Evaluation beyond y and p(y)

Thijs Kooi TKOOI@LUNIT.IO
15F, 27, Teheran-ro 2-gil, Gangnam-gu, Seoul, South Korea

Editors: Under Review for MIDL 2022

Abstract

Academic papers and challenges focus mostly on metrics that measure how well a model's output $p(y)$ approximates labels y. However, a high performance based on these metrics is not a sufficient condition for a practically useful model. Looking into the complexity of a model both in terms of hardware and software can shed more light on the practical merit. This short paper discusses several measures for medical AI system that do not focus solely on labels and predictions. We encourage the research community to consider these metrics more often.

Keywords: Evaluation, AI in practice, hardware, software

1. Introduction

A good evaluation metric for a medical AI system provides the interface between the technical and the clinical problem. Evaluation methodology for stand-alone medical AI systems typically focuses on how well the model output $p(y)$ approximates the target labels y. That is, how well the model can classify, detect, or segment abnormalities, using metrics ROC, AP and Dice (Reinke et al., 2021). A method with high classification performance is a necessary, but not a sufficient condition for a clinically meaningful application. If the model is too computationally complex or difficult to train the merit is questionable. What constitutes a 'good' model can not only be determined by a one-dimensional metric. In this short paper, we discuss some metrics focused on practical implementation to encourage researchers to consider this during development and evaluation.

2. Hardware and deployment

Hardware like GPU's enabled us to train and deploy big, complex neural networks. However, contemporary hardware can still be a bottleneck for architectures operating on, for example, 3D data like CT scans or problems where several large images need to be integrated and reasoned about such as mammograms. Two important metrics to look at are (1) the memory footprint (the space the model occupies in RAM) and (2) the inference speed (the time it takes to process a scan). Both can be approximated with the number of floating point operations (of course leaving out many details). Though sometimes reported, it is not common practice.

The relevance of these metrics depends on the specific application and reading practice. For example, in some clinics mammograms are read while the patient waits. In this case, a model that takes 10 minutes to read a scan may have limited merit. Other examples

could include models deployed to developing countries where state-of-the art hardware may not be available (such as systems for the detection of tuberculosis in chest radiographs) or systems operating in an emergency department where time is vital, e.g. the detection signs of a stroke in cranial CT scans.

3. Software and development

3.1. Code complexity

To optimize development speed, software engineers aim to reduce complexity of code. Simpler code means faster onboarding of new engineers, faster feature shipping and typically less bugs, which also speeds up development. A speed-up also means better patient care as new features and better models become available faster and so do less bugs in the code.

Measuring complexity of software is a contentious issue. Cyclomatic complexity (Ebert et al., 2016), which measures the number of paths through a piece of code is a popular metric. Some developers simply use the number of lines of code as a proxy. Although the proper metrics is debatable, most would agree simple code should be preferred.

3.2. Training complexity

A core design principle in software development is proper modularization. Code should be easy to extract and replace. In computer visions systems, end-to-end trainability is sometimes used as a selling point (Ren et al., 2015), but in real-world applications this is not always desirable. A well modularized model can be much easier to work with.

Imagine a system with 16 hyperparameters, all with 2 different settings. The search space will be 2^{16}. If we can use divide-and-conquer and tune in two phases, we could break this into $2^8 + 2^8$, for example. A properly modularized model can be broken down into simple individual components where intermediate performance is easily measurable. A good performance under that measure should translate into a good or at least equal final performance.

For example, an end-to-end model like Faster R-CNN (Ren et al., 2015) comprises a region proposal network (RPN), a region classification network (RPN) and a shared backbone, all trained in one go, requires (at least) the following parameters to be tuned:

- RPN: anchor size, number of anchors, loss parameters, weight between classification & regression loss, etc.
- R-CNN: number of parameters, number of layers, transfer functions, weight between classification & regression less, learning rate, regularization, etc.
- Backbone architecture: number of parameters, initialization, transfer functions, learning rate, regularization, etc.

Finding an optimal set of hyperparameters for this can be like spinning plates: when the performance of the RPN increases, the R-CNN can get worse again and vice versa. In practice, tuning each component individually is easier.

3.3. Hyperparameter complexity

The number of hyperparameters, and strongly related to that, the sensitivity to each hyperparameter, also determine how easy a system is to use. This can be reduced to the 'effective number of hyperparameters', the number of parameters that measurable change the model's performance. A model that gives us a quick idea of how well it would perform and only needs a bit of tuning for a final release could be preferred over a model that potentially has high performance, but requires many tries of different hyperparameter settings.

In practice only a subset of hyperparameters have a big influence on the performance, the smaller this subset the better, the way in which each parameter influences performance also differs. A simple test would be to train a couple of different models and report the mean and variance over different hyperparameters, instead of just the max over all runs. The smaller the variance for each parameter, the better.

4. Conclusion

This short paper discussed several practical measures for the evaluation of medical AI systems, that do not focus solely on labels and predictions. These include the memory footprint, inference speed, code complexity, training complexity and hyperparameter sensitivity. We encourage researchers to consider these metrics during development and evaluation.

Acknowledgments

Thanks to everyone at Lunit for fruitful discussions

References

Christof Ebert, James Cain, Giuliano Antoniol, Steve Counsell, and Phillip Laplante. Cyclomatic complexity. *IEEE software*, 33(6):27–29, 2016.

Annika Reinke, Matthias Eisenmann, Minu D Tizabi, Carole H Sudre, Tim Rädsch, Michela Antonelli, Tal Arbel, Spyridon Bakas, M Jorge Cardoso, Veronika Cheplygina, et al. Common limitations of image processing metrics: A picture story. *arXiv preprint arXiv:2104.05642*, 2021.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.