Article name: Deep Convolutional Neural Network Applied to The Liver Imaging Reporting and Data System (LI-RADS) version 2014 Category Classification: A Pilot Study
Journal name: Abdominal Radiology
Author names: Blinded for review
Affiliation and e-mail address of the corresponding author: Blinded for review

Appendix E1

Model Development

We chose the architecture, hyper parameters and data augmentation techniques that obtained the highest accuracy on our validation set.

Pre-processing

First, we resized each image to a 224 x 224 matrix using the Pillow 5.0.0 python package (1). Next, each pixel in the images was divided by 255 so all values were in the interval between 0 and 1. Observation diameters were normalized by subtracting the mean and dividing by the standard deviation across all diameters. The consensus categories were one hot encoded.

Model architecture

We implemented two different convolutional neural network architectures: 1) a VGG16 pre-trained network (2); and 2) a custom-made network.

1) A VGG16 pre-trained network

A VGG16 model was downloaded through the Keras package (3) with pre-trained weights on ImageNet (4), except for the fully-connected layers. Since the model was pre-trained on natural RGB images, triple phase images (pre-contrast, late arterial and delayed phases) were assigned to each of the three channels as an input. The output tensor was then concatenated with a fully-connected layer with 32 nodes with an input of observation diameters and processed with 3 fully-connected layers with 1024, 512, and 4 nodes to produce probabilities for the 4 LI-RADS categories (LR-1/2, LR-3, LR-4, and LR-5) with activation functions of rectified linear unit, rectified linear unit, and softmax, respectively. We inserted dropout layers of 0.5 just after the two fully-connected layers with 1024 and 512 nodes.

2) A custom-made network

A custom-made network was built with a 4-channel image input, to which quadruple phase images (pre-contrast, late arterial, portal venous, and delayed phases) were assigned. Images data were processed with batch normalization layer with first, then processed in one convolutional layer using “Conv2D” function in the Keras package (3) with a kernel size of 3×3, same padding, and activation function of rectified linear unit. Output images from the convolutional layer were then processed in a max pooling layer with “MaxPooling2D” function with a filter size of 2×2, same padding, and a stride of 2. The combination of one convolutional layer and one max pooling layer was repeated three times. The number of feature maps for each convolutional layer was 8, 16, and 32, respectively. Next, the output tensor was processed in a global max pooling layer, which was then concatenated with a fully-connected layer with 32 nodes with an input of observation diameters, and processed with 3 fully-connected layers with 64, 64, and 4 nodes to

produce probabilities for the 4 LI-RADS categories (LR-1/2, LR-3, LR-4, and LR-5) with activation functions of rectified linear unit, rectified linear unit, and softmax, respectively. We inserted dropout layers of 0.5 just after the two fully-connected layers with 64 nodes.

Model training

1) A VGG-16 pre-trained network

The model was first trained on only the fully-connected layers with having all the pre-trained layers frozen (epoch = 400), and then the weights for the latter half of the pre-trained convolutional layers as well as the fully-connected layers (epoch = 400) were fine-tuned, as part of the training process with a batch size of 32. A loss function of categorical cross entropy was applied. The optimization was carefully conducted with small learning rates by using stochastic gradient descent with momentum and stochastic gradient descent with restarts proposed by Loshchilov et al. (5, 6) with a minimal learning rate of 4e-5, a maximum learning rate of 4e-4, a learning rate decay of 0.9, a cycle length of 3, and a factor to multiply the cycle length after each cycle of 2.

2) A custom-made network

The model was trained with a batch size of 32 and a loss function of categorical cross entropy by using stochastic gradient descent with momentum and stochastic gradient descent with restarts with a minimal learning rate of 4e-4, a maximum learning rate of 4e-3, a learning rate decay of 0.9, a cycle length of 3, and a factor to multiply the cycle length after each cycle of 2. Although the training dataset did not have severe class imbalance, random oversampling was applied to mitigate class imbalance with the imbalanced-learn python package (7, 8).

Data augmentation

All images were augmented with on-the-fly technique using vertical and horizontal flip, width and height shift range of 0.2, zoom range of 0.2, and rotation range of 0.2 within the Keras framework (3). Note that when a specific source image is being used again for training on the next epoch, it will likely be flipped and rotated differently than on the previous epoch.

Hardware and Software

We developed our models with the Python programming language (version 3.6.4, Python Software Foundation, https://www.python.org/), Keras (version 2.1.5) (3) with Tensorflow backend (version 1.6.0) (9), Pillow (version 5.0.0) (1), and Scikit-learn (version 0.19.1) (10) on a workstation with a GeForce GTX 1060 (NVIDIA, Santa Clara, Calif) graphics processing unit, a Core i7-8650U 4.2GHz central processing unit (Intel, Santa Clara, Calif), and 16 GB of random access memory.

References

1. Pillow — Pillow (PIL Fork) 5.3.0 documentation. https://pillow.readthedocs.io/en/5.3.x/. Accessed December 14, 2018.
2. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv. https://arxiv.org/abs/1409.1556. 2014.
3. Chollet F, et al. Keras. https://github.com/fchollet/keras. 2015.

4. Russakovsky O, Deng J, Su H, et al. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision. 2015;115(3):211-52. doi:10.1007/s11263-015-0816-y.
5. Loshchilov I, Hutter F. Stochastic Gradient Descent with Restarts. arXiv. http://arxiv.org/abs/1608.03983. 2016.
6. Huang G, Li Y, Pleiss G, Liu Z, Hopcroft JE, Weinberger KQ. Snapshot ensembles: Train 1, get M for free. arXiv. https://arxiv.org/abs/1704.00109. 2017.
7. Buda M, Maki A, Mazurowski MA. A systematic study of the class imbalance problem in convolutional neural networks. Neural networks : the official journal of the International Neural Network Society. 2018;106:249-59. doi:10.1016/j.neunet.2018.07.011.
8. Lemaître G, Nogueira F, Aridas CK. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. The Journal of Machine Learning Research. 2017;18(1):559-63. doi:Not available.
9. Abadi M, Agarwal A, Barham P, et al. TensorFlow: large-scale machine learning on heterogeneous distributed systems. arXiv. https://arxiv.org/abs/1603.04467. 2016.
10. Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: Machine learning in Python. Journal of machine learning research. 2011;12:2825-30.