VisualBackProp: visualizing CNNs for autonomous driving

Mariusz Bojarski [1] Anna Choromanska [2] Krzysztof Choromanski [3]
Bernhard Firner [1] Larry Jackel [1] Urs Muller [1] Karol Zieba [1]

Abstract

This paper proposes a new method, that we call VisualBackProp, for visualizing which sets of pixels of the input image contribute most to the predictions made by the convolutional neural network (CNN). The method heavily hinges on exploring the intuition that the feature maps contain less and less irrelevant information to the prediction decision when moving deeper into the network. The technique we propose was developed as a debugging tool for CNN-based systems for steering self-driving cars and is therefore required to run in real-time, i.e. it was designed to require less computations than a forward propagation. This makes the presented visualization method a valuable debugging tool which can be easily used during both training and inference. We furthermore justify our approach with theoretical arguments and theoretically confirm that the proposed method identifies sets of input pixels, rather than individual pixels, that collaboratively contribute to the prediction. Our theoretical findings stand in agreement with the experimental results. The empirical evaluation shows the plausibility of the proposed approach on the road video data and reveals that it compares favorably to the LRP approach, i.e. it obtains similar visualization results and simultaneously achieves order of magnitude speed-ups.

1. Introduction

A plethora of important real-life problems are currently addressed with CNNs (LeCun et al., 1989), including image recognition (Krizhevsky et al., 2012), speech recognition (Abdel-Hamid et al., 2012), and natural language processing (Weston et al., 2014). More recently they were successfully used in the complex intelligent autonomous systems such as self-driving cars (Bojarski et al., 2016; Chen et al., 2015). One of the fundamental question that arises when considering CNNs as well as other deep learning models is: *what made the trained neural network model arrive at a particular response?* This question is of particular importance to the end-to-end systems, where the interpretability of the system is limited. Visualization tools aim at addressing this question by identifying parts of the input image that had the highest influence on forming the final prediction by the network. It is also straightforward to think about visualization methods as a debugging tool that helps to understand if the network detects "reasonable" cues from the image to arrive at a particular decision.

The visualization method for CNNs proposed in this paper was originally developed for CNN-based systems for steering autonomous cars, though it is highly general and can be used in other applications as well. The method relies on the intuition that when moving deeper into the network, the feature maps contain less and less information which are irrelevant to the output. Thus, the feature maps of the last convolutional layer should contain the most relevant information to determine the output. At the same time, feature maps of deeper layers have lower resolution. The underlying idea of the approach is to combine feature maps containing only relevant information (deep ones) with the ones with higher resolution (shallow ones). In order to do so, starting from the feature maps of the last convolutional layer, we "backpropagate" the information about the regions of relevance while simultaneously increasing the resolution, where the backpropagation procedure is not gradient-based (as is the case for example in sensitivity-based approaches (Baehrens et al., 2010; Simonyan et al., 2014; Rasmussen et al., 2012)), but instead is value-based. We call this approach VisualBackProp.

Our method provides a general tool for verifying that the predictions generated by the neural network[2], are based on reasonable optical cues in the input image. In case of autonomous driving these can be lane markings, other cars, or edges of the road. Our visualization tool runs in real time and requires less computations than forward propagation. We empirically demonstrate that it is order of mag-

[1]NVIDIA Corp. [2]ECE NYU Tandon [3]Google Brain Robotics. Correspondence to: Mariusz Bojarski <mbojarski@nvidia.com>, Anna Choromanska <ac5455@nyu.edu>, Krzysztof Choromanski <kchoro@google.com>.

[2]In the case of our end-to-end system for steering an autonomous car, a prediction is the steering wheel angle.

nitude faster than the state-of-the-art visualization method, LRP (Bach et al., 2015), while at the same time it leads to very similar visualization results.

In the theoretical part of this paper we first provide a rigorous mathematical analysis of the contribution of input neurons to the activations in the last layer that relies on network flows. We propose a quantitative measure of that contribution. We then show that our algorithm finds for each neuron the approximated value of that measure. To the best of our knowledge, the majority of the existing visualization techniques for deep learning, which we discuss in the Related Work section, lack theoretical guarantees, which instead we provide for our approach. This is yet another important contribution of this work.

The paper is organized as follows: Section 2 discusses related work, Section 3 describes our approach and Section 4 provides its theoretical analysis, Section 5 presents empirical results, and Section 6 contains final remarks.

2. Related work

A notable approach (Bach et al., 2015) addressing the problem of understanding classification decisions by pixel-wise decomposition of non-linear classifiers proposes a methodology called layer-wise relevance propagation (LRP), where the prediction is back-propagated without using gradients such that the relevance of each neuron is redistributed to its predecessors through a particular message-passing scheme. The method relies on the conservation principle which ensures that the network output activity is fully redistributed through the layers of a neural network model onto the input variables. This approach allows the visualization of the contribution of single pixels to the predictions of a previously trained network as heatmaps. Its stability and the sensitivity to different settings of the conservation parameters was studied in the context of several deep learning models (Binder et al., 2016). The LRP technique was later extended to Fisher Vector classifiers (Bach et al., 2016) and also used to explain predictions of CNNs in NLP applications (Arras et al., 2016). An extensive comparison of LRP with other visualization techniques, like the deconvolution method (Zeiler & Fergus, 2014) and the sensitivity-based approach (Simonyan et al., 2014), which we also discuss next in this section, using an evaluation based on region perturbation can be found in (Samek et al., 2016). This study reveals that LRP provides both qualitatively and quantitatively better explanation of the DNN classification decisions than considered competitors.

Another approach (Zeiler & Fergus, 2014) for understanding CNNs with max-pooling and rectified linear units (ReLUs) through visualization uses deconvolutional neural network (Zeiler et al., 2011) attached to the convolutional network of interest. This approach maps the feature activity in intermediate layers of a previously trained CNN back to the input pixel space using deconvolutional network, which performs successively repeated operations of i) unpooling, where switch variables are used to record the locations of maxima within each pooling region, ii) rectification, and iii) filtering. Since this method identifies structures within each patch that stimulate a particular feature map, it differs from previous approaches (Girshick et al., 2014) which instead identify patches within a data set that stimulate strong activations at higher layers in the model. The method can also be interpreted as providing an approximation to partial derivatives with respect to pixels in the input image (Simonyan et al., 2014). One of the shortcomings of the method is that it enables the visualization of only a single activation in a layer (all other activations are set to zero). There also exist other techniques for inverting a modern large convolutional network with another network, e.g. a method based on up-convolutional architecture (Dosovitskiy & T.Brox, 2016), where as opposed to the previously described deconvolutional neural network, the up-convolutional network is trained. This method inverts deep image representations and obtains reconstructions of an input image from each layer.

The fundamental difference between the LRP approach and the deconvolution method lies in how the responses are projected towards the inputs. The latter approach solves the optimization problems to reconstruct the image input while the former one aims to reconstruct the classifier decision (the details are well-explained in (Bach et al., 2015)).

Guided backpropagation (Springenberg et al., 2015) extends the deconvolution approach by combining it with a simple technique visualizing the part of the image that most activates a given neuron using a backward pass of the activation of a single neuron after a forward pass through the network. Finally, the recently published method (Zintgraf et al., 2017) based on the prediction difference analysis (Robnik-Sikonja & Kononenko, 2008) is a probabilistic approach that extends the idea in (Zeiler & Fergus, 2014) of visualizing the probability of the correct class using the occlusion of the parts of the image. The approach highlights the regions of the input image of a CNN which provide evidence for or against a certain class.

Understanding CNNs can also be done by visualizing output units as distributions in the input space via output unit sampling (Hinton et al., 2006). However, computing relevant statistics of the obtained distribution is often difficult. This technique cannot be applied to deep architectures based on auto-encoders as opposed to the subsequent work (Erhan et al., 2010; 2009). In their work the authors visualize what is activated by the unit in an arbitrary layer of a CNN in the input space (of images) via an activation maximization procedure that looks for input patterns of a bounded norm that maximize the activation of a given hidden unit using gradient ascent. This method extends previous approaches (Berkes & Wiskott, 2006). The gradient-based visualization method (Erhan et al., 2009) can also be viewed as a generalization of the deconvolutional network reconstruction procedure (Zeiler & Fergus, 2014) as shown in subsequent work (Simonyan et al., 2014). The requirement of careful initialization limits the method (Zeiler & Fergus, 2014). The approach was applied to Stacked Denoising Auto-Encoders, Deep Belief Networks and later on to CNNs (Simonyan et al., 2014). The latter work also proposes a technique for computing a class saliency map from a CNN, specific to a given image and class. Instead of projecting back convolutional features to the input space as in (Zeiler & Fergus, 2014), they perform the projection from the fully connected layers of the network. The method described in this work belongs to a broader family of sensitivity-based methods (other members of this family include (Baehrens et al., 2010; Rasmussen et al., 2012), which aim to understand how the classifier works in different parts of the input domain by computing scores based on partial derivatives at the given sample.

Some more recent gradient-based visualization techniques for CNN-based models not mentioned before include Grad-CAM (Selvaraju et al., 2016), which is an extension of the Class Activation Mapping (CAM) method (Zhou et al., 2016). The approach heavily relies on the construction of weighted sum of the feature maps, where the weights are global-average-pooled gradients obtained through back-propagation. The approach lacks the ability to show fine-grained importance like pixel-space gradient visualization methods (Springenberg et al., 2015; Zeiler & Fergus, 2014) and thus in practice has to be fused with these techniques to create high-resolution class-discriminative visualizations.

Other approaches for analyzing neural networks include quantifying variable importance in neural networks (Gevrey et al., 2003; Olden et al., 2004), extracting the rules learned by the decision tree model that is fitted to the function learned by the neural network (Setiono & Liu, 1995), applying kernel analysis to understand the layer-wise evolution of the representation in a deep network (Montavon et al., 2011), analyzing the visual information in deep image representations by looking at the inverse representations (Mahendran & Vedaldi, 2015), applying contribution propagation technique to provide per-instance explanations of predictions (Landecker et al., 2013) (the method relies on the technique of (Poulin et al., 2006), or visualizing particular neurons or neuron layers (Krizhevsky et al., 2012; Yosinski et al., 2015). Finally, there also exist more generic tools for explaining individual classification decisions of any classification method for single data instances, like for example (Baehrens et al., 2010).

3. Visualization method

As mentioned before, our method combines feature maps from deep convolutional layers that contain mostly relevant information, but are low-resolution, with the feature maps of the shallow layers that have higher resolution but also contain more irrelevant information. This is done by "back-propagating" the information about the regions of relevance while simultaneously increasing the resolution. The back-propagation is value-based. We call this approach VisualBackProp to emphasize that we "back-propagate" values (images) instead of gradients. We explain the method in details below.

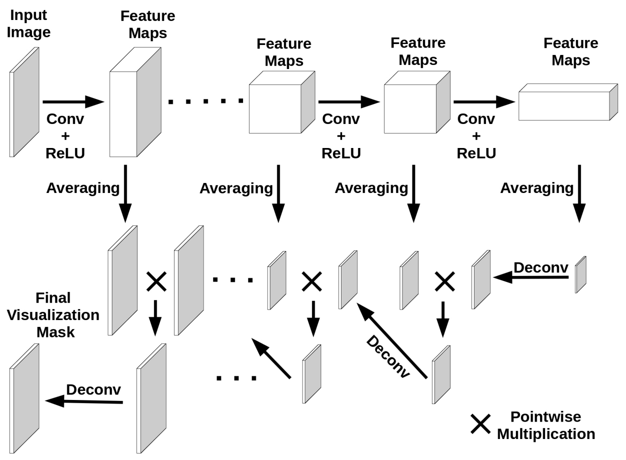

Figure 1. Block diagram of the VisualBackProp method. The feature maps after each ReLU layer are averaged. The averaged feature map of the last convolutional layer is scaled-up via deconvolution and multiplied by the averaged feature map from the previous layer. The resulting intermediate mask is again scaled-up and multiplied. This process is repeated until we reach the input.

The block diagram of the proposed visualization method is shown in Figure 1. The method utilizes the forward propagation pass, which is already done to obtain a prediction, i.e. we do not add extra forward passes. The method then uses the feature maps obtained after each ReLU layer (thus these feature maps are already thresholded). In the first step, the feature maps from each layer are averaged, resulting in a single feature map per layer. Next, the averaged feature map of the deepest convolutional layer is scaled up to the size of the feature map of the previous layer. This is done using deconvolution with filter size and stride that are the same as the ones used in the deepest convolutional layer (for deconvolution we always use the same filter size and stride as in the convolutional layer which outputs the feature map that we are scaling up with the deconvolution). In deconvolution, all weights are set to 1 and biases to 0. The obtained scaled-up averaged feature map is then point-wise multiplied by the averaged feature map from the previous layer. The resulting image is again scaled via deconvolution and multiplied by the averaged feature map of the previous layer exactly as described above. This process continues all the way to the network's input as shown in Figure 1. In the end, we obtain a mask of the size of the input image, which we normalize to the range $[0, 1]$.

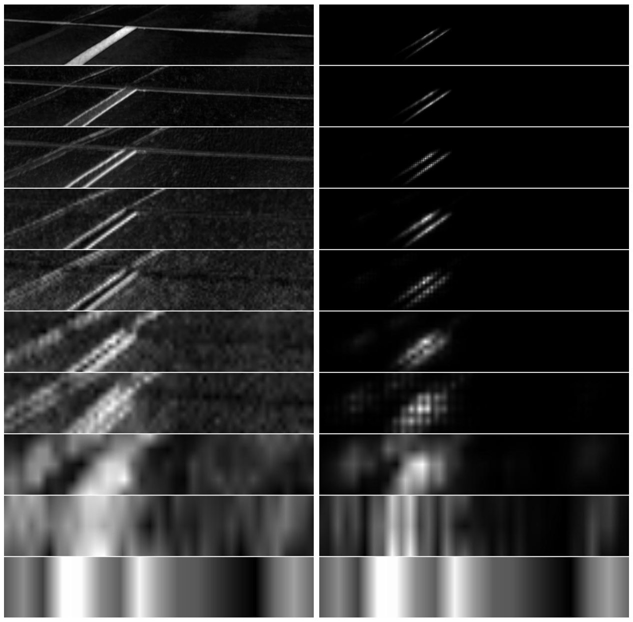
Figure 2. **Left**: Averaged feature maps of all convolutional layers from the input (top) to the output (bottom). **Right**: Corresponding intermediate masks. The top one is the final result.

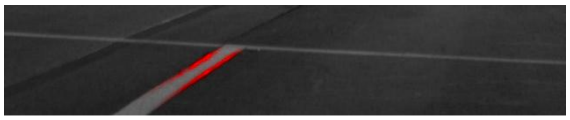
Figure 3. Input image with the mask overlaid in red.

The process of creating the mask is illustrated in Figure 2. On the left side the figure shows the averaged feature maps of all the convolutional layers from the input (top) to the output (bottom). On the right side it shows the corresponding intermediate masks. Thus on the right side we show step by step how the mask is being created when moving from the network's output to the input. Comparing the two top images clearly reveals that many details were removed in order to obtain the final mask. Finally, Figure 3 shows the mask overlaid on the input image.

4. Theoretical analysis

We now present theoretical guarantees for the algorithm (all proofs are deferred to the Supplement). We emphasize that our theoretical analysis is not about computing the sensitivity of any particular cost function with respect to the changes of values of particular input neurons. So we will not focus on computing the gradients. The reason for that is that even if these gradients are large, the actual contribution of the neuron might be small. Instead our proposed method is measuring the actual contribution that takes into account the "collaborative properties" of particular neurons. This is measured by the ability of particular neurons to substantially participate in these weighted inputs to neurons in consecutive layers that themselves have higher impact on the form of the ultimate feature maps than others.

Let's consider a convolutional neural network $\mathcal{N}$ with L convolutional layers that we index from 1 to L and ReLU nonlinear mapping with biases $-b_v$, where v stands for a neuron of the network. We assume that no pooling mechanism is used. For simplicity we assume that the strides are equal to one. However, the entire analysis can be repeated for arbitrary stride values. We denote by f_l the number of feature feature maps of the l^{th} layer, by $a_l \times b_l$ the shape of the kernel applied in the l^{th} convolutional layer and by s_l the number of neurons in each feature map of the l^{th} convolutional layer. We will denote by $a(v)$ the activation of the neuron v.

For a CNN with learned weights and biases we think about the inference phase as a network flow problem, where source nodes are the neurons of the input layer ($l = 0$) and sinks are nodes of the output layer (output of the last convolutional layer). In that setting the flow is (dis-)amplified when it travels along edges and furthermore some fraction of the flow is lost in certain nodes of the network (due to biases).

Our first observation relates the inference phase to the aforementioned flow problem on the sparse multipartite graph with $L + 1$ parts $A_0, A_1, ..., A_L$ and where vertex v in the part A_{i-1} has $a_i b_i f_i$ neighbors in part A_i.

Lemma 1 *The inference phase in the model above is equivalent to the network flow model on a multipartite graph with parts $A_1, ..., A_L$, where each edge has a weight defining the amplification factor for the flow traveling along this edge. In each node v the flow of value $\min(z_v, b_v)$ is lost, where z_v is the incoming flow and where vertex v in the part A_{i-1} has $a_i b_i f_i$ neighbors in part A_i.*

Different parts A_i above correspond to different convolutional layers (A_0 corresponds to the input layer). From now on we will often implicitly transition from the neural network description to the graph theory and vice versa using the above mapping.

The flow-formulation of the problem is convenient since it will enable us to quantitatively measure the contribution of neurons from the input layer to the activations of neurons in the last convolutional layer. Before we propose a definition of that measure, let us consider a simple scenario, where no nonlinear mapping is applied. That neural network model would correspond to the network flow for multipartite graphs described above, but with no loss of the flow in the nodes of the network.

In that case different input neurons act independently. For

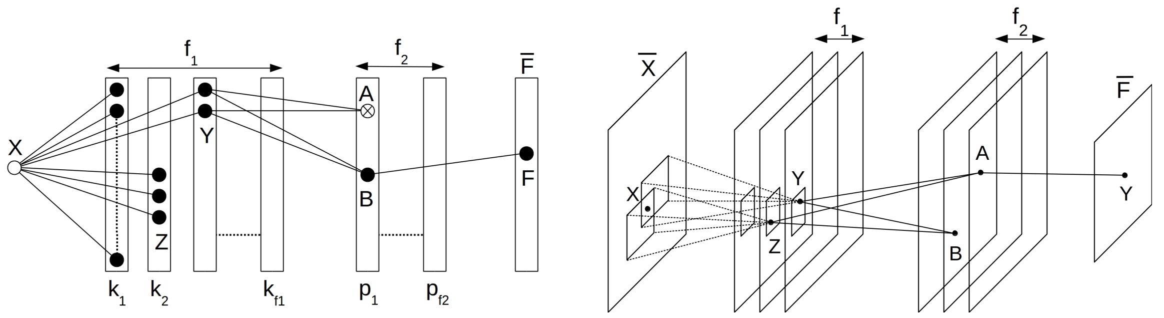

Figure 4. **Right**: General CNN $\mathcal{N}$ with biases. **Left**: Corresponding multipartite graph. Dead nodes, i.e. nodes that do not transmit any flow (corresponding to neurons with zero activation), are crossed.

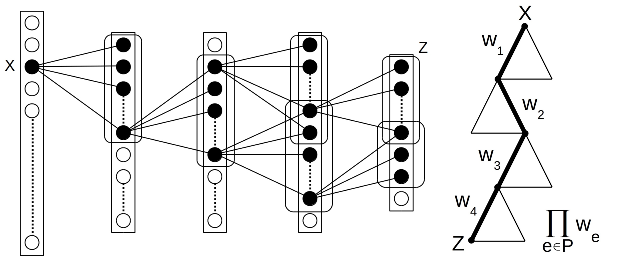

Figure 5. Convolutional architecture with no biases. The contribution of each neuron can be computed by summing the contribution along different paths originating at X and ending in the last layer. The contribution from each path is proportional to the product of the weights of edges belonging to this path.

each node X in A_0 the total flow value received by A_L is obtained by summing over all paths P from X to A_L the expressions of the form $f(X)\prod_{e \in P} w_e$, where w_e stands for the weight of an edge e and $f(X)$ is the value of the input flow to X (see Figure 5). This observation motivates the following definition:

Definition 1 (ϕ-function for CNNs with no biases) *Consider the neural network architectures $\mathcal{N}$ described above but with no biases. Then the contribution of the input neuron X to the last layer of feature maps is given as:*

$$\phi^{\mathcal{N}}(X) = v(X) \sum_{P \in \mathcal{P}} \prod_{e \in P} w_e, \tag{1}$$

where $v(X)$ is the value of X and $\mathcal{P}$ is a family of paths from X to A_L in the corresponding multipartite graph.

In our setting the biases need to be taken into account, but the definition above suggests how to do this: We transform our CNN with biases $\mathcal{N}$ to the one without biases $\tilde{\mathcal{N}}$, but with corrected weights of edges. We do it in such a way that the following property will be satisfied:

> For every neuron X and every incoming edge e_0 the input flow f_0 along e_0 is replaced by $a(X)\frac{f_0}{\sum_e f(e)}$, where the summation goes over all incoming edges e. $f(e)$ stands for the input flow along e and $a(X)$ is an activation of X in the original network.

Note that this network is equivalent to the original one, i.e. it produces the same activations for all its neurons as the original one. Furthermore, the property above just says that the contribution of the given input flow f to the activation of X is proportional to the value of the flow (this is a desirable property since the applied nonlinear mapping is a ReLU function). It also captures the observation that if the flow is large, but the activation is small, the contribution might be also small since even though the flow contributes substantially, it contributes to an irrelevant feature.

Lemma 2 *The above network $\tilde{\mathcal{N}}$ can be obtained from the original one $\mathcal{N}$ by multiplying each weight w_e of an edge $e = (n_1, n_2)$ by $c(e) = \frac{a_e}{z(e)}$, where a_e stands for the activation of the neuron n_1 and $z(e)$ is a weighted input to the neuron n_2.*

Since we know how to calculate the contribution of each input neuron to the convolutional network in the setting without biases and we know how to translate any general convolutional neural network $\mathcal{N}$ (see: Figure 4) to the equivalent one $\tilde{\mathcal{N}}$ without biases, we are ready to give a definition of the contribution of each input neuron in the general network $\mathcal{N}$.

Definition 2 (ϕ-function for general CNNs) *Consider the general neural network architecture described above. Then the contribution of the input neuron X to the last layer of feature maps is given as:*

$$\phi^{\mathcal{N}}(X) = \phi^{\tilde{\mathcal{N}}}(X), \tag{2}$$

where $\tilde{\mathcal{N}}$ stands for the counterpart of $\mathcal{N}$ with no biases described above.

We are now ready to express the contribution of the neuron X in $\mathcal{N}$ explicitly in terms of the parameters of $\mathcal{N}$.

Lemma 3 *For an input neuron X function $\phi^{\mathcal{N}}(X)$ is defined as (Figure 6):*

$$\phi^{\mathcal{N}}(X) = v(X) \sum_{P \in \mathcal{P}} \prod_{e \in P} \frac{a_e}{a_e + b_e} w_e, \tag{3}$$

where $v(X)$ is the value of pixel X, $\mathcal{P}$ is a family of paths from X to A_L in the corresponding multipartite graph, b_e is a bias in the second neuron of e and a_es are as above.

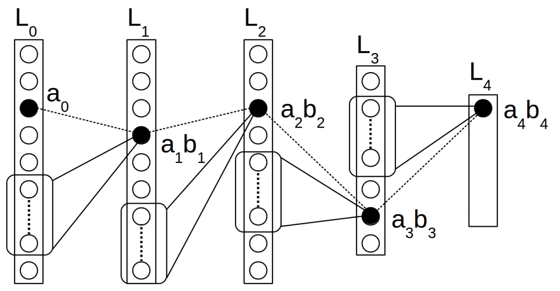

Figure 6. The contribution of any given input neuron X along a particular path P in the general CNN $\mathcal{N}$ is equal to the product $\prod_{e \in P} \frac{a_i w_i}{a_i + b_i}$, where a_is are the activations on the path and b_is are the corresponding biases.

Note that in the network with no biases one can set $b_e = 0$ and obtain the results introduced before the formula involving sums of weights' products.

We prove the following results (borrowing the notation introduced before) by connecting our previous theoretical analysis with the algorithm.

Theorem 1 *For a fixed CNN $\mathcal{N}$ considered in this paper there exists a universal constant $c > 0$ such that the values of the input neurons computed by VisualBackProp are of the form:*

$$\phi^{\mathcal{N}}_{VBP}(X) = c \cdot v(X) \sum_{P \in \mathcal{P}} \prod_{e \in P} a_e. \tag{4}$$

The statement above shows that values computed for pixels by the VisualBackProp algorithm are related to the flow contribution from that pixels in the corresponding graphical model and thus, according to our analysis, measure their importance.

The formula on $\phi^{\mathcal{N}}_{VBP}(X)$ is similar to the one on $\phi^{\mathcal{N}}(X)$, but gives rise to a much more efficient algorithm. Note that the latter one can be obtained from the former one by multiplying each term of the inner products by $\frac{w_e}{a_e + b_e}$ and then rescaling by a multiplicative factor of $\frac{1}{c}$. Rescaling does not have any impact on quality since it is conducted in exactly the same way for all the input neurons. In the next section we show empirically that $\phi^{\mathcal{N}}_{VBP}(X)$ works very well as a measure of contribution. Thus it might be the case that for the considered setting $\phi^{\mathcal{N}}(X)$ is a near-monotonic function of $\phi^{\mathcal{N}}_{VBP}(X)$.

Remark 1 *Note that for small kernels the number of paths considered in the formula on $\phi^{\mathcal{N}}(X)$ is small (since the degrees of the corresponding multipartite graph are small) thus in practice the difference between formula on $\phi^{\mathcal{N}}_{VBP}(X)$ and the formula on $\phi^{\mathcal{N}}(X)$ coming from the reweighting factor $\frac{w_e}{a_e + b_e}$ is also small. Therefore for small kernels the VisualBackProp algorithm computes good approximations of input neurons' contributions to the activations in the last layer.*

5. Experiments

In the main body of the paper we demonstrate the performance of VisualBackProp on the task of end-to-end autonomous driving, which requires real-time operation. The experiments were performed on the Udacity self-driving car data set (Udacity Self Driving Car Dataset 3-1: El Camino). The Supplement contains additional experimental results on the task of the classification of traffic signs on the German Traffic Sign Detection Benchmark data set (`http://benchmark.ini.rub.de/?section=gtsdb&subsection=dataset`). We compare our method with LRP implementation as given in Equation 6 from (Samek et al., 2016) (similarly to the authors, we use $\epsilon = 100$).

We train two networks, that we call *NetSVF* and *NetHVF*, which vary in the input size. In particular, *NetHVF* input image has approximately two times higher vertical field of view, but then is scaled down by that factor. The details of both architectures are described in Table 1. The networks are trained with stochastic gradient descent (SGD) and the mean squared error (MSE) cost function for 32 epochs.

	NetSVF	*NetHVF*		
Layer	Layer output size	Layer output size	Filter size	Stride size
conv	$32 \times 123 \times 638$	$32 \times 123 \times 349$	3×3	1×1
conv	$32 \times 61 \times 318$	$32 \times 61 \times 173$	3×3	2×2
conv	$48 \times 59 \times 316$	$48 \times 59 \times 171$	3×3	1×1
conv	$48 \times 29 \times 157$	$48 \times 29 \times 85$	3×3	2×2
conv	$64 \times 27 \times 155$	$64 \times 27 \times 83$	3×3	1×1
conv	$64 \times 13 \times 77$	$64 \times 13 \times 41$	3×3	2×2
conv	$96 \times 11 \times 75$	$96 \times 11 \times 39$	3×3	1×1
conv	$96 \times 5 \times 37$	$96 \times 5 \times 19$	3×3	2×2
conv	$128 \times 3 \times 35$	$128 \times 3 \times 17$	3×3	1×1
conv	$128 \times 1 \times 17$	$128 \times 1 \times 8$	3×3	2×2
FC	1024	1024	-	-
FC	512	512	-	-
FC	1	1	-	-

Table 1. Architectures of *NetSVF* and *NetHVF*. Each layer except for the last fully-connected layer is followed by a ReLU. Each convolution layer is preceded by a batch normalization layer.

The Udacity self-driving car data set that we use contains images from three front-facing cameras (left, center, and right) and measurements such as speed and steering wheel angle, recorded from the vehicle driving on the road. The measurements and images are recorded with different sampling rates and are independent, thus they require synchronization before they can be used for training neural networks. For the purpose of this paper, we pre-process the data with the following operations: i) synchronizing images with measurements from the vehicle, ii) selecting only center camera images, iii) selecting images where the car speed is above 5 m/s, iv) converting images to gray scale,

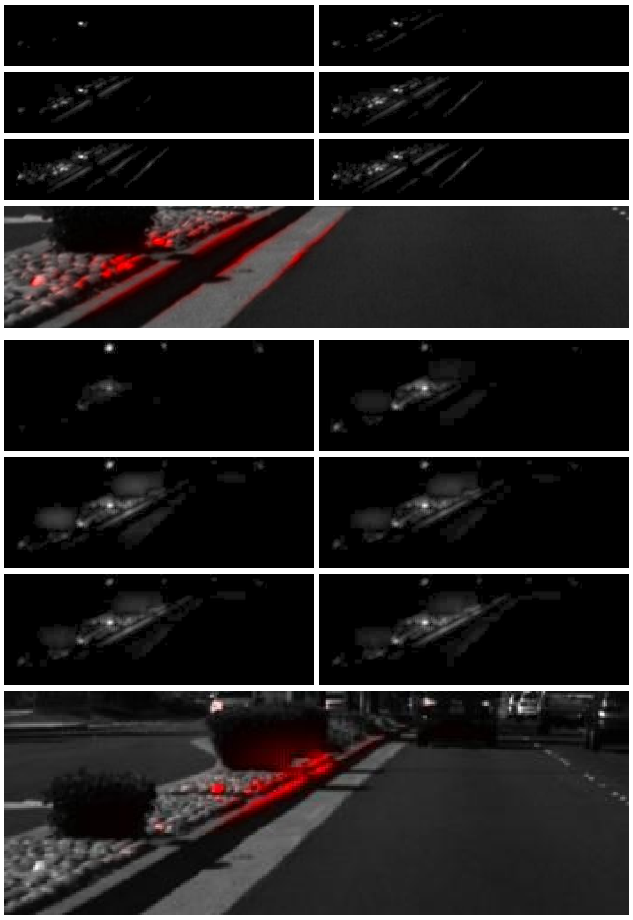

Figure 7. Visualizing the training of the network *NetSVF* (the set of top 7 images) and *NetHVF* (remaining images) with the VisualBackProp method: the mask after 0.1, 0.5, 1, 4, 16, and 32 epochs. Over time the network learns to detect more relevant cues for the task from the training data. The last picture in each set shows the input image with the mask overlaid in red.

and v) cropping and scaling the lower part of the images to a 640×135 size for network *NetSVF* and 349×173 size for network *NetHVF*. As a result, we obtain a data set with 210 K images with corresponding steering wheel angles (speed is not used), where the first 190 K examples are used for training and the remaining 20 K examples are used for testing. We train the CNN to predict steering wheel angles based on the input images.

Images in Figures 7 (the set of top 7 images), 8, 10, 12, 14, and 16 come from the same video frames as the images in Figures 7 (the set of bottom 7 images), 9, 11, 13, 15, and 17 respectively, but have smaller field of view.

We first apply VisualBackProp method during training to illustrate the development of visual cues that the network focuses on. The obtained masks for an exemplary image are shown in Figure 7. The figure captures how the CNN gradually learns to recognize visual cues relevant for steering the car (lane markings) as the training proceeds.

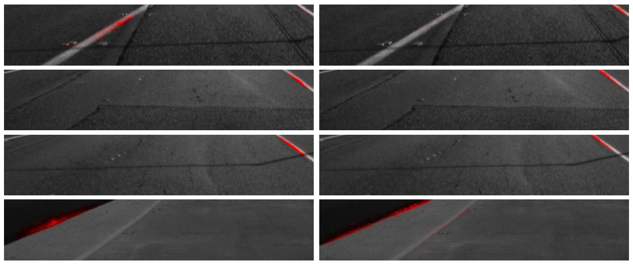

Figure 8. Network *NetSVF*. Left: VisualBackProp, right: LRP. Input test images with the corresponding masks overlaid in red. The errors are (from the top): 0.18, 2.32, 4.65, and -2.41 degrees of SWA.

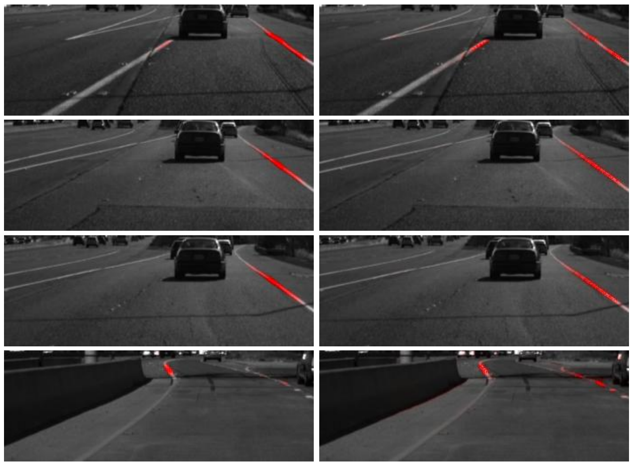

Figure 9. Network *NetHVF*. Left: VisualBackProp, right: LRP. Input test images with the corresponding masks overlaid in red. The errors are (from the top): -0.06, 3.91, 3.17, and -0.82 degrees of SWA.

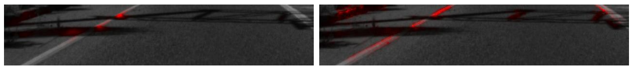

Figure 10. Network *NetSVF*. Left: VisualBackProp, right: LRP. Input test images with the corresponding masks overlaid in red. The error is 0.69 degrees of SWA.

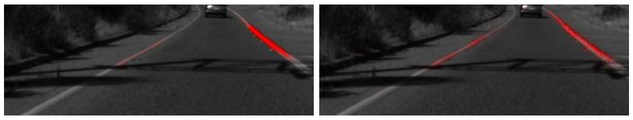

Figure 11. Network *NetHVF*. Left: VisualBackProp, right: LRP. Input test images with the corresponding masks overlaid in red. The error is -0.99 degrees of SWA.

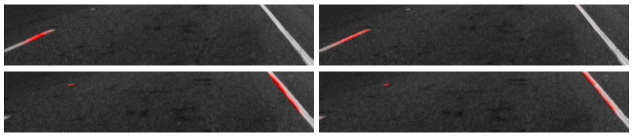

Figure 12. Network *NetSVF*. Left: VisualBackProp, right: LRP. Top and bottom of each column: two consecutive frames with the corresponding masks overlaid in red. The errors are (from the top): -2.65 and 3.21 degrees of SWA.

We next evaluate VisualBackProp and LRP on the test data. We show the obtained masks on various exemplary test input images in Figures 8–17, where on each figure the left column corresponds to our method and the right column

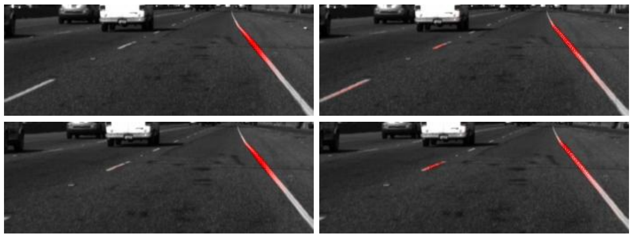
Figure 13. Network *NetHVF*. Left: VisualBackProp, right: LRP. Top and bottom of each column: two consecutive frames with the corresponding masks overlaid in red. The errors are (from the top): 1.55 and -0.66 degrees of SWA.

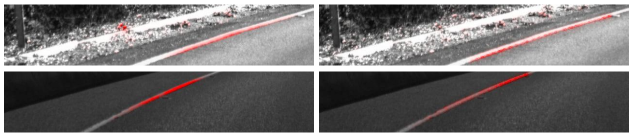
Figure 14. Network *NetSVF*. Left: VisualBackProp, right: LRP. Input test image with the mask overlaid in red. The errors are (from the top): 0.13 and -3.91 degrees of SWA.

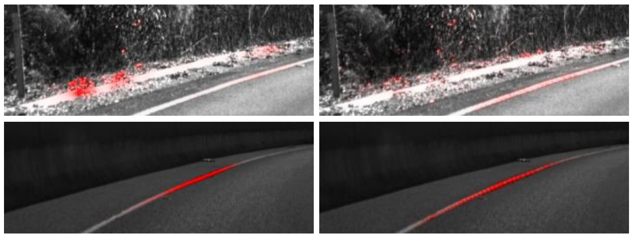
Figure 15. Network *NetHVF*. Left: VisualBackProp, right: LRP. Input test image with the mask overlaid in red. The errors are (from the top): -20.74 and -4.09 degrees of SWA.

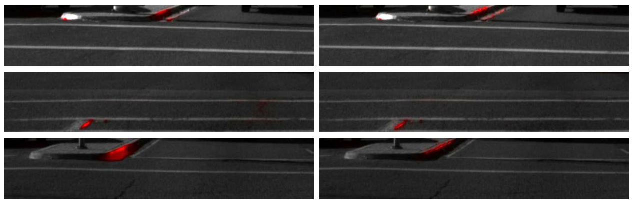
Figure 16. Network *NetSVF*. Left: VisualBackProp, right: LRP. Input test image with the mask overlaid in red. The errors are (from the top): -3.28, -0.84, and -3.97 degrees of SWA.

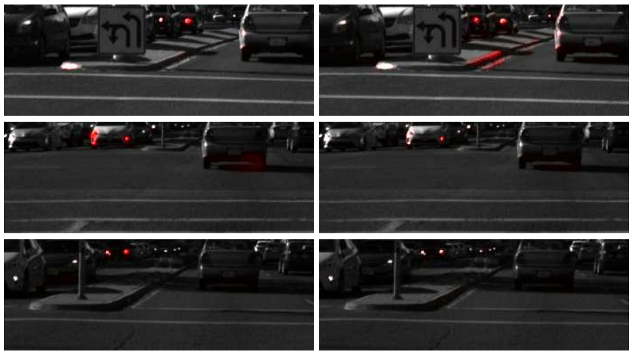
Figure 17. Network *NetHVF*. Left: VisualBackProp, right: LRP. Input test image with the mask overlaid in red. The errors are (from the top): -7.72, -0.72, and -3.17 degrees of SWA.

corresponds to LRP. For each image we also report the test error defined as a difference between the actual and predicted steering wheel angle (SWA) in degrees. Figures 8 and 9 illustrate that the CNN learned to recognize lane markings, the most relevant visual cues for steering a car. Figures 10 and 11 capture how the CNN responds to shadows on the image. One can see that the network still detects lane markings but only between the shadows, where they are visible. Each of the Figures 12 and 13 shows two consecutive frames (top and bottom) for VisualBackProp (left column) and LRP (right column). On the second frame in Figure 12, the lane marking on the left side of the road disappears, which causes the CNN to change the visual cue it focuses on from the lane marking on the left to the one on the right. Figures 14 and 15 correspond to the sharp turns. The images in the top row of Figure 15 demonstrate the correlation between the high prediction error of the network and the low-quality visual cue it focuses on. Finally, in Figure 16 we demonstrate that the CNN has learned to ignore horizontal lane markings as they are not relevant for steering a car, even though it was trained only with the images and the steering wheel angles as the training signal. The network was therefore able to learn which visual cues are relevant for the task of steering the car from the steering wheel angle alone. Figure 17 similarly shows that the CNN learned to ignore the horizontal lines, however, as the visualization shows, it does not identify lane markings as the relevant visual cues but other cars instead. As can be seen in Figures 8–17 the field of view affects the visualization results.

We implemented VisualBackProp and LRP in Torch7 to compare the computational time. Both methods used the cunn library to utilize GPU for calculations and have similar levels of optimization. All experiments were performed on GeForce GTX 970M. The average time of computing a mask for VisualBackProp was equal to $2.0\ ms$, whereas in case of the LRP method it was $24.6\ ms$. The VisualBackProp is therefore on average **12 times faster** than LRP. At the same time, as demonstrated in Figures 8–17, VisualBackProp generates visualization masks that are very similar to those obtained by LRP.

6. Conclusions

In this paper we propose a new method for visualizing the regions of the input image that have the highest influence on the output of a CNN. The presented approach is computationally efficient which makes it a feasible method for real-time applications as well as for the analysis of large data sets. We provide theoretical justification for the proposed method and empirically show on the task of autonomous driving that it is a valuable diagnostic tool for CNNs.

References

Abdel-Hamid, O., Mohamed, A.-r., Jiang, H., and Penn, G. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. In *ICASSP*, 2012.

Arras, L., Horn, F., Montavon, G., Müller, K.-R., and Samek, W. Explaining predictions of non-linear classifiers in NLP. In *ACL Workshop on Representation Learning for NLP*, 2016.

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, 10(7):e0130140, 2015.

Bach, S., Binder, A., Montavon, G., Müller, K.-R., and Samek, W. Analyzing classifiers: Fisher vectors and deep neural networks. In *CVPR*, 2016.

Baehrens, D., Schroeter, T., Harmeling, S., Kawanabe, M.i, Hansen, K., and Müller, K.-R. How to explain individual classification decisions. *J. Mach. Learn. Res.*, 11:1803–1831, 2010.

Berkes, P. and Wiskott, L. On the analysis and interpretation of inhomogeneous quadratic forms as receptive fields. *Neural Computation*, 18(8):1868–1895, 2006.

Binder, A., Bach, S., Montavon, G., Müller, K.-R., and Samek, W. Layer-wise relevance propagation for deep neural network architectures. pp. 913–922, 2016.

Bojarski, M., Testa, D. Del, Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., and Zieba, K. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

Chen, C., Seff, A., Kornhauser, A. L., and Xiao, J. Deepdriving: Learning affordance for direct perception in autonomous driving. In *ICCV*, 2015.

Dosovitskiy, A. and T.Brox. Inverting visual representations with convolutional networks. In *CVPR*, 2016.

Erhan, D., Bengio, Y., Courville, A., and Vincent, P. Visualizing higher-layer features of a deep network. Technical Report 1341, University of Montreal, 2009.

Erhan, D., Courville, A., and Bengio, Y. Understanding representations learned in deep architectures. Technical Report 1355, University of Montreal, 2010.

Gevrey, M., Dimopoulos, I., and Lek, S. Review and comparison of methods to study the contribution of variables in artificial neural network models. *Ecological Modelling*, 160(3):249 – 264, 2003.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

Hinton, G. E., Osindero, S., and Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7): 1527–1554, 2006.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *NIPS*. 2012.

Landecker, W., Thomure, M.l D., Bettencourt, L. M. A., Mitchell, M., Kenyon, G. T., and Brumby, S. P. Interpreting individual classifications of hierarchical networks. In *CIDM*, 2013.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, 1989.

Mahendran, A. and Vedaldi, A. Understanding deep image representations by inverting them. In *CVPR*, 2015.

Montavon, G., Braun, M. L., and Müller, K.-R. Kernel analysis of deep networks. *J. Mach. Learn. Res.*, 12: 2563–2581, 2011.

Olden, J. D., Joy, M. K., and Death, R. G. An accurate comparison of methods for quantifying variable importance in artificial neural networks using simulated data. *Ecological Modelling*, 178(34):389 – 397, 2004.

Poulin, B., Eisner, R., Szafron, D., Lu, P., Greiner, R., Wishart, D. S., Fyshe, A., Pearcy, B., Macdonell, C., and Anvik, J. Visual explanation of evidence with additive classifiers. In *IAAI*, 2006.

Rasmussen, P. M., Schmah, T., Madsen, K. H., Lund, T. E., Strother, S. C., and Hansen, L. K. Visualization of nonlinear classification models in neuroimaging - signed sensitivity maps. *BIOSIGNALS*, pp. 254–263, 2012.

Robnik-Sikonja, M. and Kononenko, I. Explaining classifications for individual instances. *IEEE Trans. Knowl. Data Eng.*, 20(5):589–600, 2008.

Samek, W., Binder, A., Montavon, G., Bach, S., and Müller, K.-R. Evaluating the visualization of what a deep neural network has learned. *IEEE Transactions on Neural Networks and Learning Systems*, 2016.

Selvaraju, R. R., Das, A., Vedantam, R., Cogswell, M., Parikh, D., and Batra, D. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016.

Setiono, R. and Liu, H. Understanding neural networks via rule extraction. In *IJCAI*, 1995.

Simonyan, K., Vedaldi, A., and Zisserman, A. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Workshop Proc. ICLR*, 2014.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. Striving for simplicity: The all convolutional net. In *ICLR 2015 Workshop Track*, 2015.

Weston, J., Chopra, S., and Adams, K. #tagspace: Semantic embeddings from hashtags. In *EMNLP*, 2014.

Yosinski, J., Clune, J., Nguyen, A. M., Fuchs, T., and Lipson, H. Understanding neural networks through deep visualization. *CoRR*, abs/1506.06579, 2015.

Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

Zeiler, M. D., Taylor, G. W., and Fergus, R. Adaptive deconvolutional networks for mid and high level feature learning. In *ICCV*, 2011.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A. Learning deep features for discriminative localization. In *CVPR*, 2016.

Zintgraf, L. M., Cohen, T. S., Adel, T., and Welling, M. Visualizing deep neural network decisions: Prediction difference analysis. In *ICLR*, 2017.

VisualBackProp: visualizing CNNs for autonomous driving (Supplementary Material)

7. Proof of Lemma 1

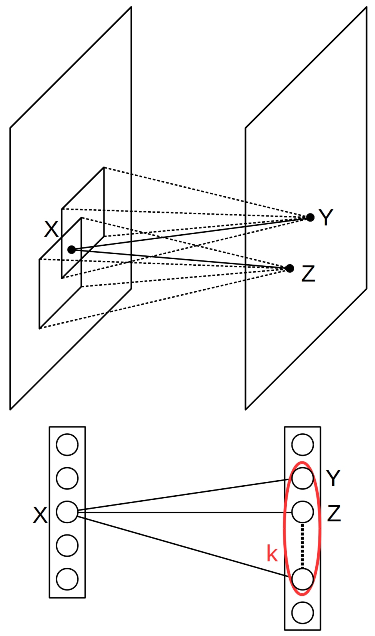

Figure 18. Neuron X is covered by patches corresponding to selected neurons in the consecutive layer. In the corresponding multipartite graph node X is adjacent to selected vertices in the next part and transports the flow to them.

Proof: As we have explained in the main body of the paper, we will identify the source-nodes for the flow problem with input neurons and the sink-nodes with the neurons of the last convolutional layer. Parts A_1,...,A_L are the sets of nodes in the corresponding convolutional layers (observe that indeed there are no edges between nodes in A_is since there are no edges between neurons in feature maps associated with the fixed convolutional layer). Part A_0 represents the input layer. Each source-node $v \in A_{i-1}$ sends $a(v)$ units of flow along edges connecting it with nodes in part A_i. This is exactly the amount of the input flow $f(v)$ to the node v with flow of value $\min(f_v, b_v)$ deducted (due to the ReLU nonlinear mapping). The activations of neurons in the last convolutional layer can be represented in that model as the output flows of the nodes from A_L. Note that for a fixed neuron v in the $(i-1)^{th}$ convolutional layer and fixed feature map in the consecutive convolutional layer (here the 0^{th} layer is the input layer) the kernel of size $a_i \times b_i$ can be patched in $a_i b_i$ different ways (see: Figure 18) to cover neuron v (this is true for all but the "borderline neurons" which we can neglect without loss of generality; if one applies padding mechanism the "borderline neurons" do not exist at all). Thus the total number of connections of v to the neurons in the next layer is $a_i b_i f_i$. A similar analysis can be conducted if v is an input neuron. That completes the proof.

8. Proof of Lemma 2

Proof: Assume that $e = (A, B)$, where A is an first node of the edge and B is second one. Note that from the definition of $\tilde{\mathcal{N}}$ for an input flow f, the flow of value $a_e \frac{f}{F}$ is transmitted by B, where F stands for the total input flow to B. Note also that $F = z(e)$. Furthermore $f = \tilde{f} w_e$, where $\tilde{f}$ is the original value of f before amplification obtained by passing that flow through an edge e. Thus the flow $\tilde{f}$ output by A is multiplied by $w_e \frac{a_e}{z_e}$ and output by B. Thus one can set: $c(e) = \frac{a_e}{z_e}$. That completes the proof.

9. Proof of Lemma 3

Proof: Straightforward from the formula on $\phi^{\tilde{\mathcal{N}}}(X)$ and derived formula on transforming weights of $\mathcal{N}$ to obtain $\tilde{\mathcal{N}}$.

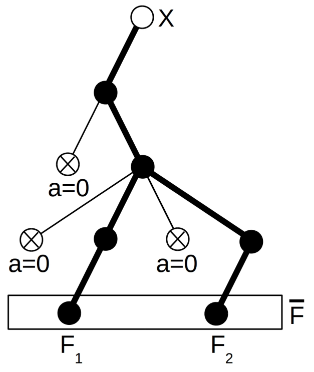

Figure 19. The family of paths from the input neuron X to output neurons F_1 and F_2 in the last set of feature maps $\bar{F}$. Crossed nodes correspond to neurons with zero activation. Paths containing these nodes do not contribute to $\bar{F}$.

10. Proof of Theorem 1

Proof: Note that averaging over different feature maps in the fixed convolutional layer with f_i feature maps that is conducted by *VisualBackProp* is equivalent to summing over paths with edges in different feature maps of that layer (up to a multiplicative constant $\frac{1}{f_i}$.) Furthermore, adding contributions from different patches covering a fixed neuron X is equivalent to summing over all paths that go through X. Thus computed expression is proportional to the union of products of activations (since in the algorithm after resizing the masks are point-wise multiplied) along paths from the given input neuron X to all the neurons in the last set of feature maps. That leads to the proposed formula. Note that the formula automatically gets rid of all the paths that lead to a neuron with null activation since these paths do not contribute to activations in the last set of feature maps (see: Figure 19).

11. Additional experiments

In this section we demonstrate the performance of VisualBackProp on the task of the classification of traffic signs. The experiments here were performed on the German Traffic Sign Detection Benchmark data set (`http://benchmark.ini.rub.de/?section=gtsdb&subsection=dataset`). We compare our method with LRP implementation as given in Equation 6 from (Samek et al., 2016) (similarly to the authors, we use $\epsilon = 100$).

We train the neural network to classify the input image containing a traffic sign into one of 43 categories. The details of the architecture are described in Table 2. The network is trained with stochastic gradient descent (SGD) and the negative log likelihood (NLL) cost function for 100 epochs.

The German Traffic Sign Detection Benchmark data set that we use contains images of traffic signs of various sizes, where each image belongs to one out of 43 different categories. For the purpose of this paper, we scale all images to the size of 125×125.

We apply VisualBackProp and LRP algorithm on trained network. The results are shown in Figures 20 and 21.

Layer	Layer output size	Filter size	Stride size
conv	$16 \times 123 \times 123$	3×3	1×1
conv	$16 \times 61 \times 61$	3×3	2×2
conv	$24 \times 59 \times 59$	3×3	1×1
conv	$24 \times 29 \times 29$	3×3	2×2
conv	$32 \times 27 \times 27$	3×3	1×1
conv	$32 \times 13 \times 13$	3×3	2×2
conv	$48 \times 11 \times 11$	3×3	1×1
conv	$48 \times 5 \times 5$	3×3	2×2
FC	64	-	-
FC	43	-	-

Table 2. Architecture of the network used for sign classification. Each layer except for the last fully-connected layer is followed by a ReLU. The last fully-connected layer is followed by a LogSoftMax. Each convolution layer is preceded by a batch normalization layer.

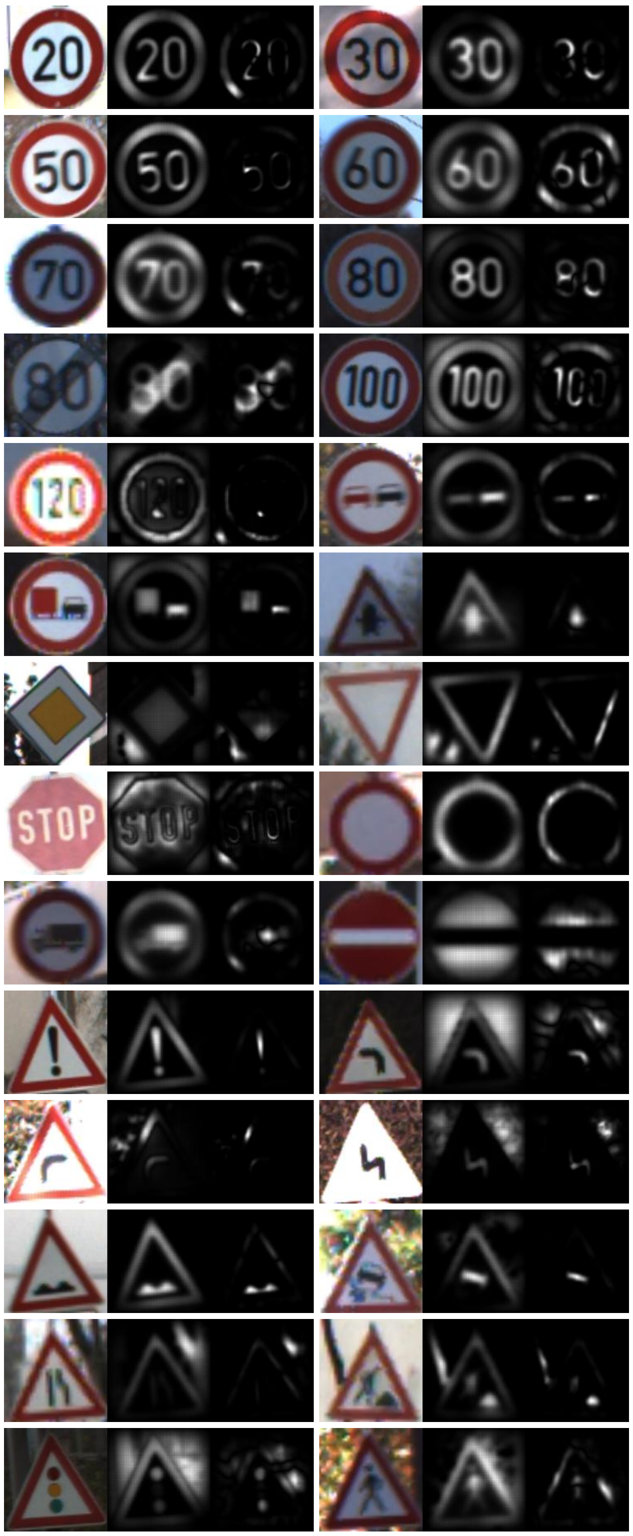

Figure 20. Sets of input images with visualization masks arranged in two columns. Each set consist of an input image (left), a visualization mask generated by VisualBackProp (center), and a visualization mask generated by LRP (right).

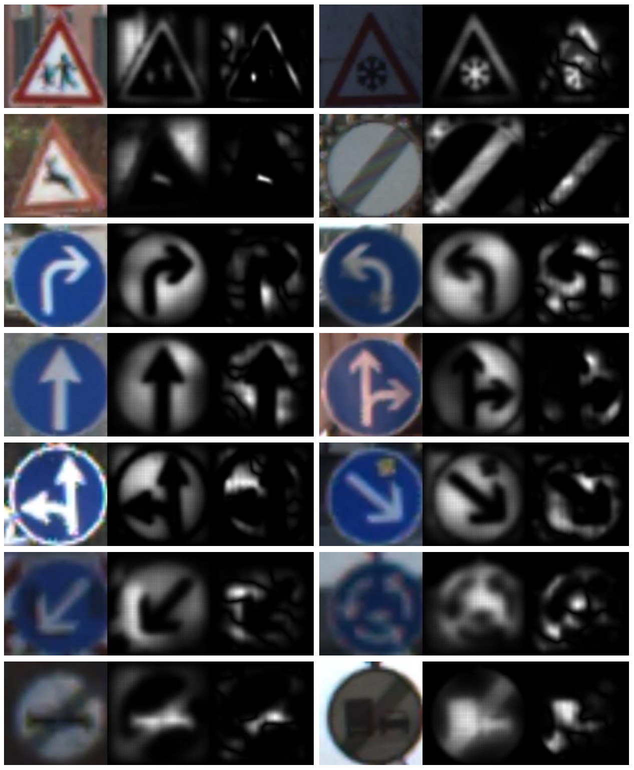

Figure 21. Sets of input images with visualization masks arranged in two columns. Each set consist of an input image (left), a visualization mask generated by VisualBackProp (center), and a visualization mask generated by LRP (right).