Neural Network Interpretability

Aleksandar Aleksandrov, Hans Hao-Hsun Hsu

July 2020

Contents

1 Project introduction

As deep learning based applications are becoming better and more widespread, more and more people are affected by their predictions in their daily lives. Recent research shows that deep learning works extremely well in computer vision and natural language processing tasks. Nevertheless, due to the black-box nature of deep learning models the public opinion of deep learning is often intertwined with numerous concerns, because its prediction can neither be fully explained nor interpreted by the general audience. Furthermore, regulations such as the European Union General Data Protection Regulation (GDPR) give the end users a *"right to explanation"* which allows the end user to request an explanation regarding an algorithmic decision that was made about them.
Based on this, a new research field, "Interpretability of Neural Network", has recently become a hot topic and aims to provide an explanation to predictions made by deep learning models. The interpretability methods should enable people to understand how and why a neural network makes its decisions. The motivations behind the methods are rooted both in the need for increased social acceptance of deep learning based application, as well as, the need for a deeper understanding of the inner workings of sophisticated deep learning models by people in the field of machine learning.
Interpretability methods can be categorized into two different types: model-based and decision-based. Model-based methods try to explain the concepts (prototypes) learned by a model. Decision-based methods concentrate on a specific input and aim to provide an explanation for a specific decision made by the model for the given input. This could be easier understood with a questioning perspective. Consider a scenario that we trained a neural network to classify cats and dogs images. Using model-based approaches to interpret our neural network we are asking the question *"How does a dog typically look like?"*. On the other hand, decision-based approaches give an interpretation of a particular decision, asking the question *"Why is this example classified as a dog?"*.
In our project we have explored 2 types of model-based interpretability methods: Activation Maximization (AM) and DeepDream; together with 7 types of decision-based interpretability methods: Backpropagation, Deconvolution, Occlusion Sensitivity, Taylor Decomposition, Layer-wise Relevance Propagation (LRP), Deep Learning Important FeaTures (DeepLIFT), and Class Activation Maps (CAM). Additionally, we demonstrate the applicability of interpretability methods on models with uncertainty awareness methods such as Monte Carlo Dropout, Evidential Deep Learning, and Temperature Scaling. Once the model is uncertain with its prediction, our interpretability methods can show the features that cause this uncertainty in the resulting saliency map or heatmap. This can further help us study the behavior of machine learning models under stochastic settings making the models more reliable in real-life applications.
In summary, the contributions of our project are as follows: (1) Investigated and implemented different neural network interpretability methods. (2) Studied how our interpretability methods can interpret uncertainty of neural network's prediction. (3) Provided an easy-to-use *nn_interpretability* package that can aid future research.

2 Neural Network Interpretability

2.1 Model-based methods

2.1.1 Activation Maximization

Activation Maximization is a model-based interpretability method that aims to find an input which produces a maximum model output [1] [5]. Mathematically, it can be represented as an optimization problem in which we try to maximize the expected output of a model by altering the input. Montavon et al. introduce three different variants of the Activation Maximization method: General AM, AM in Code Space and AM with an Expert [5].

General Activation Maximization aims to find the optimal input by optimizing directly in the input space. The optimization problem can, thus, be represented in the following form:

$$\max_{\mathbf{x}} logp(w_c|\mathbf{x}) - \lambda||\mathbf{x}||^2 \tag{1}$$

In the formula above, w_c represents the class of interest for which we are optimizing. Montavon et al. propose that the regularization term is extended with the class mean input to the form $\lambda||\mathbf{x} - x_c||^2$. Another variable in this optimization process is the initial point $\mathbf{x}$. In total, we have conducted three experiments with varying initial points.

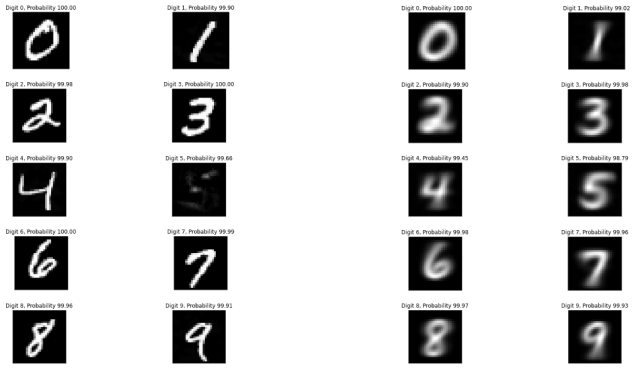


Figure 1: **Left:** Results for General AM from a random image. **Right:** Results for General AM from the class mean

In our first experiment, we set the starting point to a random representative of the class of interest drawn from the training set. The second experiment entails the class mean as the initial point. The results of these two can be observed in Fig. 1. It is important to mention that the class mean is being classified by the model under test with a high probability for the chosen class, which leads to the termination of the AM algorithm after only a few iterations.
In our third experiment, we set random noise to be the initial point of the optimization problem. Furthermore, we divide the experiment in two parts. In the first one, the class mean is part of the regularization term. In the second part, we remove the class mean from the regularization term which effectively turns it simply into the L2 regularization norm. The second part of this experiment was introduced in order to showcase the performance of AM in the absence of access to the training set. The results of these two experiments can be observed in Fig. 2

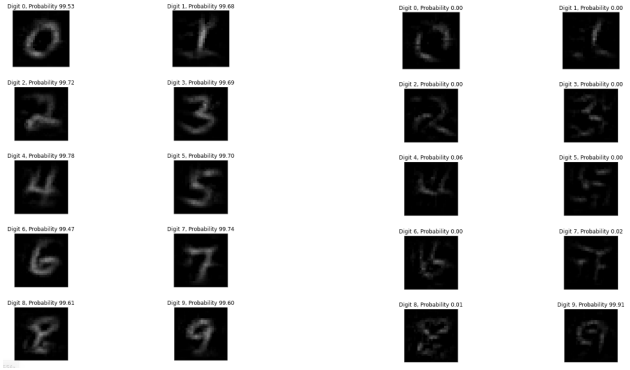


Figure 2: **Left:** Results for General AM from a random noise with class mean regularization. **Right:** Results for General AM from random noise with L2 regularization

Activation Maximization in Code Space introduces the usage of an auxiliary generative model. This method variation follows the following procedure. Firstly, we sample from the latent space. The generative function is, then, applied to the given sample. Finally, the result from the generative model is classified by the model under test. Based on that, the optimization is executed directly in the latent space instead of the input space.

$$\max_{\mathbf{z}} logp(w_c|g(\mathbf{z})) - \lambda||\mathbf{z}||^2 \tag{2}$$

We have tried the AM in Code Space method with two different generative models. In our first experiment, we have trained a simple GAN network. In the second case, we are using a pretrained DCGAN model that we have found online[1]. The results of these experiments can be found in Fig. 3. Although this method does produce less blurry results, we must remark that we faced instability in the results leading to misleading samples as observed in the outlined results. Furthermore, the method implies direct access to the training set and requires the training of a sufficiently good generative model in order to function as intended which makes the overhead associated with it significantly higher than other methods.

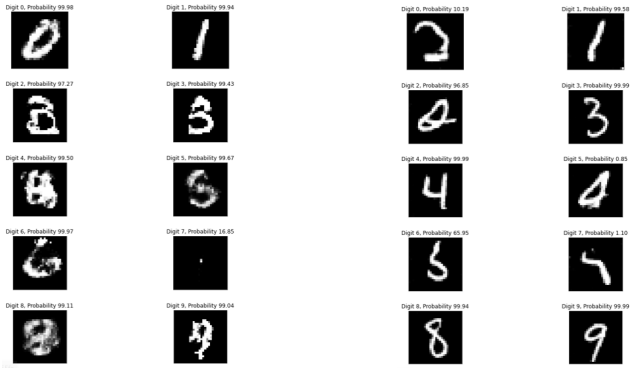


Figure 3: **Left:** Results for a simple GAN model **Right:** Results for pretrained DCGAN model

Activation Maximization with an Expert replaces the L2 regularization term with a data density model $logp(\mathbf{x})$. Possible choice for the *expert* is a Gaussian RBM. Montevan et al. argue about the difficulty of training a sufficiently well data density model.

$$\max_{\mathbf{x}} logp(w_c|\mathbf{x}) - logp(\mathbf{x}) \tag{3}$$

In our experience, a RBM implementation was given to us by the Group 5 "Hierarchical graphical models". Despite our numerous attempts, we couldn't reach a state in which both the RBM model and the AM method were performing sufficiently good. Based on this, we decided against continuing our efforts and we are leaving this method variation out of our implementation and report.

2.1.2 DeepDream

DeepDream is a model-based interpretability method that tries to enhance what is detected by a layer from an input image [6]. Following the same optimization approach from AM, we use gradient ascent to tweak an input image towards features that our neural network detects without prescribing which feature we want to amplify (Fig. 4). The main difference between DeepDream and AM is that, in the case of AM, we try to optimize class logits before the softmax, whereas in the case of DeepDream, we set a whole layer as an objective. Alternatively, a channel (filter) can be chosen as an objective,

[1]https://github.com/csinva/gan-vae-pretrained-pytorch

resulting in more consistent features of what the channel detects as in Fig. 5.
To better visualize features, we have to suppress high-frequency patterns that the network responds strongly to [7]. Two categories of regularization techniques are used in our DeepDream implementation. The first category is frequency penalization where we use $\ell 2$-norm regularizer and add Gaussian noise in each octave to penalize high frequency noise. The second category of regularization is transformation robustness where we try to find a prototype that still maximally activates our objective by transforming our input image. We first apply DeepDream on input image with different octaves in order that we get high resolution image and patterns at different granularity. In the loop of optimization, we randomly roll the input image to increase regularization. Apart from regularization techniques, we also split the input image into tiles while computing gradients of the input image and later concatenate them together. Such implementation makes DeepDream scalable and can upsample the input image to get finer details of visualization without GPU memory problem. The current DeepDream pipeline requires only 2GB GPU memory usage. This can be adapted by changing the tile parameter depending on individual hardware configurations.

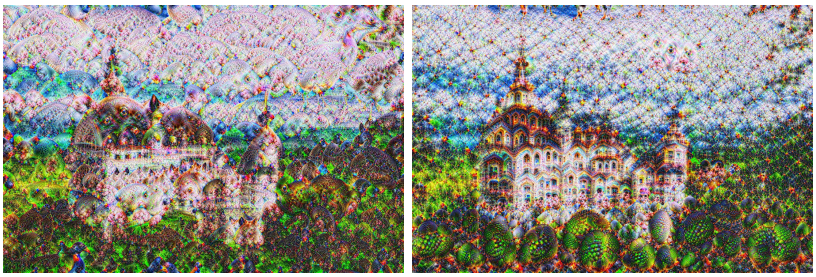


Figure 4: DeepDream objective as layer **Left:** 25th layer **Right:** 27th layer

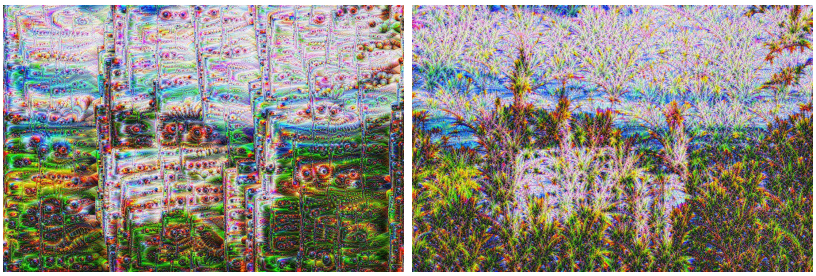


Figure 5: DeepDream objective as channel **Left:** 25th layer 30th channel **Right:** 25th layer 150th channel

2.2 Decision-based methods

2.2.1 Backpropagation

Backpropagation is a family of decision-based interpretability methods based on gradients which aims to indicate how influential a pixel is on a neural network's final decision. Gradients with respect to the input can be interpreted as a sensitivity measurement of the output [11]. Besides vanilla backpropagation, we further investigate other variants of gradient-based attribution: Guided Backpropagatation, Integrated Gradients and SmoothGrad.

Vanilla backpropagation generates a saliency map, pixels of which is the derivative of the output with respect to the input through backpropagation. Consider our neural network as a class score

function $f(x)$. We can compute the first-order Taylor expansion to approximate $f(x)$ with a linear function in the neighbourhood of $\tilde{x}$:

$$f(x) \approx f(\tilde{x}) + \frac{\partial f}{\partial x}(\tilde{x}) \cdot (x - \tilde{x}) \tag{4}$$

where the gradient is the weight of the perturbation at x in the linear function. Therefore, the gradient can be interpreted as an indicator showing which pixels need to be changed the least to affect the class score the most. However, it is important that the gradient doesn't provide an explicit relationship between class score and input pixels but rather a suggestion. When using backpropagation to interpret our decision, we are asking *"Which pixels influence this prediction the most?"* but rather the more basic question *"Which pixels cause this prediction?"*

Guided backpropatation combines the backpropagation rule of vanilla backpropagation and deconvolution while handling gradients through ReLU. It was first proposed to visualize the concepts learned by a network without pooling layers because deconvolution fails in the absence of pooling layers [13]. During backpropagation gradients will be masked out when either corresponding entries of (1) the top gradient (deconvnet) or (2) bottom data (vanilla backpropagation) is negative. This prevents backward flow of negative gradients, which cause a negative contribution to the higher layer unit we want to visualize.

Integrated Gradients uses a path method to integrate gradients along the straightline path between input x and a baseline x'. The idea is to propose an attribution method that satisfies two axioms, sensitivity and implementation invariance [14]. Sensitivity means that different features between input and baseline that bring about different predictions should be assigned a non-zero attribution. Implementation invariance means attributions should always be identical for two neural networks whose outputs are equal for all inputs, despite having different implementation. Gradients break sensitivity but satisfy implementation invariance due to chain rule. For sensitivity violation we can think of a ReLU network, if the input of ReLU is in the flatten region, the gradient method gives attribution of 0 to the pixels corresponding to the input value, no matter how we perturb these pixels in the flatten region. Integrated gradients satisfies both of these axioms, because first gradients satisfy implementation invariance and second path method add up attributions to the difference between the output at the input x and at the baseline x'. Consider an input image x, its attribution map can be computed as follows[2]:

$$IntegratedGrads(x) = \int_{\alpha=0}^{1} \frac{\partial f(x' + \alpha \times (x - x'))}{\partial x} d\alpha \tag{5}$$

In practice we can construct a sequence of images interpolating from the baseline x' to the actual input image x. The integral can be approximated by computing the average of the gradient across these images in a loop.

SmoothGrad suppresses noise on the saliency map by adding Gaussian noise. The sensitivity maps based on raw gradients (vanilla backpropagation) are visually noisy [12]. An explanation for the noise is that the derivative of the function may fluctuate sharply at small scales. In other words, due to the non-smooth derivative of the function (e.g. discontinuities caused by ReLU) there are many meaningless local variations which appear as noise in sensitivity maps. Because the local fluctuations are meaningless, we compute a local average of gradient values with a Gaussian kernel to smooth the derivative of the function. In practice we use a stochastic approximation to avoid computing in a high dimensional input space as follows:

$$SmoothGrad(x) = \frac{1}{n} \sum_{1}^{n} \frac{\partial f(x + \mathcal{N}(0, \sigma^2))}{\partial x} \tag{6}$$

[2]In order to compare with other backpropagation methods we don't multiply the integrated gradients with the input image as the paper do.

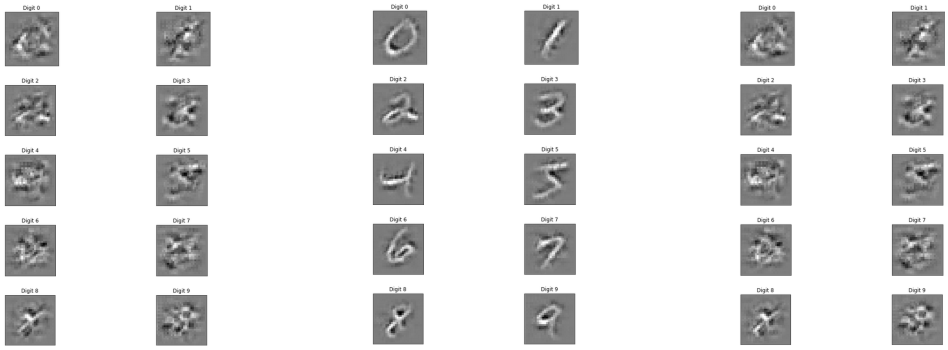


Figure 6: **Left:** Vanilla backpropagation **Middle:** Guided backpropagation **Right:** Integrated Gradients

where n is the number of samples and $\mathcal{N}(0, \sigma^2)$ represent Gaussian noise. The noise level, standard deviation σ^2, is suggested to take 10% to 20% of the input standard deviation from experience. Number of samples n greater than 50 should be enough. SmoothGrad can also be combined with other backpropagation methods as shown in Fig. 13 middle and right.

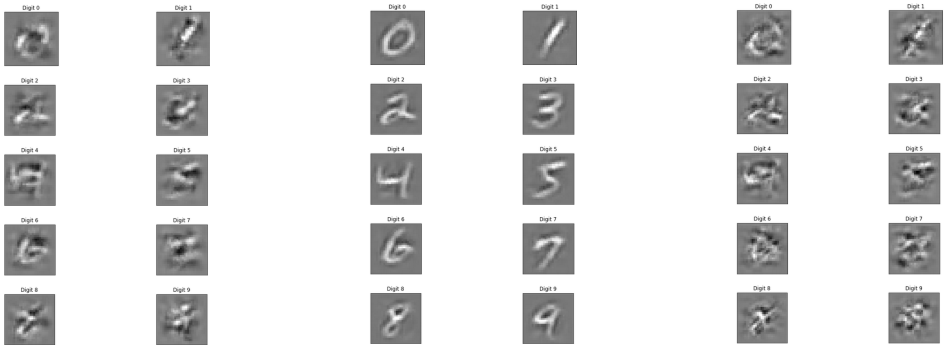


Figure 7: **Left:** SmoothGrad **Middle:** Guided backpropagation + SmoothGrad **Right:** Integrated Gradients + SmoothGrad

2.2.2 Deconvolution

Deconvolution is a decision-based interpretability method which aims to reconstruct the input based on the response of the model in the output space [16]. The reconstruction is achieved through a second network which mirrors the architecture of the model under test in a reversed order. This is achieved through using the transposed versions for convolutional and linear layers. For max pooling the indices need to be persisted in order to be able to execute the unpooling operation. ReLU units do not undergo any change. The deconvolutional network reuses the weights of the model under test.
Next to the full reconstruction of the input, Deconvolution can, also, be used to understand the role of particular filters in the reconstruction process. To do so, we choose a layer of interest and we leave only one filter of interest in the layer. The remaining filters in the layer are set to zero. The remainder of the network remains unchanged. The results of this process can be seen in Fig. 8. On the left, we can see the full reconstruction for representatives of each class. On the right, we see the results for partial reconstruction executed for the first convolutional layer which has ten filters. The first image will be the partial reconstruction if only the first filter is left and the others are set to zero. The second image represents the partial reconstruction with only the second filter and so on.

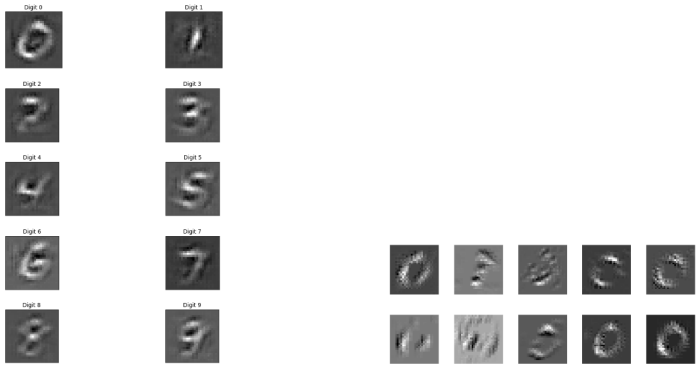


Figure 8: **Left:** Full input reconstruction for representatives of each class **Right:** Partial input reconstruction for each filter from the first convolutional layer

Furthermore, Simonyan et al. argue that there is an intrinsic correlation between Deconvolution and the gradient-based methods due to the nature of their computation [11]. Since gradient-based methods provide support for more layer types, they can be seen as a generalization of Deconvolution. Furthermore, Deconvolution can be used for the implementation of methods such as Layer-wise Relevance Propagation (LRP).

2.2.3 Occlusion Sensitivity

Occlusion Sensitivity is a decision-based interpretability method which obstructs parts of the input in order to see what influence these regions have on the output of the model under test [16]. The method helps us find the regions of the input which have the highest contribution to the final score. Fig. 9 showcases the deviations in the probability score for class 1 for each occluded image. We see the sharpest decrease in the probability score whenever central parts of the image where the object of interest resides are obstructed. Thus, we can conclude that the model bases it's decision on the actual object and not on the surrounding background.

Figure 9: Occlusion Sensitivity: Probability score for class 1 above each image

Furthermore, Occlusion Sensitivity can be combined successfully with other decision-based interpretability methods in order to further explain what influence the occluding region has on the final decision. In order to illustrate this idea, we have conducted an experiment with DeepLIFT. The results of this experiment can be seen in Fig. 10. The probability score above the images refers to the output of the given model for class 1. The grey images represent the results from the execution of the DeepLIFT method for both inputs. We can see that the occluded region leads to a sharp decrease in the confidence of the model that the image at hand is a representative of class 1. Furthermore, in the DeepLIFT result one could see that the pixels which had the highest contribution to the final score form a figure which is significantly closer to the number four than to the number one which further explains the extremely low probability score for the class 1.

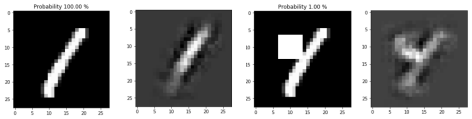


Figure 10: Occlusion Sensitivity in combination with DeepLIFT

2.2.4 Taylor Decomposition

Taylor Decomposition is a family of decision-based interpretability methods that aims to explain the decision by decomposing the model output $f(x)$ as a sum of relevance score [5]. Given an input image x, relevance score $R(x_i)$ is a measurement to quantify the amount of contribution of individual pixel x_i has on our model output $f(x)$. Taking the simple Taylor decomposition as a primitive, we also present a more sophisticated deep Taylor decomposition.

Simple Taylor decomposition applies first-order Taylor expansion to the entire network without considering the structure of our model. We now set the root point $\tilde{x}$ as the decision boundary of our model, namely $f(\tilde{x}) = 0$ and obtain

$$f(x) \approx \frac{\partial f}{\partial x}(\tilde{x}) \cdot (x - \tilde{x}) \tag{7}$$

The root point $f(\tilde{x}) = 0$ implies our model is most uncertain about its decision. $f(x) > 0$ indicates that the input contains object that is to be recognized; $f(x) < 0$ means the absence of the object. Moreover, $\tilde{x}$ can serve as a baseline for this decision. If the model classifies a given image x_p as positive $f(x_p > 0)$, then the difference $(x_p - \tilde{x})$ shows the image of pixels that caused the model to classify x_p as positive.
Finding the root point that satisfies $f(\tilde{x}) = 0$ is not easy. However, for special functions that are piecewise linear and satisfy the property $f(tx) = tf(x)$ for $t \geq 0$, we can alwyas find a root point $\tilde{x} = \lim_{\epsilon \to 0} \epsilon \cdot x \approx 0$, such that $f(\tilde{x}) = \lim_{\epsilon \to 0} \epsilon \cdot f(x) \approx 0$. The most used activation ReLU is an example of these function. Furthermore, due to the piecewise linear property, higher order terms from the Taylor expansion are zero, and therefore Eq.7 can be rewritten as

$$f(x) = \frac{\partial f}{\partial x} \cdot x = \sum_{i=1}^{d} \frac{\partial f}{\partial x_i} \cdot x_i = \sum_{i=1}^{d} R(x_i) \tag{8}$$

where d is the number of pixels in our input x. We see that Taylor expansion can be simplified as gradients times input image, or from an attribution method perspective sensitivity $\partial f / \partial x$ times saliency x. That is, an input feature is relevant, $R(x_i) \neq 0$, only if it both influences the model's decision and presents in the input. The results of simple Taylor decomposition are shown in Fig. 11 left where the red pixels indicates positive relevance score and blue pixels indicates negative relevance score.

Deep Taylor decomposition(DTD)[3] applies Taylor expansion to a layer-wise manner by considering relevance score layer by layer [5]. DTD is also a variant of LRP based upon Taylor Decomposition. Here we want to show that DTD is equivalent to LRP-$\alpha 1 \beta 0$. Applying Taylor expansion on the relevance score of higher layer neurons R_k we obtained an expression of a function of the activations from the previous layer a_j with a root point $\tilde{a}$

$$R_k(a) = R_k(\tilde{a}) + \sum_j \frac{\partial R_k}{\partial a_j}(\tilde{a}_j) \cdot (a_j - \tilde{a}_j) + \dots \tag{9}$$

[3]It is recommended to read LRP first before reading this section

Instead of modeling the true relevance function $R_k(a)$, we have to model the *relevance neuron* $\hat{R}_k(a)$ which is $R_k(a)$ times a constant positive term c_k.

$$\hat{R}_k(a) = \max(0, \sum_j a_j w_{jk}) \cdot c_k \tag{10}$$

After we apply Taylor expansion on the relevance neuron $\hat{R}_k(a)$ and consider the piecewise linear properties of ReLU that we use to derive simple Taylor decomposition, we obtain

$$\hat{R}_k(a) = \sum_j \frac{\partial \hat{R}_k(a)}{\partial a_j}(\tilde{a}_j) \cdot (a_j - \tilde{a}_j) \tag{11}$$

Comparing Eq.11 and 14 we can infer a closed-form expression as

$$R_{j \leftarrow k} = \frac{\partial \hat{R}_k(a)}{\partial a_j}(\tilde{a}_j) \cdot (a_j - \tilde{a}_j) = \frac{a_j w_{jk}^+}{\sum_j a_j w_{jk}^+} R_k \tag{12}$$

Summing $R_{j \leftarrow k}$ over all neurons k results in the exact LRP-$\alpha 1 \beta 0$ propagation rule. The results of DTD are shown in Fig. 11 right where the range of $R(x_i)$ is limited to non-negative values, meaning $R(x_i) = 0$ indicates no contribution and $R(x_i) > 0$ indicates positive contribution.

$$R_j = \sum_k R_{j \leftarrow k} = \sum_k \frac{a_j w_{jk}^+}{\sum_j a_j w_{jk}^+} R_k \tag{13}$$

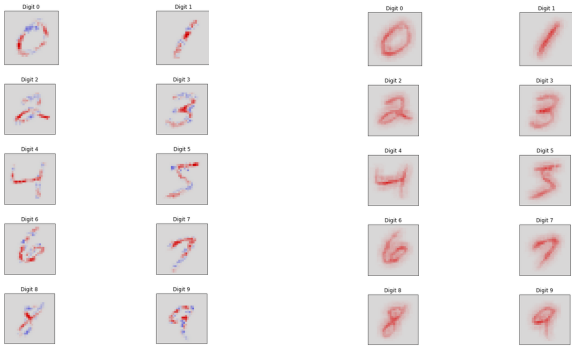


Figure 11: **Left:** Simple Taylor Decomposition **Right:** Deep Taylor Decomposition

2.2.5 Layer-wise Relevance Propagation (LRP)

Layer-wise Relevance Propagation (LRP) is a decision-based interpretability method which aims to backpropagate relevance score of every pixel from a predicted class logit using a set of LRP rules [4] [5]. Propagating relevance score at a given layer to neurons of the lower layer follows the layer-wise relevance conservation constraint:

$$\sum_j R_{j \leftarrow k} = R_k \tag{14}$$

where j denotes indices for neurons at the lower layer and k for neurons at the successive higher layer. We define $R_{j \leftarrow k}$ as the portion of relevance flow from neuron k to neuron j. That is, the relevance score of a neuron in the higher layer is equal to the total relevance it propagates downwards. The relevance score of a lower layer R_j can be defined as a similar manner:

$$R_j = \sum_k R_{j \leftarrow k} \tag{15}$$

Combining Eq.14 and 15 leads us to a global conservation constraint from the model output to input relevance score. In other words, we don't loss any quantity of relevance during the backpropagation and the sum of the relevance score of the image pixels will equal to the output of our model.

$$\sum_{i=1}^{d} R_i = \ldots = \sum_j R_j = \sum_k R_k = \ldots = f(x) \tag{16}$$

Next we model neurons at layer with ReLU activation as

$$a_k = \max\left(0, \sum_j a_j w_{jk}\right) \tag{17}$$

with a_j activations from the previous layer and w_{jk} the weight of the neuron. Based on Eq.16, 17 we can derive four variations of propagation rules: LRP-0, LRP-ϵ, LRP-γ, and LRP-$\alpha\beta$.
LRP-0 is the basic rule of LRP. However, it can be shown that a uniform application of this rule to the whole neural network provides an explanation that is equivalent to Gradient×Input. The result heatmap seems noisy, therefore one needs to design more robust propagation rules.

$$R_j = \sum_k \frac{a_j w_{jk}}{\sum_j a_j w_{jk}} R_k \tag{18}$$

LRP-ϵ is the enhancement of LRP by adding a small positive term in the denominator. Epsilon is to absorb some relevance when the contributions to the activation of neuron is to weak or are weak or contradictory. Bigger epsilon leads to a sparser heatmap and less noisy.

$$R_j = \sum_k \frac{a_j w_{jk}}{\epsilon + \sum_j a_j w_{jk}} R_k \tag{19}$$

LRP-γ favors the effect of positive contributions over negative contributions we obtain this rule. As gamma increases, negative contributions start to disappear. The prevalence of positive contributions can limit how much positive and negative relevance can grow in LRP backpropagation, which leads to a more interpretable manner. Note when gamma is close to infinity large, LRP-gamma is equivalent to LRP-$\alpha\beta$.

$$R_j = \sum_k \frac{a_j \cdot (w_{jk} + \gamma w_{jk}^+)}{\sum_j a_j \cdot (w_{jk} + \gamma w_{jk}^+)} R_k \tag{20}$$

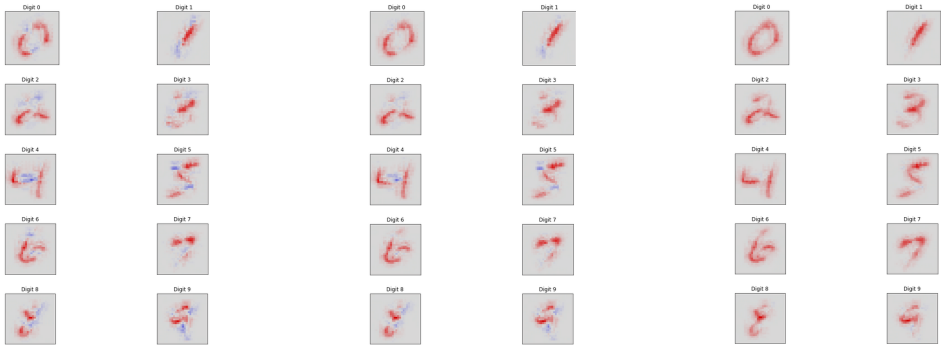


Figure 12: **Left:** LRP-0 **Middle:** LRP-ϵ **Right:** LRP-γ

LRP-$\alpha\beta$ handles positive $()^+$ and negative $()^-$ parts of the weight separately. The parameter α and

β should be chosen subject to $\alpha - \beta = 1$ and $\beta \geq 0$. The positive term can be considered as excitation flow that is proportional to the excitatory effect lower-layer neuron activation a_j has on the higher-layer neuron activation a_k in the prediction forward pass. Similarly, the negative term is proportional to the inhibitory effect a_j has on a_k in the prediction forward pass.

$$R_j = \sum_k \left(\alpha \frac{a_j w_{jk}^+}{\sum_j a_j w_{jk}^+} - \beta \frac{a_j w_{jk}^-}{\sum_j a_j w_{jk}^-} \right) R_k \tag{21}$$

In practice, we use composite LRP rules because it gives better understanding and fidelity. LRP-0 is chosen for the upper layers because in the upper layers many concepts from different classes are entangled. Choosing a propagation rule that most close to the gradient propagation better preserves these entanglements. LRP-ϵ is suitable for the middle layers because it filters out spurious variations from the stacking layers and keeps only the most salient features. LRP-γ and LRP-$\alpha\beta$ are chosen for the lower layers as they tend to spread relevance uniformly to the feature and makes human easier to understand the interpretation. In our implementation of composite LRP we use a combination of LRP-0, LRP-ϵ and LRP-$\alpha\beta$.

Input layer(input image) has to be treated differently because it doesn't receive ReLU activations as input but pixel values. A recommended rule suggested by Montavon et al. is the $z^{\mathcal{B}}$-rule that will map the relevance score to a pixel domain $x_i \in [l_i, h_i]$ with $l_i \leq 0 \leq h_i$

$$R_j = \sum_k \left(\frac{x_i w_{jk} - l_i w_{ij}^+ - h_i w_{ij}^-}{\sum_j x_i w_{jk} - l_i w_{ij}^+ - h_i w_{ij}^-} \right) R_k \tag{22}$$

Built upon the relevance conservation constraint, these LRP rules bridge an explicit relationship between relevance score of each pixel $R(x_i)$ and the class score $f(x)$. Hence we are able to answer the question *"Which pixels cause this prediction?"* from the heatmap of LRP.

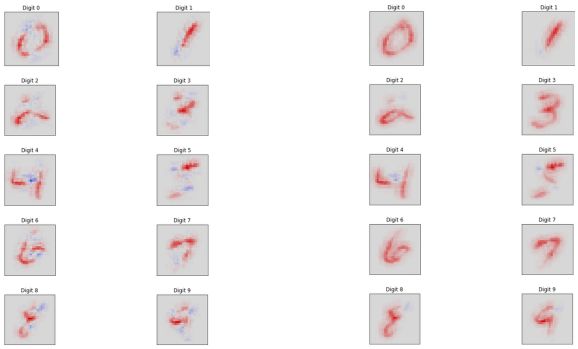


Figure 13: **Left:** LRP-$\alpha 2\beta 1$ **Right:** Composite LRP

2.2.6 Deep Learning Important FeaTures (DeepLIFT)

Deep Learning Important FeaTures (DeepLIFT) is a decision-based interpretability method which aims to explain the output of a model under test by assigning contribution scores to every part of the input [10]. The method computes the contribution scores based on explaining the difference in output from a *reference* output in the context of the difference from the corresponding *reference* input. The *reference* input is domain and model specific. In the case of MNIST, a fully black image can be set as the *reference* input. By using the difference-from-reference the method is able to backpropagate information even in the case that the gradient is equal to zero.

If we have a single neuron of interest with inputs $x_1, x_2, ...x_n$ and output t, $\Delta t = t - t^0$ represents

the difference-from-reference for this neuron. The contribution scores can, then, be defined in terms of Δt in the following manner: $\sum_i^n C_{\Delta x_i \Delta t} = \Delta t$. Based on that, we can define multipliers in the following form: $m_{\Delta x \Delta t} = \frac{C_{\Delta x \Delta t}}{\Delta t}$. Since the multipliers fulfill the chain rule, we can efficiently compute the multipliers for each neuron based on multipliers of it's predecessors by using backpropagation.
DeepLIFT is based on the use of predefined rules which handle the computation of contribution scores for specific layers based on the contribution scores of the predecessor layer. The **Linear** rule applies to linear and convolutional layers without the non-linear activations. The specifics of its implementation can be found in the appendix to the original paper [4]. The **Rescale** rule applies to non-linear activations such as the ReLU, sigmoid or tanh operations. Its computation is described mathematically in Eq. 23.

$$\Delta y^+ = \frac{\Delta y}{\Delta x}\Delta x^+ = C_{\Delta x^+ \Delta y^+} \quad \Delta y^- = \frac{\Delta y}{\Delta x}\Delta x^- = C_{\Delta x^- \Delta y^-} \tag{23}$$

The **RevealCancel** rule also applies to non-linearities, but unlike the **Rescale** rule, it provides a separate handling of positive and negative contributions.

$$\begin{aligned}
\Delta y^+ &= \frac{1}{2}\left(f(x^0 + \Delta x^+) - f(x^0)\right) \\
&\quad + \frac{1}{2}\left(f(x^0 + \Delta x^- + \Delta x^+) - f(x^0 + \Delta x^-)\right) \\
\Delta y^- &= \frac{1}{2}\left(f(x^0 + \Delta x^-) - f(x^0)\right) \\
&\quad + \frac{1}{2}\left(f(x^0 + \Delta x^+ + \Delta x^-) - f(x^0 + \Delta x^+)\right) \\
m_{\Delta x^+ \Delta y^+} &= \frac{C_{\Delta x^+ y^+}}{\Delta x^+} = \frac{\Delta y^+}{\Delta x^+} ; m_{\Delta x^- \Delta y^-} = \frac{\Delta y^-}{\Delta x^-}
\end{aligned}$$

Figure 14: Formula for the DeepLIFT RevealCancel Rule [10]

We have experimented with 6 different modes of the DeepLIFT method: NoRule, Rescale, RevealCancel, Linear, LinearRescale, LinearRevealCancel. The NoRule mode refers to simple vanilla backpropagation and serves as a base point. The Rescale, RevealCancel and Linear refer to modes in which only the specified rule is used. The LinearRevealCancel and LinearRescale modes refer to the modes in which the Linear rule is used for linear and convolutional layers and either the RevealCancel or the Rescale rule is used for the ReLU units. A subset of our results can be found in Fig 15.

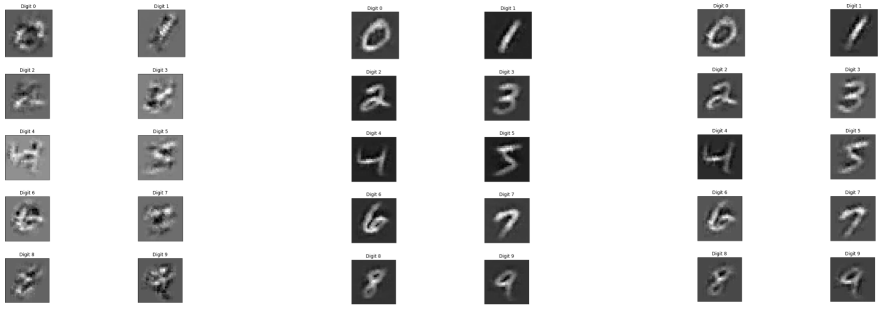


Figure 15: **Left:** DeepLIFT Rescale **Middle:** DeepLIFT Linear **Right:** DeepLIFT RevealCancel

2.2.7 Class Activation Maps (CAM)

Class Activation Maps (CAM) is a decision-based interpretability method which utilizes global average pooling (GAP) to define discriminative input regions which are used by the model under test to classify the input [17]. CAM supports only a limited family of convolutional networks which consist

[4] http://proceedings.mlr.press/v70/shrikumar17a/shrikumar17a-supp.pdf

of convolutional blocks followed by a single GAP layer. Following the GAP layer, there should be a single linear layer which outputs the class logits. If a network does not have the described architecture, it can be changed by removing all of the layers after the last convolutional block and replacing them by a GAP layer followed by a single linear layer. However, this would require additional training for the new layers.

$$M_c(x, y) = \sum_k w_c^k f_k(x, y) \tag{24}$$

CAM can be computed using the formula above. $f_k(x, y)$ represents the activation of the kth unit in the last convolutional layer at the (x, y) position. w_c^k is the weight of class c for the kth unit. Summing these two over the k units gives us M_c which we refer to as the class activation map for the class c. In order to be able to fully map the class activation map to the input space, we need to upsample the CAM to the size of the input.

In our experiments, we have found out that the upsampling procedure is another limitation of the CAM method. More specifically, the quality of the results from the CAM method is directly correlated to the ratio of the input size to the size of the class activation map before the upsampling procedure. A high ratio leads to more blurry results which often have little meaningful information to offer. Models with a lower ratio have better results from the CAM method. We exemplify this behavior in Fig. 16. In the left picture, one can find the result from the CAM method for a model based on which the class activation maps have a size of 3x3 before the upsampling. On the right side of the figure, the results for a model based on which the class activation maps have a size of 6x6 before the upsampling are displayed. It is easy to see that the model with a lower ratio allows for significantly better and more accurate results in comparison to its counterpart. It is noteworthy to say that we have tried out different upsampling methods (e.g. bilinear, bicubic, linear, etc) without a significant change in the final results.

Figure 16: **Left:** CAM results for a high input/feature map ratio **Right:** CAM results for a low input/feature map ratio

Gradient-weighted Class Activation Maps (Grad-CAM) is a decision-based interpretability method which offers a generalization of the Class Activation Maps (CAM) method by utilizing the gradient signal [8]. Since it is no longer bound to the weights of a specific layer, Grad-CAM can be used for practically any convolutional model. Furthermore, the flexibility gained by utilizing the gradient makes it possible to execute Grad-CAM for any convolutional layer within a network and not only for the very last one as it is the case for CAM.

$$L^c = ReLU(\sum_k \alpha_k^c A^k) \quad where \quad \alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\mathrm{d}y^c}{\mathrm{d}A_i j^k} \tag{25}$$

Grad-CAM can be computed using the formula above. α_k^c is "a neuron importance weight" and is computed by average pooling the incoming gradient over the width and the height dimensions. A^k

refers to the kth feature map produced by the chosen convolutional layer in the forward pass. We refer to the linear combination of these two variables as the class-discriminative localization map L^c. Exactly as we do for CAM, we need to upsample the L^c map to the size of the input.
In our experiments, Grad-CAM produces more precise results than CAM. In Fig. 17, the Grad-CAM results for the first three layers of a CNN model are displayed. It is self-evident that the precision of the discriminative regions directly correlates to the depth of the convolutional layer. Executing Grad-CAM for convolutional layers which are deeper in the model generates results which are more general and less precise. The results for Grad-CAM for the last convolutional layer are similar to the ones we obtained from CAM, but with a little higher precision in terms of the discriminative regions.

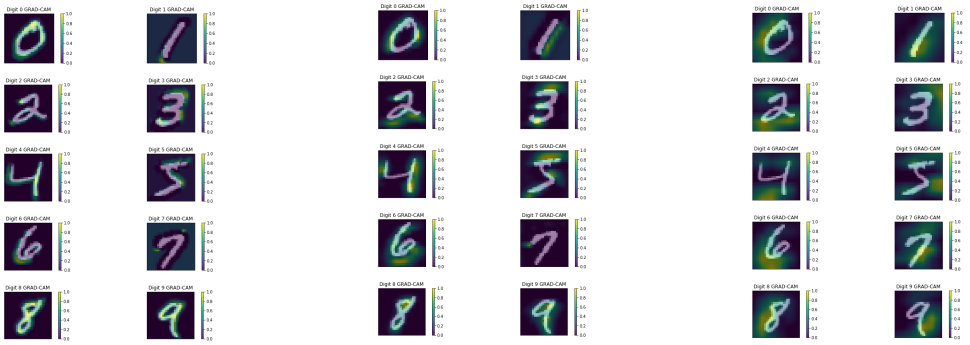


Figure 17: **Left:** Grad-CAM for the 1st CONV layer **Middle:** Grad-CAM for the 2nd CONV layer **Right:** Grad-CAM for the 3rd CONV layer

3 Uncertainty-aware Interpretability

3.1 Monte Carlo Dropout

Dropout in neural networks can be interpreted as a Bayesian approximation of Gaussian process [2]. While the training phase stays the same as the standard training approach, during the test phase the network will enable dropout and perform a number of stochastic forward predictions to approximate a distribution of predicted classes. To analyze how Monte Carlo dropout(MC dropout) can estimate uncertainty we evaluate our model with a continuously rotated MNIST digit 1. The rotation process introduces uncertainty to the prediction. For a deterministic prediction from the same model, the middle rotated images tend to predicted as "8" and "2". For MC dropout prediction, we set the number of predictions as 1000 and visualize the distribution for class "1"(Green), "2"(Blue) and "2"(Red). Fig. 19 shows both the class logits before softmax and the class score values after softmax. If the colors in the scatter plot are well separated this means our model is more confident with the prediction. For the middle rotated images the colors are very overlapped meaning that the uncertainty of prediction is high. Interestingly, by comparing the two scatter plots we can see that softmax tends to push the highest class score apart from others, which makes the predictions of softmax more "certain" than those of class logits. However, softmax output doesn't reveal uncertainty. For example, the 3rd rotated image is classified as "1" with a score of 82% from softmax, but the MC dropout prediction preserves high uncertainty in Fig. 19.
To interpret uncertainty using composite LRP we start from the following approach 1: For 1000 predictions we observed, LRP propagates relevance score from the *predicted* class score to the input layer. Heatmaps from the same predicted class will sum together. We evaluate this approach with the 3rd rotated image, prediction of which should involve high uncertainty. The results are shown in Fig. 19 left side. The sum of the numbers below the square figure, meaning the number of predictions of each class, is equal to 1000. Unfortunately, we see that directly interpret MC dropout with LRP fails

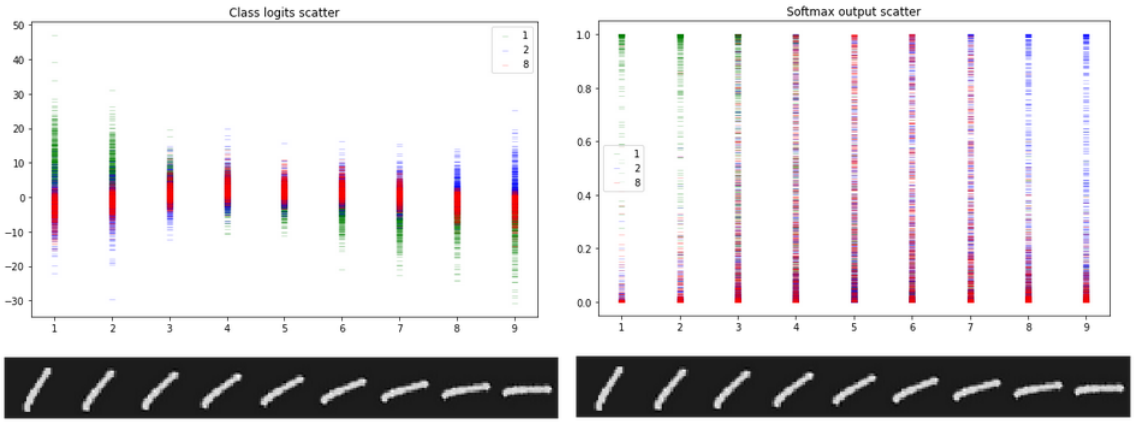


Figure 18: **Left:** MC dropout results for class logits values **Right:** MC dropout results for softmax values

Algorithm 1 MC dropout interpret with LRP

```
for i = 1 to number of stochastic predictions do
    for cls in all classes do
        prediction ← model(image)
        heatmap ← LRP.interpret(prediction)        ▷ LRP backpropagates from assigned cls
        heatmaps[cls] ← heatmap + heatmaps[cls]
    end for
end for
return heatmaps
```

to tell us the uncertainty of the model's predictions.

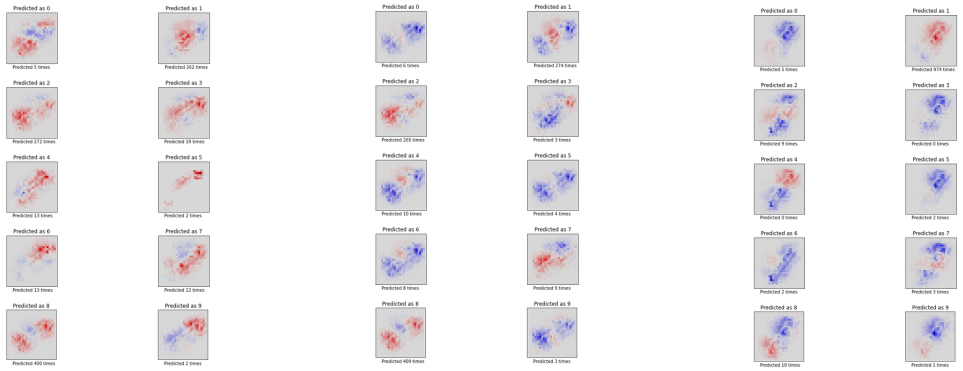


Figure 19: **Left:** 3rd rotated image results from approach 1 **Middle:** 3rd rotated image results from Algorithm 1 **Right:** 1st rotated image results from Algorithm 1

By slightly changing LRP backpropagation starting point from predicted class to an *d class*. We developed Algorithm 1 that can capture uncertainty. We assume the predictions are independent and identically distributed (*i.i.d.*) and therefore can observe predictions separately for each class. Note the numbers below now don't sum to 1000 in total. The results of the algorithm on the 3rd rotated image are shown in Fig. 19 middle. Our algorithm can reveal the classes that our model tends to predict, the resulting heatmap of these classes will contain more red pixels than others. Furthermore, it points

out the features that induce the model to make the wrong decision(e.g. both endpoints of "1" induce the prediction of "8"). Namely, the uncertainty of prediction is attributed to these features. We also provide results on the 1st rotated image as a control group for low uncertainty prediction as shown in Fig. 19 right. Comparing the two results, we can see that if the prediction is under high uncertainty, the results will contain more heatmaps consists of red pixels comparing to the results from the low uncertainty prediction.

3.2 Evidential Deep Learning

Evidential deep learning is an uncertainty aware approach that handles predictions as subjective opinions which are leaned by collecting evidence in the training phase [9]. The resultant predictor is a Dirichlet distribution parametrized over the subjective opinions and models uncertainty following the Dempster–Shafer Theory of Evidence(DDST) [15]. From the DST, consider K classes in our dataset we can formulate

$$u + \sum_{k=1}^{K} b_k = 1 \tag{26}$$

subject to $b_k \geq 0$ and $u_k \geq 0$. b_k is the belief mass for class k and m is the overall uncertainty mass. In our experiments, the model architecture is the same as the ones trained on MNIST dataset, except the last layer is replaced with ReLU to enforce non-negative evidence e_k. We can then define b_k and u as

$$b_k = \frac{e_k}{S} \quad \text{and} \quad u = \frac{K}{S} \quad \text{where} \quad S = \sum_{k=1}^{K}(e_k + 1) \tag{27}$$

$e_k + 1$ corresponds to the parameter $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_k]$ in the Dirichlet distribution and capture the prior belief that is learned in the training phase. The Dirichlet distribution is defined as

$$D(\boldsymbol{p}|\boldsymbol{\alpha}) = \begin{cases} \dfrac{1}{B(\boldsymbol{\alpha})} \displaystyle\prod_{k=1}^{K} p_k^{\alpha_i - 1} & \text{for } \boldsymbol{p} \in S_K \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

S_K is the multinomial distribution and $\beta(\boldsymbol{\alpha})$ is the multinomial beta function. The expected probablity of class k can tan be calculated as the mean of the corresponding Dirichlet distribution

$$\hat{p}_k = \frac{\alpha_k}{S} \tag{29}$$

We choose the mean squared error as our loss, suggested by the author based on empirical findings, and add a Kullback-Leibler (KL) divergence term into our loss as a regularization to penalize those divergences from equally distributed beliefs. The results of the evidential deep learning model are shown in Fig. 20. We visualize the class that the model tends to predict class "1" and "4". Note predictions of the middle of rotated images contain high uncertainty. We can see that the probability of class "1" and "4" is near 10% in these predictions, meaning that all 10 classes are assigned with nearly the same belief mass and the model just answer *"I don't know"* as an opinion over possible states.

We test our interpretability methods to see whether it can interpret uncertainty with an uncertainty aware model. The evidential deep learning model is compared with a basic model that is built upon the same architecture and trained under standard backpropagation. Fig. 21a and 21b show the results from composite LRP to interpret predictions on all rotated images. Interestingly, for the model that introduced uncertainty in training, the heatmaps show more blue pixels if the prediction is under high uncertainty. On the other hand, the heatmaps show very concentrated red pixels if the prediction is under low uncertainty. We also try to interpret with DTD as shown in 21c and 21d. For the basic model DTD cannot interpret any uncertainty because it filters out inhibitory relevance scores while

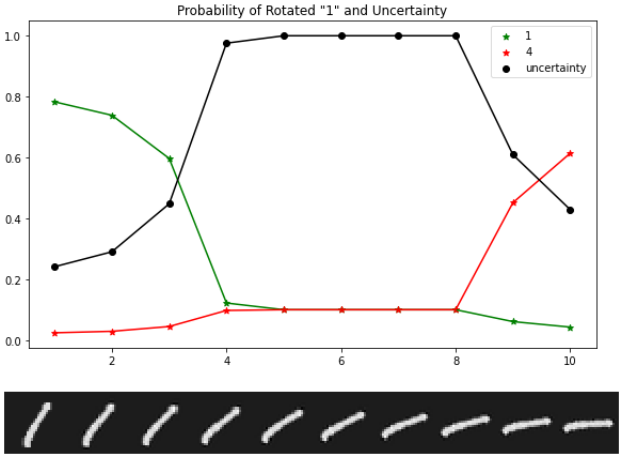
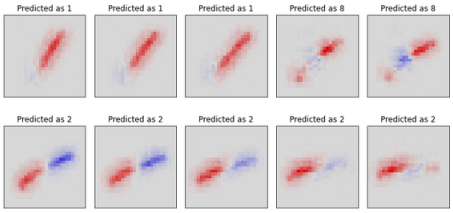
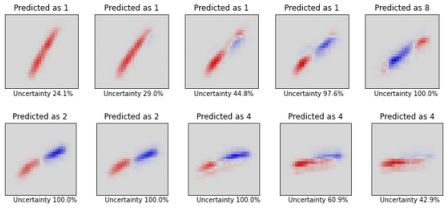


Figure 20: Evidential deep learning results for continuously rotated digit "1"

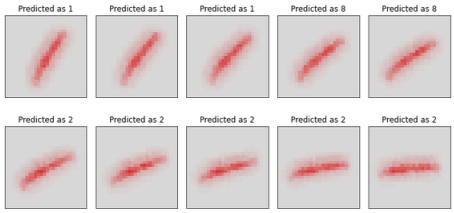
propagating to the input layer. Contrastingly, regarding evidential deep learning, model DTD can only capture model uncertainty if the uncertainty score is 100% as shown in Fig. 21c. This makes DTD not a suitable interpretability method for uncertainty estimation.



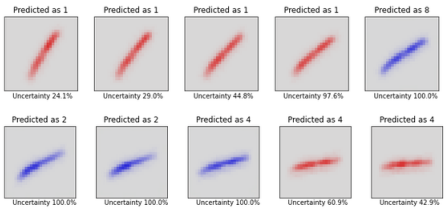
(a) LRP interpret basic model



(b) LRP interpret evidential deep learning model



(c) DTD interpret basic model



(d) DTD interpret evidential deep learning model

Figure 21: Heatmap results of basic model and evidential deep learning model interpreted with composite LRP (a,b) and DTD (c,d)

3.3 Uncertainty-aware DeepLIFT

In order to test the performance of DeepLIFT under uncertainty, we have executed three separate experiments. In the first experiment, we observe the performance of DeepLIFT for a normal CNN and for a CNN with Dropout layers. The results of this experiment can be seen in Fig. 22. In the multi-pass case, we do see significant improvement from the single-pass one, but the final result still lacks in detail in comparison to the normal CNN. Nevertheless, the results from the Multi-Pass DeepLIFT still provide a good enough interpretation of the decision of the model under test.
In the second experiment, we are once again using the Dropout-CNN, but this time we are going to

Figure 22: **Experiment 1:** Left: DeepLIFT with normal CNN Middle: Single-Pass DeepLIFT with Dropout-CNN Right: Multi-Pass DeepLIFT with Dropout-CNN

add random noise to our input image. Here, we choose a random image from the dataset and execute 2000 iterations of DeepLIFT for it. In each iteration, we, also, add random noise to the input. The results are depicted in Fig. 23. It is evident by the averaged images for each classification category that whenever the random noise mislead the model under test to miscategorize the input, the contribution score distribution changed in a way that resembles the wrongly predicted class.

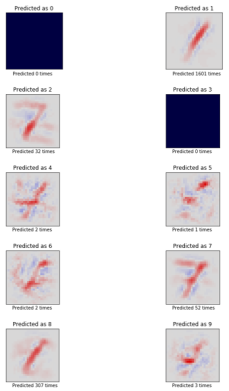


Figure 23: **Experiment 2:** Heatmaps for the consolidated results of 2000 iterations of DeepLIFT for an input image with added random noise

In our third and final experiment, we are examining the behavior of DeepLIFT for calibrated neural networks. A fully calibrated model outputs confidence scores which are equal to the true probability. Guo et al. discuss different calibration techniques and recognize *Temperature Scaling* as the best one in terms of performance [3]. Temperature Scaling is a calibration method which requires the division of the class logits by a constant T, which is referred to as the temperature. The value of the temperature is determined through optimization using the validation set. In our experiment, we have tried out Temperature Scaling for models for both MNIST and ImageNet. In the ImageNet case, we are using a pretrained AlexNet network and a subset of 1000 images from the ImageNet validation set. We have found out that the Temperature Scaling has a minimal effect on the results from DeepLIFT with the only difference being that the DeepLIFT results are a little less noisy. This is easily explained by the fact that the simplistic nature of Temperature Scaling does not provide a strong enough impact on the backward-flowing contribution signal so bigger changes were not expected. This argument is further demonstrated by the results displayed in Fig. 24.

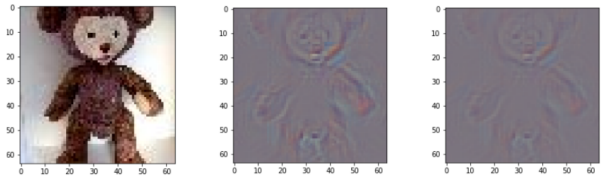


Figure 24: **Experiment 3:** *Left:* Input Image *Middle:* DeepLIFT for AlexNet *Right:* DeepLIFT for TS-AlexNet

4 Conclusion

In the last few years, the interest in machine learning and more specifically in neural networks has skyrocketed. Their ever-increasing popularity is predicated on the wide variety of applications that they have in many different activity domains which were formerly thought to be exclusively tied to human expertise. The demand for intelligent solutions and the common use of neural networks in many areas requires trust in machine learning models. Among others, trust is built upon the ability to explain and reason about the decisions that neural networks make.
As part of our project work, we have concerned ourselves with many different interpretability methods that try to either explain a single decision or find the prototypes which the model has learned for any of its classification categories. As outlined and visualized throughout the report, plenty of methods exist which provide meaningful interpretations of decisions made by neural networks. However, it is, also, evident that every method comes with it's positive and negative sides, it's limitations and problems. Based on that, the use of interpretability methods should always be predicated on both domain and model knowledge. Furthermore, we have conducted experiments including the use of interpretability methods in the uncertainty-aware settings which is a topic that is still relatively new and has not been heavily researched.
The full project work can be found in the project repository where also a detailed description of every folder inside the repository can be found. We are contributing all our weekly reports and our three presentations. Furthermore, we are providing notes for 14 research papers and a single video tutorial. The main deliverable of our project is the *nn_interpretability* package which provides an easy-to-use interface for all of the reviewed interpretability methods. The usage of any interpretability method from the packages is as easy as creating a new object and invoking a single function. We are also contributing an accompanying Jupyter Notebook for every implemented interpretability method which details its usage. Results in this report and in our presentations are all generated by the Jupyter Notebooks.

5 Workload Distribution

The project work has been contributed by Aleksandar Aleksandrov and Hans Hao-Hsun Hsu in the following manner. Aleksandar Aleksandrov has implemented General Activation Maximization, Activation Maximization in Code Space, Deconvolution, Occlusion Sensitivity, LRP Transpose version, DeepLIFT, CAM, Grad-CAM, Uncertainty-aware DeepLIFT, Temperature Scaling. Hans Hao-Hsun Hsu has implemented Deep Dream, Vanilla Backpropagation, Guided Backpropagation, Integrated Gradients, SmoothGrad, Taylor Decomposition, LRP, Monte Carlo Dropout, Uncertainty-aware LRP, Evidential Deep Learning. For every implementation there is, also, an accompanying Jupyter Notebook which showcases the implementation and generates all of the results and visualization which we have used throughout this report. Next to that, we have reviewed and provided notes for 14 papers and a single video tutorial. Hans Hao-Hsun Hsu has reviewed and provided notes for "Tutorial on Interpretable Machine Learning", "Striving for Simplicity: The All Convolutional Net", "Axiomatic Attribution for Deep Networks", "Layer-Wise Relevance Propagation: An Overview", "SmoothGrad:

removing noise by adding noise", "Inceptionism: Going Deeper into Neural Networks". Aleksandar Aleksandrov has reviewed and provided notes for "Methods for Interpreting and Understanding Deep Neural Networks", "Feature Visualization", "The Building Blocks of Interpretability", "Visualising image classification models and saliency maps", "Visualizing and understanding convolutional networks", "Learning Deep Features for Discriminative Localization", "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization", "Learning Important Features Through Propagating Activation Differences", "On Calibration of Modern Neural Networks". Everything that has not been explicitly mentioned here is the result of collaborative effort.

References

[1] Dumitru Erhan, Y. Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *Technical Report, Univeristé de Montréal*, 01 2009.

[2] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, 2015.

[3] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks, 2017.

[4] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. *Layer-Wise Relevance Propagation: An Overview*, pages 193–209. 09 2019.

[5] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, Feb 2018.

[6] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks, 2015.

[7] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

[8] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016.

[9] Murat Sensoy, Lance Kaplan, and Melih Kandemir. Evidential deep learning to quantify classification uncertainty, 2018.

[10] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences, 2017.

[11] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2013.

[12] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda B. Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *CoRR*, abs/1706.03825, 2017.

[13] Jost Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. 12 2014.

[14] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *CoRR*, abs/1703.01365, 2017.

[15] Ronald R. Yager and Liping Liu. *Classic Works of the Dempster-Shafer Theory of Belief Functions*. Springer Publishing Company, Incorporated, 1st edition, 2010.

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

[17] B. Zhou, A. Khosla, Lapedriza. A., A. Oliva, and A. Torralba. Learning Deep Features for Discriminative Localization. *CVPR*, 2016.