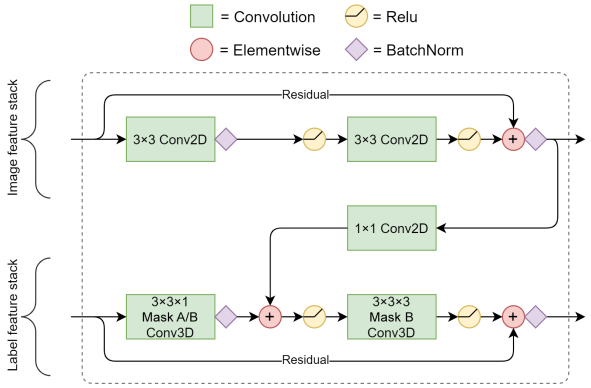

Figure 4: Architecture of the LayeredCNN convolutional block.

equipped with residual connections [4]. Information is passed from the image feature stack to the label feature stack as the network needs information from both to predict the labels correctly. Batch normalization [6] is necessary to keep training stable.

The image feature stack uses normal (unmasked) 2D convolutions, allowing the network to see all image information. The label feature stack uses masked 3D convolutions. The depth dimension (third dimension) represents the label classes given to the stack as a one hot encoding. Since the network is fully convolutional in all three dimensions, we can use a label of the arbitrary depth, such that the same network can be used for different number of classes. The bottom right convolution in Figure **4** propagates information throughout the depth dimension so the network can consider the different layers in relation to each other.

Figure **5** shows the full network, which consists of convolutional block stacks, with three blocks in each block stack. Our network uses three down and upsamplings. All the convolutional blocks use 32 feature channels. Strided convolutions and transposed strided convolutions are used for downscaling and upscaling, respectively. The network ends with two convolutional layers to process the information from all the skip connections. The network starts with a single convolutional block which uses mask A, and the block includes no residual connections. This is in order to satisfy the left label dependence as required by the first layer in Figure **3**. All other convolutional blocks use mask B.

The one hot encoded label given to the network is always ordered from top (first depth channel) to bottom (last depth channel). It is important that the order is consistent to make training easier. Note that the label is both an input and output of the model and we are minimizing the negative log likelihood between the predicted and target label. The left label dependence makes this task non-trivial since the network has to extend the label by one column of pixels. It does this simultaneously for all pixels since we are using convolutions. This means that during training time all the iterations can be trained with a single forward pass, which speeds up training significantly.

During inference, when the network sequentially labels columns of pixels they have a tendency to become increasingly blurry due to the uncertainty of the network. To combat this we introduce a post processing step after each new column of labels has been produced by our network (see Figure **6**). The sharpening is done by converting all probabilities in the column to one-hot labels, except for one pixel at each border between two different labels. These are converted to a mixture of the two labels to allow for sub-pixel accuracy.

3.2 Training Data

Our models are trained exclusively on synthetic data. We have constructed a data generation algorithm based on Brodatz textures [2] that aims at imitating the structure and visual appearance of real layered data (see Figure **7**). We generate 30000 training images with one to five layer curves (6000 of each).

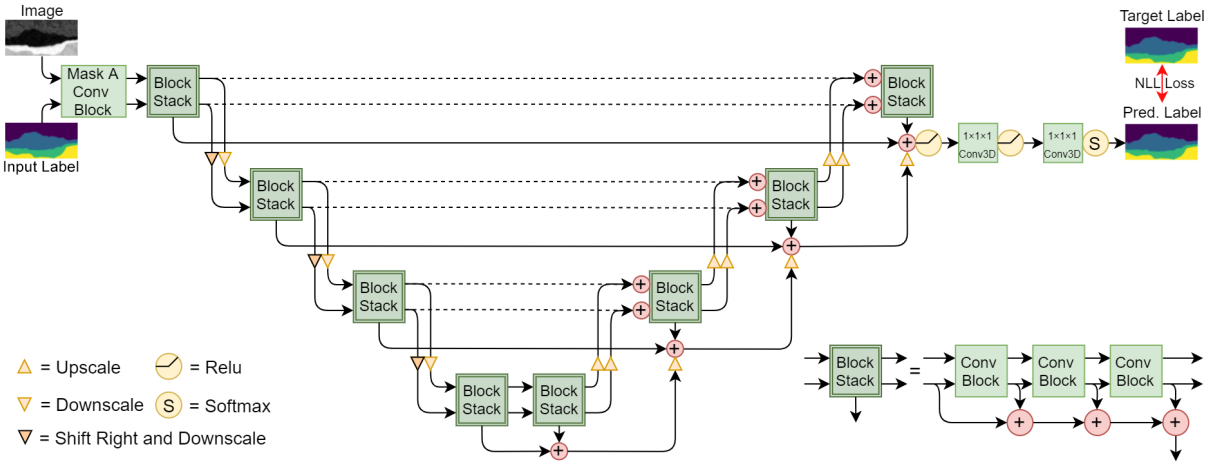

Figure 5: Architecture of the LayeredCNN. Dashed lines are residual skip connections between the encoder and decoder. The downward connections coming from the block stacks are skip connections connected to the end of the network to make learning easier.

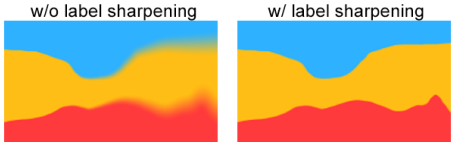

Figure 6: The effect of label sharpening.

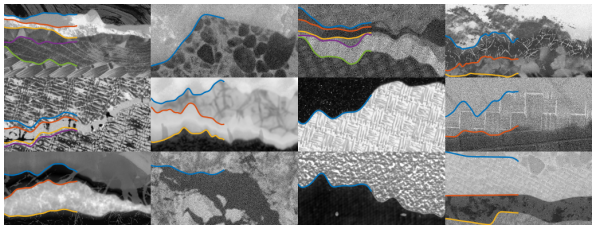

Figure 7: Synthetic images from the data generation algorithm with layer curves in the left half.

Another 2000 images were generated and split into a validation and test set. Additionally, we want to find out how well our method works on real data and we have therefore manually segmented 40 layered images [11] coming from 4 vastly different domains. We use 20 of these as a validation set and 20 as a test set. All models are trained with images of shape $(H \times W) = (64 \times 128)$ pixels. We apply the following data augmentations during training:

Layer dropping where layer curves are ignored from the label with a probability of 20%. This is to make sure the network does not assume everything that looks like a layer is supposed to be segmented (e.g. we might want to only segment some of the layers in a layered image). All layer curves cannot be removed, and therefore one curve (chosen at random) is always kept.

Horizontal and vertical flipping are used with a probability of 50% each.

Border warping is a deformation that we define by

$$\hat{\boldsymbol{l}} = \boldsymbol{l} + \boldsymbol{l}_{\text{warp}}, \quad \boldsymbol{l}_{\text{warp}} = \frac{\boldsymbol{l}_{\text{noise}} * g}{\sqrt{\text{Var}[\boldsymbol{l}_{\text{noise}} * g]}} \sigma_H, \quad (1)$$

where $\boldsymbol{l} \in \mathbb{R}^W$ is a layer curve and $\hat{\boldsymbol{l}} \in \mathbb{R}^W$ is the corresponding warped layer curve that has been warped by $\boldsymbol{l}_{\text{warp}} \in \mathbb{R}^W$. The numbers in the layer curve vectors represent the height (y) position of the layer at each image width index. The operator $*$ is a cross-correlation filtering and g is a Gaussian kernel with standard deviation σ_W. The noise vector $\boldsymbol{l}_{\text{noise}} \in \mathbb{R}^W$ contains I.I.D Gaussian noise which we filter with g to get $\boldsymbol{l}_{\text{warp}}$. The variable σ_H represents the vertical warping height and σ_W represents the horizontal smoothness of the warping. We sample $\sigma_W \in \mathcal{U}(1, 3)$ and $\sigma_H \in \mathcal{U}(1, 1.5)$ (in pixels) where $\mathcal{U}(a, b)$ is the uniform distribution. We apply border warping to the input labels during training, but not the target labels. This encourages the network to

undo the border warping.

Training on network outputs is when we pass an input through the network without gradients a few times before doing the final forward pass with gradients. However, The target label in the final forward pass with gradients is kept unchanged. The input label will be deformed by the network as if it was continuing a column of labels per forward pass. We are therefore training the network to fix its own mistakes. The number of forward passes without gradients is sampled with equal probability from $\{0, 1, \ldots, \min(\lfloor e/3 \rfloor, 5)\}$, where e is the number of completed epochs.

3.3 Automatic Initialization

We formulate a method of producing an automatic initialization (partial label). We can give a trained network a batch of 20 linearly spaced layer curve initializations ranging from the bottom to the top of an image. We initialize just the rightmost column of labels and then segment the image from right to left. The linearly spaced layers will find nearby ground truth layers and end up clustering together. The position of these clusters in the leftmost column of labels can be used as our initializations. We cluster layers with single-linkage clustering [3] with a linkage distance of 1.5 pixels on the positions in the leftmost column.

We want to select the best clusters. We can measure how confident the network is in a layer by how blurry the one hot label is before layer sharpening. The summed absolute difference between the unsharpened and sharpened one-hot label is the cost associated with a layer. The cost associated with a cluster is the minimum cost of the layers contained in it.

The positions of the best m clusters are used, where m is the number of layers we want to segment. The actual segmentation can begin with the leftmost column initialized at those positions (see Figure **8**).

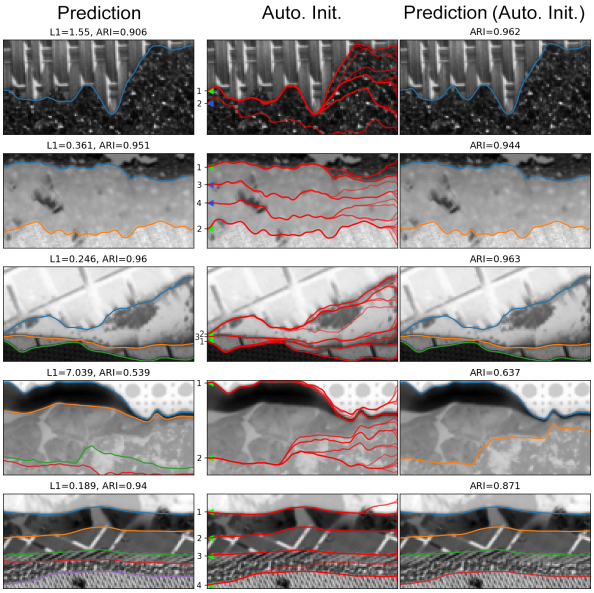

Figure 8: Segmentations of the synthetic data from the test set. **Left:** Prediction with one columns of target labels initialized. **Middle:** Linearly spaced auto. init. curves with smaller numbers indicating lower cluster costs. **Right:** Prediction made using auto. init. positions (green arrows).

4 Experiments

4.1 Segmentation Results

We implemented LayeredCNN using PyTorch and the model was trained with the Adam [8] optimizer for 30 epochs (0.9 million images). Training the model on a single Nvidia V100 GPU with 32 GB RAM took approximately 15 hours. Segmenting a (64×128) image with 2 layer curves, which requires 127 sequential forward passes through the network, takes approximately 5 seconds.

We are using two measures of performance: the mean absolute distance between target layer curves and predicted layer curves (denoted L1) and the Adjusted Rand Index [5] (denoted ARI). ARI is useful since it can also compare different numbers of labels, which is a possibility when using auto. init. Larger scores are better with ARI and ARI=1 represents a flawless segmentation while ARI=0 is as good as random.

Results on synthetic data are very accurate (see Figure **8** and Table **1**). Some qualitative results of real data are displayed in Figure **10**, where the label was initialized with only the leftmost column. Most segmentations are accurate despite our model only training on synthetic data. The model is able to handle images where the segmentation is both intensity and texture based. The prediction does

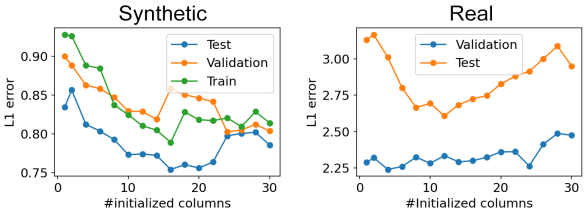

Figure 9: The mean L1 error (for segmented columns) plotted against the number of initialized columns of label.

Network	Measure	Synthetic			Real	
		Train	Vali.	Test	Vali.	Test
Base network	L1 ↓	**0.921**	**0.893**	**0.828**	2.270	3.106
	ARI ↑	**0.916**	**0.917**	**0.920**	0.835	0.797
Base network (auto. init.)	ARI ↑	0.870	0.873	0.859	0.642	0.595
No TONO & no BW	L1 ↓	1.556	1.644	1.539	4.831	3.531
	ARI ↑	0.869	0.863	0.866	0.742	0.811
No TONO	L1 ↓	0.973	1.012	0.854	2.720	2.890
	ARI ↑	0.913	0.914	**0.920**	0.822	0.828
No border warping (BW)	L1 ↓	1.066	1.103	0.886	**2.148**	**2.710**
	ARI ↑	0.904	0.903	0.912	**0.843**	**0.829**

Table 1: The mean performance measures of our model on the different datasets. Training on network outputs is abbreviated as *TONO* and border warping as *BW*.

deviate from the target in some cases. On *Test Bone #1* (top image) the blue layer follows the porous bone protrusions, however it is unclear from the initialization whether they should be segmented or not. In the third image, *Test Bone #8*, the model has detected the blue curve correctly but loses track of the orange curve. Perhaps this is because the image is quite dissimilar from the synthetic training data. The target orange curve indicates a subtle boundary between a coarse texture to fine texture. The synthetic training data rarely had such wide fades. The network mostly fails on images with few similarities to the synthetic training data.

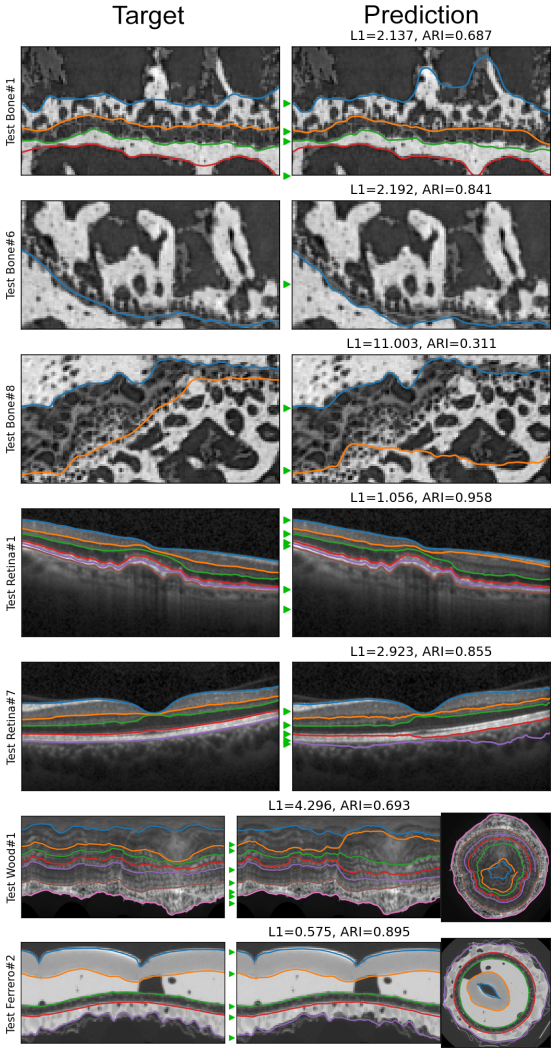

Figure 10: Segmentation results on real data from the test set. Green arrows are auto. init. positions (not used as initialization here). The bottom two images showcase that circular data can be segmented by our model in polar coordinates.

4.2 Ablation study

To investigate the effect of the initialization size we segmented images with an initialization ranging from 1 to 30 label columns (see Figure **9**). Performance improves slightly with more columns.

To test the effect of our data augmentations we train a network without border warping, without training on network outputs and without both. Mean performance measures are shown in Table **1**. The table shows that the proposed augmentations improve performance. Either training on network outputs or border warping is necessary, although both might not be needed. The model is more likely to lose track of layers without any data augmentations. The loss in ARI when using automatic initializations is very low on the synthetic datasets (2%-6%), but much larger on real data (23%-26%).

The base network has 1.16 million trainable parameters and we found that increasing the model size improved synthetic data performance slightly while having no impact on real data performance. This probably indicates that the distribution overlap between real and synthetic images is limited.

5 Discussion and Conclusion

We present the LayeredCNN model architecture which has relatively good success in segmenting real images when considering the fact that it has only trained on synthetic data. The model is extremely good at finding layers in the synthetic data, and it could even consistently segment them with no human supervision (automatic initialization). Human supervision would however be required to get consistent results on real data. The best improvement would likely come from training a model on a large dataset of real layered images, as most of the poor performance was seen in data with few similarities to the synthetic data. A good use of our model is to reduce a long and arduous segmentation task to just a couple of clicks. The main strength of our model is the class-agnostic formulation that makes it applicable to a large variety of different data that it has not trained on.

6 Acknowledgements

This work is supported by The Center for Quantification of Imaging Data from MAX IV (QIM) funded by The Capital Region of Denmark.

References

[1] J. Beutel, H. L. Kundel, and R. L. Van Metter. *Handbook of medical imaging*, volume 1. Spie Press, 2000.

[2] Dong-Chen He and Abdelmounaime Safia. Brodatz's texture database. URL `http://multibandtexture.recherche.usherbrooke.ca/original_brodatz.html`. [Online; accessed 27-May-2021].

[3] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CVPR*, 2016. doi: 10.1109/CVPR.2016.90.

[5] L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985. doi: 10.1007/BF01908075.

[6] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML*, 2015. URL `https://arxiv.org/abs/1502.03167`.

[7] N. Jeppesen, A. N. Christensen, V. A. Dahl, and A. B. Dahl. Sparse layered graphs for multi-object segmentation. *CVPR*, 2020. doi: 10.1109/CVPR42600.2020.01279.

[8] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR*, 2015. URL `https://arxiv.org/abs/1412.6980`.

[9] H. A. Leopold, J. Orchard, J. S. Zelek, and V. Lakshminarayanan. Pixelbnn: Augmenting the pixelcnn with batch normalization and the presentation of a fast architecture for retinal vessel segmentation. *Journal of Imaging*, 5, 2019. doi: 10.3390/jimaging5020026.

[10] K. Li, X. Wu, D. Chen, and M. Sonka. Optimal surface segmentation in volumetric images - a graph-theoretic approach. *PAMI*, 28(1):119–134, 2006. doi: 10.1109/TPAMI.2006.19.

[11] Paul Mooney. Retinal OCT Images (optical coherence tomography). Kaggle dataset, 2018. URL `https://www.kaggle.com/paultimothymooney/kermany2018`. [Online; accessed 27-May-2021].

[12] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *MICCAI*, 2015. doi: 10.1007/978-3-319-24574-4_28.

[13] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *ICLR*, 2017. URL `http://arxiv.org/abs/1701.05517`.

[14] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. *ICML*, 2016. URL `http://arxiv.org/abs/1601.06759`.

[15] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. Conditional image generation with pixelcnn decoders. *NIPS*, 2016. URL `http://arxiv.org/abs/1606.05328`.

[16] X. Wu and D. Z. Chen. Optimal net surface problems with applications. *ICALP*, 2002. doi: 10.1007/3-540-45465-9_88.