Seminar elaboration of Learning how to explain neural networks: PatternNet and PatternAttribution

Florian Kleinicke

September 9, 2018

1 Introduction

Convolutional neural networks can be trained on multiple areas and move more and more into the focus of widely used technologies. They are excellent at detecting objects in an image, f.e. a pedestrian.
They do this by filtering out non-relevant and distracting information, the here called distractor, and keeping a signal supporting the object they found.
In certain areas, it's really important to understand the reasons behind a decision a neural network made. Therefore the authors of the paper "Learning how to explain neural networks: PatternNet and PatternAttribution" [5] focused on making the signal in an image visible with the algorithm PatternNet and tried to find the relevance of certain pixel, the attribution to the classification, with the algorithm PatternAttribution.
To get more familiar with the terminology, take a look at figure 1. In the first image in the first row, the complete data $\mathbf{x}$ is shown. In the second image, only the signal $\mathbf{s}$ is left. The non-relevant parts and noise, called distractor $\mathbf{d}$, are removed.
In the second row, the attribution is shown. The attribution indicates how relevant which part of an image is. In order to determine the attribution, some earlier approaches already exist.
The most straight forward approach is to use a simple backpropagation through the layers as described in [9].
The backpropagation determines the weights $\mathbf{w}$, that need to be applied to the pixels of the image to receive the correct classification y. The derivative of the equation $\mathbf{w}^T\mathbf{x} = y$ is $\frac{\partial y}{\partial \boldsymbol{x}} = \mathbf{w}$. Therefore this is called gradient method. The problem with this simple approach is, that it's prone to noise and highlights only roughly interesting regions.
Other approaches have been proposed. Deconvolutional networks [12] and Guided Backpropagation [11] are two methods, that are compared to PatternNet.
This paper shows that it's also possible to gain more precise information about the position of the classified object. It will even show which areas of the object are the most

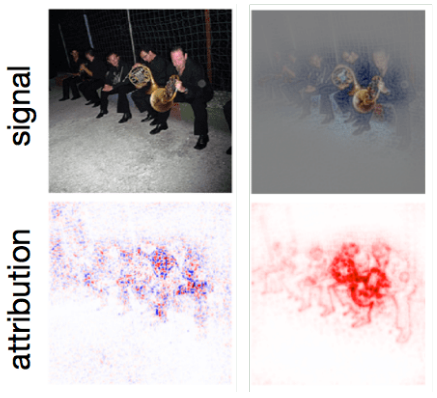

Figure 1: Terminology: Top left input data, the complete image. Top right: Signal (input data without distractor). Bottom row: Attribution. That's the estimated importance of a specific pixel for the outcome of the image classification.

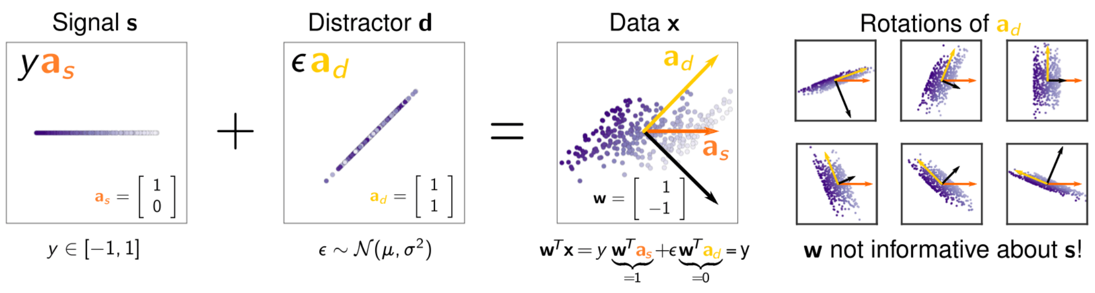
Signal **s**
$y\mathbf{a}_s$
$\mathbf{a}_s = \begin{bmatrix}1\\0\end{bmatrix}$
$y \in [-1, 1]$
Distractor **d**
$\epsilon\mathbf{a}_d$
$\mathbf{a}_d = \begin{bmatrix}1\\1\end{bmatrix}$
$\epsilon \sim \mathcal{N}(\mu, \sigma^2)$
Data **x**
$\mathbf{a}_d$
$\mathbf{a}_s$
$\mathbf{w} = \begin{bmatrix}1\\-1\end{bmatrix}$
$\mathbf{w}^T\mathbf{x} = y\,\underbrace{\mathbf{w}^T\mathbf{a}_s}_{=1} + \epsilon\,\underbrace{\mathbf{w}^T\mathbf{a}_d}_{=0} = y$
Rotations of $\mathbf{a}_d$
w not informative about **s**!

Figure 2: Linear Model

important for classification.
To develop a new approach the authors take a step back and work in a controlled environment, where it's easier to analyze, how methods work. They compare the previously mentioned methods applied on a linear model and develop an improved solution.

2 Linear Model

The goal of image classification is to determine which object is visible in an image. This task is complicated and it's hard to keep track of what is happening. To simplify the problem, the authors of the paper proposed to construct a linear data model. With such a generated model, it's easier to see where the problems of an approach are and it's possible to propose a solution that performs perfectly on such a simplified model.
The linear model consists of the complete image, the input data $\mathbf{x}$, and is decomposed into the helpful signal ($\mathbf{s}$) and a distractor ($\mathbf{d}$). The distractor contains noise and information that can not be used to classify the image.

Let's first describe a simple example distractor.

$$\mathbf{d} = \mathbf{a}_d\epsilon, \text{ with } \mathbf{a}_d = (1,1)^T \text{ and } \epsilon \sim N(\nu, \sigma^2)$$

and a signal

$$\mathbf{s} = \mathbf{a}_s y, \text{ with } \mathbf{a}_s = (1,0)^T \text{ and } y \in [-1,1]$$

$\mathbf{a}_s$ and $\mathbf{a}_d$ are the directions of their related properties. In figure 2 is shown, what such a linear model looks like graphically. A good weight vector that can separate the signal from the input data can be determined with a linear regression and is in this case $\mathbf{w} = (1,-1)^T$.

$$\mathbf{w}^\mathbf{T}\mathbf{x} = \mathbf{w}^\mathbf{T}\mathbf{s} + \mathbf{w}^\mathbf{T}\mathbf{d} = \mathbf{w}^\mathbf{T}\mathbf{a}_s y + \mathbf{w}^\mathbf{T}\mathbf{a}_d\epsilon = 1y + 0\epsilon = y$$

The goal of a learning algorithm is to find the best fitting $\mathbf{w}$. For the linear case, this is always possible, as long as the directions of the signal and the distractor are not parallel. The filter/weight vector is highly dependent on the direction of the distractor. Figure 2 shows in every image the same signal vector and with different distractor directions. The weight vector is always orthogonal to the distractor, which leads to a strongly changing weight vector. When the weight vector is used to determine the important areas in the image, the result would be highly dependent on the distractor. The direction of the signal $\mathbf{a}_s$ only alters the amplitude of the filter, so that $\mathbf{w}\mathbf{a}_s = 1$. Therefore the gradient method, which only highlights the weight vector, won't lead to good results.
In the here summarized paper, the authors propose a more sophisticated approach, that leads to ideal results for this linear model, and still lead to really good results applied on a nonlinear image. In natural images there is no linear distractor, that can be found. The distractor is a composition of Gaussian noise and irrelevant information that is displayed in the image. To detect the relevant information, a way to find the signal direction $\mathbf{a}_s$ has to be learned.

3 Signal estimators and attribution

It's essential to differentiate between the relevant signal of an image and its distractor that contains the not relevant parts. In the first row of figure 1 the difference between an image and its relevant information is shown. The signal can later be utilized to determine the attribution. The attribution is the importance of a certain pixel on the outcome of the classification. As shown in the last section for a linear model, the filter is highly dependent on the distractor. Therefore it's essential for determining the attribution, to have the signal separated from the distractor. The attribution is computed with a point-wise multiplication of the weight vector (filter) with the estimated signal.
In this section an identity estimator, which interprets the entire image as the signal, and the filter-based estimator are introduced. For better performance on a linear model a linear estimator and a for convolutional neural networks adapted version, a two-component estimator, are proposed. To understand the calculations some basic statistics is required.

3.1 Statistics

Expectation value The expectation value E describes the mean value of a distribution. A constant factor that multiplies the random variable can be put in front of the expectation value.

$$E[ax] = a\,E[x]$$

Variance The variance describes the expectation of the squared deviation of a random variable from its mean. At the same time, it's the square of the standard deviation.

$$var(X) := E[(x - \mu)^2] = \sigma_x^2$$

Covariance The covariance describes the joint variability of two variables. If the two variables are independent, the covariance is 0.

$$cov(X, Y) = E[(x - \mu_x)(y - \mu_y)] \Rightarrow cov(X, X) = var(X)$$

$$cov(X, Y) = E[\mathbf{xy}] - E[\mathbf{x}]E[\mathbf{y}]$$

Correlation The correlation normalizes the Covariance to the range -1 to 1.

$$\rho(X, Y) := \frac{cov(X, Y)}{\sigma_x \sigma_y}$$

Two variables, that are highly correlated tend to be positive.
For example $\rho(X, X) = \frac{cov(X,X)}{\sigma_x^2} = 1$. When two variables are uncorrelated, their correlation tends to be 0. But it doesn't mean that when the correlation is 0, that the variables are uncorrelated.
The correlation is used in the next chapter for the quality measurement.

3.2 Identity estimator S_x

The identity estimator S_x is interpreting the complete image as the signal, ignoring the existence of a distractor. This means formally: $S_x(\mathbf{x}) = \mathbf{x}$. When the attribution is computed it is applied to the signal and the distractor.

$$\mathbf{r} = \mathbf{w} \bigodot S_x = \mathbf{w} \bigodot \mathbf{s} + \mathbf{w} \bigodot \mathbf{d}$$

With $\bigodot$ as the element wise multiplicator.
This leads to noisy outputs, since the distractor is not filtered out at all.

3.3 Filter-based estimator S_w

The filter-based estimator S_w is implicitly assumed by the Deconvolution Network and Guided Backpropagation approach. It is more complex than the identity estimator.

$$S_w(\mathbf{x}) = \frac{\mathbf{w}}{\mathbf{w}^T\mathbf{w}}\mathbf{w}^T\mathbf{x} = \frac{1}{\mathbf{w}^T\mathbf{w}}\mathbf{w}y$$

That leads to the attribution

$$\mathbf{r} = \mathbf{w} \bigodot S_w(\mathbf{x}) = \frac{\mathbf{w} \bigodot \mathbf{w}}{\mathbf{w}^T\mathbf{w}}y$$

This attribution delivers much better estimations, but still doesn't even manage to reconstruct the optimal solution for the linear example. The estimated signal is still dependent on the weight vector, which is dependent on the distractor. This leads the authors of this paper to suggest a different estimator.

3.4 Linear estimator S_a

In the linear model, the distractor should be 0. Therefore $d = x - S(x) = 0$ which means reformulated with co-variances

$$\text{For } cov[y, \mathbf{d}] = 0 \Rightarrow cov[\mathbf{x}, y] = cov[S(\mathbf{x}), y]$$

For a linear model the signal estimator applied on the image should result in the signal

$$S_a(\mathbf{x}) = \mathbf{a}_s\mathbf{w}^T\mathbf{x} \ \ (= \mathbf{a}_s y) = \mathbf{s}.$$

For this model S_a performs perfectly. To determine $\mathbf{a}_s$ the covariance term can be used

$$cov[\mathbf{x}, y] = cov[\mathbf{a}_s\mathbf{w}^T\mathbf{x}, y] = \mathbf{a}_s cov[y, y] = \mathbf{a}_s\sigma_y^2 \Rightarrow \mathbf{a}_s = \frac{cov[\mathbf{x}, y]}{\sigma_y^2}.$$

To determine $\mathbf{a}_s$, the algorithm has to be trained to be able to determine the covariance for every given data x. Its attribution is

$$\mathbf{r} = \mathbf{w} \bigodot \mathbf{a}_s y.$$

Since the classifier should be used on nonlinear neural networks and extended approach is introduced.

3.5 Two-component estimator S_{a+-}

The estimator is later not used to estimate the signal in a linear model but inside a convolutional neural network. Therefore the structure of the estimator should be adapted to the structure of the neural network. The convolution layer itself is linear, but nonlinearity is added through the ReLU layers. Therefore a two-component estimator

S_{a+-} is proposed, that acts similar to a ReLU different for a positive and a negative activation. Therefore the model $\mathbf{x} = \mathbf{s} + \mathbf{d}$ is extended into two components.

$$\mathbf{x} = \begin{cases} \mathbf{s}_+ + \mathbf{d}_+ & \text{if } y > 0 \\ \mathbf{s}_- + \mathbf{d}_- & \text{otherwise} \end{cases}$$

Which leads to a two-component signal estimator.

$$S_{\mathbf{a}+-}(\mathbf{x}) = \begin{cases} \mathbf{a}_+ \mathbf{w}^T \mathbf{x} & \text{if } \mathbf{w}^T \mathbf{x} > 0 \\ \mathbf{a}_- \mathbf{w}^T \mathbf{x} & \text{otherwise} \end{cases}$$

To determine $\mathbf{a}_+$ and $\mathbf{a}_-$ a similar procedure as for the linear estimator is used. This time it's only slightly more complicated, because there are more parameters involved. The in section 3.1 mentioned reformulation of the covariance is used.

$$cov[\mathbf{p}, \mathbf{q}] = E[\mathbf{p}\mathbf{q}] - E[\mathbf{p}]E[\mathbf{q}]$$

The following computation is limiting itself to the positive case $\mathbf{a}_+$. For $\mathbf{a}_-$ the calculation can be done analogously.

$$\begin{aligned} cov[\mathbf{x}_+, y] &= cov[S(\mathbf{x})_+, y] \\ \Rightarrow E_+[\mathbf{x}y] - E_+[\mathbf{x}]E_+[y] &= E_+[S(\mathbf{x})y] - E_+[S(\mathbf{x})]E_+[y] \end{aligned}$$

With the positive signal estimator $S_{\mathbf{a}+}(\mathbf{x}) = \mathbf{a}_+ \mathbf{w}^T \mathbf{x}$

$$\Rightarrow E_+[\mathbf{x}y] - E_+[\mathbf{x}]E_+[y] = E_+[\mathbf{a}_+ \mathbf{w}^T \mathbf{x} y] - E_+[\mathbf{a}_+ \mathbf{w}^T \mathbf{x}]E_+[y].$$

Since the expectation value doesn't depend on $\mathbf{a}_+$ and $\mathbf{w}$ these terms can be put in front of the expectation value. This means the equation can be refactored to $\mathbf{a}_+$

$$\mathbf{a}_+ = \frac{E_+[\mathbf{x}y] - E_+[\mathbf{x}]E_+[y]}{\mathbf{w}^T E_+[\mathbf{x}y] - \mathbf{w}^T E_+[\mathbf{x}]E_+[y]}.$$

Similar to the linear case the expected parameters have to be learned by applying a linear regression model.

3.6 Deep Taylor Decomposition and PatternAttribution

To compute the attribution with this estimator, a deep Taylor decomposition (DTD) [7] is used. This method takes the inputs of a neuron and determines the importance of their contributions. To do so, it uses a first order Taylor expansion around root point x_0 with $\boldsymbol{w}^T \boldsymbol{x}_0 = 0$. It's an algorithm that propagates backward through the layers. In the last layer it's $r_i^{output} = y$ for the correct output class and otherwise $r_{j \neq i}^{output} = 0$.

$$\mathbf{r}^{l-1,i} = \frac{\mathbf{w} \bigodot (\mathbf{x} - \mathbf{x}_0)}{\mathbf{w}^T \mathbf{x}} r_i^l$$

With r_i^l as the attribution in the ith neuron of the lth layer. $\mathbf{r}^{l-1,i}$ is a vector of all contributions of a single neuron r_i^l on the neurons in the previous layer. These contributions

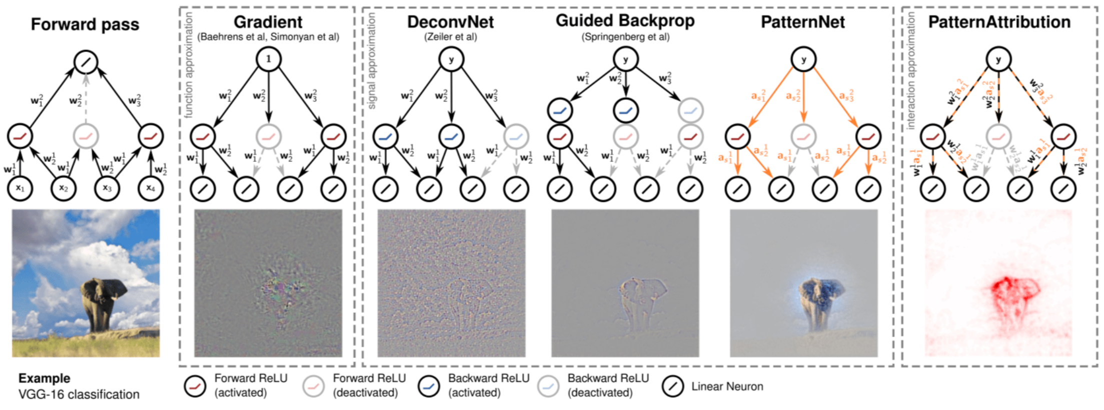

Figure 3: Architecture of compared approaches. For the gradient method, a saliency map is shown, for the other three methods the estimated signal is shown. The architecture of PatternAttribution and the Forward pass are shown for comparison.

have to be summed up, to get the attribution of a single neuron in the lower layer.
When a ReLU isn't activated on a forward pass, the re-distribution on the backward pass is stopped and it's save to assume that $\boldsymbol{w}^T\boldsymbol{x} > 0$.
The extension around x_0 is used by the authors. When it's extended around 0, this method is equal to the layer-wise relevance propagation (LRP) [2].
For the proposed PatternAttribution algorithm the Deep Taylor Decomposition is extended around the distractor. The negative component of the two-component estimator, isn't required, since the ReLU's stopped the negative attribution in the forward pass.
$\mathbf{x}_0 = \mathbf{d} = \mathbf{x} - S(\mathbf{x})_{+-} = \mathbf{x} - \mathbf{a}_+\mathbf{w}^T\mathbf{x}$ included into the formula:

$$\mathbf{r}^{l-1,i} = \frac{\mathbf{w} \odot (\mathbf{x} - \mathbf{x} - \mathbf{a}_+\mathbf{w}^T\mathbf{x})}{\mathbf{w}^T\mathbf{x}} r_i^l = \mathbf{w} \odot \mathbf{a}_+ r_i^l$$

In comparison, the gradient method uses a backward propagation with

$$\mathbf{r}^{l-1,i} = \mathbf{w} r_i^l$$

3.7 Compared Approaches

After giving the theoretical background behind the signal estimators, let's compare how these are used in the different algorithms. Figure 3 compares 4 different approaches and their effectiveness to abstract the signal.
The first method the gradient method [10] simply takes the backward pass through the network and determines the most relevant areas, by comparing its influence on the outcome. It uses the identity estimator S_x, so the distractor isn't canceled out. The displayed result is a saliency map, and not the signal itself (which would be the complete image). The outcome is really noisy with a focus in the area of the elephant.
The second method, DeconvNet [12] use implicitly the filter-based estimator S_w. To

compute the backward pass it exchanges the forward ReLU's used in the original neural network with backward ReLU's. These backward ReLU's become 0 when the backward propagation at this place becomes negative. The result seems to be noisy with no focus on the area of the elephant. Therefore some minor structures of the elephant are visible. The outcome is the signal, normalized to maximize the contrast for better visibility. Nevertheless, it's hard to see the structures.
The third method, Guided Backpropagation [11] also uses implicitly the filter-based signal estimator. To compute the backward pass it keeps the forward ReLU's for the backpropagation in place and adds additional backward ReLU's. This leads so much less noise and to some visible structures of the elephant.
PatternNet uses for backpropagation the signal direction instead of the weight vector. This way not only the contours, but a detailed elephant is visible in the output.
On the right side, the outcome of PatternAttribution is shown.

4 Quality measurements

To get a signal estimator is an ill-posed problem with multiple differently scaled solutions. To find the best solution a quality estimator is used for optimization. The quality estimator turns to be 1 when the optimal result is reached is formulated. A solution is desired with no correlation between the distractor and the output. This would mean, that no relevant information is left in the distractor and the signal estimator is optimal. Therefore a vector v is multiplied on the distractor to maximize the visibility of these correlations.

$$\rho(S) = 1 - \max_{\mathbf{v}} corr[\mathbf{v}^T(\mathbf{x} - S(\mathbf{x})), \mathbf{w}^T\mathbf{x}] = 1 - \max_{\mathbf{v}} corr[\mathbf{v}^T\mathbf{d}, y]$$

Using this definition of the correlation

$$corr(\mathbf{x}, \mathbf{y}) = \frac{cov(\mathbf{p}, \mathbf{q})}{\sqrt{\sigma_{\mathbf{p}}^2 \sigma_{\mathbf{q}}^2}}$$

the final equation gets

$$\rho(S) = 1 - \max_{\mathbf{v}} \frac{\mathbf{v}^T cov(\mathbf{d}, y)}{\sqrt{var(\mathbf{v}^T\mathbf{d}) var(\mathbf{y})}}$$

The equation further can further be simplified by the assumption $var(\mathbf{v}^T\mathbf{d}) = var(\mathbf{y})$. To compute the covariance and the v, a least-squares regression is used.

5 Experiments and Results

The proposed approaches were implemented using the Lasagne library [4]. "Lasagne is a lightweight library to build and train neural networks in Theano." Theano [1] is a Python library, developed mainly by the University of Montreal, that can be used for

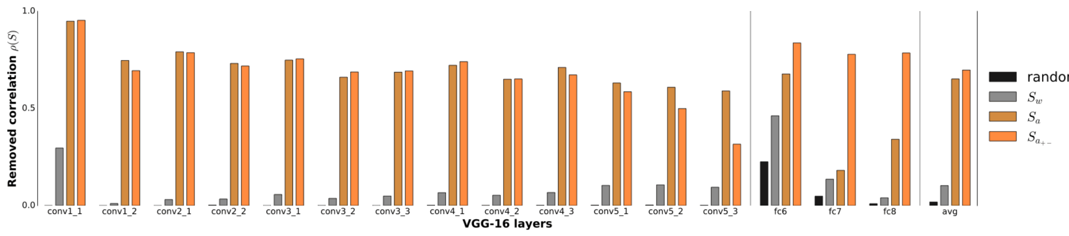

Figure 4: Evaluation of the Quality estimator $\rho(S)$ in certain layers of the VGG16 network. A random signal estimator was chosen as the baseline.

deep learning.
To be comparable with other papers, ImageNet [3], a widely used dataset of images, with labeled objects, was used for training and testing. All images were cropped to 224 times 224 pixels.
The authors used a VGG16 [8] network that was already pretrained on ImageNet for performance reasons. They split the training dataset into two equally sized sets. The signal estimators were trained on the first half of the training set, and the quality estimator on the second half. The validation of the training was run on the official validation dataset with 50000 samples.
They trained the linear and the two-component estimator, which takes 3-4 hours on 4 GPU's.
To train the quality estimator, it would require them to compute it for every single weight vector in the network. To speed this process up, they developed an equivalent least-squares problem using stochastic mini-batch gradient descent with the ADAM [6] optimizer until convergence. This still took 1 day on a Tesla K40.
These periods seem long, but they only have to be done for each signal estimator and quality estimator whenever the network or training set is exchanged. In comparison to the training for the classification of the neural network, which takes several days, these times are acceptable. Therefore the usage of the trained quality estimator and the signal estimator are computationally cheap.

In figure 4 they compare the result of the quality estimator for different signal estimators. As a baseline they use a random estimator, that randomly chooses a signal and distractor. A random signal estimator leads to a very bad quality in every single layer of the VGG16 network. This is exactly what is expect. The filter-based estimator S_w performs better but still not really good. It's fine in the first layer of the convolutional network and the fully connected network, but getting worse in the deeper layers. This is probably because the signal gets more and more non-linear. The linear and two-component estimator perform much better. Interestingly the linear estimator performs in the fully connected layers even better than the two-component estimator. This effect still needs to be understood. Due to much better performance in the fully connected layers, the two-component estimator performs overall better.

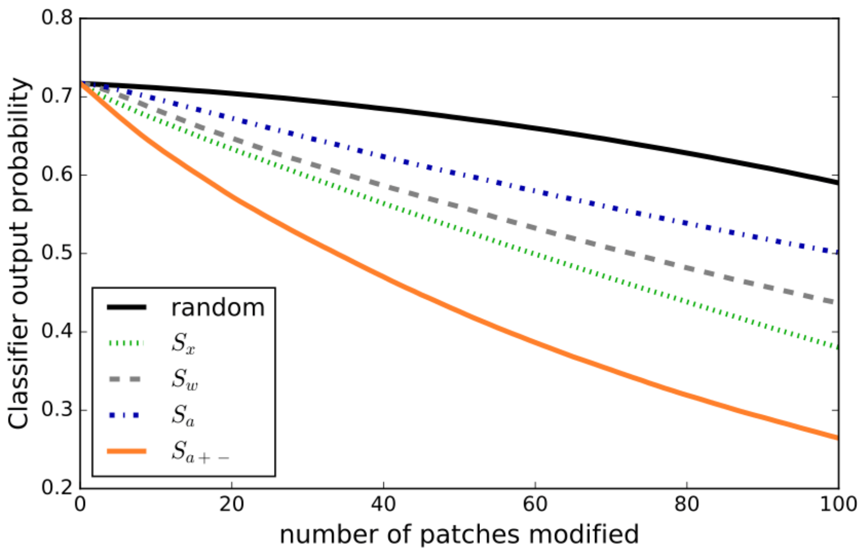

Figure 5: The different attributions ordered importance of 9×9 pixel patches. The most important ones were averaged. The steeper the degeneration of classification success, the better was the ordering of the method.

The quality estimator is only one indicator of the overall quality. A good signal estimator manages to find the most relevant areas of a signal. When the most relevant areas of a signal get disturbed, the classification task of the network gets much harder. In figure 5 the most relevant image patches in the size of 9×9 pixel get averaged. It's clearly shown, that the two-component estimator is best suited to find the most relevant image patches. The probability of the correct output drops from about 72 to 59 percent, after just 20 modified patches. When the patches are chosen randomly, this drop is below 1 %. The other methods are in between these two estimators. The linear estimator performs here even worse than the remaining estimators. This is because it can only represent linear information, so the ordering is not perfect.
Interestingly the identity estimator leads to a better result than the filter-based estimator, despite the fact that the filter-based estimator is an improved version of the identity estimator. The filter-based estimator seems to be relatively good to find the most relevant areas, but classifies also multiple not relevant patches as relevant.

To compare the results qualitatively, the different outputs of the signal estimators are compared in figure 6. In the first row, the outputs of the different estimators are compared. The identity estimator classifies the whole image as the signal, for the filter-based estimator the outcome is relatively noisy and not very detailed. With the linear estimator, some contours can be seen and the two-component estimator nicely selects the important areas of the image. The area and color of the important parts can nicely bee seen.
In the second row, the attribution for the four classifiers is shown. For the identity

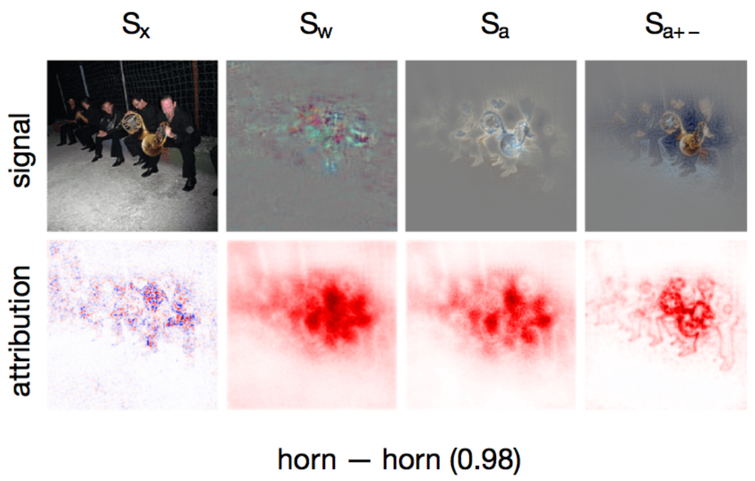

Figure 6: The identity estimator S_x, the filter-based estimator S_w, the linear estimator S_a and the two-component estimator S_{a+-} compare their estimated signal (first row) and their pixel-wise attribution (second row).

estimator, the attribution is really noisy but the most relevant areas of the image can be approximately estimated. The filter-based estimator points out the most important area of the image but doesn't show any details. The linear estimator is more precise, but still, there are barely any details. The attribution of the two-component estimator nicely shows what parts of the objects in the image are the most relevant. In this case, its main focus is on the corners of the music instrument.

In figure 7 the signal estimation and the attribution is shown for the different estimators, with multiple images. The previously observed problems and limitations can also be seen here. It's interesting to see, that according to the PatternNet algorithm for classification of the mailbox, the person next to it was important. The person itself had it's most important areas where the eyes and the knees are.

6 Summary and Discussion

The paper analyses the problem of finding the most relevant parts of an image for a classification decision from two new angles. At first, it starts with a linear model, which can be solved perfectly, but wasn't in the focus of the other authors yet. Secondly, it trains a linear regression model to perform well with the data. It sounds like a good idea to train an algorithm to get better insights into trained neural networks. Maybe this idea can be even improved by training a neural network to highlight the most relevant areas of the image and gain insights into neural networks.
The method was adapted to work well on convolutional neural networks that make use of ReLU's to gain nonlinearities. When it should be applied to a completely different looking architecture, this approach should be updated.

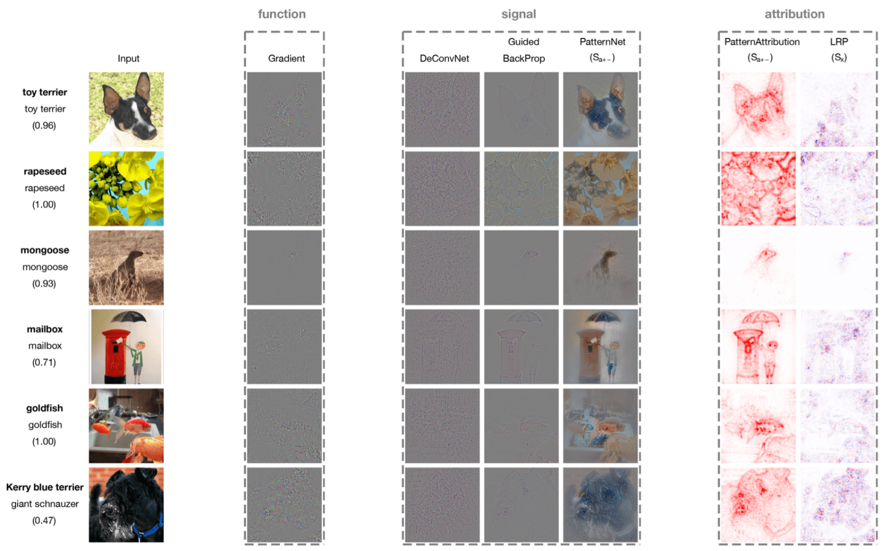

Figure 7: Some more images for comparison between the proposed methods. The function and signal methods are back-projections to input space with their original color channels and were normalized to maximize contrast. The attribution methods are heat maps showing pixel-wise attribution.

The result is much better than the compared approaches. It's hard to judge for me, whether it was compared with the current state of the art approaches of this field, or if there was already some cherry picking. The resulting images they produced with the existing approaches are authentically and look similar to the results in the original papers. Nevertheless, it's not surprising, that their method performs much better on a linear model than the other approaches. It's easy to construct a simple toy model, where other approaches fail, but a new specially optimized algorithm performs very well.
I didn't like a few points in the paper. The formula 5 in their paper is:

$$cov[\boldsymbol{x}, y] = \pi_+ \left(\mathbb{E}_+[\boldsymbol{x}y] - \mathbb{E}_+[\boldsymbol{x}]\mathbb{E}[y]\right) + (1 - \pi_+) \left(\mathbb{E}_-[\boldsymbol{x}y] - \mathbb{E}_-[\boldsymbol{x}]\mathbb{E}[y]\right)$$

π_+ is the expected ratio of $\boldsymbol{w}^T\boldsymbol{x} > 0$ and is immediately dropped again. This formula and formula 6 weren't necessary to explain the following formula. They should have used the formula

$$cov[\mathbf{x}_+, y] = cov[S(\mathbf{x})_+, y]$$

like I did. From this easy to understand formula follow their further formulas.
Overall the results were a huge improvement over the existing methods and a nice approach.

References

[1] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, Yoshua Bengio, Arnaud Bergeron, James Bergstra, Valentin Bisson, Josh Bleecher Snyder, Nicolas Bouchard, Nicolas Boulanger-Lewandowski, Xavier Bouthillier, Alexandre de Brébisson, Olivier Breuleux, Pierre-Luc Carrier, Kyunghyun Cho, Jan Chorowski, Paul Christiano, Tim Cooijmans, Marc-Alexandre Côté, Myriam Côté, Aaron Courville, Yann N. Dauphin, Olivier Delalleau, Julien Demouth, Guillaume Desjardins, Sander Dieleman, Laurent Dinh, Mélanie Ducoffe, Vincent Dumoulin, Samira Ebrahimi Kahou, Dumitru Erhan, Ziye Fan, Orhan Firat, Mathieu Germain, Xavier Glorot, Ian Goodfellow, Matt Graham, Caglar Gulcehre, Philippe Hamel, Iban Harlouchet, Jean-Philippe Heng, Balázs Hidasi, Sina Honari, Arjun Jain, Sébastien Jean, Kai Jia, Mikhail Korobov, Vivek Kulkarni, Alex Lamb, Pascal Lamblin, Eric Larsen, César Laurent, Sean Lee, Simon Lefrancois, Simon Lemieux, Nicholas Léonard, Zhouhan Lin, Jesse A. Livezey, Cory Lorenz, Jeremiah Lowin, Qianli Ma, Pierre-Antoine Manzagol, Olivier Mastropietro, Robert T. McGibbon, Roland Memisevic, Bart van Merriënboer, Vincent Michalski, Mehdi Mirza, Alberto Orlandi, Christopher Pal, Razvan Pascanu, Mohammad Pezeshki, Colin Raffel, Daniel Renshaw, Matthew Rocklin, Adriana Romero, Markus Roth, Peter Sadowski, John Salvatier, François Savard, Jan Schlüter, John Schulman, Gabriel Schwartz, Iulian Vlad Serban, Dmitriy Serdyuk, Samira Shabanian, Étienne Simon, Sigurd Spieckermann, S. Ramana Subramanyam, Jakub Sygnowski, Jérémie Tanguay, Gijs van Tulder, Joseph Turian, Sebastian Urban, Pascal Vincent, Francesco Visin, Harm de Vries, David Warde-Farley, Dustin J. Webb, Matthew Willson, Kelvin Xu, Lijun Xue, Li Yao, Saizheng Zhang, and Ying Zhang. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[2] A. Binder, G. Montavon, S. Bach, K.-R. Müller, and W. Samek. Layer-wise Relevance Propagation for Neural Networks with Local Renormalization Layers. *ArXiv e-prints*, April 2016.

[3] Jia Deng, Wei Dong, Richard Socher, Li jia Li, Kai Li, and Li Fei-fei. Imagenet: A large-scale hierarchical image database. In *In CVPR*, 2009.

[4] Sander Dieleman, Jan Schlüter, Colin Raffel, Eben Olson, Søren Kaae Sønderby, Daniel Nouri, Daniel Maturana, Martin Thoma, Eric Battenberg, Jack Kelly, Jeffrey De Fauw, Michael Heilman, Diogo Moitinho de Almeida, Brian McFee, Hendrik Weideman, Gábor Takács, Peter de Rivaz, Jon Crall, Gregory Sanders, Kashif Rasul, Cong Liu, Geoffrey French, and Jonas Degrave. Lasagne: First release., August 2015.

[5] P.-J. Kindermans, K. T. Schütt, M. Alber, K.-R. Müller, D. Erhan, B. Kim, and S. Dähne. Learning how to explain neural networks: PatternNet and PatternAttribution. *ArXiv e-prints*, May 2017.

[6] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *ArXiv e-prints*, December 2014.

[7] G. Montavon, S. Bach, A. Binder, W. Samek, and K.-R. Müller. Explaining NonLinear Classification Decisions with Deep Taylor Decomposition. *ArXiv e-prints*, December 2015.

[8] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[9] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013.

[10] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013.

[11] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.

[12] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535, June 2010.