Explaining Decisions in Image Recognition

Understanding black-box classifiers using Layerwise Relevance Propagation

Robin Schmitt
Matr. Nr. 88353

Data Science (M.Sc.)
Practical work
Module 40200

Hochschule
Albstadt-Sigmaringen
Albstadt-Sigmaringen University

Germany
08.02.2020

As the adaptation rate of machine learning in a multitude of tasks is accelerating, it is more important than ever to enable developers to understand decisions of their machine learning models. Additionally, there are regulatory demands e.g. in autonomous driving that might require manufacturers in the automotive field to explain driving behaviour in case of accidents. As machine learning models, especially neural networks, are refined to solve more complex tasks, it is also getting harder to intuitively understand the decisions they make. In order to allow a developer or user of neural networks to understand the decisions of the implemented and trained neural network model, the framework of layerwise relevance propagation that was described by Bach et al [2] is implemented in the keras/tensorflow framework.

1 Introduction/State of the art

Machine learning models like decision trees (and variations like random forests) are not only a viable and powerful alternative to complex models like neural networks [10] , but also have a distinct advantage: Their decisions or regressions are easy to understand and debug. In the case of a decision tree, reviewing a classification by following a path along all nodes is a way to gain intuition and confidence in the classifier. This is key when it comes to using a classifier in a production environment, since stakeholders like quality departments or management often have to approve the presented solution. Besides the advantages during development that come from easier debugging, it has been shown that an explanation next to a recommendation (e.g. in movie recommendations) increases the user's acceptance and understanding. [6]

Recent accidents that happened in the testing fleet of companies such as Uber show how important it is to know why a self-driving vehicle chose to act in a certain way. The report that was created by the National Transport Safety Board of the USA for accident HWY18MH010 dedicates a whole paragraph describing how the pedestrian was classified as different objects before the impact [8]. This example crroborated how there is a big public interest in explanation of automated systems in general.

Additionally the General Data Protection Regulation of the European Union defines transparency as one of the principles relating to processing of personal data in article 5 [16]. Depending on how the GDPR will be interpreted by jurisdiction, transparency can mean that the subject must be informed about the reasons why e.g. a profiling algorithm made a decision. If such requests for information will be allowed and should their number increase steadily, an established way to understand decisions of machine learning models will be necessary.

Putting complex neural neworks in contrast to simple decision trees, it can be seen that classifications calculated through them can not be understood easily due to the multitude of layers and neurons. There has been an uprising number of research in the field of explainable artificial intelligence [11] focusing on different aspects and ways to understand the relationship between input and output of models [5]. There have been attempts to gain insights into the black-box. One attempt is shrinking complex models (e.g. neural networks) into simpler models (e.g. decision trees) that are easier to understand while trying to maintain the predicting accuracy [7]. A different approach is giving explanations based on the input (e.g. for each pixel in a picture) why a certain classification was chosen [2, 15].

This work will focus on the latter and explains a concrete implementation of Layerwise Relevance Propagation (LRP) [2] and explores the mathematical foundations, the implementation as well as parameters.

Besides LRP, other methods that aim to explain a classification and denote to each input a positive or negative contribution have been proposed. A commonly used technique is Locally Interpretable Model-agnostic Explanations (LIME). LIME is a method that aims to give local explanations of classifications by first permuting the input data close to the local feature, then calculating a similarity score between the original and permuted data, after that calculating the predicition score for all permuted data and using this information to find the most influential permutations by fitting simple models and using their feature weights [15]. Since this method is model-agnostic and does not rely on access to the model itself, it can be used accross different domains such as natural language processing of image recognition, where feature context information is relevant for a model but also for finance and use cases like fraud detection, where context between input dimensions is not always relevant [15]. However since LIME is based on permuting the input data and local explanations, the results are of stochastic nature and are not deterministic. The quality of an explanation is dependend on the number of permutations that the algorithm applies to the input data.

Other papers that aim to give explanations for black-box classifiers use gradient-based methods [4]. Local gradients for a sample are approximated and used as vectors describing whether an increase in value contributes positively or negatively to a certain classification. This process is similar to sensitivity-analysis and only gives local explanations for numerical input data. In the paper, Parzen windows are referenced as substitute if the classifier $f(x)$ is not available itself and a local approximation of the gradient has to be made. Since fitting the Parzen windows themselves is a learning task that depends on hyper-parameters, this method has to be adapted and supervised for each classification task [4].

2 Mathematical foundations of LRP

Layerwise Relevance Propagation is a technique that, similar to error-backpropagation, calculates a relevance matrix that is propagated from the model output to the input. The goal is to attribute the quantitative contribution to the output $f(x)$ to every pixel of the input x. The assumption is that the model output is thresholded between 0 and 1 [2]. The following notation is used throughout this work:

- R, Relevance score
- l, Layer
- d, Dimension
- $R^l_{(d)}$, Relevance score of layer l at dimension d

In the first section of this chapter, the LRP framework as defined in [2] is explained with regards to neural networks for image recognition. The second section adds further definitions and narrows the mathematical explanations down to fully-connected layers. Sections three and four respectively describe how the LRP framework can be vectorized for simple LRP and the a/b-rule. Last, the LRP algorithm is described in section five.

2.1 General framework of LRP

Since LRP is only a proposed logic to calculate the relevance scores of each input pixel, a framework of rules/equations is proposed [2]. Equation (1) depicts that the sum of the relevance score accross all dimensions per layer stays constant. The intuition behind this is that the model's decision is distributed accross all input pixels and the sum of those pixel relevances can't be greater than the classification result (denoted as $f(x)$).

$$f(x) = \sum_{d \in l+1} R_d^{(l+1)} = \sum_{d \in l} R_d^l = \ldots = \sum_d R_d^1 \tag{1}$$

Since the propagation of relevance scores through the layers can be viewed as messages between the neurons, the second equation determines that the sum of all messages a neuron sends is equal to the neurons relevance.

$$R_j^{l+1} = \sum_{i:\, i\, is\, input\, for\, neuron\, j} R_{i \leftarrow j}^{(l,l+1)} \tag{2}$$

The messages that are referenced in equation (1) can be understood similarly to messages in factor-graphs. A neuron can reference all incoming and outgoing messages. And the sum of all these messages per layer must be equal so that the relevance is conserved. This property will be used in the final implementation to calculate a check-sum per layer. A simplified example for a relevance propagation that satisfies equations (1) and (2) in a simple multilayer neural network where neuron i with x_i sends a message $x_i w_{ij}$ to neuron j and h denoting all neurons in layer $l+1$ can be written as follows:

$$R^{(l,l+1)}_{i \leftarrow j} = R^{(l+1)}_j \frac{x_i w_{ij}}{\sum_h x_h w_{hj}} \tag{3}$$

This equation assumes the relevance of neuron j in layer $l+1$ was already calculated. The local contribution of neuron i is then weighted by the global contributions of all neurons h in layer l.

Figure 1 is a visualization of a simple network during feed-forward and relevance propagation. The relevance is backpropagated through each layer. During feed-forward the sum of the inputs to neuron 5 is $f(x)$. During the relevance propagation $f(x) = R^{(1)}_5$. This holds also for multiclass classification, where the relevance score of the output layer in relation to a specific class is equal to the label's one-hot encoding. Furthermore the principle of propagating messages from the output layer all the way to the input layer can be seen.

2.2 Possible implementations of LRP

The description of the LRP framework in chapter 2.1 allows for different implementations that just have to follow equations (1) to (3). All vectorized formulas in the upcoming section are written for fully-connected layers. Adapting them to other types of layers like convolutions or pooling is possible but will not be discussed in this work. The implementation of LRP however is done for all types of layers in a later section.

According to how multilayer neural networks are defined, the following definitions are made:

$$z_{ij} = x_i * w_{ij} \tag{4}$$

$$z_j = \sum_i z_{ij} + b_j \tag{5}$$

$$x_j = g(z_j) \tag{6}$$

Here, the function $g()$ is a non-linear activation function, such as $tanh()$.

2.3 Simple LRP

The formula to calculate the relevance score of a layer l given the relevance of layer $l+1$ as well as the activation and weights is given in equation (3). The intuition behind the messages, as defined in equation (3), is calculating for each neuron in layer l the local contribution to each neuron in layer $l+1$ divided by its global contribution to all neurons in layer $l+1$.

To avoid division by zero, a normalization factor is added. In the case of equation (7), the lowest possible value $epsilon = eps$ is chosen.

$$R^{(l,l+1)}_{i \leftarrow j} = R^{(l+1)}_j \frac{x_i w_{ij}}{\sum_h x_h w_{hj} + eps} \tag{7}$$

Scaling the LRP calculation to a whole layer by vectorizing the operations, can be expressed by a succession of matrix multiplications. Considering $d1$ as the dimensionality of layer l, $d2$ as the dimensionality of layer $l+1$ and n as the batch size, the first step is calculating the denominator of equation (7). A matrix notation is used where the lowered brackets show the dimensions:

$$Z_{(n,d2)} = input_{(n,d1)} weights_{(d1,d2)} + eps \tag{8}$$

The next step consists of normalizing the relevance matrix of layer $l+1$ by element-wise dividing by Z.

$$S_{(n,d2)} = R^{l+1}_{(n,d2)} / Z_{(n,d2)} \tag{9}$$

As last step, the relevance matrix of layer l is calculated by multiplying S with the transposed weight matrix and then element-wise with the layer input.

$$R^l_{(n,d1)} = S_{(n,d2)} weights^T_{(d1,d2)} * input_{(n,d1)} \tag{10}$$

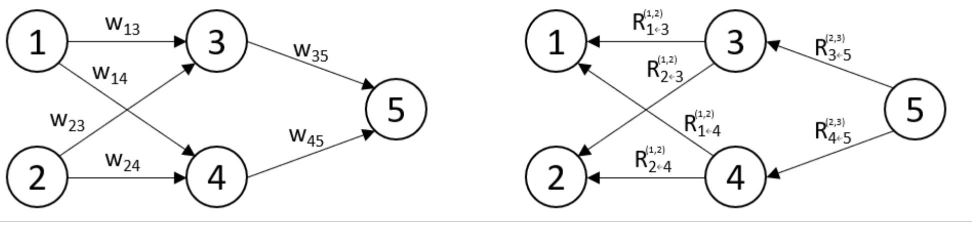
1
2
3
4
5
w_{13}
w_{14}
w_{23}
w_{24}
w_{35}
w_{45}
$R^{(1,2)}_{1\leftarrow 3}$
$R^{(1,2)}_{2\leftarrow 3}$
$R^{(1,2)}_{1\leftarrow 4}$
$R^{(1,2)}_{2\leftarrow 4}$
$R^{(2,3)}_{3\leftarrow 5}$
$R^{(2,3)}_{4\leftarrow 5}$

Figure 1: A feed-forward network and the back-propagation of Relevance

2.4 a/b - Rule

A more advanced version of LRP as proposed in [2] can be described as a/b-rule. The underlying idea is to separate the positive and negative relevance scores during the propagation in order to be able to handle them separately. Use-cases are e.g. to use only positive or only negative relevances or to deliberately weigh the positive and negative impacts differently. In accordance to equation (3), the relevance messages are calculated, but with separated multipliers a and b. To split the positive and negative relevance contributions, the following definitions are made:

$$z_j^+ = \sum_i z_{ij}^+ + b_j^+ \quad (11)$$

$$z_j^- = \sum_i z_{ij}^- + b_j^- \quad (12)$$

The $+$ and $-$ indicate the positive and negative parts of z and b.

The calculation of relevance messages (similar to equation (3)) is conducted as follows:

$$R_{i\leftarrow j}^{(l,l+1)} = R_j^{(l+1)} * (a * \frac{z_{ij}^+}{z_j^+} + b * \frac{z_{ij}^-}{z_j^-}) \quad (13)$$

The vectorization is done similarly to equations (8, 9, 10).

2.5 LRP Algorithm

For either of the previous detailed implementations of LRP, an algorithm that calculates the relevance scores throughout the network architecture is defined. Here, the loop is structured similar to how feed-forward and backpropagation work and L denotes the layers of the multilayer network. As described above, the relevance propagation starts with the network's output and propagates until the input layer.

```
input : $R^L = f(x)$
for $l \in (L, \dots, 0)$ do
    Calculate $R_{i\leftarrow j}^{(l,l+1)}$ (using simple
      LRP and equation (7) or the
      a/b-rule and equation (13))
    $R_i^{(l)} = \sum_j R_{i\leftarrow j}^{(l,l+1)}$
end
output: $\forall d:\ R_d^1$
```

Algorithm 1: propagating relevance through the network's layers [12]

3 Python-implementation of a/b LRP

Up to this point, all definitions and equations were done programming language and framework agnostic. The acutal implementation is done for python with the tensorflow and keras modules. The following versions are used:

- python: 3.6.7
- numpy: 1.15.4
- tensorflow: 1.12.0
- keras: 2.1.6

In [12], a brief introduction is given on how to implement LRP. The implementation below was aligned with the source's recommendation.

3.1 Helper functions

The calculation of the relevance score requires pre-computed matrices. Helper functions were defined in order to:

- get precalculated input, weight, bias and output matrices per layer
- define a standardized plotting function, that combines the original image with a relevance score heatmap

As an example, the helper function to get each layer's weights is lined out in listing 1. The function is implented in such a way that a list of weight matrices is returned. In the case of layers without defined weights (e.g. pooling-layers), $None$ is appended to the list.

Listing 1: extracting weights from the model

```
def get_weights(model):
  weights = []
  for i in range(0,len(model.layers)):
    try:
      weights.append(model.layers[i] \
          .get_weights()[0])
    except:
      weights.append(None)
return weights
```

3.2 a/b-rule for fully connected layers

Since fully connected layers were already discussed in the sections beforehand, the code in listing 2 is a straightforward implementation of equations (8) to (10) but with the extra steps necessary as defined in equation (13). The positive z^+ and negative z^- are calculated separately and then weighted according to a and b. This specific implementation of the a/b-rule leads to a restriction where $a - b = 1$ has to be ensured, because the sum of the negative and positive components must be equal to 100%. Here, values of $a = 2$ and $b = 1$ are chosen. Their impact on the result is discussed in a later section.

3.3 a/b-rule for pooling layers

In general, two different pooling-layers are used in convolutional neural networks, namely either max-pooling or average-pooling. Usually, pooling layers are used after convolution layers with the goal to make the convolution output less influenced by local translation of the input. Just like convolution layers, a window is moved across the input and a certain function (in this case either average or max) is calculated on the window. Since max-pooling layers are chosen for the neural networks in this work, the relevance propagation was only implemented for that type of pooling layers.

Calculating the relevance propagation in a max-pooling layer is done by reversing the operation:

- Calculate Z and S according to equations 8 and 9
- Re-distribute the R matrix by reversing the max-pooling operation (which is similar to calculating the gradient of a max-pooling layer)
- Multiply the re-distributed R with the layer input

After re-distributing R by reversing the max-pooling operation, the feature dimensionality is increased. However, since the number of R-values stays constant, the R-matrices are sparser. Listing 3 shows the final python implementation.

3.4 a/b-rule for convolution layers

The implementation for convolutional layers was done using a predefined gradient-function in the tensorflow library. The function $tf.nn.conv2d\_backprop\_input$ computes the gradient of the convolution step with regard to the input into the convolution layer. In listing 4, the python code can be found. Since it is based on the a/b-rule, all operations are split into the positive and negative components and afterwards weighted according to a and b.

Listing 2: a/b-rule for fully connected layers

```
def relprop_lin_ab(layer, R, inputs, weights, biases, a=2, b=1):
  Z_p = np.matmul(inputs[layer],weights[layer]*(weights[layer]>=0))+ \
          biases[layer]*(biases[layer]>=0)+eps
  Z_n = np.matmul(inputs[layer],weights[layer]*(weights[layer]<0))- \
          biases[layer]*(biases[layer]<0)-eps
  S_p = R/Z_p
  S_n = R/Z_n
  C_p = np.matmul(S_p,(weights[layer]*(weights[layer]>=0)).T)
  C_n = np.matmul(S_n,(weights[layer]*(weights[layer]<0)).T)
  R = (C_p*inputs[layer]*a-C_n*inputs[layer]*b)
return R
```

Listing 3: a/b-rule for max-pooling layers

```
def relprop_pooling(layer, R, inputs, outputs, model):
  pool_height, pool_width = model.layers[layer].pool_size
  stride_up, stride_side = model.layers[layer].strides
  placeholder = np.zeros(inputs[layer].shape)
  Z = outputs[layer]
  S = R/(Z+eps)
  for l in range(outputs[layer].shape[3]):
      for i in range(outputs[layer].shape[2]):
          for j in range(outputs[layer].shape[1]):
              placeholder[0,i*stride_side:(i*stride_side+pool_width),j*stride_up:\
              (j*stride_up+pool_height),l]=S[0,i,j,l]*((inputs[layer][0,i*stride_side:\
              (i*stride_side+pool_width),j*stride_up:(j*stride_up+pool_height),l]==\
              outputs[layer][0,i,j,l])&(outputs[layer][0,i,j,l]!=0))
  R = placeholder*inputs[layer]
return R
```

Listing 4: a/b-rule for convolution layers

```
def relprop_conv2d_ab(layer, R, inputs, weights, model,biases, a=2, b=1):
  Z_p = K.eval(K.conv2d(tf.constant(inputs[layer]),tf.constant(weights[layer]* \
        (weights[layer]>=0)),strides=(1,1), padding=model.layers[layer].padding))+ \
        biases[layer]*(biases[layer]>=0)+eps
  Z_n = K.eval(K.conv2d(tf.constant(inputs[layer]),tf.constant(weights[layer]* \
        (weights[layer]<0)),strides=(1,1), padding=model.layers[layer].padding))- \
        biases[layer]*(biases[layer]>=0)-eps
  S_p = R/Z_p
  S_n = R/Z_n
  C_p = K.eval(K.tf.nn.conv2d_backprop_input(inputs[layer].shape,weights[layer]* \
        (weights[layer]>=0),S_p, (1,1,1,1),padding=model.layers[layer].padding.upper() ))
  C_n = K.eval(K.tf.nn.conv2d_backprop_input(inputs[layer].shape,weights[layer]* \
        (weights[layer]<0),S_n, (1,1,1,1),padding=model.layers[layer].padding.upper() ))
  R = (C_p*inputs[layer]*a-C_n*inputs[layer]*b)
return R
```

3.5 Algorithm implementation

At last, the algorithm as described in algorithm 1 has to be implemented as a loop through all layers. The function recieves six inputs when it is called:

- $model$: the keras model object of the classifier that shall be explained
- img: the input image in the same shape and format that was used for training
- R: the initial R that is handed over to the LRP-function depends on the use-case and will be explained in a later chapter
- a, b: as written above, these values can be chosen based on the premise $a - b = 1$
- $verbose$: a boolean flag on whether a debug information shall be printed

When the function $relprop$ is called, the first step is calculating per layer the inputs, outputs, weights and biases. Then the function loops through all layers, starting with the output layer. Depending on the type of layer, it chooses the correct LRP-function to calculate the relevance propagation. In this module, the LRP algorithm is implemented for the following types of layers:

- fully-connected layers
- flatten layers
- max-pooling layers
- Conv2D layers

4 Evaluating the LRP implementation and parameters

With the framework of LRP described and the concrete implementation lined out, this section is dedicated to first setting up and training two neural networks (one for the MNIST [14] and one for the CIFAR-10 [13] dataset), then comparing differences due to model architecture, parameter tuning and inital values for relevance in the output layers.

4.1 Datasets MNIST and CIFAR-10

The MNIST dataset is a widely used, very popular dataset containing handwritten digits from 0-9 in a monochrome 28x28 pixel format. Due to the small picture size (and thus dimensionality) as well as the well organized and normalized structure, it is widely used in test applications and also referenced by Bach et al [2]. It consists of 70.000 samples of which 60.000 were used for training and 10.000 as test set.

The CIFAR-10 dataset on the other hand consists of 32x32 pixels with RGB-channels and contains 10 classes, namely airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships and trucks. Since the CIFAR-10 images are not much larger than the MNIST pictures (regarding resolution) but contain much more complex objects, they are a fitting alternative to see whether the LRP results are usable also for tougher classification tasks. Of the 60.000 samples, 50.000 were used for training and 10.000 as test set.

4.2 Neural-networks used for the classification task

As mentioned in section 3, keras is used as machine-learning library in this work. Keras is an open-source high-level application programming interface (API) built on top of tensorflow, CNTK and theano [9] and aims to enable a quick realization of ideas and experiments. Keras offers two ways to declare the architecture of a neural network: the sequential model and the functional API. The sequential model is a simple way to create a model by stacking layers. Changing or adding a layer is as simple as adding or changing a line of code. The functional API on the other hand is more complex to use but also offers more flexibility. Each layer is represented as a function call that returns a result, thus individual layer outputs can easily be referenced.

For the given task of multiclass classification of pictures, a neural network with fully-connected, convolutional and pooling layers is chosen. In general, it is common to build the

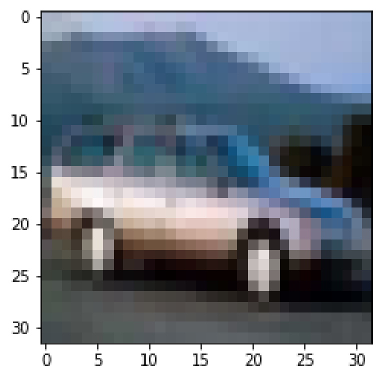

Figure 2: Example of the CIFAR-10 training set, depicted class is car

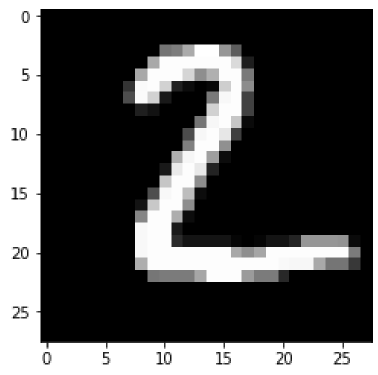

Figure 3: Example of the MNIST training set, depicted class is 2

neural network according to the following order:

- Input
- Convolution
- Pooling
- Fully-Connected
- Output

Convolutional Neural Networks (CNN) are used in image recognition/classification tasks due to their good performance in that field. In CNNs, convolution filters are used for feature extraction. These features are then used to identify objects or correctly classify an image. For both datasets, a similar architecture was chosen based on empirical evidence after a testing phase. The architecture can be seen in figure 4.

4.3 LRP Results and interpretation

Using the LRP-module that was defined in the sections above, at first the relevance matrix was calculated for the MNIST dataset. Since the relevance propagation assigns to each input dimension a relevance score, the relevance matrix can be depicted as an image with the same dimensions as the input. In order to display the relevance matrices, a heatmap with the following attributes will be used:

- Centered at 0
- Positive values are shown blue
- Negative values are shown red
- In case of RGB-Inputs, the sum of all channels is calculated

As first example, an image of the MNIST dataset that belongs to class 3 is depicted in figure 5 and the according relevance matrix with $a = 2; b = 1$ and the target class 3 is depicted in figure 6.

A first observation is that more areas with positive than negative relevance scores are present. Only the part where the lower arc points upwards on the left side is marked with negative relevance. This can be explained by the fact that the relevance was calculated in regards to class 3, which is the correct class of the example. The highest relevance scores are present in the upper arc, indicating that this area is more important for the classification regarding class 3 than the lower arc.

When calculating the relevance matrix for the image in figure 5 with regards to target class 9, the result differs as can be seen in figure 7. On a first glance, the resulting relevance matrix is very similar to figure 6, however an important difference is, that the lower arc now has higher relevance scores than the upper arc. This indicates, that in this example, the upper arc is a strong indicator for class 3, while the lower arc is an indicator for class 9.

Taking this example as a baseline, in a first step, the initially chosen a and b values are varied

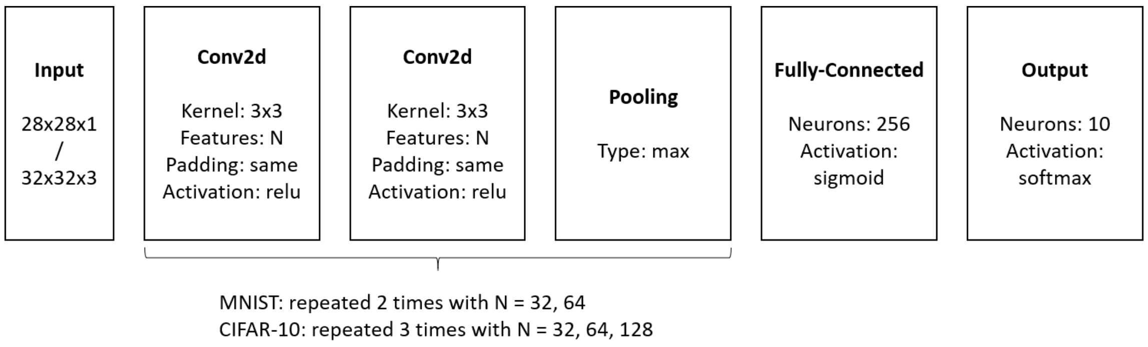

Figure 4: The neural network architecture for the MNIST and CIFAR-10 dataset.

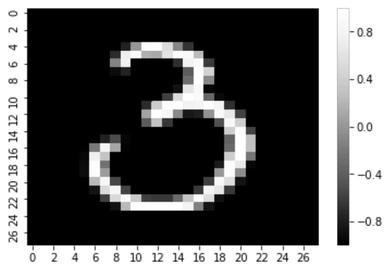

Figure 5: MNIST image of class 3

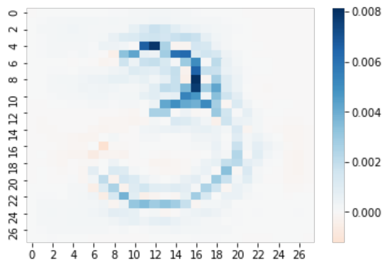

Figure 6: Relevance matrix on actual class 3; target class 3; $a = 2, b = 1$.

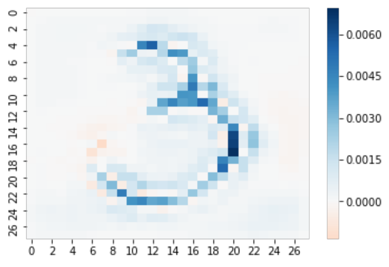

Figure 7: Relevance matrix on actual class 3; target class 9; $a = 2, b = 1$.

now. Keeping the restriction $a - b = 1$ in mind, first values of $a = 3; b = 2$ are chosen.

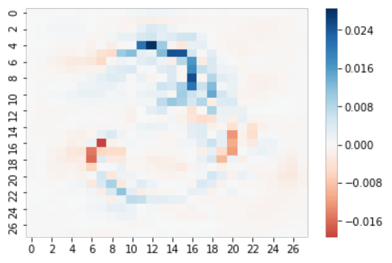

Figure 8: Relevance matrix on actual class 3; target class 3; $a = 3, b = 2$.

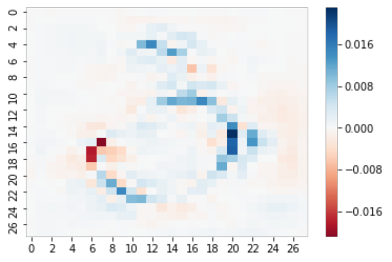

Figure 9: Relevance matrix on actual class 3; target class 9; $a = 3, b = 2$.

Comparing figure 8 and figure 9 to the corresponding results in figure 6 and figure 7, a first observation is that there is more noise in the result. The general observation that high relevance scores in the upper arc correspond to the correct class 3 while higher relevance scores in the lower arc correspond to target class 9 is similar to the example above. Additionally, the areas without input pixels (e.g. at the end of the upper arc) are now represented whereas they were not shown to be of relevance in the previous example. Even though the result shows noise, the values of $a = 3, b = 2$ seem to give a more detailed explanation since it keeps more of the negative relevances. Additionally, figure 10 shows the relevance matrices of actual class 3 and all possible target classes.

Next, the number of training epochs is varied in order to identify whether a difference in training duration can be seen in the relevance matrices. Again, values of $a = 3, b = 2$ are chosen, as well as the same sample image.

For the relevance matrix in figure 8, a second model was trained with only 2 epochs. Comparing figure 8 with figure 11, significant differences can be observed. The relevance values in the end of the lower arc show strong negative values in figure 8, in figure 11 on the other hand there are not significant negative values in this area. Additionally, the central part where both arcs connect show a positive relevance in figure 9 whereas they show a negative relevance in figure 11.

5 Conclusion

After the foundations of LRP were defined in section two and a concrete implementation in python was done as outlined in section three, figures 6 - 11 show first results of LRP using the MNIST dataset and a CNN. Last, the question whether or not LRP helps to intuitively understand the decision making of models such as neural networks must be discussed.

5.1 Validating the results

In order to validate the results of section 4 and to indicate whether LRP correctly identifies the relevant input pixels, an MNIST example picture of actual class 3 is used and the following steps are done:

- Calculate prediction
- Calculate the relevance matrix
- Find the 10 pixels with lowest relevance
- Flip these pixels (multiply by -1)
- Recalculate the prediction

If these steps lead to an increased y-value for the actual class, it can be concluded that LRP correctly identifies the pixels with positive and negative contributions. Taking the example of figure 5, the y-value of class 3 is 0.9962. The result of flipping the 10 pixels with the lowest negative relevance as shown in figure 6 to -1 is depicted in figure 12. With this as input for the CNN, the y-value for class 3 now is 0.9978 and thus increased.

Repeating this process for 500 random MNIST-images gave the following results:

- The average prediction improvement was 0.016, the standard deviation was 0.056
- In 17% of all cases, the prediction was worse
- The other 83% had an improved prediction result
- The highest improvement was 0.523, the lowest -0.046

Additionally, doing the same experiment but flipping the 10 pixels with the highest relevance, gave the following results:

- The average prediction change was -0.138, the standard deviation was 0.217
- In 100% of all cases, the prediction was worse
- The smallest change was 0, the highest change was -0.941

This leads to the conclusion that the goal to attribute to every input pixel the relevance to a target class was achieved. Additionally, this information can not only be used to understand the decision making of classifiers, but also to possibly enhance the input data.

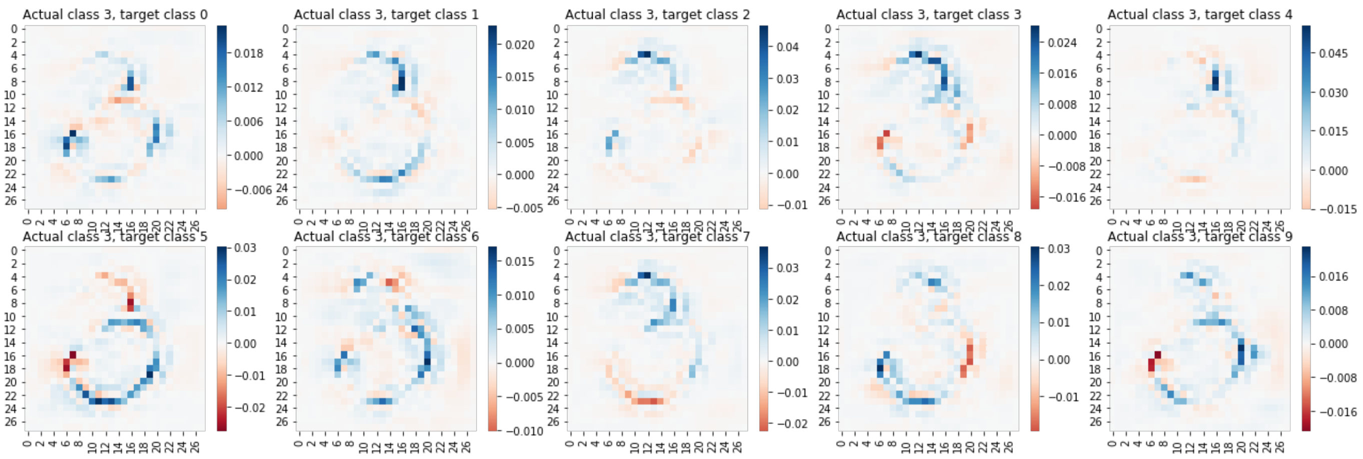

Figure 10: Relevance matrix on actual class 3 with all other target classes; $a = 3, b = 2$.

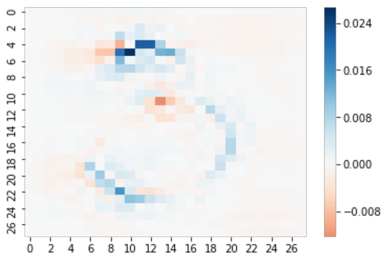

Figure 11: Relevance matrix on actual class 3; target class 9; $a = 3, b = 2$; 2 training cycles.

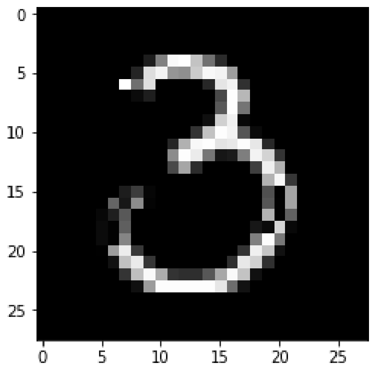

Figure 12: Adapted figure 5 by flipping the 10 pixels with lowest relevance.

Using LRP with the CIFAR10 dataset gave similar results. As an overview, figures 13 - 15 depict examples.

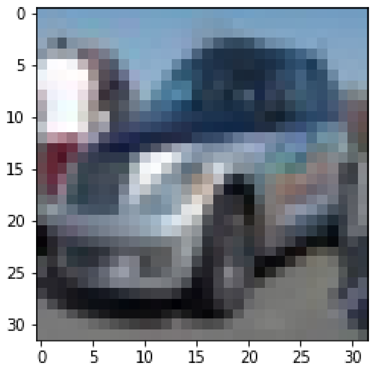

Figure 13: Sample image of CIFAR10 dataset, class automobile.

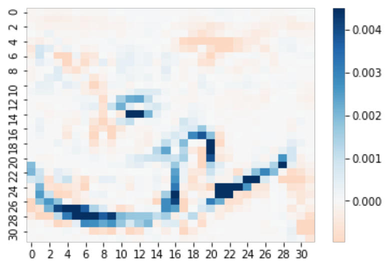

Figure 14: Relevance matrix on actual class automobile, target class automobile; $a = 3, b = 2$.

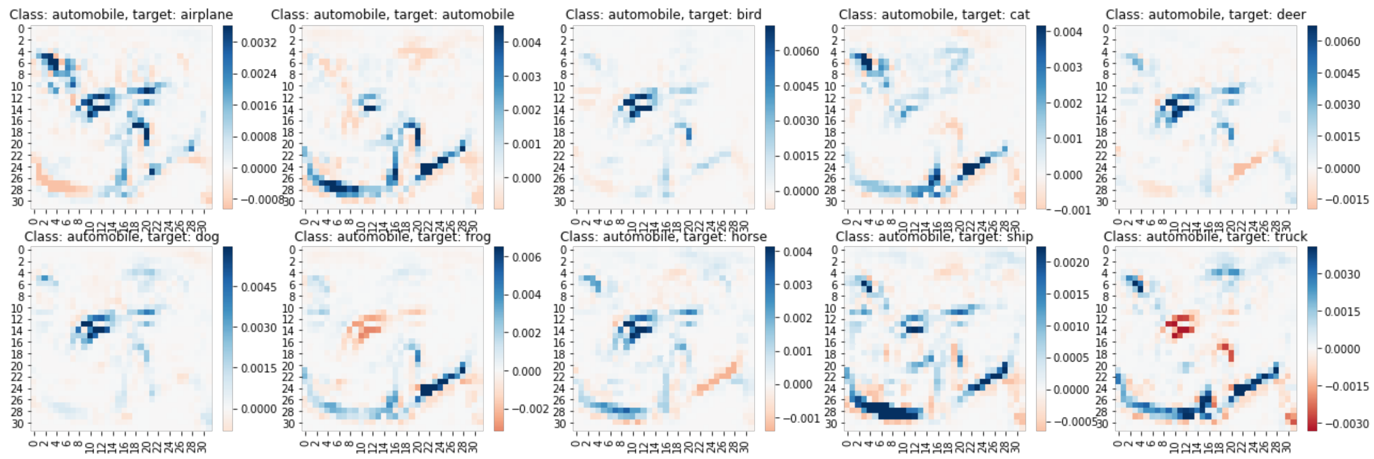

Figure 15: Relevance matrix on actual class automobile with all other target classes; $a = 3$, $b = 2$.

5.2 Next steps and further research

In order to make this LRP implementation usable with available image recognition models such as AlexNet, further layer types must be implemented. Next, a comparison to other techniques such as LIME can be done in order to evaluate whether other approaches to explaining decisions of classifiers are superior.

Recent research aims to further enhance the explanation of decisions by not only attributing to each input pixel the relevance, but to identify the concepts that are identified by classifiers [3, 1]. Additionally, major players in machine learning start to include tools (such as what-if, Shapley Values) in their framework to enable developers to seamlessly debug their models

References

[1] Amirata Ghorbani et al. *Towards Automatic Concept-based Explanations*. 2019 (cit. on p. 13).

[2] Bach et al. *Layerwise Relevance Propagation*. 2015 (cit. on pp. 2, 3, 5, 8).

[3] Been Kim et al. *Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV)*. 2018 (cit. on p. 13).

[4] David Baehrens et al. *How to Explain Individual Classification Decisions*. 2010 (cit. on p. 3).

[5] Joe Vaughan et al. *Explainable Neural Networks based on Additive Index Models*. 2018 (cit. on p. 2).

[6] Jonathan L. et al. *Explaining Collaborative Filtering Recommendations*. 2000 (cit. on p. 2).

[7] Nicholas Frosst et al. *Distilling a Neural Network Into a Soft Decision Tree*. 2017 (cit. on p. 2).

[8] National Transport Safety Board. *Preliminary report HWY18MH010*. 2018 (cit. on p. 2).

[9] François Chollet et al. *Keras*. `https://keras.io`. 2015 (cit. on p. 8).

[10] Manuel Fernandez Delgado. *Do we Need Hundreds of Classifiers to Solve Real World Classification Problems*. 2014 (cit. on p. 2).

[11] David Gunning. *Explainable Artificial Intelligence*. 2017 (cit. on p. 2).

[12] heatmapping.org/tutorial/. *Tutorial: Implementing Deep Taylor Decomposition / LRP*. 2017 (cit. on p. 5).

[13] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. “CIFAR-10 (Canadian Institute for Advanced Research)”. In: (). URL: `http://www.cs.toronto.edu/~kriz/cifar.html` (cit. on p. 8).

[14] Yann LeCun and Corinna Cortes. “MNIST handwritten digit database”. In: (2010). URL: `http://yann.lecun.com/exdb/mnist/` (cit. on p. 8).

[15] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. *Why Should I Trust You?: Explaining the Predictions of Any Classifier*. 2016 (cit. on pp. 2, 3).

[16] European Union. *General Data Protection Regulation*. 2018 (cit. on p. 2).