# {RE} VISUAL EXPLANATION BY INTERPRETATION: IMPROVING VISUAL FEEDBACK CAPABILITIES OF DEEP NEURAL NETWORKS

**Cukran, Yunus**
yunus.cukran@mail.mcgill.ca

**Tang Guokai**
guokai.tang@mail.mcgill.ca

**Zhang, Wenbo**
wenbo.zhang@mail.mcgill.ca

## ABSTRACT

**In this reproduction effort for ICLR 2019, we have taken the paper *"Visual Explanation by Interpretation: Improving Visual Feedback Capabilities of Deep Neural Networks"* (4) and sought to reproduce parts of the authors work. We have focused on validating the automatic feature extraction method proposed by the authors, explored its hyperparameter sensitivity and other characteristics across different convolutional neural network models. We replicated the ablation test performed by the authors to achieve similar results, and validated the importance of selected features.**

## 1 INTRODUCTION

Visual Explanation by Interpretation: Improving Visual Feedback Capabilities of Deep Neural Networks is a work that proposes a pipeline for bridging the gap between methods aiming at model interpretation, i.e., understanding what a given trained model has actually learned, and methods aiming at model explanation, i.e., justifying the decisions made by a model.

### 1.1 PIPELINE

The authors have summarized their contributions as fourfold

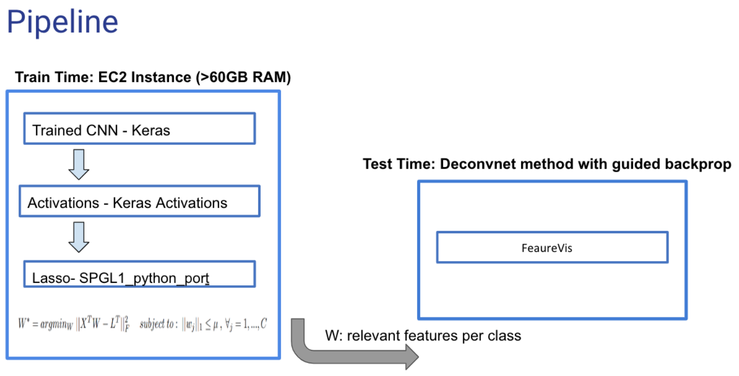

Pipeline
Train Time: EC2 Instance (>60GB RAM)
Trained CNN - Keras
Activations - Keras Activations
Lasso- SPGL1_python_port
$W^* = argmin_W \|X^T W - L^T\|_F^2 \quad subject\ to: \|w_j\|_1 \leq \mu, \forall_j = 1, ..., C$
Test Time: Deconvnet method with guided backprop
FeaureVis
W: relevant features per class

Figure 1: Project Pipeline

1. Proposition of an automatic method based on feature selection to identify the network-encoded features that are important for the prediction of a given class
2. Providing visual feedback with improved quality over previous Deconvolution based saliency map methods.
3. Proposition of a method that is general enough to be applied to any type of network.
4. Releasing a dataset and protocol specifically designed for becoming the benchmark for model explanation methods

## 2 GOAL

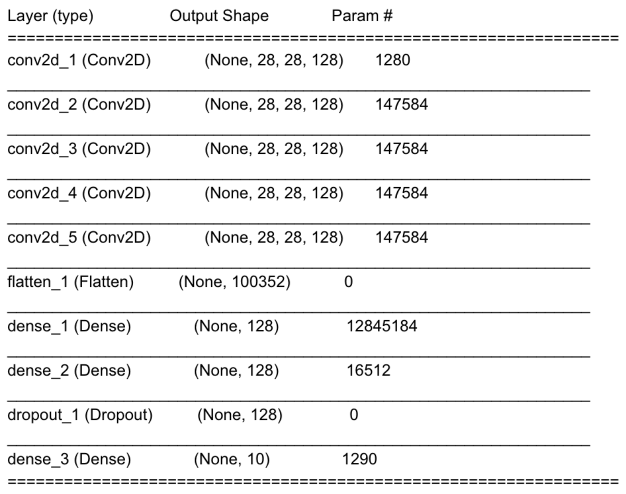
The authors have proposed the following pipeline:

1. Solving a -lasso optimization problem using the Spectral Gradient Method to identify the sparse relevant features.
$$W^* = argmin_w ||X^T W - L||_F^2 \text{ subject to} ||W_j||_1 < \mu$$
2. Feeding the extracted relevant features to the Deconvnet-based method with guided back-propagation from Grn et al. (2016) (3) (using Grn et. all's open sourced implementation: FeatureVis) to visualize the important features
3. A resampling approach (using nearest-neighbor interpolation) that addresses the grid-like artifacts introduced by other deconvNet based visualizations(5).

We have decided to focus on component one, as we believe that selected features lend themselves to more objective testing criteria while the visualizations are more subjective. The recent criticism of Adebayo et al (2) of Deconvolution based visualization methods was another factor in us deciding to spend our efforts on feature extraction. An8Flower, a new dataset introduced in this paper, was not in the scope of our replication effort.

## 3 IMPLEMENTATION METHODOLOGY

The authors have demonstrated their methods on three different datasets: MNIST, Fashion144k, imageNet. We have focused on the MNIST dataset because of our time and budget constraints. We wanted to follow in the spirit of Adebayo et al. (1), which proposes various invarience tests as sanity checks for visualization methods, including a model parameter randomization test which compares outputs of a trained model on a randomly initialized untrained network of the same architecture. The authors own randomization test was similar to this. We have tried to test further invariance by employing slightly different CNN models on the same dataset. We have used the default MNIST CNN of MATLABs MathConvNet library, which has 8 layers (4 Convolution, 2 Pooling, 1 Fully Connected, 1 Softmax). We have also used the following Keras CNN, which has a dropout layer. The dropout layer was added to explore the details of the filter number problem we encountered. Further details will be expanded upon in the results section.



```
Layer (type)              Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)         (None, 28, 28, 128)       1280
_________________________________________________________________
conv2d_2 (Conv2D)         (None, 28, 28, 128)       147584
_________________________________________________________________
conv2d_3 (Conv2D)         (None, 28, 28, 128)       147584
_________________________________________________________________
conv2d_4 (Conv2D)         (None, 28, 28, 128)       147584
_________________________________________________________________
conv2d_5 (Conv2D)         (None, 28, 28, 128)       147584
_________________________________________________________________
flatten_1 (Flatten)       (None, 100352)            0
_________________________________________________________________
dense_1 (Dense)           (None, 128)               12845184
_________________________________________________________________
dense_2 (Dense)           (None, 128)               16512
_________________________________________________________________
dropout_1 (Dropout)       (None, 128)               0
_________________________________________________________________
dense_3 (Dense)           (None, 10)                1290
=================================================================
```

Figure 2: Keras Model Summary

Training models for the MNIST dataset was done on a consumer level personal computer. For the Lasso optimization problem we used an AWS EC2 p2.xlarge instance, as we encountered a bottleneck and required more than 50GB of RAM. We have used the Python wrapper of the spgl1 library to implement the Spectral Gradient Projection method, as SGP was recommended by the authors.

## 4 Results and Discussions

On the MathConvNet implementation, we achieved very consistent results with authors, except hyperparameter differences. The authors have gotten significant results at a mu of 10, while we required a lot more filters to be dropped.

| Mu | Original Model | Conv + Dense | Conv + Dense (Random) | Conv | Conv (Random) | FIlters Dropped |
|---|---|---|---|---|---|---|
| 10 | 1 | 0.86 | 0.99 | 0.875 | 0.9034 | 17 |
| 100 | 1 | 0.30 | 0.96 | - | - | 175 |
| 200 | 1 | 0.21 | 0.32 | - | - | 387 |

Figure 3: MatConvNet Classification Accuracy Summary

In our Keras implementation, the performance drop was very dependent on the number of ablated filters. We experimented with different filter amounts in the architecture, and all models showed different tipping points, but in all models below that tipping point, selected features would not cause much of a difference. Removing dropout seemed to increase this effect. In a non dropout model we had to drop 394 filters to get an accuracy of 80.11, compared to 301 ablations and a performance of 80.03 with dropout. After reaching that certain tipping point however, our findings were similar to those in the paper. This seems to suggest that further exploration about filter sparsity might reveal useful information on interpretation methods.

| Mu | Selected Filters Ablated | Random Filters Ablated (Filter amount Controlled - Averaged over 5 runs) | Total Filters Dropped (out of 640) |
|---|---|---|---|
| 10 | 99.01 | 99.14 | 23 |
| 100 | 98.81 | 99.08 | 69 |
| 500 | 80.03 | 98.71 | 301 |
| 600 | 58.61 | 98.47 | 333 |
| 700 | 11.42 | 98.73 | 397 |

Figure 4: Keras Classification Accuracy Summary

## 5 Challenges

Since we used different models in different frameworks, filter ablation required understanding the implementation details of these different libraries. The usage of different frameworks also affected our ability to build an efficient pipeline that can be reproduced easily, as MATLAB for example would not work in the EC2 instance due to our license limitations. This was especially a problem with the optimization routine, as it required the capabilities of a AWS compute instance. We ended up manually transferring a lot of files because of this framework discrepancy. This however, has kept an open avenue for incorporating the visualization methods in our pipeline, as the FeatureVis library is implemented in MATLAB. The biggest challenge was trying to determine whether the filter number-performance relationship and the existence of the tipping point that we observed was a genuine property of the system or an implementation quirk.

## 6 Conclusion

We have taken the proposed methods of the authors and focused on their feature extraction method. We have implemented the method on different CNN architectures, and used the authors ablation tests on the MNIST dataset. Some of our findings on the relationship between the sparsity constant and performance difference suggest an open avenue for further investigation in this topic. Overall, our reproducibility effort returned similar result to the authors' original findings in the paper.

## 7 Code

Code URL: `https://github.com/Krestone/iclr_2019_code`

## References

[1] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps, 10 2018.

[2] V. Escorcia, J. C. Niebles, and B. Ghanem. On the relationship between visual attributes and convolutional networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1256–1264, June 2015.

[3] Grn, Felix, Rupprecht, Nassir, and Federico. A taxonomy and library for visualizing learned features in convolutional neural networks, Jun 2016.

[4] Jose Oramas, Kaili Wang, and Tinne Tuytelaars. Visual explanation by interpretation: Improving visual feedback capabilities of deep neural networks. In *International Conference on Learning Representations*, 2019.

[5] Zeiler, Matthew D, Fergus, and Rob. Visualizing and understanding convolutional networks, Nov 2013.