The Journal of Open Source Software

Sensie: Probing the sensitivity of neural networks

Colin Jacobs[1]

1 Center for Astrophysics and Supercomputing, Swinburne University of Technology

DOI: 10.21105/joss.02180

Software

- Review
- Repository
- Archive

Editor: George K. Thiruvathukal

Reviewers:

- @ejhigson
- @omshinde

Submitted: 06 March 2020
Published: 19 June 2020

Introduction

Deep neural networks (DNNs) are finding increasing application across a wide variety of fields, including in industry and scientific research. Although DNNs are able to successfully tackle data problems that proved intractable to other methods, for instance in computer vision, they suffer from a lack of interpretability. Some well-known methods for visualising and interpreting the outputs of DNNs include directly inspecting the learned features of a model (e.g. convolutional kernels and activation maps); or producing "saliency maps" which allow a human researcher to inspect which parts of an input (such as an image) played the most crucial role in the model reaching a particular determination. These algorithms include occluding parts of an image (Zeiler & Fergus, 2014), guided backpropagation (Simonyan, Vedaldi, & Zisserman, 2013), and more complex algorithms such as SmoothGrad (Smilkov, Thorat, Kim, Viégas, & Wattenberg, 2017) and Layer-wise relevance propagation (Binder, Montavon, Lapuschkin, Müller, & Samek, 2016).

However, simply inspecting the most salient regions of an input image (or other input) may not always be sufficiently interpretable. Here we present `Sensie`, a python package to quickly inspect and quantify the sensitivity of a trained model to user-specified properties of the input domain, or under the arbitrary transformation of a test set.

Several quality software packages exist to probe artificial neural network internals, such as `Innvestigate` (Alber et al., 2018), `tf-explain`[1], and `keras-vis` (Kotikalapudi & contributors, 2017). These packages are geared towards traditional image-based applications, and cannot explore relationships between the neural network performance and arbitrary properties of the examples presented to the network. `Sensie` is designed to fill that gap.

Method

`Sensie` helps the user explore the sensitivity of a machine learning model to properties of data (using a test set with known, correct outputs). `Sensie` wraps a pre-trained model, and assists in quantifying and visualising changes in the performance of the model as a function of either a specified property of the data (one that is explicitly passed to the model at test time, or other metadata); or a perturbation of the data, parameterised by a value p. In addition, `Sensie` offers a convenience method to display the sensitivity of a classification model to the class itself, though this should be used with caution.

`Sensie`'s algorithm is summarised as follows as applied to a network trained for classification, for two use cases:

A) Using data or metadata: a scalar property p of the test set $X_{\rm test}$, such that each example $\boldsymbol{x}_i$ has a corresponding value p_i and a supplied ground truth $\hat{y}$:

[1] https://github.com/sicara/tf-explain

1. Segment the data into bins by p.
2. Collect predictions from the model (using `model.predict()` or a user-supplied function), and note the scores for the correct classes $\hat{y}$.
3. Calculate the mean score $\bar{s}$ in the correct class for each bin, and the standard deviation.
4. Plot $\bar{s}$ as a function of p.
5. Estimate the significance of the effect using Bayesian linear regression, producing a scalar value representing $\partial\bar{s}/\partial p$, with 50% and 95% credible intervals.

B) Using a perturber function $f_{\mathrm{perturb}}(\boldsymbol{x}, p)$—an arbitrary transformation of the inputs—applied to the test set, where the magnitude of perturbation is parameterised by p:

1. Choose discrete values of p to be tested.
2. For each value of p to be tested, p_j, transform the test set such that $X'_j = f_{\mathrm{perturb}}(X_{\mathrm{test}}, p_j)$.
3. Collect predictions from the model for X'_j (using `model.predict()` or a user-supplied method), and note the scores for the correct classes, $\hat{y}$.
4. Calculate the mean score $\bar{s}$ in the correct class for each p_j, and the standard deviation.
5. Plot $\bar{s}$ as a function of p.
6. Estimate the significance of the effect using Bayesian linear regression[2], producing a scalar value representing $\partial\bar{s}/\partial p$, with 50% and 95% credible intervals.

In the first case, A), `Sensie` can optionally forego binning by P, and treat every element as a data point in determining the trend.

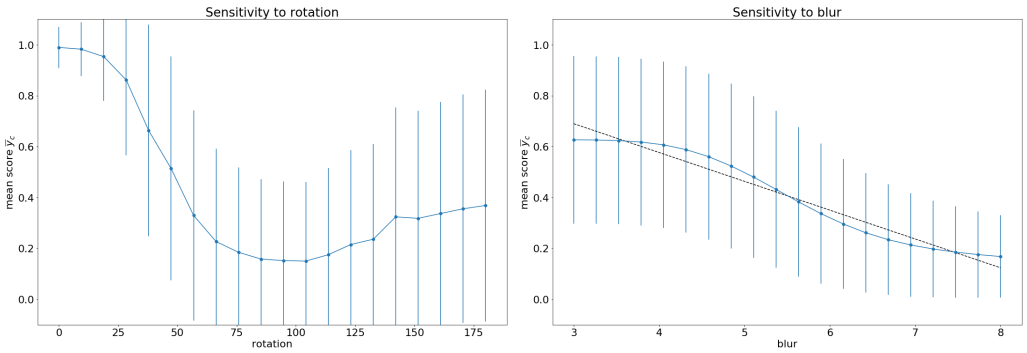

Figure 1: Left: Output from `Sensie` for a model trained to recognise handwritten digits, testing model sensitivity to rotation. Error bars show the standard deviation for the mean ground-truth-class score. Right: Sensitivity of a model to an applied blur of the input image data, showing a linear fit to a significant region.

Discussion and conclusion

`Sensie` helps the model user to quickly identify sensitivities of the accuracy to various properties of the input. This may identify deficiencies in the training set; for instance, if we expect an image classifier to be robust under a particular transformation such as rotation or translation, `Sensie` can quickly quantify whether such a transformation has a statistically significant effect on the accuracy across the test set. Figure 1 shows two examples; on the left, an output plot from `Sensie` showing the sensitivity of a trained model to rotation of the input images; the fact that the curve is not flat indicates that the model is highly sensitive to

[2] See https://docs.pymc.io/notebooks/GLM-linear.html for an example using the PyMC3 probabilistic programming framework.

input orientation and that this is a significant feature (or bug) of the training process. On the right, the sensitivity of a trained model to a Gaussian blur applied to the model, along with a fit indicating the magnitude of the effect. The sensitivity to any given property, i.e. the curve $\bar{s}$ vs p, may not be linear. In this case, a fit using linear regression may produce a less meaningful result.

`Sensie` has been successfully applied to a model trained for astronomical classification (Jacobs et al., 2019) to indicate the model's sensitivity to the scale of features present in the input; the signal-to-noise ratio of the input data; the colours of the input; and astrophysical properties of the objects present in an input image.

Acknowledgements

The author acknowledges support from Karl Glazebrook's Australian Research Council Laureate Fellowship FL180100060.

References

Alber, M., Lapuschkin, S., Seegerer, P., Hägele, M., Schütt, K. T., Montavon, G., Samek, W., et al. (2018). iNNvestigate neural networks! *arXiv:1808.04260 [cs, stat]*. Retrieved from http://arxiv.org/abs/1808.04260

Binder, A., Montavon, G., Lapuschkin, S., Müller, K.-R., & Samek, W. (2016). Layer-wise relevance propagation for neural networks with local renormalization layers. In *International Conference on Artificial Neural Networks* (pp. 63–71). Springer. doi:10.1007/978-3-319-44781-0_8

Jacobs, C., Collett, T., Glazebrook, K., Buckley-Geer, E., Diehl, H. T., Lin, H., McCarthy, C., et al. (2019). An Extended Catalog of GalaxyGalaxy Strong Gravitational Lenses Discovered in DES Using Convolutional Neural Networks. *ApJS*, *243*(1), 17. doi:10.3847/1538-4365/ab26b6

Kotikalapudi, R., & contributors. (2017). Keras-vis. https://github.com/raghakot/keras-vis; GitHub.

Simonyan, K., Vedaldi, A., & Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*. Retrieved from https://arxiv.org/abs/1312.6034

Smilkov, D., Thorat, N., Kim, B., Viégas, F., & Wattenberg, M. (2017). SmoothGrad: Removing noise by adding noise. *arXiv:1706.03825 [cs, stat]*. Retrieved from http://arxiv.org/abs/1706.03825

Zeiler, M. D., & Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. In D. Fleet, T. Pajdla, B. Schiele, & T. Tuytelaars (Eds.), *Computer Vision ECCV 2014* (Vol. 8689, pp. 818–833). Cham: Springer International Publishing. ISBN: 978-3-319-10589-5 978-3-319-10590-1