Automatic linear measurements of the fetal brain on MRI with deep neural networks

Appendix

This appendix provides details of the pipeline presented in the main paper. It consists of five sections: A.1. Network architectures; A.2. Reference slice selection; A.3. Fetal brain components segmentation; A.4 Parameters selection, and; A.5. Implementation details.

A.1 Network architectures

We use two variants of the ResNet architecture described in reference [26] of the manuscript for the reference slice selection (ResNet50) and the brain structures segmentation (ResNet34). The networks are 50- and 34-layers deep variants of Residual Networks. The configurations and parameters of these networks are described in reference [26] are:

Layer	Output size	ResNet34	ResNet50
Conv1	112x112	Conv 7x7, 64, stride 2	
Conv2_x	56x56	MaxPool 3x3, stride 2	
		3 blocks of 2 Conv 3x3, 64 Conv 3x3, 64	3 blocks of 3 Conv 1x1, 64 Conv 3x3, 64 Conv 1x1, 256
Conv3_x	28x28	4 blocks of 2 Conv 3x3, 128	4 blocks of 3 Conv 1x1, 128 Conv 3x3, 128 Conv 1x1, 512
Conv4_x	14x14	6 blocks of 2 Conv 3x3, 256 Conv 3x3, 256	4 blocks of 3 Conv 1x1, 256 Conv 3x3, 256 Conv 1x1, 1024
Conv5_x	7x7	3 Blocks of 1 Conv 3x3, 512	3 blocks of Conv 1x1, 512 Conv 3x3, 512 Conv 1x1, 2048
fc_1000	1x1	Average pooling FC 1000 + softmax	

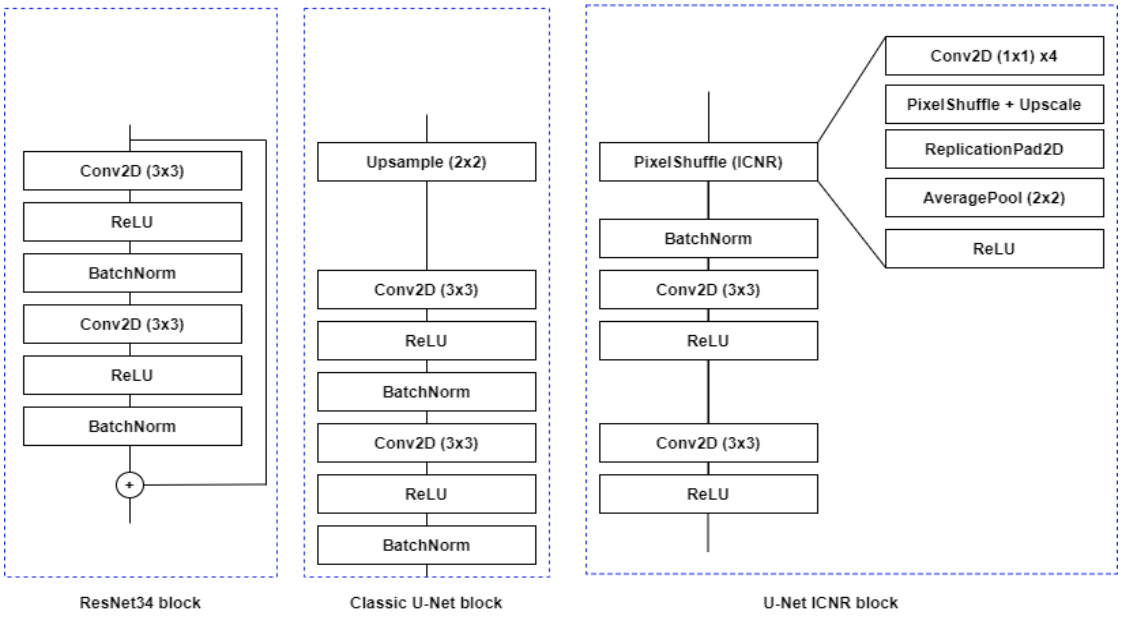

Fig. A.1: Blocks of the fetal brain structures segmentation network: a ResNet34 block for the first scale of the encoder (left), a U-Net ICNR block for the decoder (right) and a comparison block between the classic U-Net and the U-Net ICNR blocks (middle).

Each convolution is followed by ReLU and BatchNorm layers (Fig. A.1) and each ResNet block ends with identity skip connection with summation between original block input to the block output.

We use the network models that were pre-trained on the ImageNet dataset available from the torchvision.models python package (https://pytorch.org/vision/stable/models.html).

A.2. Reference slice selection

The ResNet50 network is modified as follows. First, we replace the last multi-class classification layer, fc-1000, with a two-class softmax classification layer. Second, we copy the single gray-value channel to the three RGB channels. The network training strategy reflects these modifications. First, we train all the network layers for 10 epochs to account for the difference in the data, e.g., medical images vs. natural images in ImageNet and then train the classification layer for 20 more epochs. We use Stochastic Gradient Descent with momentum optimizer, with learning rate of 10^{-3}, *alpha* = 0.9 and batch size of 18 slices (12 background and 6 reference slices in each batch).

For training and inferencing, we use four augmentations: 1) cropping with a bounding box with scaling in the range x1.0-1.7 for training and x1.5 for inference; 2) sampling with a negative/positive ratio of 2:1 for training only; 3) scaling to size 224x224 and normalization in the range [0,1] for both training and inference, and; 4) rotation in the range of 0-90 degrees for training and horizontal and vertical flip for both training and inference.

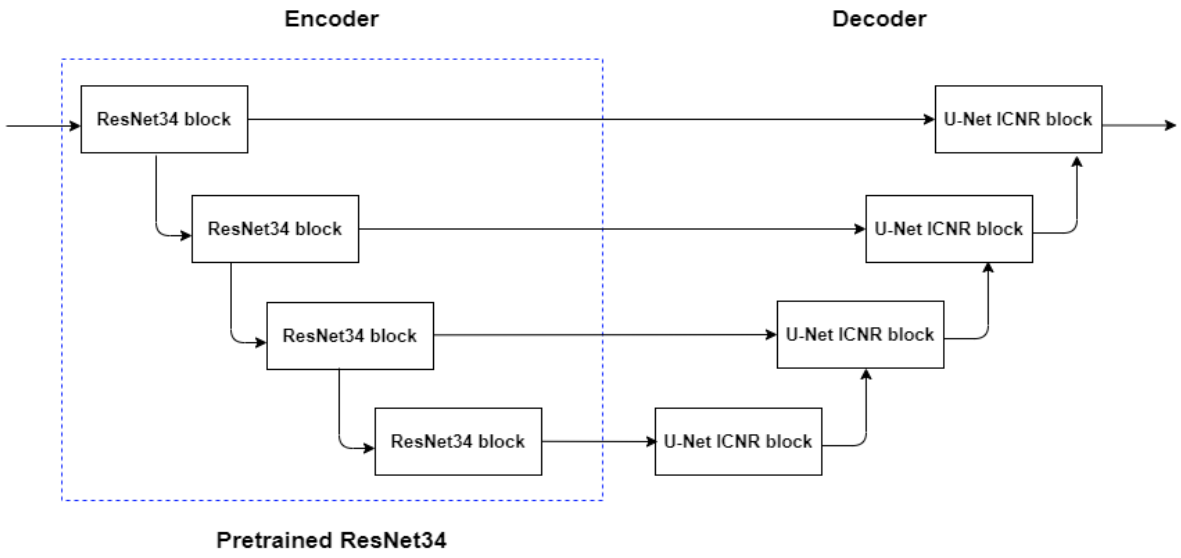

Fig. A.2: Custom network for the fetal brain components segmentation. The encoder (right) is an ImageNet-pretrained ResNet34, with four scales; the decoder (left) has four scales of U-Net ICNR blocks.

A.3 Fetal brain components segmentation

Fig. A.2. shows the block diagram of the network. The pretrained ResNet34 network is used as the encoder. The decoder consists of U-Net PixelShuffle ICNR blocks for each scale described in [A1]. The encoder and decoder are connected with the fast.ai Dynamic U-Net [A2].

To train the network, we use the OneCycle optimizer in [A3] with a learning rate of 10^{-3} for 12 epochs twice -- the first time only the decoder layers (U-Net ICNR blocks), and the second time for the entire network with batch size of 8 slices.

For training and inferencing, we use four augmentations: 1) cropping with a bounding box with scaling in the range x1.0-1.7 for training and x1.5 for inference; 2) intensity with a contrast in the range 0.4-0.8 and brightness in the range 0.4-1.6 for training only; 3) scaling to size 160x160 and normalization in the range [0,1] for both training and inference, and; 4) rotation in the range of 0-90 degrees for training and horizontal and vertical flip for training. For the inference, the slice output segmentation is computed by nearest-neighbor interpolation followed by zero-padding (background class) to obtain the original slice size.

A.4 Parameters selections

The following parameters/hyperparameters were empirically set on the training and validation set results (45 volumes for training, 10 for validation, Section 4.5).

For the reference slice selection network (Section 3.2), all parameters, including the ratio between positive and negative samples (1:2), were selected empirically from the results of their validation set. For the brain component segmentation (Section 3.3), all parameters, including augmentation hyperparameters, were selected empirically. For the mid-sagittal line and brain orientation computation (Section 3.4), the SVM is executed for up to 10^8 iterations with $\lambda = 10$. For the BBD measurement computation (Section 3.5), the inner skull contour pixels are identified by selecting the point with the maximum value from the two local extrema closest to the segmented cerebral brain boundary above a predefined threshold set to 60 intensity units difference. For the computation reliability estimation (Section 3.6), all parameters were selected empirically.

A.5 Implementation details

The code is written in Python and runs on Linux. It uses the Python libraries dicom2nifti, NiBabel [A4], Numpy [A5], Scipy [A6], scikit-image [A7]. Tensorflow and Keras [A8] are used for training and inferencing of the custom anisotropic 3D U-Net for fetal brain ROI identification (stage 1, Section 3.2). OpenCV [A9], Fast.AI [A2] and Pytorch [A10] are used for training the reference slice selection CNN (stage 2, Section 3.3) and for the fetal brain structure segmentation (stage 3, section 3.4). Scikit-learn [A11] is used for the midsagittal line computation (stage 4, Section 3.5). To speed up training and inference of all networks, the implementation uses NVIDIA CUDA library [A12].

The program runs on a PC: Intel Xeon4110 CPU, 128GB RAM, 10GB used, and a 1x1080TI GPU. The training phase takes ~360 mins for fetal brain ROI identification, ~60 mins for the selection of two reference slices CNNs, and ~120 mins for the segmentation of fetal brain structures. The online inference computation of the entire pipeline takes ~5 secs (3-9 secs) per volume.

References

[A1] Aitken A, Ledig C, Theis L, Caballero J, Wang Z, Shi W (2017) Checkerboard artifact free sub-pixel convolution: A note on sub-pixel convolution, resize convolution and convolution resize. arXiv preprint arXiv:1707.02937.

[A2] Howard J, Gugger S. (2020) Fastai: A layered API for deep learning. Information 11(2), 108. (https://docs.fast.ai/vision.models.unet.html).

[A3] Smith LN (2018) A disciplined approach to neural network hyper-parameters: Part 1 -- learning rate, batch size, momentum, and weight decay. arXiv preprint arXiv:1803.09820.

[A4] Brett M, Hanke M, Markiewicz C, Côté MA, McCarthy P, Ghosh S, Wassermann D (2018) nipy/nibabel: 2.3.0. Zenodo 1287921.

[A5] Harris CR, Millman, KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, Wieser E, Taylor J, Berg S, Smith NJ, Kern R (2020) Array programming with NumPy. Nature 585, 357–362.

[A6] Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, van der Walt SJ (2020) SciPy 1.0: fundamental algorithms for scientific computing in Python. Nature Methods 17, 261–272.

[A7] Van der Walt S, Schönberger JL, Nunez-Iglesias J, Boulogne F, Warner JD, Yager N, Gouillart E, Yu T (2014). scikit-image: image processing in Python.

[A8] Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M (2016). Tensorflow: A system for large-scale machine learning, 12th USENIX Symp. on Operating Systems Design and Implementation (OSDI 16), pp. 265-283.

[A9] Bradski G, Kaehler A. (2000) OpenCV. Dr. Dobb's Journal of Software Tools.

[A10] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T (2019) Pytorch: An imperative style, high-performance deep learning library. arXiv preprint arXiv:1912.01703.

[A11] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V (2011). Scikit-learn: Machine learning in Python. J. of Machine Learning Research 12, 2825-2830.

[A12] Kirk D (2007). NVIDIA CUDA software and GPU parallel computing architecture. Proc. 6th Int. Symp. on Memory Management, pp. 103-104.