Automatic 3D segmentation of pulmonary lobes using progressive dense v-net

Abdullah-Al-Zubaer Imran[1], Ali Hatamizadeh[1], Shilpa P. Ananth[2], Demetri Terzopoulos[1,2], Xiaowei Ding[1,2], and Nima Tajbakhsh[2]

[1] University of California, Los Angeles, Los Angeles, CA 90095, USA
[2] VoxelCloud Inc. , Los Angeles, Los Angeles, CA 90024, USA

[**This report contains a short description of our lobe segmentation model. Our complete paper is currently under peer-review. Once the review finishes, this report will be replaced by the full paper.**]

1 Algorithm-Checklist

Give the overall structure of the algorithm. Does your algorithm search for boundaries or fissures? Does it use airway or other information? Does it use an atlas and/or region growing? our algorithm is an end-to-end deep-learning based network which utilizes 3D context of the input and is supervised at different resolutions without the need for any prior knowledge of airways/vessels or anatomical knowledge or atlases.

Briefly describe each step in the structure of the algorithm (If applicable, which type of algorithms were used for preprocessing? How are different types of information combined?) We propose an end-to-end solution for automatic lung lobe segmentation in CT scan images via a progressive architecture inspired by dense V-networks, as depicted in Fig. 1. In its architecture, the proposed method employs convolutional layers, dense feature blocks, convolutional downsampling, and upsampling. In each convolutional layer, 3D convolutional operations are followed by batch normalization and rectified linear units (ReLu), while in each dense feature block, every layer is connected with every other layer in a feed-forward manner, resulting in each layer receiving the feature maps of all the preceding layers as an input. In addition, batch-wise spatial dropout is incorporated for regularization purposes. The inputs to the network are first down-sampled using a strided convolutional layer, and then passed to the dense feature blocks. Consecutively, the outputs of the dense feature blocks are utilized in low and high resolution passes via convolutional down-sampling and skip connections. This enables the generation of feature maps at three different resolutions. The outputs of the second and third dense feature blocks' skip connections are further up-sampled in order to be consistent with the size of the output in the first skip connection. These feature maps are concatenated

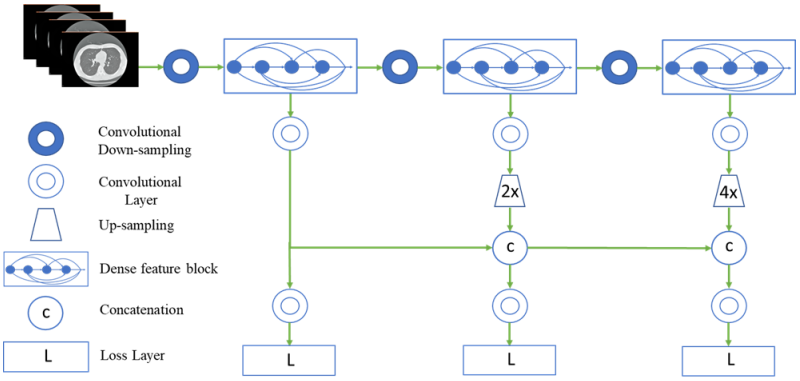

Fig. 1. Architecture of the progressive dense V-network.

and passed to a convolutional layer followed by a softmax layer, which outputs the probability maps. At each stage, we define three separate dice loss layers, as discussed in the subsequent section, with the aim of progressively improving the previous outputs. All the convolutional down-sampling layers have a kernel size of $3 \times 3 \times 3$ with strides of 2, except for the initial convolutional down-sampling layer, which has a kernel size of $5 \times 5 \times 5$. In addition, each of the employed dense feature stacks has 2, 4, and 8 output channels for high, medium, and low resolution stacks, respectively, with a kernel size of $3 \times 3 \times 3$ and a stride of 1. All the Skip layers have a kernel size of $3 \times 3 \times 3$ and a stride of 1.

Volumetric predictions with the same spatial resolution as with the corresponding ground truth are fed into each of the loss functions. Such predictions denote the probabilities with which each voxel belongs to the corresponding class. Since the background region often occupies more volume in comparison to the foreground, we employ a dice loss function which directly maximizes the similarity between the predicted values and the ground truth over all voxels for a multi-class segmentation task:

$$D = \sum_{l=1}^{L} \frac{\sum_i^N p_i^l g_i^l}{\sum_i^N (p_i^l)^2 + \sum_i^N (g_i^l)^2}, \tag{1}$$

where N is the total number of voxels, L is the number of classes, p_i^l denotes the predicted probabilities for each class, and g_i denotes the corresponding ground truth for each class.

List limitations of the algorithm. Is the algorithm specifically designed to segment only certain types of scans? Is your algorithm intended for segmenting pathological lungs? Was it optimized to work for scans with thick or thin slices, are other technical scan parameters expected to influence segmentation performance? For non-pathological scans, our model shows robustness against scan parameters such as slice thickness and

reconstruction kernel. However, we did observe slightly lower performance for pathological cases. This trend can be explained by 1) the absence of pathological cases in our training set, 2) the suboptimal, machine-generated ground truth that we received for pathological cases from NIH.

Was the algorithm trained with example data? If so, describe the characteristics of the training data. we selected a subset of chest CT volumes (354 cases) from the LIDC dataset for annotation. To ensure variation in the data, the CT scans were selected such that both challenging and visible fissures are well-represented in the dataset. The lobe segmentation ground truth masks were generated in a semi-automatic fashion. To mitigate a bias in the ground truth, mask generation was performed by multiple observers and the generated masks were later refined and validated by a radiologist.The dataset was split into 270 training and 84 test cases. The CT scans used in the experiment have a variable number of slices with each CT volume consisting between 100 to 672 slices of size 512×512 pixels. The voxel dimensions vary between [0.49–0.98, 0.49–0.98, 0.45–3.00] mm in the x, y and z axis. Therefore, the selected CT scans used for pulmonary lobe segmentation not only exhibit varying shapes of fissures and lobes, but also show a variable number of slices and voxel sizes.

If the algorithm has been tested on other databases, you could consider including those results. the proposed algorithm has been tested on 84 cases of the LIDC data set with variable levels of fissure visibility, and we achieved an overall dice score of 0.95 for all the lobe classes.

What is the average runtime of your algorithm, and on which system is this runtime achieved? our method processes each 3D CT images in one minute on average using 1 Nvidia Titan X GPU.

Is your algorithm automatic or semi-automatic? If user input is used, how much is needed and in what way? the proposed algorithm is fully automatic and does not require user input or interaction.