An investigation of different interpretability methods used to evaluate a prediction from a CNN model

Mona Heggen

Thesis submitted for the degree of
Master in Computational Science: Imaging and Biomedical Computing
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2021

An investigation of different interpretability methods used to evaluate a prediction from a CNN model

Mona Heggen

An investigation of different interpretability methods used to evaluate a prediction from a CNN model

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

In this thesis we investigate different interpretability methods for evaluating predictions from Convolutional Neural Networks. We look at research on several explanation methods with a focus on Local Interpretable Model-agnostic Explanations (LIME) and Layer-wise Relevance Propagation (LRP). Our goal is to investigate different interpretability methods and how robust they are in comparison to each other.

We do initial experiments by testing a set of images with Guided Backpropagation, Gradient-weighted Class Activation Mapping (Grad-CAM), LIME and LRP. In the next set of experiments we focus on LRP and LIME. The models we use are VGG16 with and without batchnorm layers. We use rotation and Gaussian noise to transform the input images. To measure the robustness we use Root Mean Square Error (RMSE). The transformation is added to the input and sent through the model. The output from the model is sent through the interpretability method. The resulting heatmap for the transformed image is then compared with the original heatmap to measure the RMSE score. We use a set of small transformations and a set of more extreme transformations. The transformations we use for rotation are between 0.5-10 degrees and 15-40 degrees. For the Gaussian noise we use σ between 0.01-0.10 and 0.25-10.0.

We observe that LIME focuses on super pixels and will therefore be less robust for transformations compared to LRP. We find that methods which emphasises on both positive and negative contributions, such as LRP and Grad-CAM are more helpful since they highlight the regions that contribute and work against the prediction in the image.

When implementing LRP with models using batchnorm layers we find that this give unreliable results. We handle this by merging the batchnorm layers with the corresponding convolutional layer before backpropagating LRP.

Our experiments show that the explanation from the interpretability method correlates significantly with the models robustness. Though in some cases the robustness of the model is not reflected in the interpretability method and this is especially noticeable when Gaussian noise are applied to the input in the LIME experiments.

Foreword

First I would like to express my sincere gratitude to my supervisor Anne H. Schistad Solberg (IFI) for invaluable help and guidance while working on this thesis.

Second I would like to thank Johan M. Grønstad, my rubber duck. I would not have completed this without your encouragement, feedback and patience.

Third I would like to thank Odd M. Heggen for his support, motivation and finding the name for this thesis while proofreading it.

Finally, I would like to thank Magnus and Signe for their endless patience and optimism.

Contents

Acronyms

ADAM	Adaptive Moment Estimation
B-LRP	Bayesian Layer-wise Relevance Propagation
BN	Batchnorm
BNN	Bayesian Neural Network
CAM	Class Activation Mapping
CIFAR	Canadian Institute for Advanced Research
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CSC	Cosine Similarity Convergence
Deconvnet	Deconvolutional Network
DNN	Deep Neural Network
DTD	Deep Taylor Decomposition
FC	Fully Connected
GAP	Global Average Pooling
GDPR	General Data Protection Regulation
GPU	Graphics Processing Unit
Grad-CAM	Gradient-weighted Class Activation Mapping
LIME	Local Interpretable Model-agnostic Explanations
LRP	Layer-wise Relevance Propagation
MNIST	Modified National Institute of Standards and Technology database
MSE	Mean Squared Error
PASCAL VOC	Pattern Analysis, Statistical Modelling and Computational Learning Visual Object Classes
RDE	Rate-Distortion Explanation
ReLU	Rectified Linear Unit
RMSE	Root Mean Square Error
SENN	Self-Explaning Neural Network
SGD	Stochastic Gradient Descent
SVM	Support-vector Machines
VGG	Visual Geometry Group
XAI	Explanation Methods
YOLO	You Only Look Once

Chapter 1

Introduction

In this thesis we look at deep neural networks for image recognition and explanation methods developed for them. Specifically we look at Convolutional Neural Networks (CNN) [1] as a tool to recognize objects in images and different explanation methods for these models such as LRP (Layer-wise Relevance Propagation) [2] and LIME (Local Interpretable Model-agnostic Explanations) [3].

The deep learning and in particular CNN, have made major impact for image recognition and outperformed traditional classification methods. In recent years, for selected applications CNNs have outperformed human classification ability.

A major issue with deep learning systems is that they act as a black box meaning that we do not know why they predict as they do. Even though the model produces high softmax score for a classification, it is not given that the actual labeled object is detected or even present. In [3] they showed an example of this: The model predicted a wolf, but by looking at the explanation map it was established that the model was only looking at the snowy surrounding and not the actual wolf. This shows that it is important to make these black boxes more interpretable. The image and explanation from [3] is shown in Figure 1.1.

(a) Husky classified as wolf (b) Explanation

Figure 1.1: An example of a prediction of an object that is not present in the image. The explanation indicates bias in the models trained dataset. Left: the input image. Right: the corresponding explanation. Image from [3].

CNN architectures are commonly used for image analysis. In a regular Deep Neural Network (DNN) all weights in layer l is connected to the weights in layers $l-1$ and $l+1$. This results in a huge amount of parameters. A regular Deep Neural Network (DNN) requires to learn all weights in the network, while CNN learn weights locally and reduces the amount of parameters. A CNN model reduces the amount of neurons by taking advantage of the image 3D size. Unlike a regular deep neural network each neuron is not connected to all the neurons in the layer before.

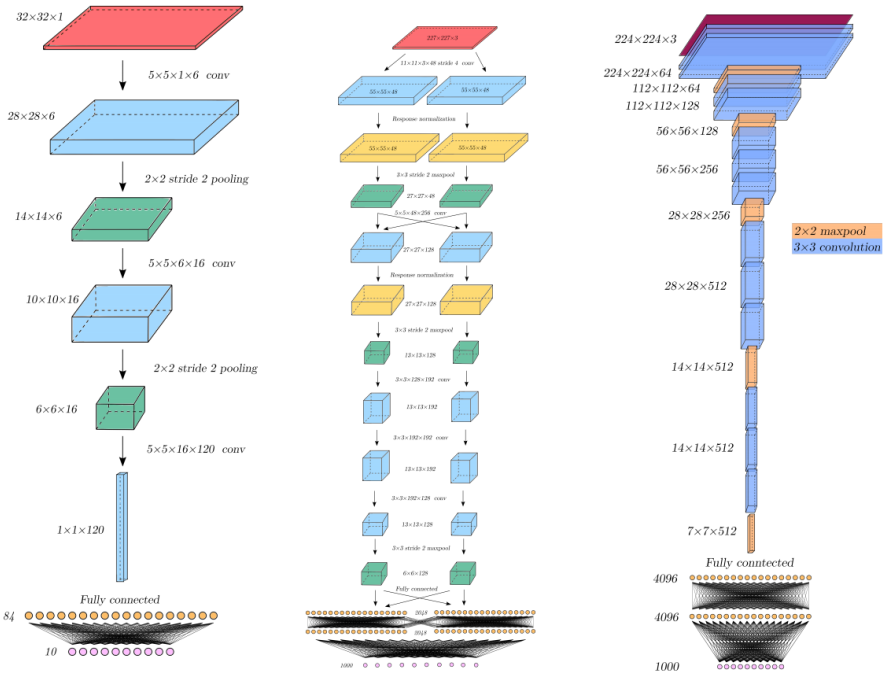

Figure 1.2: Architectures of three CNN models. Left: LeNet-5. Middle: AlexNet. Right: VGG. Figures from [4].

Figure 1.2[1] shows the design of three of the most known CNNs. In 1989 LeCun et al. [1] developed the first convolutional neural network (CNN), LeNet-5. In 1998 LeCun et al [5] showed that the CNN model trained on a dataset (MNIST) outperformed all other recognition systems for similar data. In 2012 AlexNet was introduced by Krizhevsky et al [6]. While MNIST was a relatively small grayscale dataset with approximately 60000 training images of size 28×28, the dataset (ImageNet) that was used to train AlexNet contains over 14 million annotated and labeled color images of size 256×256. The model was based on LeNet with more convolutional layers and the image sizes and the layers required even more computational data. AlexNet used two GPUs to parallellize the training. In 2014 VGG was introduced. In 1.2 the three architectures of the models are shown. The reason for their popularity was the outperformance of other computational systems and later also the human perceptron. These models have inspired the design of even more complex architectures such as DenseNet [7] and ResNet [8]. As the access to computational power increased rapidly, the design of the models could afford to be more intricate demanding previously impossible amounts of computational resources.

[1] https://www.uio.no/studier/emner/matnat/ifi/IN5400/v19/material/week7/in5400_2019_lecture6_training.pdf

1.1 Thesis goal

For machine learning to be truly useful we have to know why the system interprets the way it does. By understanding this we can discover errors both in datasets and how much context plays a role in classification. In this thesis we have mainly focused on the explanation methods used in conjunction with CNNs.

A common approach for investigating the interpretability in a CNN is to study each pixels contribution to the decision. These methods include e.g. Guided Backpropagation [9], class activation maps [10] and LRP [2]. The reliability for a CNN can also be studied by looking at changes to the input pixels. The occlusion experiment [11] is an example of this and how this affects the explanation.

A negative side to these approaches is that they only investigate one image at a time and therefore not giving a global explanation that can be used for similar images/predictions.

In this thesis we investigate different interpretability methods for evaluating predictions from CNN models. We look at research on several explanation methods with a focus on LIME and LRP. Our goal is to investigate different interpretability methods and how robust they are in comparison to each other.

1.2 Thesis structure

The thesis is structured:

- **Chapter 2:** ***Convolutional Neural Networks (CNN)*** In this chapter we give a short introduction to convolutional networks and how they are used and trained.
- **Chapter 3:** ***Relevant literature review*** In this chapter we take a look at different interpretability methods and how they are structured. This is represented by deconvnet, Guided Backpropagation, CAM/Grad-CAM, LIME and LRP.
- **Chapter 4:** ***Applying different models with LRP*** In this chapter we investigate LRP and how to integrate it with different models. We also present some of the issues and decisions of the model used in the later chapters.
- **Chapter 5:** ***Initial investigation of selected methods from the literature*** In this chapter we use a few of the interpretability methods described in chapter 3 and a pretrained VGG16 model.
- **Chapter 6:** ***Robustness of explanation methods - literature review*** In this chapter we review later research about the area: Interpretability methods and how to measure their robustness and how to make them more robust.
- **Chapter 7:** ***Experiments on measuring robustness of LRP and LIME*** In this chapter we introduce our experiments and the methods we use.

- **Chapter 8:** ***Results*** In this chapter we present the results from the experiments in chapter 7.
- **Chapter 9:** ***Model Sensitivity*** In this chapter we look at different models sensitivity regarding LRP.
- **Chapter 10:** ***Further work*** In this chapter we look at potential further work based on our experiments.
- **Chapter 11:** ***Conclusion*** In this chapter we conclude our thesis by summarizing our learnings from the previous chapters.
- **Source code:** Some of the code used in this thesis can be found at GitHub.[2]

[2]https://github.com/SignusRobotics/master-thesis

Chapter 2

Convolutional Neural Networks (CNN)

In this chapter we take a look at Convolutional Neural Networks, how they are structured and used.

Convolutional layers, pooling layers and the fully connected layers are the main blocks of a CNN model. Each building block have different functionality constrained by given rules such as hyperparameters and activation functions.

The convolutional layer is dependent of four hyperparameters. These are the number of filters (K), filter size (F), the stride of the convolution steps (S) and the padding of the input (P). The dimensions of the output, and therefore the input to the next layer, are given by the input dimensions and these hyperparameters:

$$\begin{aligned} kw_{(l+1)} &= \frac{(kw_l - F + 2P)}{S} + 1, \\ kh_{(l+1)} &= \frac{(kh_l - F + 2P)}{S} + 1, \\ d_{(l+1)} &= K \end{aligned}$$

where kw_l, kh_l and d_l is the width, height and depth of the input to layer l and likewise for the output of layer l. In addition the image has to be reshaped to the dimensions of the input layer of the network. For LeNet this is 28×28 and for VGG16 256×256. The pooling layer is only dependent on the hyperparameters filter size and the stride. The most common type is the Maxpool where the maximum number in the kernel is preserved and therefore reduces the output volume and thus less neurons. The main function of this layer is to reduce the amount of neurons surpassed through the next layer and control overfitting. The most common type is a kernel of 2×2 and stride of 2. A kernel that is greater will give too much information loss. It can be discussed on how necessary pooling layers are.

ReLU [12] is a type of activation function. Here only the positive parameters from the the layer before is kept and all negative parameters are set

to 0. These functions are used to prevent the vanishing/exploding gradients problem [13].

The last units of a CNN model often consist of fully connected layers (FC) and a softmax layer. The FC layers can be seen as a regular neural network where all neurons in one layer are connected to all neurons in the previous layer. It is usually more than one FC layer in a model, and the first can be seen as a $1 \times 1 \times$ N where N is the amount of neurons in the previous layer. The last FC layer is converted to the number of classes used to train the model.

The model is then trained by feeding a dataset of images to the model. For a classification problem, the images are labeled and might also be annotated with a bounding box of the labeled object. ImageNet[1], MNIST[2], CIFAR[3], PASCAL VOC[4] and COCO[5] are examples of such datasets. The model is designed and bounded by the image dimensions of the dataset used to train the model. Each dataset consist of a unique test set to prevent false positive accuracy.

The earliest CNNs consisted of convolutional layers, pooling layers and fully connected layers ending with a softmax layer. Later it was improved by other layers such as Droput [14] and Batchnorm [15] layers. Recently newer forms of layers have been designed, such as convolutions with dilations. Dropout is used as a regularization technique to prevent overfitting but also reduces computational time. The main idea is to randomly drop weights and connections in the network while training [14].

Normalization is a regularizer method that prevents weights in the network to explode. As an added benefit this makes the training process faster. Batch Normalization between layers was introduced in 2015 [15]. The Batchnorm layers main function is normalizing the layer inputs to prevent internal covariate shift. This is done for each mini-batch in the training process. These layers are used as a regularizer but also gives a flexibility to the learning process. Some of the benefits of adding these layers to the architecture is to use higher learning rates that results in less training time and more flexible initialization. The Batchnorm layers can in some cases be used instead of Dropout [15]. It is possible to fine-tune an already trained model for a similar case by using some of the model architecture and train the modified new model with the weights from the old with a new dataset. This is appropriate when the required dataset is small and the pretrained models dataset is similar to the new dataset.

2.1 Training

Training of a CNN model and propagating the gradients of the loss are done by sending appropriate data through the network, using backpropagation. A dataset is required to train a network. This is usually divided into three parts: training, validation and test. The training and validation sets are

[1]https://image-net.org/download.php
[2]http://yann.lecun.com/exdb/mnist/
[3]https://www.cs.toronto.edu/ kriz/cifar.html
[4]http://host.robots.ox.ac.uk/pascal/VOC/
[5]https://cocodataset.org/home

used for the actual training, while the test set is used to evaluate the model. In early stopping the validation sets loss is checked against the training sets loss.

The training process consists of a forward pass and a backward pass. In the forward pass the input data is passed through the network. The main purpose of the the backward pass is to update the weights in the network by backpropagating the gradients. The softmax function is as follows:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.1}$$

where $\sigma(\vec{z})_i$ is the softmax vector and z_i is the vector before softmax. K is the number of classes and i and j represents which specific class number between 0-K to look at. The softmax helps to distribute the networks class scores to a valid probability distribution.

In the softmax layer the prediction is calculated. This dense layer is designed with number of nodes equal to the number of classes from the dataset the network is trained on. The activation function in this layer is normally the softmax function. The purpose is to get a statistical score from the models prediction. Further the Cross-Entropy Loss gives the prediction score for each class.

$$L(p, q) = -\sum p(x) log(q(x)) \tag{2.2}$$

where L represents the cross entropy loss for each true (p) and predicted (q) value in the distribution defined by x. For the backward phase the gradients for a class is dependent and calculated only for that specific class.

The focus of training a network is to minimize the loss. An optimizer is added to the network to minimize the gradient of the loss function with respect to the weights of the models output. The loss is typically calculated by looking at the true label and the predicted label of the class prediction. Mean Square Error (MSE) is a regular loss function to this case. The loss is minimized by using backpropagation and an optimization algorithm such as ADAM [16] or Stochastic Gradient Descent (SGD) [17]. SGD with momentum [18] is an improvement of SGD. By adding a momentum to the previous gradient function this will results in a faster convergence/training. ADAM gives additionally a more stable and an improvement over previous methods such as SGD. The learning rate for the ADAM optimizer is decided after each iteration. This results in a faster and more flexible training process. AdaGrad [19] and RMSprop [20] are two other commonly used optimizers.

The optimization function updates the weights and is controlled by the learning rate. Ideally the training is stopped before overfitting. To prevent overfitting a regularization method such as L1, L2 or Dropout can be used. The regularization method penalize large image gradients. In addition a technique called early stopping is often used. Early stopping checks the validation loss versus the training loss and stops the training when the difference of the two losses reaches a break point. Data augmentation of the training dataset is an initial preparation to prevent overfitting. This is done

by generating more training data using one or combinations of transformations which can give a more robust model that is more flexible for similar images.

Chapter 3

Relevant literature review

In this chapter we look deeper into some of the existing interpretability methods. The interpretability methods reviewed here are: Deconvolutional Network [21], Guided Backpropagation [9], CAM [10], Grad-CAM [22], LIME [3] and LRP [2].

Interpretability methods are important for understanding the output of CNN models. It can detect anomalies of the architecture and catch bias from the dataset. As the models have gotten more complex the interpretability methods are increasingly becoming more important.

To understand CNN better researchers started to inspect the different layers by visualizing the outputs of them. Over the past years it has been developed several methods to visualize different aspects of CNN's. It was discovered in [9] that the first layers of the model usually detects simple patterns such as edges and Gabor filters. More complex patterns are typically identified in the later layers of the model. An example of this is an image of a face through a suited model. Here the first layers of the model will reveal that it typically detects patterns less complex than layers nearer the FC layers. For example will eyes and the nose be detected earlier in the layers versus the whole face [11].

As more research and approaches have been tested new better methods have been discovered. The newest approach is to make the explanation method as part of training the model and give a more robust method of how too understand and visualize the output of the model.

Originally the challenge with CNNs was a lack of computational power, but now this is mitigated by powerful desktop GPUs and custom processing units. Now one of the main challenges is to actually understand why CNNs perform as they do and how to improve them using a scientific approach.

In this chapter we review some existing approaches for understanding the performance of CNNs.

3.1 Deconvolutional Network, deconvnet [21] and Guided Backpropagation [9]

One of the first papers on understanding CNNs was introduced by Zeiler and Fergus [21]. Until then, the main process to understand, improve and

look into the network was trial and error, which is not sustainable. There were also a limited number of methods for looking into the activations inside the model and interpreting them. A natural approximation to this could be using a Hessian matrix, but for the deeper layers the Hessian cannot be easily computed. According to Zeiler and Fergus [11], "[..]visualizations differ in that they are not just crops of input images, but rather top-down projections that reveal structures within each patch that stimulate a particular feature map."

Zeiler and Fergus presented two approaches, one based on deconvnet [21], and one based on occlusion maps [11].

The principle used in Deconvnet is fairly similar to backpropagation, the difference between backpropagation and the deconvnet approach is how the ReLU function is performed. In backpropagation the negative gradients flowing backwards through a ReLU function are removed. In Deconvnet the ReLU is applied to the error signal, and therefore only the positive error signals is backpropagated different from backpropagating. According to [11] "A deconvnet can be thought of as a convnet model that uses the same components (filtering, pooling) but in reverse, so instead of mapping pixels to features does the opposite. In (Zeiler et al., 2011) [21], deconvnets were proposed as a way of performing unsupervised learning."

They also showed the correspondence between image structure and feature map activities by occlusion of images [11]. This was done by occluding portions of the input image and then performing a sensitivity analysis of the classifier output. It was discovered that not only the probability of the class dropped when the object or part of the object for the class was occluded, but also the activity in the feature map.

It was also established in [11] that the deeper layers in the model gives more complex structures and that these have to be trained for more epochs to converge. Small changes in scaling and translation gave dramatic changes in the shallow layers while deeper in the model it had lesser impact. They also showed that the model they used had problems with input rotation.

In 2015 a new approach for visualization of filters was proposed [9], guided backpropagation. According to [9] networks without max-pooling layers could be challenging to visualize using deconvnet [21]. By combining backpropagation and deconvnet, Guided Backpropagation [9] was created. In this approach all the negative gradients from backpropagation and deconvnet are set to zero. This gave better results, particularly for visualization of the deeper layers with less artefacts and it highlights pixels that are important for the classification of class c.

3.2 CAM/Grad-CAM

3.2.1 CAM [10]

Around 2016 a visualization method for CNNs called Class Activation Mapping (CAM) was proposed by Hansen et al. [10]. This method is able to visualize why the input image is classified as a given class c, by highlighting which regions that have contributed to the classification. Based on the last convolution layer in the model and the pre-softmax score for a given class,

the importance of the different regions in the input image is computed. By using Global Average Pooling (GAP) the localization ability of the network is maintained.

The result after performing GAP on each filter in given convolution layer, $f_k(x, y)$, is then:

$$F^k = \sum_{x,y} f_k(x, y), \tag{3.1}$$

where k is the filter and F^k represent the activation of unit k.

Then for each predicted class the new input after GAP to the softmax layer, S_c is:

$$S_c = \sum_k w_k^c F_k, \tag{3.2}$$

where w_k^c is the weights from the original forward pass in the layer before softmax and k is filter number and c the specific class. w_k^c indicates how important the corresponding F_k is for a given class. To find the softmax the new values S_c is used.

M_c defines the class activation map for a given class, c:

$$M_c(x, y) = \sum_k w_k^c f_k(x, y). \tag{3.3}$$

This is upsampled to the original size of the input image, and will show the most important regions in the image that gives the classification for a given class c.

3.2.2 Grad-CAM [22]

In 2019 a generalization of CAM called Grad-CAM, [22] was found. This method is more flexible and it is possible to look at any layer with no retraining requirement. This is done in one operation after a given class prediction score is computed by partially backpropagating to the given convolutional layer. This was a great improvement from other methods such as CAM and occlusion method.

The class-discriminative localization map for class c with respect to a convolutional layer of size $u \times v$ is defined as $L^c_{Grad-CAM} \in R^{u \times v}$. This is computed with respect to the given feature map activations A_k and the classification score, y_c, for class c, and backpropagated to the selected convolutional layer. The neuron importance weights, α, is computed by doing a GAP on the gradients on equivalent to the position of the convolutional layer:

$$\alpha_k^c = \frac{1}{ij} \sum_i \sum_j f_k(i, j) \frac{\partial y^c}{\partial A_{ij}^k}, \tag{3.4}$$

where Aij^k represent the activations of the convolutional layer and y^c the output prediction score for the class c. Like for CAM the weight α_k^c represents the importance of each feature map for a given class c.

Before the class-discriminative localization map, $L^c_{Grad-CAM}$ is obtained, each combination of weights α_k^c and feature maps with respect to the given convolutional layer is sent through a ReLU. This results in a heatmap corresponding to the size of the feature maps at this layer. The ReLU function is added to highlight only the positive neurons that give an increased prediction score for class c and therefore also increase the performance of the localization maps.

$$L^c_{Grad-CAM} = ReLU\left(\sum_k \alpha_k^c A^k\right). \tag{3.5}$$

This method gives the opportunity to also only highlight the negative regions which prevents classification to the specific prediction score for the given class. Instead of looking at the positive gradient one only looks at the negative while following the other rules for Grad-CAM.

3.2.3 Guided Grad-CAM [22]

Guided Grad-CAM [22] is a combination of Grad-CAM and guided backpropagation [9]. By combining these two methods it is possible to get more fine-grained details like pixel space gradient visualization methods. The class-discriminative localization maps are upsampled to the input image resolution using bilinear interpolation and then performed elementwise multiplication with the Guided Backpropagation map. Combining these methods will remove all pixel information that gives negative information about the class, and only highlights the fine-grained information about the prediction score for class c.

3.3 Local Interpretable Model-agnostic Explanations (LIME) [3]

In 2016 LIME was introduced in [3]. LIME is according to the article a method "that explains the predictions of any classifier in an interpretable and faithful manner, by learning an interpretable model locally around the prediction". LIME is a flexible method that can handle different models such as random forests and neural networks and therefore different datasets containing different data types such as text and images. The main focus of this method is the aspect of trusting a model and trusting a prediction.

There are two aspects of this method one is LIME: "an algorithm that can explain the predictions of any classifier or regressor in a faithful way, by approximating it locally with an interpretable model.". The other is SP-LIME: "that selects a set of representative instances with explanations to address the "trusting the model" problem, via submodular optimization.".

The algorithm for LIME is as following:

$$\xi(x) = \underset{g \in G}{argmin} \quad L(f, g, \pi_x) + \Omega(g) \tag{3.6}$$

where $f(x)$ is the probability of the prediction for a given class. $\pi_x(z)$ measures how close an instance z is to x and $L(f, g, \pi_x)$ measures how unfaithful g is in approximating f in the locality defined by $\pi_x(z)$. $\Omega(g)$ states the complexity of the explanation $g \in G$.

The output gives for an example patches of image with contributions/non contributions to why the image/patch is classified as it is. From the article it is a dog with a guitar and it is clearly logical why the image is classified as the different labels. This is also applicable for text datasets with random forests where the words is highlighted for or against.

The article addresses different problems from a human perspective when to trust a model and why. They argue for that when the user know why the model have predicted as it did it is easier for the worker, such as medical personnel or engineer, that not necessary have machine learning background to understand and rely on the prediction. When the interpretability method makes the prediction logical it is easier to discard or use it further. This is especially important for systems that handles life-critical situations, such as medical diagnosis.

The method in itself is not a straight forward implementation. It is built on different libraries and the execution is time consuming. The output in itself is based on number of iterations, that can give different outputs, and higher the iterations the more time is used to run the method. The output of this method gives both positive and negative areas of contributions.

3.4 Layer-wise Relevance Propagation (LRP) [2]

Layer-wise relevance propagation [2] is a visualization method introduced in 2015. This method is currently among the most popular methods used for explaining CNN models.

LRP could be implemented with several different classifiers. In the paper that introduced this method they showed how to use it on two different classifier architectures, Bag of Words [23] features with non-linear Support-vector Machines (SVM) [24] and neural networks.

The main problem is to figure out how and why the model classifies as it classifies. LRP is an explanation method that makes an interpretation of the prediction to help understand this. In many fields it is very important to understand the model. An example of this could be the medical field, where the classification gives a result with a decision that is essential for a patients life and death.

Another important result with LRP is that the result would give an explanation on which pixels that gives the highest importance for the class. Thus, the pixels with the highest score would also indicate what the model finds most important. This could reveal/expose anomalies in the model that is used, such as bias and errors in the dataset that the model is trained on. Therefore it would give an indication on how to improve the dataset.

An example of this type of bias is to consider two classes young woman and old woman where smile gives high relevance score for the class young woman, while for old woman it gives indication against. Another similar example is doctor vs nurse where woman/long hair contraindicate doctor classification.

Unlike other explanation methods such as sensitivity analysis, LRP locates which pixel that contributed most to the given predicted class. For each of the classes in the model a LRP score is calculated for each pixel of the image being tested. This is a result of the LRP method implementation having to pass several unit-tests such as continuity, conservation, selectivity and positivity. The method can be implemented to different classifiers such as SVM and neural networks.

Thus, if the pixels with highest score is removed, then you will see that the prediction score, $f(x)$ is lower than it was before. The score will decrease proportionally with how much of the highest LRP score pixels that is removed.

3.4.1 LRP in detail

LRP is an approximation of Deep Taylor Decomposition (DTD) [25] when the function is highly non-linear. A problem that Layer-wise Relevance propagation solves is the necessity of a root point in calculating the DTD [25]. LRP can be calculated only with the input points and the classifier score. The Layer-wise Relevance propagation score, R, of the current layer (l), is calculated from the output layer $(l+1)$. Therefore it is easy to back-propagate the LRP score from the prediction, $f(x)$, back to the input pixels, x_d for a given classifier f and an image x.

$$f(x) \approx \sum_{d=1}^{V} R_d, \tag{3.7}$$

where V is the number of pixels in the image.

The input pixels, x_d, are sent from the input layer through the network to the last layer that ends in the real-valued softmax score of the classifier f. The l-th layer is modeled as a vector:

$$z = (z_d^l)_{d=1}^{V(l)}, \tag{3.8}$$

where z is the vector of the relevance score for each node in layer l and $V(l)$ is the dimensionality.

Assuming that layer $l+1$ has a relevance score, $R(l+1)$, then it is possible to find the relevance score of the previous layers, $R(l)$. This is repeated until reaching the input layer to find the relevance score for each pixel, $R(1)$. Then the conservation rule gives:

$$f(x) = ... = \sum_{d \in l+1} R_d^{(l+1)} = \sum_{d \in l} R_d^{(l)} = ... = \sum_{d} R_d^{(1)}. \tag{3.9}$$

$R_d < 0$ means that the respective pixel, x_d, does not contribute to the class, and $R_d > 0$ means that the pixel has a relevance for the class.

The procedure for a neural network starts the same, the first message is the classifier score for the analyzing class. The message, $R_{i\leftarrow k}^{(l,l+1)}$, for layer, l, is calculated from the neurons connected to the layer, $l+1$. From Figure 3.1[1] you can see a visualization of this.

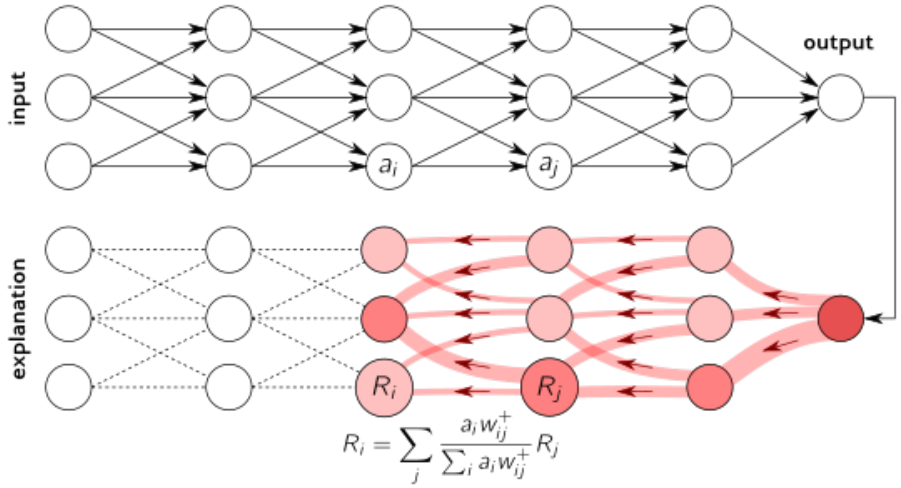

Figure 3.1: This Figure shows how the relevance score is calculated from the output layer $l+1$, from the current layer, l. Here you can see the backward pass [26].

The main formulas to calculate the message contribution for each layer are:

$$R_i^{(l)} = \sum_{\text{k: i is input for neuron k}} R_{i\leftarrow k}^{(l,l+1)}, \tag{3.10}$$

$$R_i^{(l+1)} = \sum_{\text{i: i is input for neuron k}} R_{i\leftarrow k}^{(l,l+1)}, \tag{3.11}$$

$$R_{i\leftarrow k}^{(l,l+1)} = R_k^{(l+1)} \frac{a_i w_{ik}}{\sum_h a_h w_{hk}}. \tag{3.12}$$

This last equation is another way to look at computing the relevance for layer l from layer $l+1$. The messages for each layer are calculated based on the next layer and starts with the softmax layer (R_output) and is calculated back to the pixel level.

In a neural network it is possible to find every neurons relevance score, R_j^{l+1}. For each node in current layer, i, the contribution of the relevance score can be computed from the previous layers connected nodes, j, $R_{i\leftarrow j}$. These messages is calculated with the following equation:

[1] heatmapping.org

$$R_{i\leftarrow j}^{(l,l+1)} = \frac{z_{ij}}{z_j} \cdot R_j^{(l+1)}, \tag{3.13}$$

where z_{ij} is the message from the previous layers node j connected to node i in the current layer and z_j is the sum of all the preactivation scores from the node j. $R_{i\leftarrow j}^{(l,l+1)}$ is each neurons messages from layer $l+1$ to layer l.

This can be confirmed by the conservation rule. The relevance score for each node j in the previous layer, $l+1$, can be calculated back again from the i nodes in the current layer, pointing to node j.

$$\sum_i R_{i\leftarrow j}^{(l,l+1)} = R_j^{(l+1)} \cdot \left(1 - \frac{b_j}{z_j}\right). \tag{3.14}$$

To implement LRP for a network it has to fulfill the constraints that LRP requires. A problem with the straightforward implementation is that it might give a problem with unbounded values. This happens when z_j is small. This can be solved by using a predefined stabilizer, $\epsilon \geq 0$.

$$\begin{cases} z_j = z_j + \epsilon, \text{if } z_j > 0 \\ z_j = z_j - \epsilon, \text{if } z_j < 0. \end{cases} \tag{3.15}$$

Another way to solve this is to separate positive and negative preactivations. This approach will avoid relevance leakage and allows for control of the importance of negative and positive evidence. This is defined by $\alpha + \beta = 1$, and the equation 3.16.

$$R_{i\leftarrow j}^{(l,l+1)} = R_j^{(l+1)} \cdot \left(\alpha \cdot \frac{z_{ij}^+}{z_j^+} + \beta \cdot \frac{z_{ij}^-}{z_j^-}\right). \tag{3.16}$$

3.4.2 LRP-rules

Since the introduction of LRP in the article [2] new rules for the different states and conditions have been established for LRP. In [27] different rules for the different layers in the network were proposed. These rules were:
Basic rule, LRP-0:

$$R_j = \sum_k R_k \frac{a_j w_{jk}}{\sum_{0,j} a_j w_{jk}}. \tag{3.17}$$

Epsilon rule, LRP-ϵ

$$R_j = \sum_k R_k \frac{a_j w_{jk}}{\epsilon + \sum_{0,j} a_j w_{jk}}. \tag{3.18}$$

Gamma rule, LRP-γ

$$R_j = \sum_k R_k \frac{a_j \cdot (w_{jk} + \gamma w_{jk}^+)}{\sum_{0,j} a_j \cdot (w_{jk} + \gamma w_{jk}^+)}. \tag{3.19}$$

The different rules have different properties. It is possible to only use one rule for all layers, but the best result is to use a combination with the suitable rules for the right layers [28]. LRP-0 highlights artefacts and therefore gives a result that is not understandable if it is used for the whole network. This rule is used for the last layers in the network since it contains a small amount of neurons, 4096 for VGG-16. The LRP-ϵ rule is used for the middle layers and removes noise and results in keeping only the most relevant pixels for the predicted object, but it gives a sparse result and this rule can not be used alone. For the first layers one would like to highlight all the features, relevant or not, before it is sent through LRP-ϵ rule. LRP-γ does this. For the first layers it can also be used the α and β ruled mentioned in the section before. By combining these three rules the LRP heatmap gives a faithfully and understandable output.

A special case is when the data, such as pixel values are sent to the model. Here it is appropriate to use the z^B rule from the DTD framework.

Since a large variation of models exist it is important to evaluate the model before applying the different rules to the layers of the model.

For models with batchnorm layers it was primarily recommended to use the identity rule or the ϵ rule. This have later been shown to not give good results [28].

Another approach is to merge the convolutional layer before the batchnorm layer together and form a new convolutional layer [29, 30, 31] This new architecture is then used when the LRP relevance is backpropagated to the pixel level. One constraint of this method is that the model used have to retain the weights and biases when the model is trained. If this is not the case another solution is to remove the batchnorm from the architecture and use this architecture when the LRP relevance is calculated.

In [28] the problem with batchnorm layers in the model was addressed and new LRP rules for batchnorm layers were presented. Here they stated that the bias had more to say than first assumed. They also showed that the first proposed rules, ϵ rule, $\alpha\beta$ rule and the identity rule were not suited for these layers. This is especially observed for the MobileNetv2 [32] architecture where both ϵ rule and $\alpha\beta$ rule shows little or no information on the LRP heatmaps. This is due to the bias conservation and shows that the biases of the network gives a higher impact than assumed. The identity rule ignores bias and therefore may give a poorer LRP heatmap result. This is shown for the DenseNet121 model [7].

They presented a new rule used for the Batchnorm layers, $|z|$ rule. This rule considers the fact that bias in the batchnorm layers should not be ignored like the Identity rule does. It is also critical that the rule handles both positive and negative contributions, that is not the case for the $\alpha\beta$ rule shown for the MobileNetv2. In addition the rule avoids bias cancellations unlike the ϵ rule.

3.4.3 LRP implementation

In this thesis we base or work on the tutorial from the LRP website, heatmapping.org and the original rules described first in this section. Initial testing and implementations showed major flaws when use of models with batchnorm. We therefore avoid use of batchnorm layers.

Chapter 4

Applying different models with LRP

In this chapter we present some of the challenges encountered during the beginning of this thesis. This had an impact on the rest of the project. We look at the model used for the rest of the thesis, the calibration of the LRP rules and a discussion of models with batchnorm.

The main focus in this chapter is to investigate LRP on different models and get reasonable results. It was tested with pretrained networks from PyTorch and models trained on different datasets. The LRP implementation was based on the tutorial on the LRP website[1] and the initial rules referred to in section 3.4.2.

The implementation of the LRP heatmaps are based on utils.py of the demo from the tutorial on the website with minor changes such as adding implementation for showing the four first predicted heatmaps. The color scheme used is the library ListedColormap from matplotlib.

When testing more complicated networks such as ResNet [8] and DenseNet [7] the block based architecture gave problems with the basic LRP implementation. The LRP implementation would have to be deconstructed further to work with the demo as is and this was not investigated further here. Another problem were models with batchnorm layers. The original suggested rules for batchnorm layers referred to in the LRP rule section did not give acceptable results. It was observed that the LRP heatmaps gave similar results as discussed in [28]. This is shown in Figures 4.1 and 4.2. All the testing in this chapter was done before the suggested rules for batchnorm was presented [28].

[1] heatmapping.org

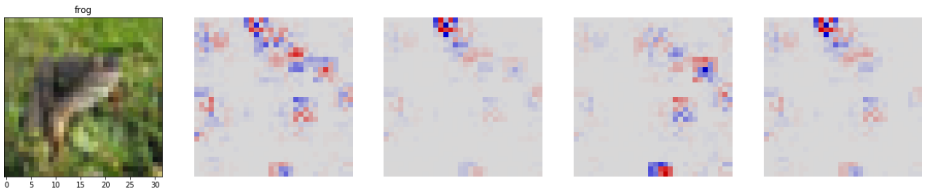

Figure 4.1: Trained model with architecture from [33] saved BN parameters. LRP results for the first 4 predicted classes. Not absorbed CNN layers with corresponding BN layers. Heatmaps from pixel layers.

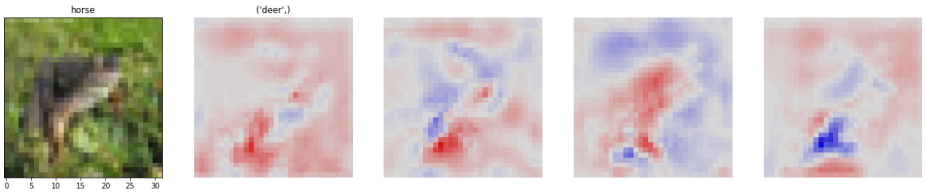

Figure 4.2: Trained model with architecture from [33] saved BN parameters. LRP results for the first 4 predicted classes: deer, cat, frog, bird. Absorbing CNN layers with corresponding BN layers to new layers. Heatmaps from pixel layers.

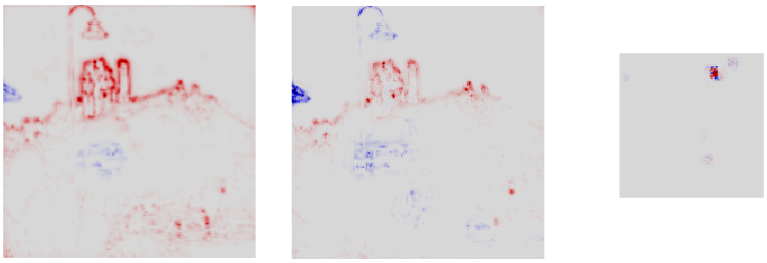

Figure 4.3: Pretrained VGG16 from pytorch LRP results. Left: VGG16 without batchnorm. Middle: VGG16 with batchnorm layers absorbed to new CNN layers. Right: VGG16 with batchnorm layers used without absorbing of the layers. LRP output from layer 11.

Since complex architectures such as ResNet [8] and DenseNet [7] gave problems we decided to use a VGG based architecture initially to get a functional implementation of LRP as a baseline. The pretrained VGG with 16 layers, VGG16, from PyTorch [34] with and without batchnorm layers was tested. Here it was discovered that the batchnorm layers gave problems with the LRP output. When testing with the initial suggested rules the heatmaps looked compressed and clearly not matching the predicted results.

The implementation was also tested with the pretrained model for AlexNet [6] from PyTorch. Both VGG16 and AlexNet models are trained on ImageNet.

For the batchnorm issues we use a method that merged the convolutional layer and the corresponding batchnorm layer to a new convolution layer, this method was found from [35]. A demonstration of the results is found in Figure 4.3. For this specific image where the predicted label is castle it is observed that the model with batchnorm layers gives better LRP results than the model without. This is also observed from the prediction scores where the model with batchnorm layers give higher prediction score for the class castle.

The LRP was also tested for models trained on CIFAR10 and CIFAR100. Here we used pretrained models from ModelZoo [33, 36]. These models contained batchnorm layers, but the parameters for bias was not preserved. To use the fuse function bias parameters have to be preserved. This particular problem was not a problem for the pretrained models from PyTorch since it is a choice in the function call. Our first attempt was to train models not containing batchnorm layers, but this gave inaccurate results. The quick fix was to train a model based on the models from ModelZoo with a minor fix to preserve the bias parameters. By doing this the fuse function performed as expected. A problem with the CIFAR datasets is that it only contain one label on each image.

4.1 Architecture of VGG models

The architectures of the networks used in this thesis are shown in Appendix A.1 and A.2.

4.2 Calibration of the LRP-rules for the specific model

The parameters for the LRP rules for each model were tuned by following the recommendations from [27]. In Figure 4.4 it is shown output for using different LRP rules alone and in combinations. The description of each test is listed in Table 4.1. The idea was to calibrate other models by the same process.

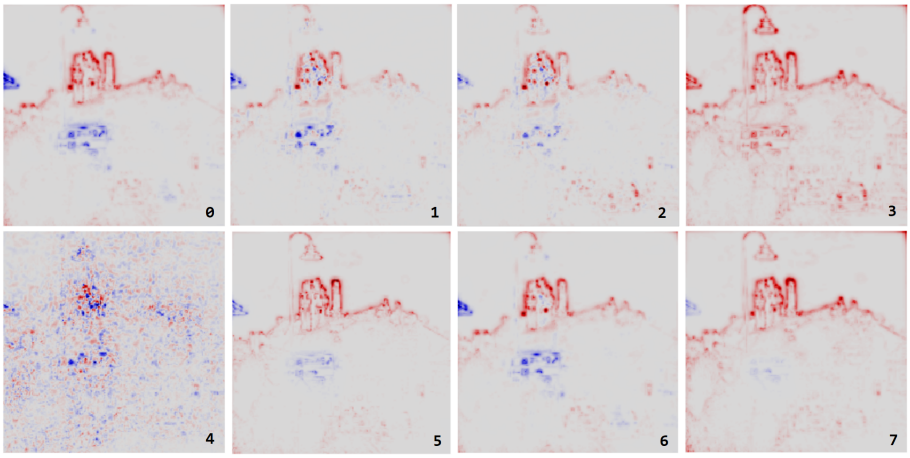

Figure 4.4: Testing of different LRP rules alone and together. The model used is a pretrained VGG16 network from PyTorch

Initial:	Original LRP parameters from demo.
Test 1:	LRP-ϵ for all layers in the features part and LRP-0 for the layers in the classifier part.
Test 2:	Only using LRP-ϵ for the whole network.
Test 3:	Only using LRP-γ for the whole network.
Test 4:	Only using LRP-0 rule for the whole network.
Test 5:	LRP-γ: layer 0-25, LRP-ϵ: 26-30 and LRP-0 for the classifier.
Test 6:	LRP-γ: layer 0-10, LRP-ϵ: 11-30 and LRP-0 for the classifier.
Test 7:	LRP-γ for all layers in the features part and LRP-0 for the layers in the classifier part.

Table 4.1: List of tests for different LRP rules on a VGG16 model.

All tests gave expected results [27]. Tests 1, 2 and 6 gives quite similar results due to the use of the LRP-ϵ rule. Test 5 and 6 gives the best results as they are using all three rules. Test 3 and 7 shows that the LRP-0 rule on the classifier part is important for the detection of negative pixels. It is also observed that the LRP-γ rule registrates important pixels as pixels and do not distinguish between positive or negative importance. The three rules are important for a well operating LRP method. By following this rules and tests it was easier to decide where to set the thresholds between the layers in the other networks. For example for the pretrained VGG16 with batchnorm the thresholds where tuned as for the VGG16 without batchnorm layers. The pretrained VGG16 without batchnorm layers is set to: LRP-γ: layer 0-16, LRP-ϵ: 17-30 and LRP-0 for the classifier. For the VGG16 with batchnorm layers this corresponds to the layer partitioning: LRP-γ: layer 0-23, LRP-ϵ: 24-43 and LRP-0 for the classifier.

Chapter 5

Initial investigation of selected methods from the literature

In this chapter we investigate a few of the explanation methods presented in chapter 3. The methods used are: guided backpropagation, Grad-CAM, LIME and LRP. The first section investigates images with one labeled object and section two repeats the the experiment with images with more than one object. The chapter concludes with the interpretability methods we will look further into in the following chapters.

5.1 One label

In this section we look at the visualization methods introduced in section 2 considering interpretability. By using example images and a pretrained network we decide which visualization method that is most interesting to look further into.

Before these initial test is done some of the methods like deconvolution and CAM are not considered further. Deconvolution alone is shown to give more artefacts and is more noisy than using guided backpropagation [9] [22]. Deconvolution is also incorporated in guided backpropagation. The original CAM is discarded due to the architecture of the method being dependent on retraining with small changes of the model.

The methods analyzed in this section are guided backpropagation, Grad-CAM, LIME and LRP. The experiment used the pretrained Pytorch VGG-16 model [37]. The example images are: a tabby cat, a bee, a zebra and a castle with surrounding landscape. Table 5.1 presents the results of the first four predicted categories of each image.

True label	tabby cat	castle	bee	zebra
1.prediction	tabby: 87.1%	castle: 52.5%	bee: 82.3%	zebra: 99.9%
2. prediction	egyptian cat: 9.0%	church: 11.6%	fly: 17.5%	-
3. prediction	tiger cat: 3.2%	monastery: 10.3%	rapeseed: 0.07%	-
4. prediction	lynx: 0.2%	bell cote: 7.2%	ant: 0.05%	-

Table 5.1: Predicted label and score for each image with pretrained VGG-16 model and Pytorch. Each label name is abbreviated to the first whole name of the ImageNet name.

As seen in Table 5.1 the first prediction of the four images has a much higher score than the next three and the image of the zebra gives high prediction score for category zebra and very low score for the other classes.

The guided backpropagation method highlights fine-grained structures of the interpretation, but is less class-discriminative compared to the other three methods. The output based on the different prediction score from a specific image gives almost the same results, see Figures 5.1, 5.5, 5.9 and 5.13.

Grad-CAM, LIME and LRP gives class-discriminative results. Both LIME and LRP are able to show regions of negative and positive contributions in the same visualization. Unlike LIME, LRP are able to give high-resolution results down to specific pixels. An example of visualization of a negative contribution is the LRP heatmap for the cat image, Figure 5.4, with prediction of a lynx. Here it is shown that the nose and fur gives more counter contribution for the lynx prediction than the three other. This seems logical since the fur of a lynx and a cat is different.

Figure 5.16 shows the visualization method LRP with an image of a zebra. For the first prediction the model is 99% confident that this is a zebra and the corresponding heatmap shows that it is almost no negative parts. For the three other LRP results, the score is too low to actually analyze it. LRP is dependent on a relative high score to give reliable results [2]. For the LIME visualization of the same image, shown in Figure 5.15, it is clearer that the first prediction gives much higher score than the three other visualizations where the three images shows less positive contributions to the respective predictions. Grad-CAM for this set of predictions show that it is small differences between the visualizations. See Figure 5.14.

By itself Guided Backpropagation is not tuned to determine class-discriminative regions, but in combination with a class-discriminative method it can make an improvement, a good example of this is Guided Grad-CAM [22]. Grad-CAM seems to give more noisy output than LRP. It is also beneficial to see which regions that contribute and work against the prediction in the same image. This can among other things discover positive and negative biases and therefore decide how robust the model/dataset is. Overall LRP have a benefit with its high-resolution and class-discriminative output.

Image: Cat[1]

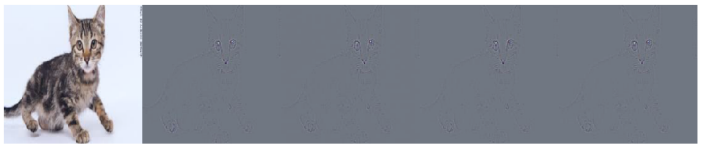

Figure 5.1: Guided Backpropagation results for image of tabby cat with respect to the four first predicted classes. See table 5.1 for specific labelnames

Figure 5.2: Grad-CAM results for image of tabby cat with respect to the four first predicted classes. See table 5.1 for specific labelnames

Figure 5.3: LIME results for image of tabby cat with respect to the four first predicted classes. See table 5.1 for specific labelnames

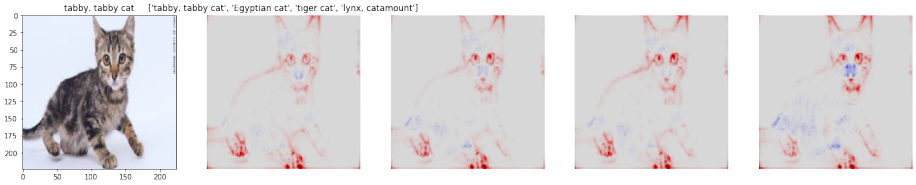

Figure 5.4: LRP results for image of tabby cat with respect to the four first predicted classes. See table 5.1 for specific labelnames

[1]https://www.kaggle.com/c/dogs-vs-cats

Image: Castle[2]

Figure 5.5: Guided Backpropagation results for image of castle with respect to the four first predicted classes. See table 5.1 for specific labelnames

Figure 5.6: Grad-CAM results for image of castle with respect to the four first predicted classes. See table 5.1 for specific labelnames

Figure 5.7: LIME results for image of castle with respect to the four first predicted classes. See table 5.1 for specific labelnames

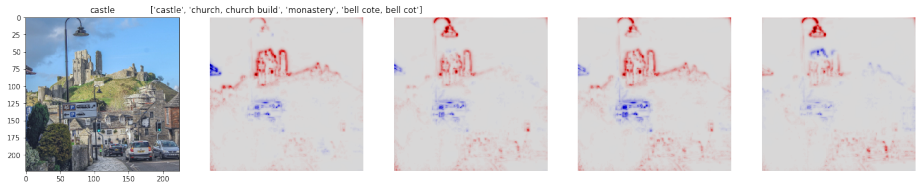

Figure 5.8: LRP results for image of castle with respect to the four first predicted classes. See table 5.1 for specific labelnames

[2]heatmapping.org

Image: Bee[3]

Figure 5.9: Guided Backpropagation results for image of bee with respect to the four first predicted classes. See table 5.1 for specific labelnames

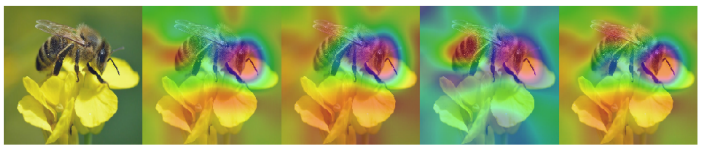

Figure 5.10: Grad-CAM results for image of bee with respect to the four first predicted classes. See table 5.1 for specific labelnames

Figure 5.11: LIME results for image of bee with respect to the four first predicted classes. See table 5.1 for specific labelnames

Figure 5.12: LRP results for image of bee with respect to the four first predicted classes. See table 5.1 for specific labelnames

[3]https://commons.wikimedia.org/wiki/File:Apis_mellifera__Brassica_napus__Valingu.jpg

Image: Zebra

Figure 5.13: Guided Backpropagation results for image of zebra with respect to the four first predicted classes. See table 5.1 for specific labelnames

Figure 5.14: Grad-CAM results for image of zebra with respect to the four first predicted classes. See table 5.1 for specific labelnames

Figure 5.15: LIME results for image of zebra with respect to the four first predicted classes. See table 5.1 for specific labelnames

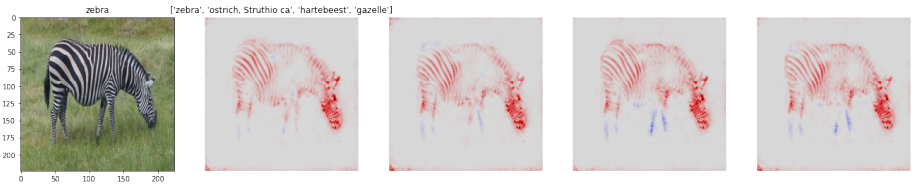

Figure 5.16: LRP results for image of zebra with respect to the four first predicted classes. See table 5.1 for specific labelnames

5.2 Images with more than one label

In this section we look at images with more than one label. The images are obtained from the dataset cats vs dogs [38]. The experiment is the same as described in section 3. Table 5.2 shows the four highest predicted scores for

each image. The images was found by manually testing different images from cats vs dogs dataset.

True label	dog/tub	cat/maraca	cat/hamper
1. prediction	bucket: 55.19%	ping-pong ball: 47.29%	hamper: 44.8%
2. prediction	Norwich terrier: 18.42%	pool table: 43.70%	shopping basket: 23.2%
3. prediction	tub: 11.97%	tennis ball: 5.17%	tabby: 12.3%
4. prediction	chow: 6.75%	maraca: 1.73%	tiger cat: 7.2%

Table 5.2: Predicted label and score for each image with pretrained vgg16 model and pytorch. Each label name is abbreviated to the first whole name of the ImageNet name.

As expected the guided backpropagation method highlights fine-grained structures, but is less class discriminative and the heatmap for the four predictions are almost identical. The three other methods highlights and focuses on objects that seem natural from my point of view. The LRP is the most detailed of the four methods used.

Image: Dog/Tub

An interesting observation is that for the bucket and the tub predictions the heatmaps focuses on different shapes. The bucket prediction focus on the pixels forming half of the tub. For the tub prediction the shape of the pixels is the edges of the whole object. This can indicate that the model recognizes different areas of the object and that the bucket is more round-shaped than the tub. This is not so clear from the LRP heatmap. The question is why this happens and if it is bias related, LRP related or just coincidence.

As in section 3, it is observed that LIME gives a more precise result than Grad-CAM. Especially the edge of the tub shows this. This is also observed by comparing Grad-CAM and LIME for prediction two and four. Here it is shown that the two heatmaps for Grad-CAM is slightly different while the corresponding heatmaps for LIME shows more differences and also follows the shape of the dog and negative pixels are also highlighted.

As expected the LRP heatmaps focuses only on the dog for prediction two and four. For the two predictions labeled respectively bucket and tub it is shown that the dog is present for both heatmaps. This is unexpected but may be due to bias in the dataset that was used when training the model. Another possibility is a mismatch due to computational error or from the visualization method. This should be analyzed further.

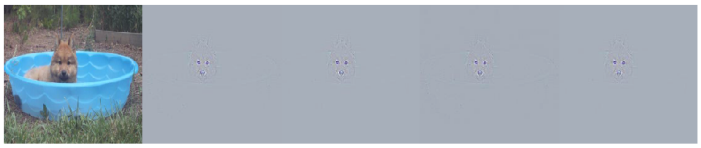

Figure 5.17: Guided Backpropagation results for image of dog and tub with respect to the four first predicted classes. See table 5.2 for specific labelnames

Figure 5.18: Grad-CAM results for image of dog and tub with respect to the four first predicted classes. See table 5.2 for specific labelnames

Figure 5.19: LIME results for image of dog and tub with respect to the four first predicted classes. See table 5.2 for specific labelnames

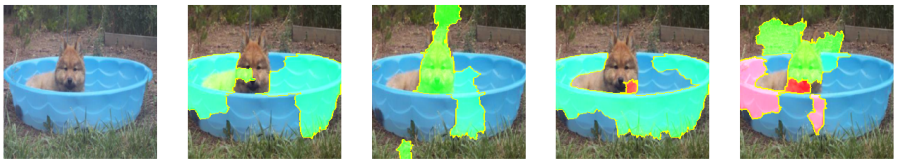

Figure 5.20: LIME results for image of dog and tub with respect to the four first predicted classes. See table 5.2 for specific labelnames

Figure 5.21: LRP results for image of dog and tub with respect to the four first predicted classes. See table 5.2 for specific labelnames

Image: Cat/Maraca

In this image the focus of all of the methods are mainly the maracas. The four first predictions contain round objects as shown in table 5.2. The outputs from the visualization methods indicates that the pixels that are recognized corresponds to the object in the image that is logically corresponding to the predictions. In this case this is the round shape of the maracas. Compared to the other methods, LIME noticeably include more of the cat and the surroundings.

It was experimented further on the cat in the image by isolating the cat and the result was that the model predicted the cat as a Persian when the maracas was not present. This indicates a weakness for predictions in images containing multiple objects with different labels, in effect highly confident label predictions obstruct other labels. Possible further work might be looking into how YOLO [39] solves this.

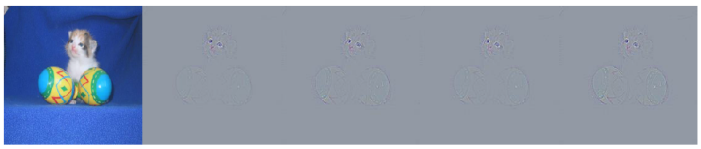

Figure 5.22: Guided Backpropagation results for image of cat and maraca with respect to the four first predicted classes. See table 5.2 for specific labelnames

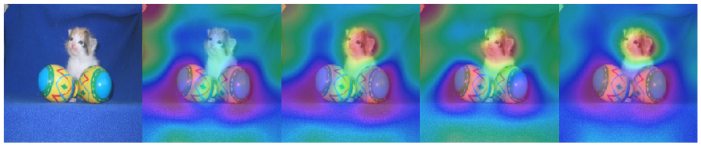

Figure 5.23: Grad-CAM results for image of cat and maraca with respect to the four first predicted classes. See table 5.2 for specific labelnames

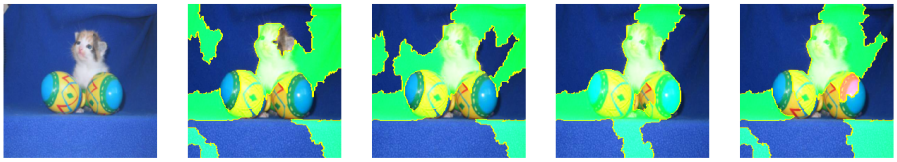

Figure 5.24: LIME results for image of cat and maraca with respect to the four first predicted classes. See table 5.2 for specific labelnames

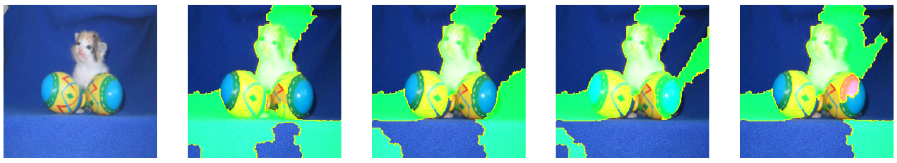

Figure 5.25: LIME results for image of cat and maraca with respect to the four first predicted classes. See table 5.2 for specific labelnames

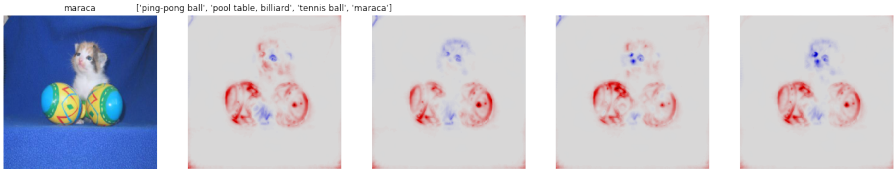

Figure 5.26: LRP results for image of cat and maraca with respect to the four first predicted classes. See table 5.2 for specific labelnames

Image: Cat/Hamper

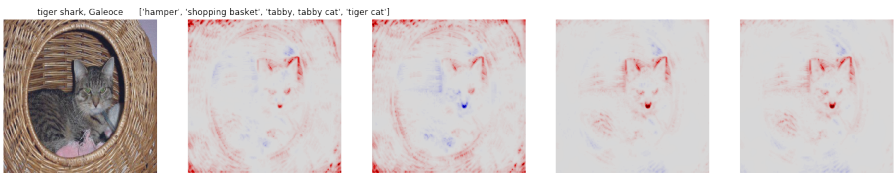

Figure 5.27: LRP results for image of cat and hamper with respect to the four first predicted classes. See table 5.2 for specific labelnames

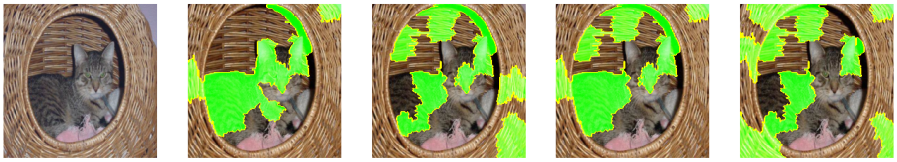

Figure 5.28: LIME results for image of cat and hamper with respect to the four first predicted classes. See table 5.2 for specific labelnames

Figure 5.29: LIME results for image of cat and hamper with respect to the four first predicted classes. See table 5.2 for specific labelnames

For the hamper/shopping basket it is also observed here that some of the surroundings is predicted as positive contribution as for tub/bucket

5.3 Discussion of initial experiments

In this section we have looked into some of the different interpretability methods introduced in section 2. By running different images we have considered the output for the explanation methods and the validity of the results. Here we have seen that as expected the Guided Backpropagation method gives pixel specific results but is not class sensitive.

Grad-CAM, LIME and LRP gives class specific results, LRP and LIME gives in addition positive and negative contributions in the same heatmap. Grad-CAM gives a more noisy output than LIME. LIME is more time consuming than the three other methods. LRP gives pixel specific results, but have some issues with specific classes, such as bucket/tub and hamper/shopping basket where the surroundings are predicted as positive contributions. It is uncertain why this happens and if it is a bias in the dataset or LRP related. Overall LRP have a benefit with its high-resolution and class-discriminative output.

For the rest of the thesis we are going to look further into LIME and LRP.

Chapter 6

Robustness of explanation methods - literature review

In this chapter we look deeper into some of the robustness analysis of the existing interpretability methods that is currently used.

When GDPR was introduced in 2016, a higher focus on the importance of transparency of digital processes followed it. As an example insurance companies have to make sure their systems do not make decisions on invalid background such as skin color, gender or the weekday you were born on. Therefore it is very important to know how the neural network is trained and if there is such biases in the dataset used. Thus the focus on understanding the prediction output from a neural network and the explanation methods (XAI) was increased. In medical and terror context it is essential that the models and the XAI gives a correct result, if not it can give critical consequences. A tool to decide if a method produces a faithfull result or not is to estimate how robust the method is.

Saliency/a posteriori methods was the first approach to explain an output from a neural network. These explanations can at first look good, but when breaking down the methods the results are revealed as deceptive.

Several articles have addressed the problem with the lack of robustness in saliency methods such as LRP, LIME and guided backpropagation. By showing that these methods fail and investigate why they fail, one can help to improve/optimize existing methods or create new more robust explanations methods.

In this section we look at different works related to investigation and solutions of existing a posteriori methods, where the output from a neural network is sent through a saliency method.

In 2018 Alvarez-Melis et al [40] investigated the robustness of explanation methods and argued that robustness in an interpretability method is very important to give a meaningful explanation. The main two arguments for why an interpretability method should strive for robustness is: The first is to remain constant - “in order for an explanation to be valid around a point, it should remain roughly constant in its vicinity, regardless of how it

is expressed". Second: "robustness of the simplified model implies that it can be approximately used in lieu of the true complex model, at least in a small neighborhood".

Most explanation methods is based on an a posteriori approach, that means that the explanation is found by looking at the output of the machine learning model and then working backwards to finding the cause. Since this does not actually give a explicit pattern of the cause it gives potentially many pitfalls. The vulnerability of an a posteriori explanation is among other things raw features such as pixels can give noisy explanations and it is not robust to transformations.

The main idea is that by making a small transformation on the input image the explanation should stay the same. Their experiments showed that this is not the case for the methods analysed. By using a measure for robustness of each method they were able to estimate how robust each method was.

The experiment was performed by using different datasets such as ImageNet and MNIST and then training a suitable model such as a random forest classifier. Then there was fetched 200 randomly samples from the test set and then use of the explanation method. At last the suitable Lipschitz continuity was used to estimate the robustness. The transforms that was used was Gaussian noise.

To estimate the robustness of the explanations they used a variation of Lipschitz continuity, a stability that measures relative changes in the output with respect to the input, but here on the local stability and not the typically global. The output is unitless quantities. The same equation is used in adversarial attacks [41].

They evaluated methods such as Saliency Maps, LRP and Occlusion sensitivity. By using a simple neural network they could verify that the explanations for a linear model were stable but not for a simple two layer network, with significant difference between the two explanation methods.

Other results were that for a small perturbed image the explanation was dramatically changed, this was especially the case for LIME and occlusion methods. LIME is a sparse superpixel based explanation and therefore make it prone to small input perturbations. For a more complex model and a set of perturbed images with the same predictions the explanations also show differences.

They also suggested strategies to enforce robustness on existing interpretability methods or design methods robust by construction. One tecnique is to train an interpretable CNN with robust explanations by using a slight generalization of criterion such as the Lipschitz continuity. This approach is also similar to Adversarial training, [41].

In [42] the same authors expanded the work to include two new requirements, explicitness and faithfullness. They argue that the ideal way to explain a model would be to train the model such that it is possible to make an interpretability method that could explain on a deeper level and not only on an a posteriori manner. The idea is to make a complex self-explaining model where interpretability is a key in the implementation and enforced through regularization. This type of model will therefore reassure the three proposed requirements for interpretability: explicitness, faithfulness and stability. This is referred to as a self-explaining neural network (SENN).

The three requirements should answer how understandable the explanations from SENN are, if the relevant features really are relevant and how coherent explanations for similar inputs are.

Explicitness stands for the fact that instead of simple features such as pixels it is focused on higher level features such as strokes in an image. These concepts have to be defined to make sense. Faithfulness is to estimate the feature relevance, the particular feature is removed and then the prediction drop is measured from the original prediction score. Stability ensures that similar inputs gives similar explanations.

To estimate the stability of the explanation model they use a variation of the Lipschitz constant and this is done to make sure the particular interpretability method is robust. The stability estimate can be time consuming for traditional interpretability methods such as LIME. This is solved by using Bayesian Optimization [43]. When they evaluated the robustness of different datasets and interpretability methods the conclusion was that the new approach, SENN, outperformed all other methods and also showed that it is a big problem with the lack of robustness of the a posteriori methods. It was also clear that the model in itself can be robust but not the interpretability method.

In 2017 P. Kindermans et al. suggested two new explanation techniques, PatternNet and PatternAttribution [44].

Here they demonstrated that interpretability methods such as LRP, Guided Backpropagation and DeConvNet was giving weak results even for linear models. They showed that the model gradient is not an estimate for the signal in the data. Instead it was revealed that it is a relation between signal direction and the distracting noise contribution.

They divided the interpretability methods in three groups of visualization: function, signal and attribution. The groups gives different information about the network, but complement each other. Where function and signal groups visualize the explanation using the original color channels, the attribution group is visualized as a heatmap of the relevance for each input pixel. From their analysis they concluded that none of these visualizations shows what the signal in the neural network is. The function group such as Gradients and saliency gives information about how to extract the signal. The signal group included DeConvNet and Guided Backpropagation shows the filter and not the direction as supposed. The last group such as LRP and DTD shows the relevance from the output through the input pixels.

To measure the robustness of the XAI they used a quality measure for neuron-wise signal estimators. These estimators were used to create an optimized explanation method. They presented solutions for the different groups. For function and signal groups the PatternNet was introduced and for the attribution methods PatternAttribution. Their experiments showed both qualitative and quantitatively that by including the data distribution the visualization methods can be optimized.

In 2019 Macdonald et al took the optimisation further by using a rate distortion framework, [45]. Here they proposed the minimisation procedure Rate-Distortion Explanation (RDE). Their analysis showed that use of heuristic explanation methods in practical applications can be useful.

By looking at the dropping distortion it is possible to find out an estima-

tion of correctness. How fast distortion drop indicates how many relevant components that are correctly identified. By comparing this method with a posteriori interpretability methods it was found that RDE performs better. This method finds the most relevant components and thus giving the steepest rate-distortion graph compared to the other

In 2020 Bykov et al. made a new approach to investigate the uncertainties of XAI where they mixed a XAI with a Bayesian Neural Network (BNN) [46]. Here they convert the XAI into an explanation method for BNN, with an integrated modeling of uncertainties. This new approach quantifies the uncertainties and visualize them.

The specific XAI they based their work on here was LRP and the new method was called B-LRP. This new approach can in theory be used on any XAI and not just LRP. The output of this was not only one heatmap as for regular LRP, but a distribution of heatmaps. As for LRP the heatmaps shows negative and positive contributions. In particular they demonstrated the results by using the 5th, 25th, 50th, 75th and 95th percentile. By looking at the different heatmaps from B-LRP it is possible to figure out uncertainties of the LRP. The pixels that are positive in the 5th percentile heatmap will stay positive in the rest of the distribution from 5-100. The same is the case for the negative pixels in the 95th percentile. The pixels that are negative in the 95th percentile will be negative from 0-95th percentile heatmaps. The 50th percentile was equal to the original LRP heatmap. By only looking at the positive pixels in the 5th and the negative pixels in the 95th it is revealed that the pixels in the heatmaps are quite changeable and thus the uncertainty are quite big for a LRP heatmap. The Clever Hans effect was also verified were the text in an image was predicted as positive even if the text was not a part of the object that was predicted. Another approach to checking the reliability of a saliency method is proposed in [47]. Here they argued that a reliable XAI method should be invariant to transformations that does not impact the model. This property is called input shift invariance. A XAI should give the same results for two corresponding equally trained models, where the only difference can be a transformation of the dataset used for training. They demonstrated that this is not the case by several examples. By comparing two models with the same XAI and where the only differences in the two models are that one is trained with the same dataset than the other but adding a constant shift. Since the process for both input images are the same the interpretability should also be the same and therefore the two heatmaps should be the same. They also showed that input shift invariance can easy be used to purposefully manipulate the explanation of predictions.

In 2020 [46] proposed a new process to investigate attribution methods such as PatternAttribution, Guided Backpropagation and LRP. Cosine similarity convergence (CSC) focuses on how much the later layers contribute to the explanation. This work was inspired by the sanity check from [48]. Here they showed that by randomizing the last layers parameters, the saliency maps stayed the same, thus showed that the last layers parameters did not contribute to the explanation. The fact that the XAI is ignoring the last layer is clearly a weakness of the interpretability method and thus can not explain the prediction faithfully. In [46] they took the work further by looking at more methods and create a new estimate, CSC, to calculate how

information from the later layers are ignored.

They also addressed the problem with class insensitivity by especially the different LRP methods [49], [50]. By recognizing the issue the community was able to optimize the method. The Datasets CIFAR and ImageNet [51] have a problem by not having more than one class pr image and thus not revealing class sensitivity.

Their results of this particular article [46] shows that almost every attribution method investigated failed the test. An exception was DeepLIFT [52] that was the only method that passed the test. A proposition to solving the class insensitivity was to backpropagate negative relevances similar to DeepLIFT.

In this section we have looked at works related to investigations of XAI methods and their solutions. These methods can be optimized by making a new model with an integrated explanation method such as SENN [42], PatternNet and PatternAttribution [44] or measure of the robustness of the existing models such as B-LRP [46], RDE [45] and CSC [53]. Inspecting where the XAI methods fail can help to improve/optimize and understand how to create new more robust XAI methods. It was also clear that a neural network in itself can be robust even though the XAI is not.

Chapter 7

Experiments on measuring robustness of LRP and LIME

In this chapter we look at how to measure explanation method robustness. The goal of this chapter is to find a method to measure how robust the explanation method is and decide on which method to use.

The robustness can be estimated on different approaches such as B-LRP [46], Lipschitz continuity [40], RDE [45] and Root Mean Square Error (RMSE) [54]. To test the robustness of an explanation method and a network/model one can use similar techniques as when augmenting datasets used for training a model. Examples of this are adding different types of noise, rotation, color shift and zoom. For testing of robustness others have also used perturbations such as patching out parts of the image [11] as described in chapter 3.1.

We decided to use RMSE to estimate the robustness for these experiments. To test the robustness we decided to use rotation and Gaussian noise. We use minor transformations to verify that the model and the XAI method perform as it should. If the prediction changes it might indicate a weakness in the model and on the contrary if the RMSE for the explanation before and after the transformation increases, it is an contraindication of robustness in the explanation method.

By looking at robustness indicators like RMSE with these basic transformations it is possible to decide if further investigations should be done. These investigations might include more complex robustness investigations such as Lipschitz continuity [40, 42] or adversarial attacks [41].

7.1 Quantitative Analysis

For the quantitative analysis we use Root Mean Square Error (RMSE) as a tool. The RMSE value that is close to 0 is best, where the ideal value is 0. The higher the RMSE value the more inadequate result, thus signaling a brittle method.

RMSE is given as:

$$RMSE = \sqrt{\sum_{i=1}^{N} \frac{(y_{transformed_i} - y_{ref_i})^2}{N}}, \tag{7.1}$$

where $y_{transformed_i}$ is the transformed heatmap pixels and y_{ref_i} is the reference heatmaps pixels. N is the number of pixels in the heatmap and i denotes the pixel.

The RMSE is calculated for each LRP pair, with LRP after transformation as $y_{transformed}$ and the original LRP as y_{ref}. For the rotations the reference LRP output is rotated by the same angle as the image was rotated before LRP estimation and then RMSE is calculated. This is done to get the original heatmap and the transformed heatmap to be as similar as possible and thus prevent an additional source of error.

To use RMSE as a measurement of the explanation methods correctness, it is dependent on the models robustness. The original and the transformed image should produce similar results from the model. If the model can do this, a divergence in the RMSE score for the image pair can reveal deficiencies in the explanation method.

7.2 Transformations

We have decided to use the following transformations in this experiment:

- rotation of the input image
- adding of noise to the input image

It is also possible to transform the image either by changing the grayscale or zooming. Two potential issues by doing this are that most models look at the texture and not the shape of the object when predicting [55]. By changing the color scheme, the observed texture changes, potentially resulting in a different prediction by the model. When zooming, important parts of the object might be clipped resulting in erroneous predictions i.e. distinct ears of a lynx being cut resulting in a prediction of a cat instead. Randomized zoom may only leave a small part of the object which might be outside the scope of the models training. This might discover flaws of the model such as bias in the dataset.

Figure 7.1: Examples of an image before and after transformations. The original image to the left, 10 degree rotated image in the middle and image with added Gaussian noise with sigma = 5.0

As seen the middle image in Figure 7.1 the rotation gives gray edges, and this again can generate a boost in the LRP detection. To avoid this the image is cropped before sent through RMSE estimation. Since the model used here, VGG-16, requires a set input size and the images used are approximately the same size, the cropping have to be done after the LRP estimation. This can therefore affect the result of the prediction estimation. Another problem with the images used in this experiment is that the object spans over substantial parts of the image. Additionally the size of the images are small 224×224 and therefore the cropping may remove significant parts of the object of interest. Small angles will work, but larger angles will remove to much of the object. A solution to this would be to use images that can be cropped such that the whole object is not affected by the cropping of the image and still have proper size required by the model. The grey edges in this experiment are removed by calculating where the grey edges are and cropping them away. Another approach could be to take a portion of the object in the image and then find that part of the image after rotation. This could then be sent through LRP and estimate the RMSE for this portion on each image.

7.3 Explanation methods

The two chosen interpretability methods for these experiments are LIME and LRP. The implementation we use for LRP is the one based on [26] as described in chapter 4. The LIME implementation is based on the tutorial from [3, 56]. We use Jupyter Notebooks to execute all code.

The LIME algorithm used is from the lime package introduced in [56]. This method is more complex to implement and it is also dependent of images, and not tensors.

7.4 Experiment process

The model used in this section is a pretrained VGG-16 model from PyTorch [37, 34]. A discussion of why this is selected is found in chapter 4. The images used are the same as presented in 5.2. These images contain

more than one object and have almost the same shapes as the ImageNet dataset that the model is trained on. In this way we avoid the displacement/cropping of the object.

The model is trained with different types of augmentation [37, 57]. Here it is used techniques to recognize objects at different scales, horizontal flipping and color shift and subsections of the images. The model is not trained with vertical flipping and rotated augmentations.

We use the images with transformations and compare them to the original images. The experiment can in theory be used on any CNN model or explanation method.

7.4.1 Rotation of input image

The rotation angles used are: 0.5, 1.0, 5.0 and 10.0 degrees. It is also tested with significantly greater angles between 15 to 40 degrees. The first prediction of the rotated image from a pretrained VGG-16 model is used further to compare the changes in rotation in the original image. In table 8.1 the prediction score and the corresponding angle used is shown. To estimate the RMSE for each image pair, the LRP heatmaps before and after rotation are compared. The original heatmap is rotated by the same angle as the original image sent through LRP.

It was in addition experimented with greater rotation angles (15, 20, 25 and 40 degrees). Since the image is quite small and the object is not surrounded by much background it is best to do this experiment on images with more background than the images used here.

7.4.2 Noise on input image

For the experiment we added Gaussian noise to each image before sending it through LRP. For the noise we use Gaussian distributed noise on input with sigmas from 0.01 to 0.1 with step of 0.01. The used explanation heatmaps is then compared with the original explanation heatmaps. The explanation methods we use here are LRP and LIME. We also experimented with significantly higher levels of sigma between 0.25-10.0. The small sigma parameters was selected such that the changes in the image with noise was minor enough to still be recognized by the human eye. The higher sigma values was to investigate how much noise could be added before the model were unable to predict accurately.

Chapter 8

Results

In this chapter we investigate how robust the selected explanation methods, LRP and LIME are by doing a quantitative analysis. We specifically evaluate how the method performs with transformed data using noise and rotation. This chapter presents the results from the experiments described in chapter 7. The results are divided in a noise and a rotation section where an investigation for each explanation method have been done. Finally we conclude our findings in the end of the chapter.

Here we investigate the LRP and LIME methods and compare the original heatmaps to the transformed heatmaps. We focus on the first prediction, the reason being that this gives the highest prediction score and the LRP method is unreliable for low prediction scores [2], as discussed earlier in chapter 3.

Due to the sparse results LIME was giving, this method was only investigated with the minor transformations i.e. rotations between 0.5-10.0 degrees and Gaussian noise with σ between 0.01-0.10.

8.1 Rotation

In this section we present the results for the rotation part of the experiment. This includes the explanation methods heatmaps after the specific angles and RMSE graphs for rotation pr image.

From tables 5.2 and 8.1 it is observed that the first prediction for the images cat/maraca and cat/hamper the prediction label is similar before and after transformation. This is not the case for the dog/tub image where the original image prediction label is bucket and for the angles 1.0 and 10.0 gives prediction Norwich terrier. The difference is also observed in the RMSE graphs in Figures 8.14 - 8.16, where the similar prediction labels before and after transformation is more even and the graph for the dog/tub gives a jump when the label is different. This is expected since the two labels are quite different and will mostly focus on different pixels in the image and therefore give higher RMSE values.

By comparing the original labels to the labels after rotation it can imply a bias in the dataset, and that the object is not represented in different positions. Most objects in real life can exist in different poses, such as a

plane taking off. The plane upside down is not a logical position, but if it is a system that searches for a plane crash this might be a potential orientation in the required models dataset. On the other hand a model that detects numbers should be sensitive to rotations, since 6 and 9 are equal upside down. In this example it would be better if the system have some requirement such as that the document scanned should be placed in a given position or some kind of context based rule for detecting orientation.

It is observed that for some of the rotations the LRP predicts a bit more positive/negative pixels in the edges than the reference LRP. This will affect the RMSE estimation and may therefore give a higher deviation in RMSE than without these effects. To avoid this the edges can be cropped away. Particularly if the pixels in the edges are not clearly important for the prediction of the object and not part of the object to investigate the edges can be cropped away from the heatmaps before RMSE is estimated. Another approach is to fill the gray parts due to the rotation with similar pixels from the edges. In this particular case it is observed that the gray edges are not affecting the models predictions in a considerable amount and therefore the edges from the heatmaps are cropped away after the prediction and before the RMSE estimation. For the image containing cat/hamper the object detected is in the edges. In this particular case it is observed that the edges are part of the hamper/shopping basket label, but also maintain as positive pixels through the rotations. The edges from the heatmaps are also here cropped away before the RMSE estimation.

For the cat/hamper image the prediction consistently gives similar labels (i.e. hamper/shopping basket and tabby/egyptian cat) through label 1-4 with respect to reference images. This shows in the graphs for RMSE, as they are quite similar and seemingly increases proportionally with the increased rotation. This is also true for the cat/maraca images with an exception in the fourth prediction where the angles 0.5 and 10.0 gives a different label than the reference, as shown in the RMSE graph. For the dog/tub images the angles that show different prediction label gives higher RMSE and the similar labels give a lower RMSE score. Especially prediction 1 and 2 where the label respectively are terrier/bucket and bucket/terrier. It is also observed that bucket prediction shows positive contribution for the dog pixels as in chapter 3.2.

For the graphs it is observed that the cat/hamper image is the one that gives lowest RMSE value for all the rotations and the predictions. This is unexpected since it has the most positive/negative contributions of the three images. One would have expected that this image would have given the highest RMSE values. One reason can be the edges in the heatmaps, another can be that the object of the predicted label is stretched more out in the image and thus the range of the positive contributions are higher and will potentially be more sources for errors since LRP is an interpretability method that gives pixel specific output and not area such as LIME and Grad-CAM.

The image with the dog/tub might reveal some issues with the model since the model predicts different labels for the original image and the transformations 1.0 and 10.0 degrees. The RMSE values for these degrees must be disregarded since it clearly will give a higher RMSE than for two similar predictions. The heatmaps for the other rotations shows that there

are some differences in the positive contributions of the pixels. The 5.0 degree heatmap is closer to the original heatmap, and this prediction score is also higher than the 0.5 degree. The 0.5 degree focuses on the detail of the tub as opposed to the original and the 5.0 degree heatmap.

For the cat/maraca image it is observed that the angles 0.5, 1.0 and 10.0 results in predicted label ping pong ball as for the original image with the corresponding score: 44.7%, 58.8% and 43.4%. In this case the prediction is wrong but the shape and texture are similar to the maracas in the image as the LRP take as positive contributions. In these heatmaps it is a distinct difference in the heatmaps. Especially the negative contributions are variant for the three heatmaps. These variations indicates that the explanation method may not be robust for these transformations.

The cat/hamper image rotated gives prediction label hamper and shopping basket. Even for a small change in rotation 0.5 degrees, the score drops approximately 10% which can indicate that the model in itself have a dataset bias and the hamper is only represented in one position, and not for example upside down. Hamper and shopping basket have similar texture and therefore the heatmaps looks similar.

Although the shape of the predicted label is similar it is proved that cnn systems as opposite to humans do not look at shapes but on textures such as fur. This was tested with a shape of a cat but with elephant skin. The models predicted elephant and not cat, [55]. This can be a reason why machine learning systems are fragile to transformations.

For an interpretability method it is expected that the explanation shows the same when the model predicts the same for the different transformations, this is addressed in several articles and discussed in chapter 6. 10 degrees for a rotation is not a big transformation and one would expect that the models prediction would stay the same as the original input and therefore the RMSE values would stay close to zero. This is not the case and it indicates that even though the model in itself is robust for transformations the interpretability method is not necessarily that.

Angle/True label	dog/tub	cat/maraca	cat/hamper
0.5	bucket: 43.8%	ping-pong ball: 44.7%	hamper: 34.7%
1.0	Norwich terrier: 63.4 %	ping-pong ball: 58.8%	shopping basket: 45.4%
5.0	bucket: 51.0%	tennis ball: 38.8%	shopping basket: 36.6%
10.0	Norwich terrier: 43.4 %	ping-pong ball: 43.4%	shopping basket: 26.0%

Table 8.1: First prediction for each rotated image with pretrained vgg16 model and Pytorch. The images dog/tub, cat/maraca and cat/hamper are rotated for a given angle and then the first prediction is fetched for the LRP visualization. The column shows the first prediction label and score for each rotated image. Each label name is abbreviated to the first whole name of the ImageNet name. Angles are in degrees.

8.1.1 LRP results

In this section we look at different LRP heatmaps for rotation of the images. We also look at RMSE effects between cropped and not cropped images before estimating RMSE. The main experiments are done with minor angles

and the VGG16 model. In a second experiment we use significantly higher rotations on the cropped input images. Finally the experiment with minor rotations are briefly repeated using a VGG16 architecture with batchnorm layers.

RMSE results without cropping the images

The heatmaps for the first prediction of each rotation are shown in Figure 8.1, The RMSE is estimated for these heatmaps. The RMSE graphs are found in Figures 8.2 - 8.4.

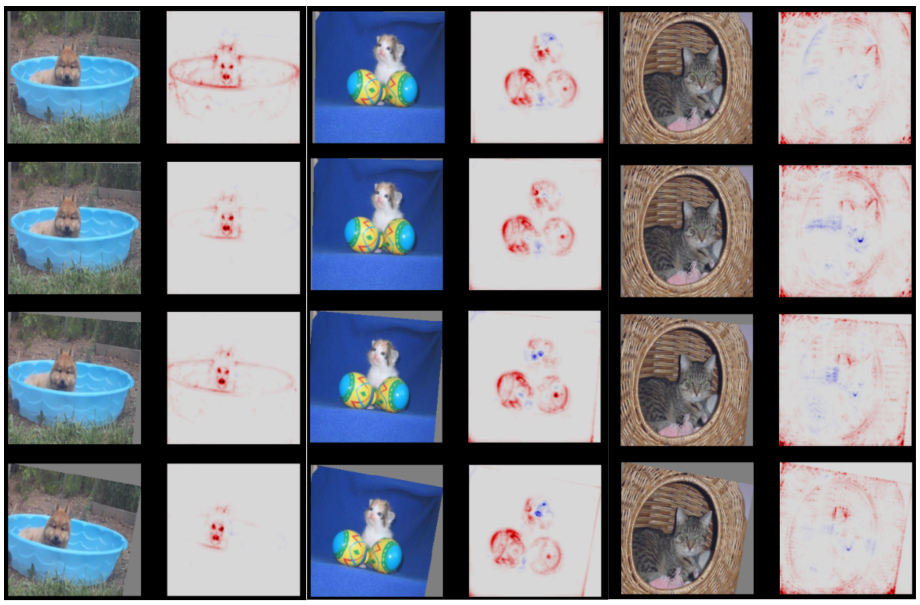

Figure 8.1: LRP heatmaps for the first predictions for each rotation of all three images with angles 0.5, 1.0, 5.0 and 10.0 degrees. The smallest angle from the top to the highest angle in the bottom, 10.0 degrees. The image of dog and tub to the left, cat/maraca in the middle and cat/hamper to the left.

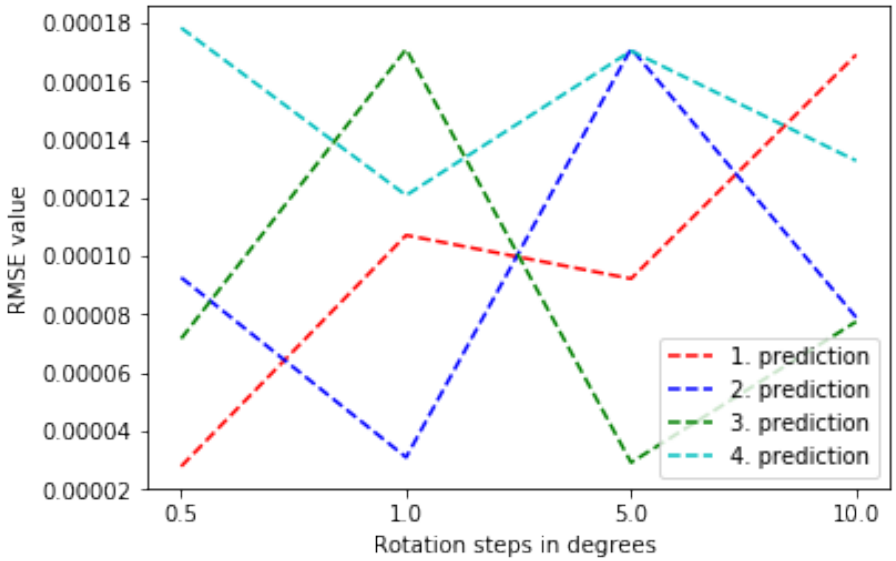

Figure 8.2: Quantitative Analysis of image of dog and tub rotated using RMSE with respect to original LRP heatmaps.

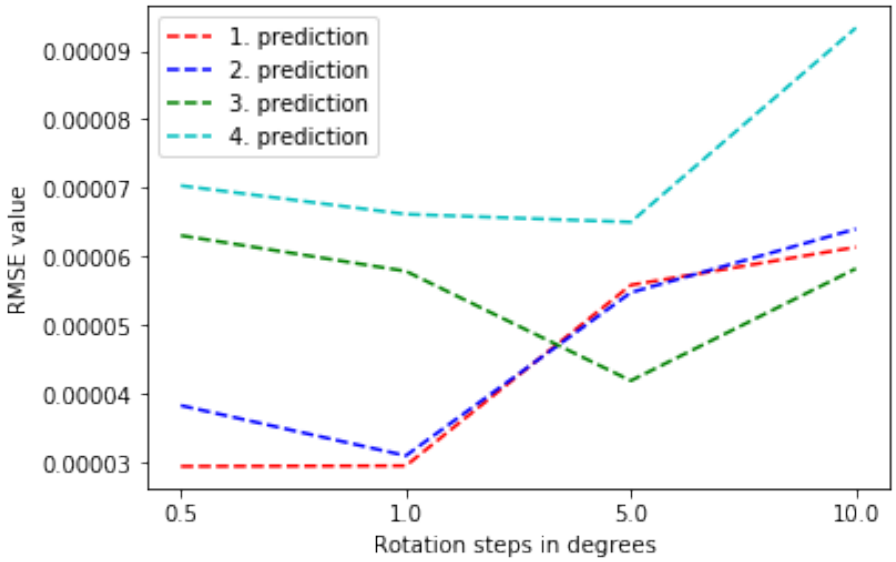

Figure 8.3: Quantitative Analysis of image of cat and maraca rotated using RMSE with respect to original LRP heatmaps.

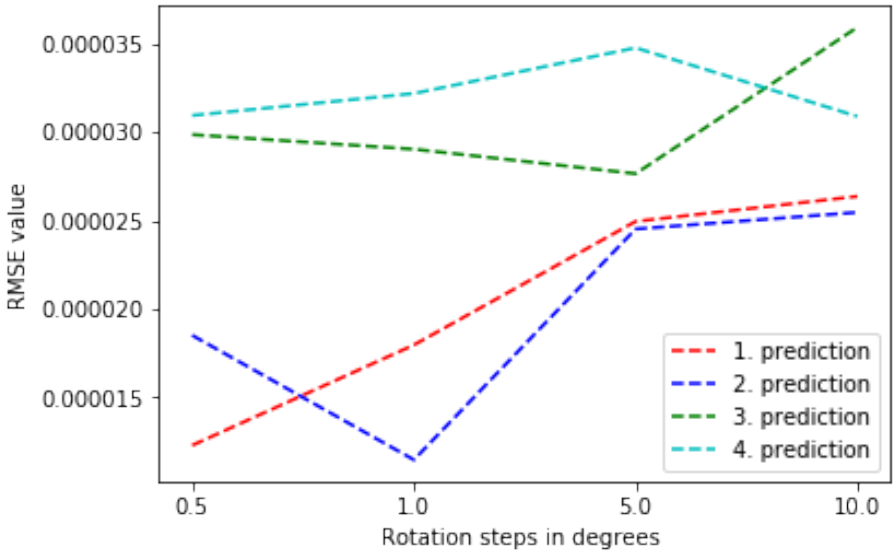

Figure 8.4: Quantitative Analysis of image of cat and hamper rotated using RMSE with respect to original LRP heatmaps.

RMSE results for cropped heatmaps

The cropped images and corresponding heatmaps are found in Figures 8.5-8.7. The corresponding RMSE graphs for the first 4 prediction for each image are found in Figures 8.8-8.10.

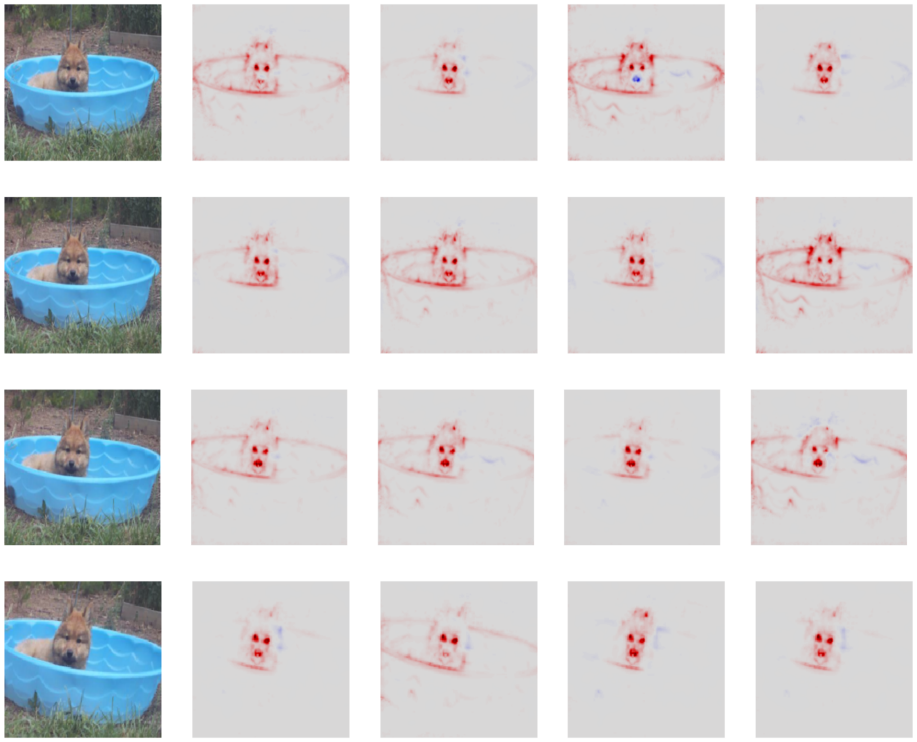

Figure 8.5: LRP heatmaps for the first predictions for each rotation of image of dog and tub with angles 0.5, 1.0, 5.0 and 10.0 degrees. The smallest angle from the top to the highest angle in the bottom, 10.0 degrees.

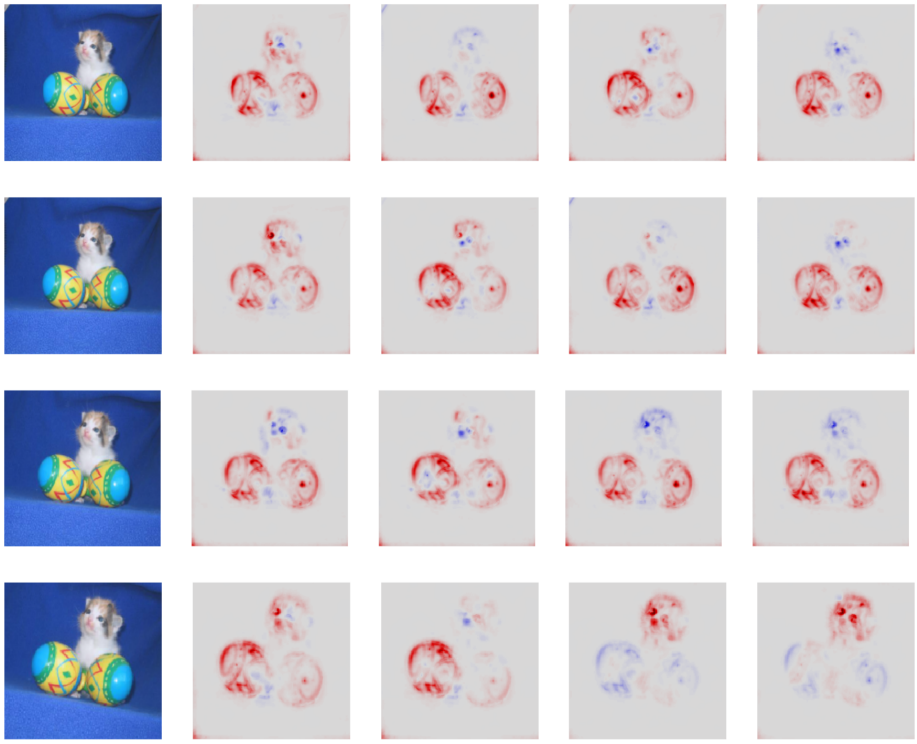

Figure 8.6: LRP heatmaps for the first predictions for each rotation of image of cat and maraca with angles 0.5, 1.0, 5.0 and 10.0 degrees. The smallest angle from the top to the highest angle in the bottom, 10.0 degrees.

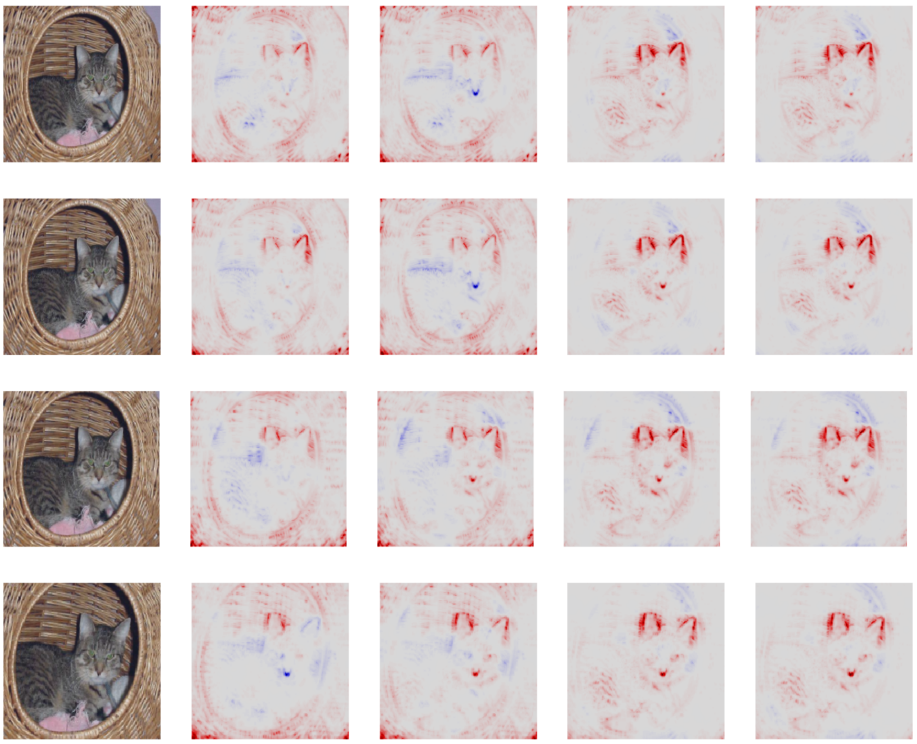

Figure 8.7: LRP heatmaps for the first predictions for each rotation of image of cat and hamper with angles 0.5, 1.0, 5.0 and 10.0 degrees. The smallest angle from the top to the highest angle in the bottom, 10.0 degrees.

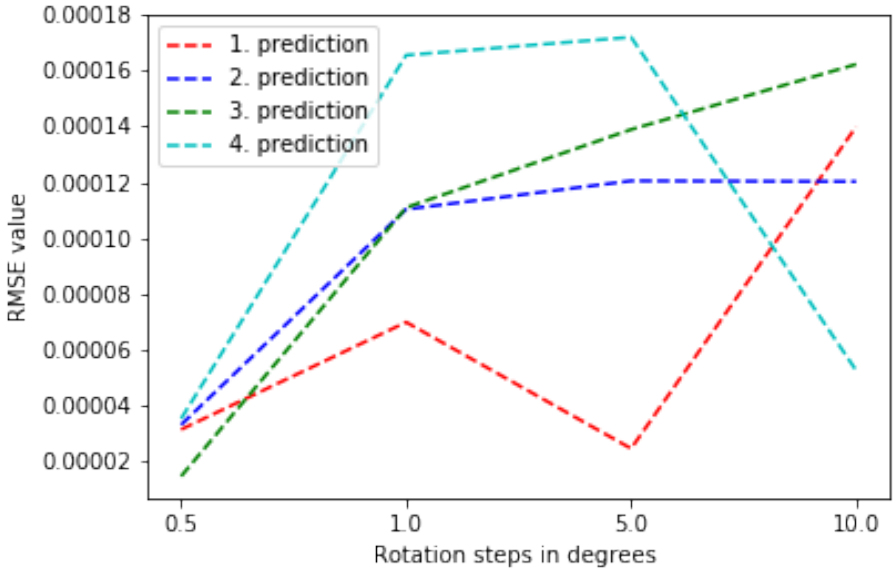

Figure 8.8: Quantitative Analysis of image of dog and tub rotated using RMSE with respect to original LRP heatmaps.

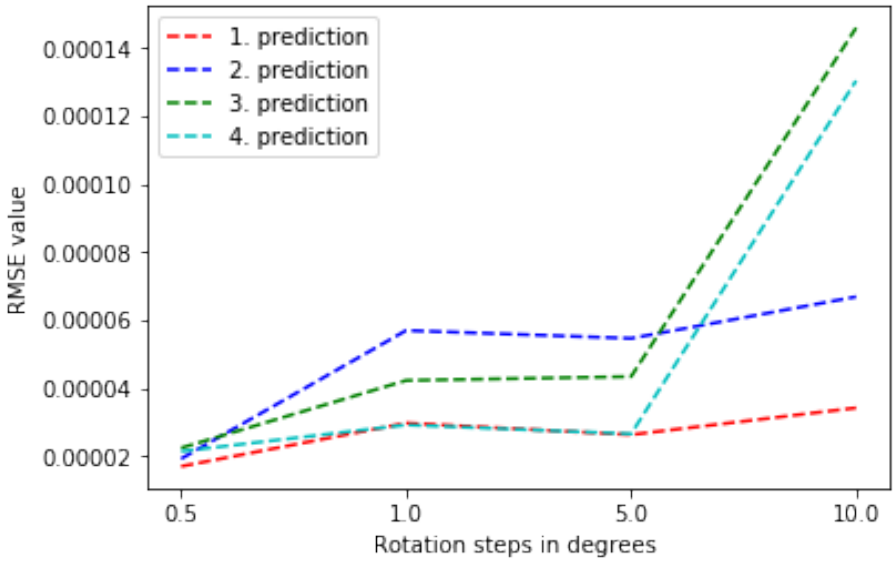

Figure 8.9: Quantitative Analysis of image of cat and maraca rotated using RMSE with respect to original LRP heatmaps.

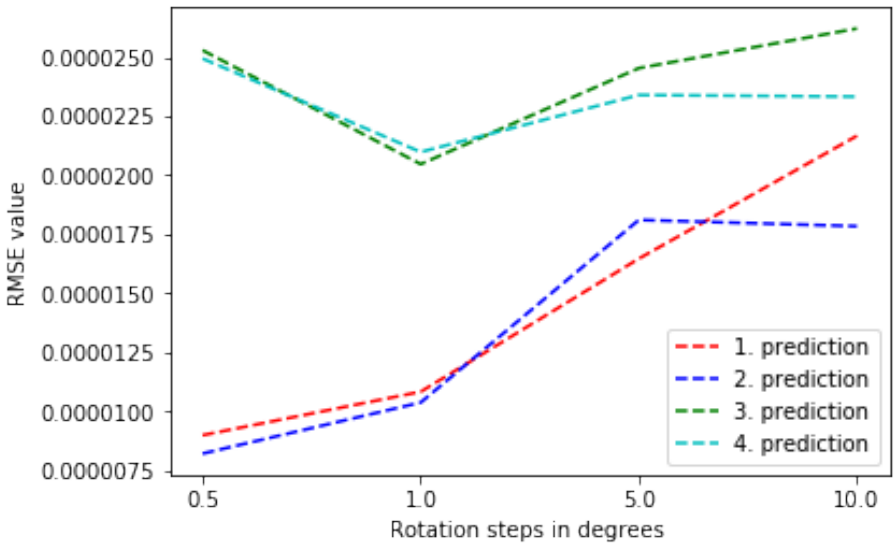

Figure 8.10: Quantitative Analysis of image of cat and hamper rotated using RMSE with respect to original LRP heatmaps.

By rotating the cat/hamper image even more it is observed that the predicted label changes to 'tabby cat' for all rotation in this part of the experiment. While the original image is predicted as 'hamper'. This will affect the RMSE results. The rotation/cropping of the image leads to important parts of the object hamper disappears from the part that is sent through the LRP. It is observed that as expected when the angle is to high the cropped image is not recognizable. The RMSE calculation give inconsistent results since the rotated and the original images have different prediction labels. This is observed for all three images, both for cropped and not cropped versions.

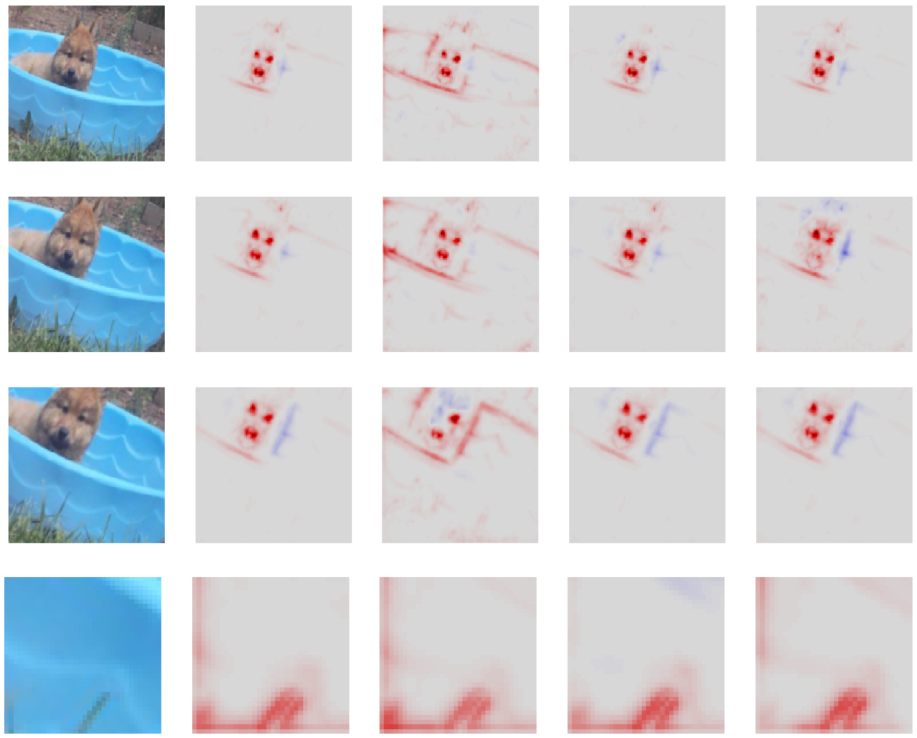

Figure 8.11: LRP heatmaps for the first predictions for each rotation of image of dog and tub with angles 15.0, 20.0, 25.0 and 40.0 degrees. The smallest angle from the top to the highest angle in the bottom, 40.0 degrees.

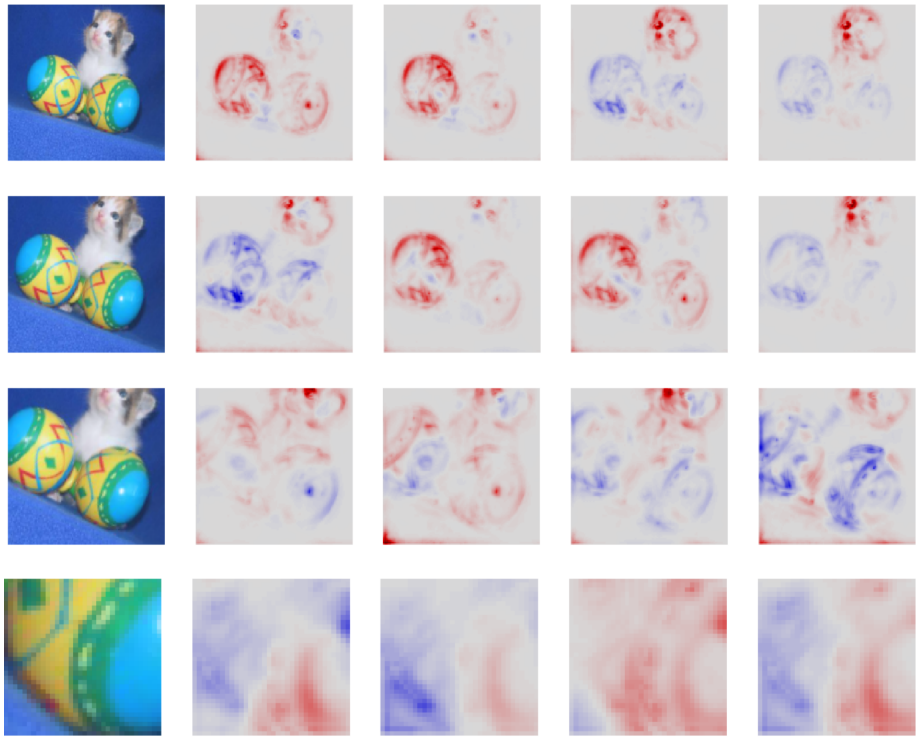

Figure 8.12: LRP heatmaps for the first predictions for each rotation of image of cat and maraca with angles 15.0, 20.0, 25.0 and 40.0 degrees. The smallest angle from the top to the highest angle in the bottom, 40.0 degrees.

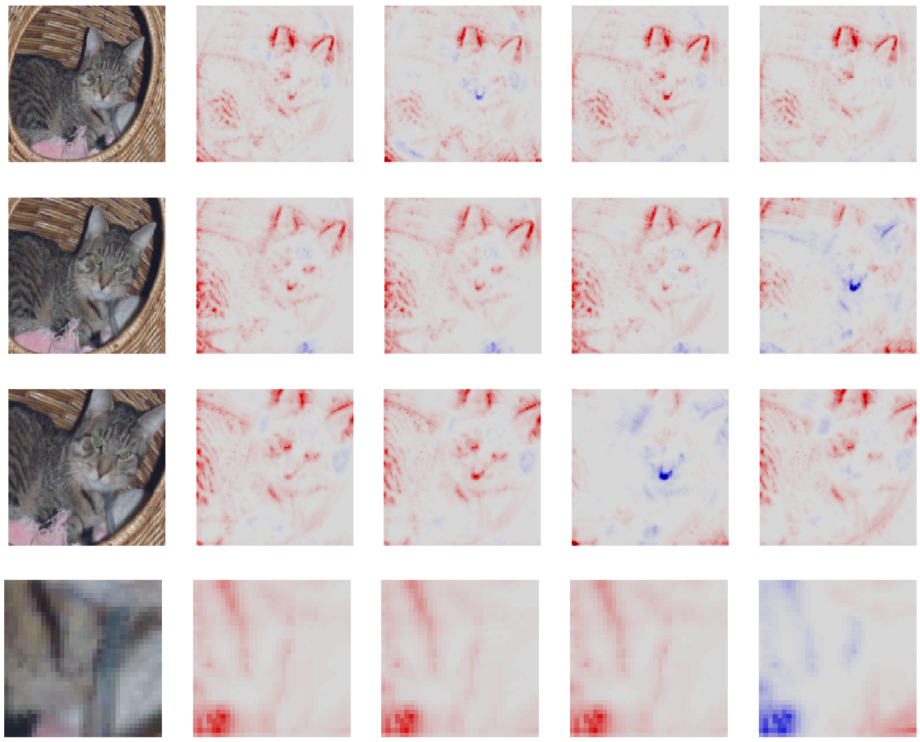

Figure 8.13: LRP heatmaps for the first predictions for each rotation of image of cat and hamper with angles 15.0, 20.0, 25.0 and 40.0 degrees. The smallest angle from the top to the highest angle in the bottom, 40.0 degrees.

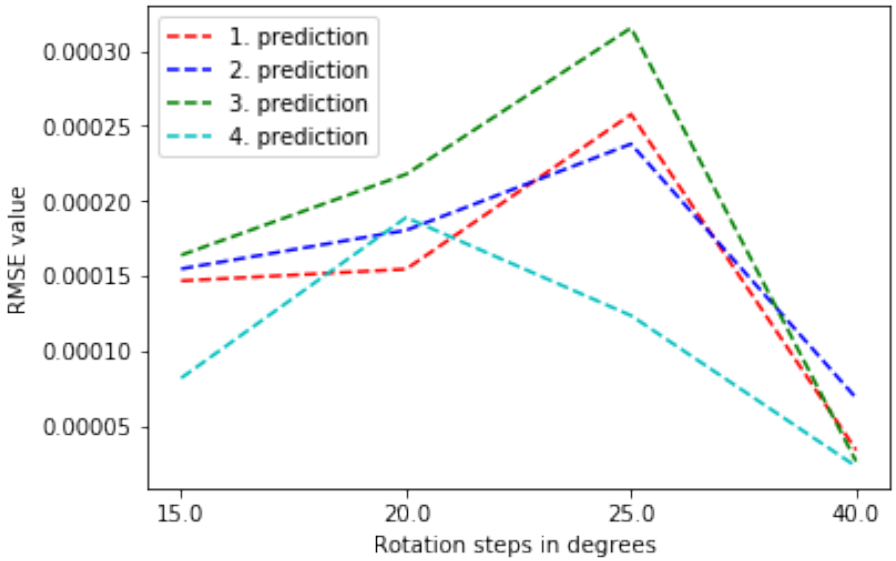

Figure 8.14: Quantitative Analysis of image of dog and tub rotated using RMSE with respect to original LRP heatmaps.

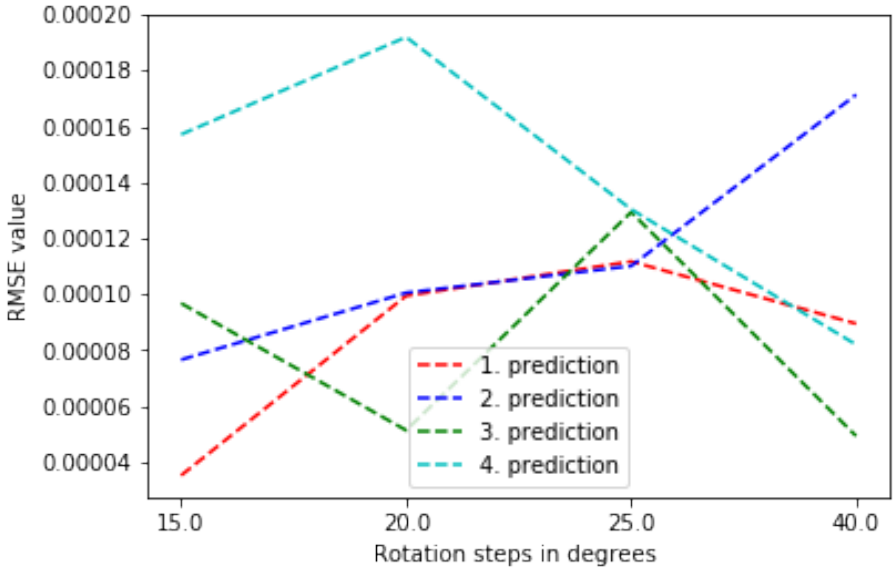

Figure 8.15: Quantitative Analysis of image of cat and maraca rotated using RMSE with respect to original LRP heatmaps.

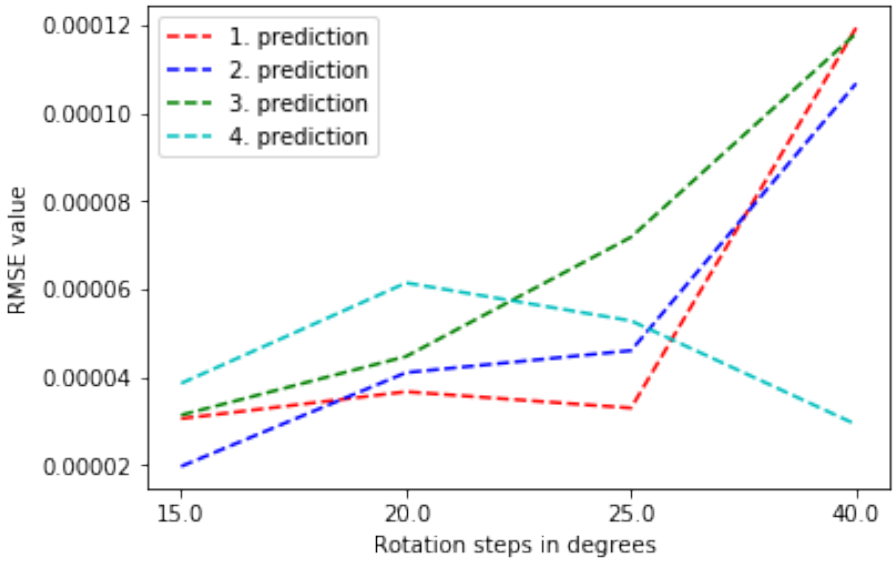

Figure 8.16: Quantitative Analysis of image of cat and hamper rotated using RMSE with respect to original LRP heatmaps.

Comparison of cropped and not cropped images

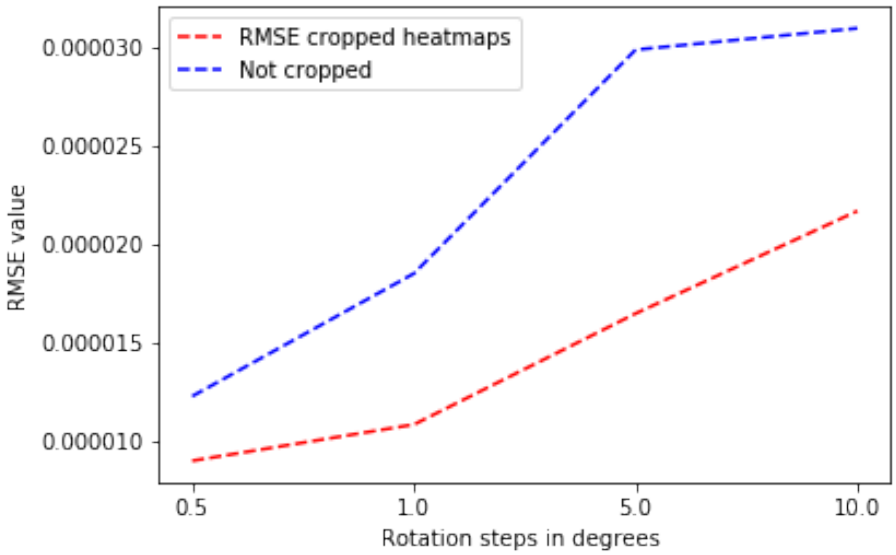

Figure 8.17: Quantitative Analysis of image of cat and hamper rotated using RMSE with respect to original heatmaps.

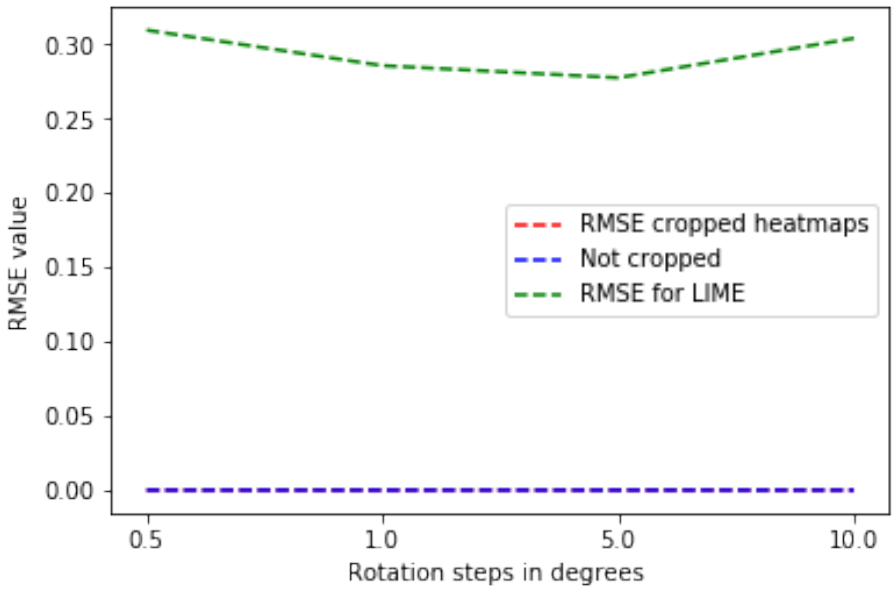

Figure 8.18: Quantitative Analysis of image of cat and hamper rotated using RMSE with respect to original heatmap. LIME vs LRP.

LRP results using VGG16 with batchnorm

In this section we present the results for the rotation experiment using minor rotation and a pretrained VGG16 with batchnorm layers. In Figure 8.19 the heatmaps for the rotated input images are shown. The four first predicted labels are shown in Figure 8.20. In conclusion a comparison of the two models used in this chapter are shown in Figure 8.21.

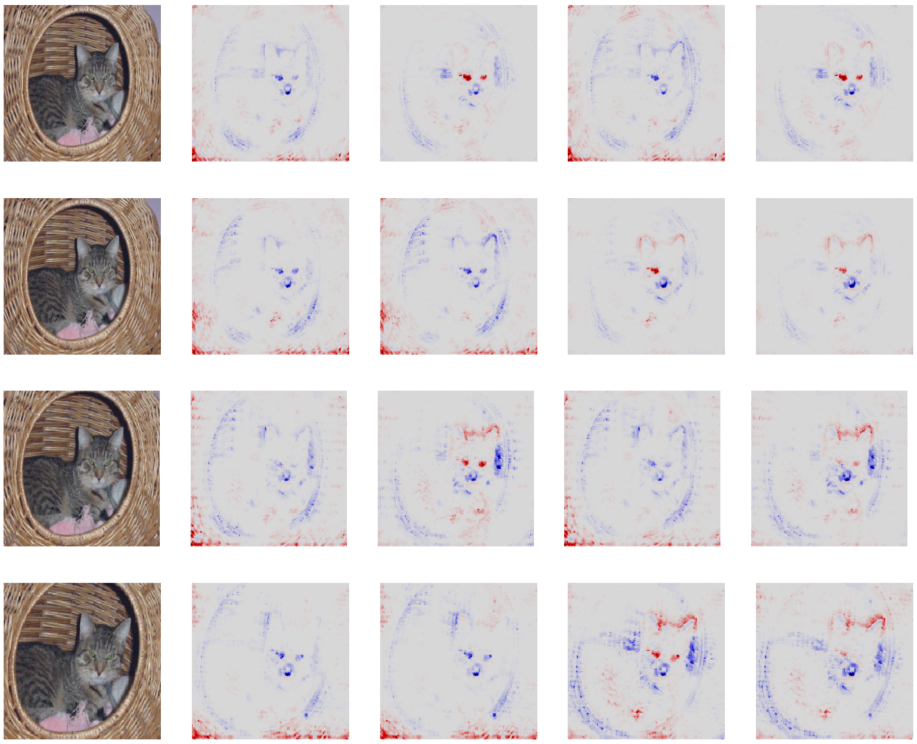

Figure 8.19: The four first LRP heatmaps generated for each rotation.

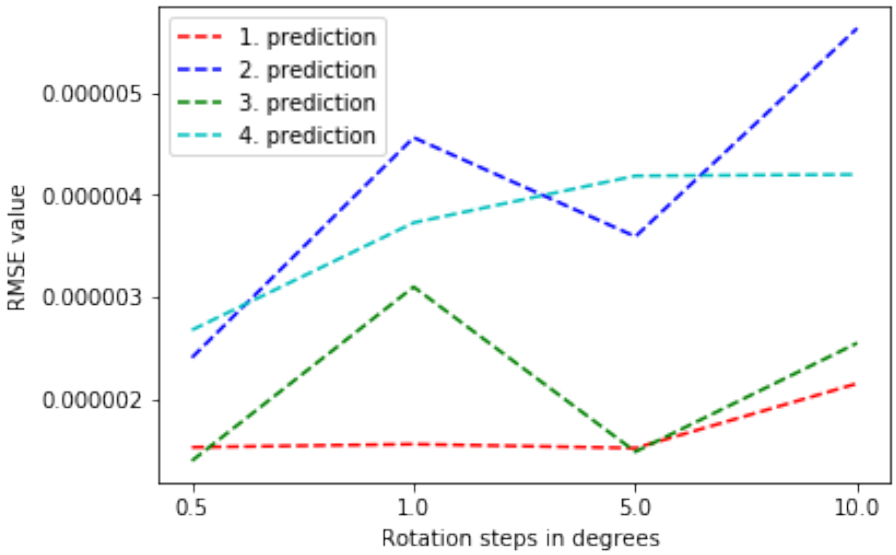

Figure 8.20: The four first RMSE for each rotation.

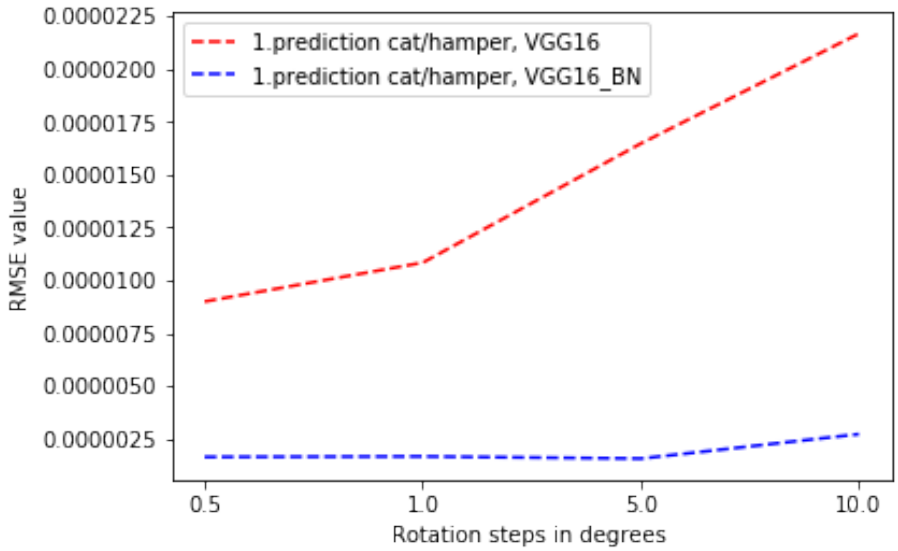

Figure 8.21: Comparison of Quantitative Analysis of image cat/hamper using two different models. RMSE calculated by using rotation.

8.1.2 LIME results

In Figure 8.22 shows the results for the experiment with rotation of the input image and using the explanation method LIME. Only the first prediction pr image is shown in this Figure. The corresponding RMSE graphs is shown in Figure 8.23-8.25.

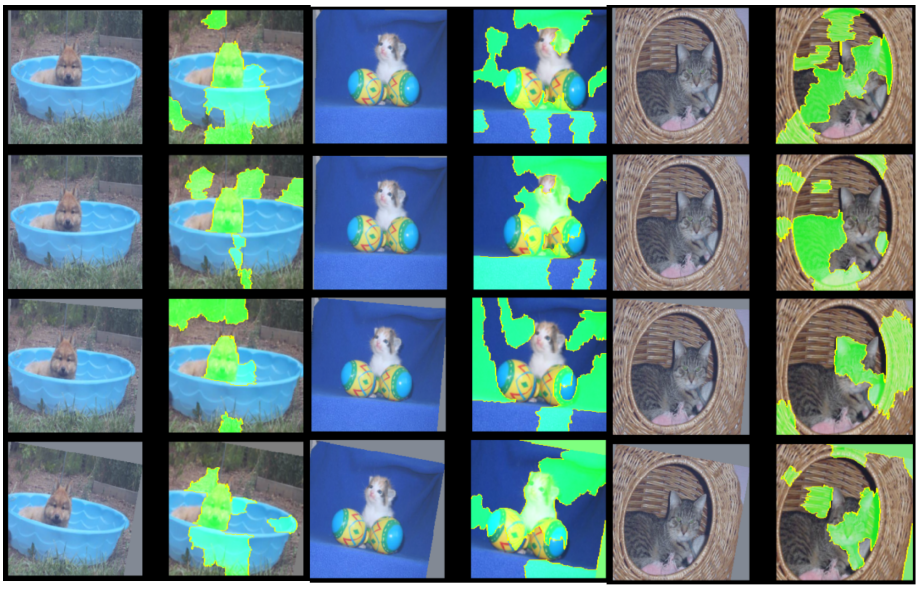

Figure 8.22: LIME heatmaps for the first predictions for each rotation of all three images with angles 0.5, 1.0, 5.0 and 10.0 degrees. The smallest angle from the top to the highest angle in the bottom, 10.0 degrees. The image of dog and tub to the left, cat/maraca in the middle and cat/hamper to the left.

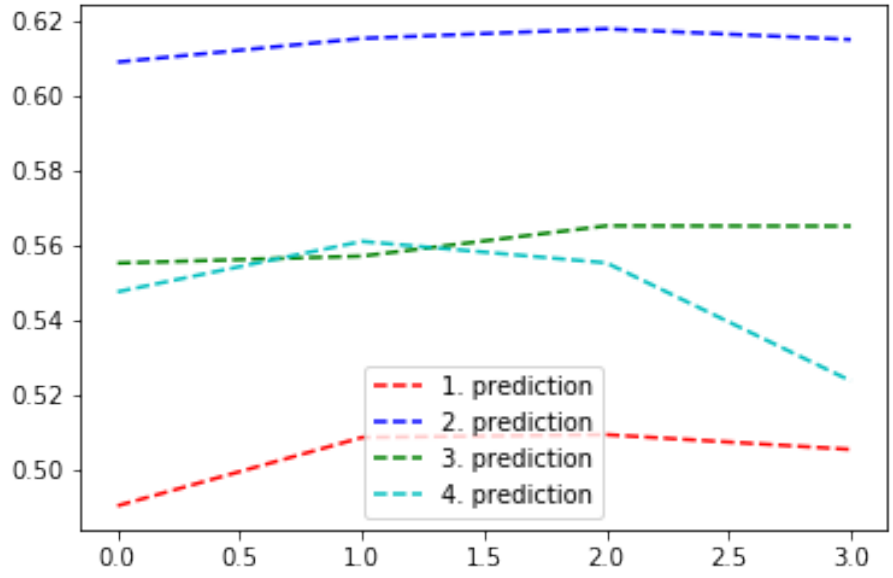

Figure 8.23: Quantitative Analysis of image of dog and tub rotated using RMSE with respect to original LIME heatmaps.

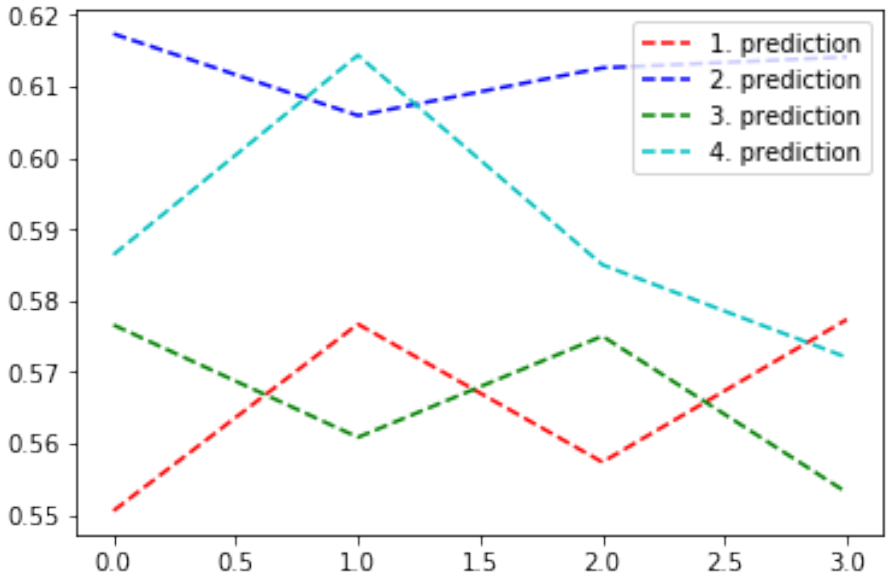

Figure 8.24: Quantitative Analysis of image of cat and maraca rotated using RMSE with respect to original LIME heatmaps.

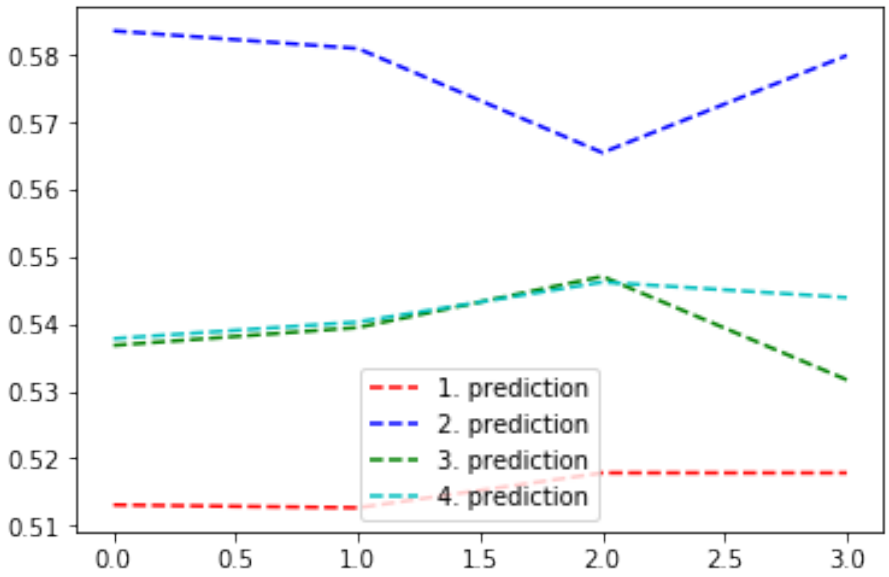

Figure 8.25: Quantitative Analysis of image of cat and hamper rotated using RMSE with respect to original LIME heatmaps.

8.2 Noise on original image

As observed in Table 8.2 it is cat/hamper that gives the most stable prediction scores of the three images. This indicates that the changes in the heatmaps should be from the explanation method and not due to instability of those particular networks used here.

We focuses here on the image cat/hamper since this image have the same predicted label through the smallest adding of noise. It is observed that for these adding of noise there are drastically changes for the LIME heatmaps than the LRP heatmaps. Since this image gives the most stable result we decided to test it with higher levels of noise.

LIME finds superpixels and small changes may have major effect on the explanation heatmap. This is also discussed in [40].

Added noise/True label	dog/tub	cat/maraca	cat/hamper
0.01	bucket: 53.0%	ping-pong ball: 45.7%	hamper: 44.3%
0.02	bucket: 54.1 %	pool table: 50.1%	hamper: 45.0%
0.03	bucket: 50.4%	ping-pong ball: 48.0%	hamper: 45.6%
0.04	bucket: 41.6 %	pool table: 51.5%	hamper: 43.4%
0.05	bucket: 43.6%	ping-pong ball: 45.8%	hamper: 44.6%
0.06	bucket: 39.3 %	ping-pong ball: 40.1%	hamper: 42.8%
0.07	bucket: 36.4%	ping-pong ball: 41.5%	hamper: 42.7%
0.08	bucket: 32.3 %	pool table: 41.1%	hamper: 49.8%
0.09	bucket: 38.0%	pool table: 36.6%	hamper: 41.7%
0.1	bucket: 20.9 %	ping-pong ball: 41.7%	hamper: 41.5%

Table 8.2: First prediction for each transformed image with pretrained vgg16 model and Pytorch. The images dog/tub, cat/maraca and cat/hamper are added Gaussian noise and then the first prediction is fetched for the LRP visualization. The column shows the first prediction label and score for each image added noise. Each label name is abbreviated to the first whole name of the ImageNet name.

8.2.1 LRP results

The first four predictions and corresponding LRP heatmaps for the image cat/hamper are found in Figures 8.26-8.27.

The RMSE graphs for the first prediction for all images are found in Figure 8.28. Table 8.2 shows the first prediction score and label for each image with added Gaussian noise. The noise added are from 0.01 to 0.1, with step size of 0.01.

The fact that the bucket prediction decreases from 53.0% to 20.9% indicates that the model in itself is not as robust as expected. A model should manage to give the same output regardless of adding a small amount of noise. The image is not changed enough to see the difference from the original image. Here the labels are consistently the same as the original image.

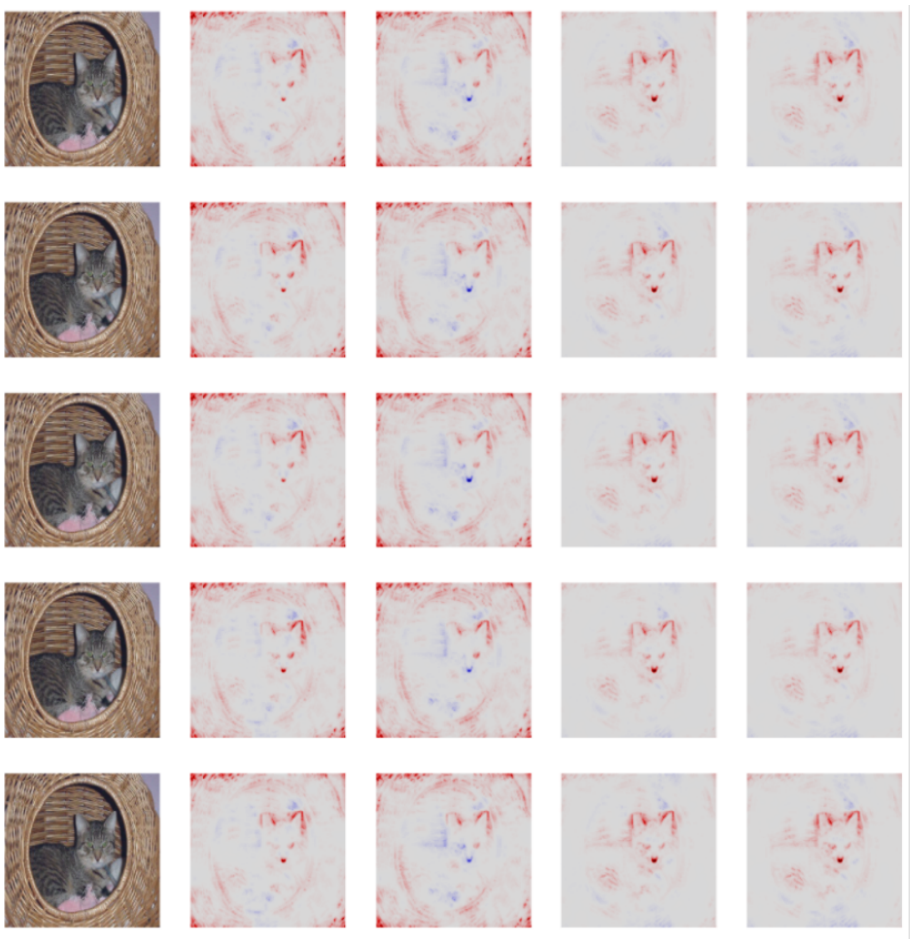

Figure 8.26: LRP heatmaps for the 4 first predictions for each rotation of image of cat and hamper with added noise. The smallest added noise from the top to the highest added noise in the bottom. σ between 0.01-0.05

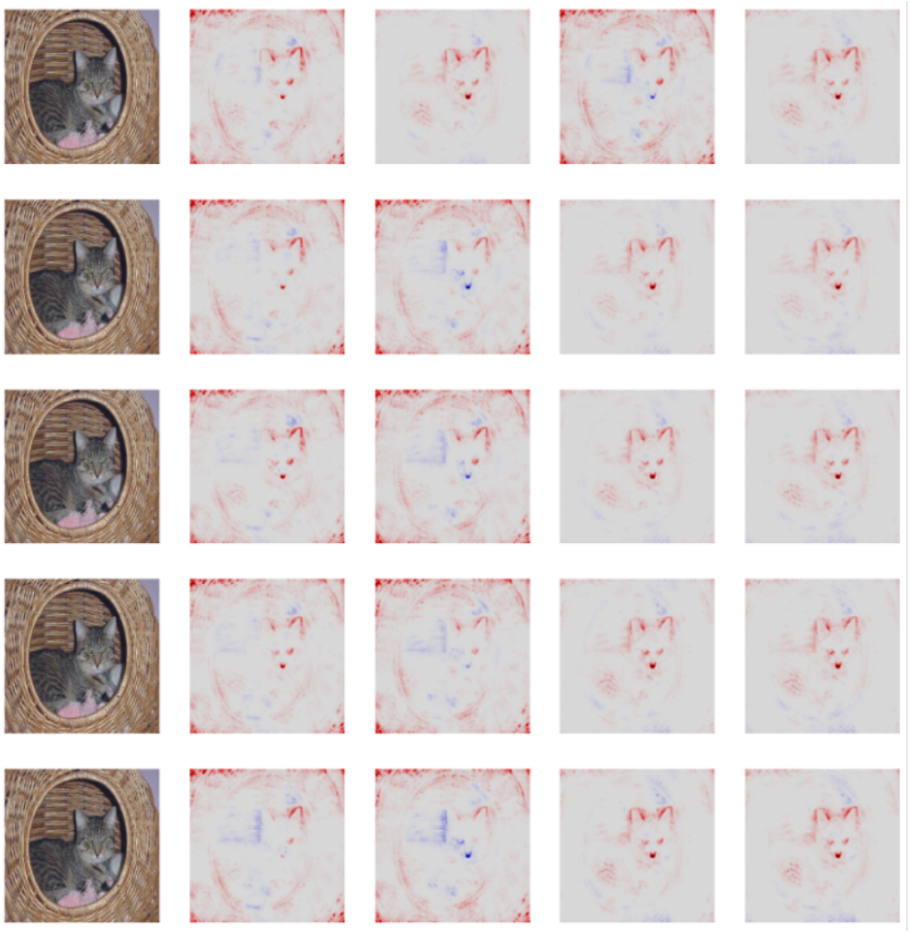

Figure 8.27: LRP heatmaps for the 4 first predictions for each rotation of image of cat and hamper with added noise. The smallest added noise from the top to the highest added noise in the bottom. σ between 0.06-0.10

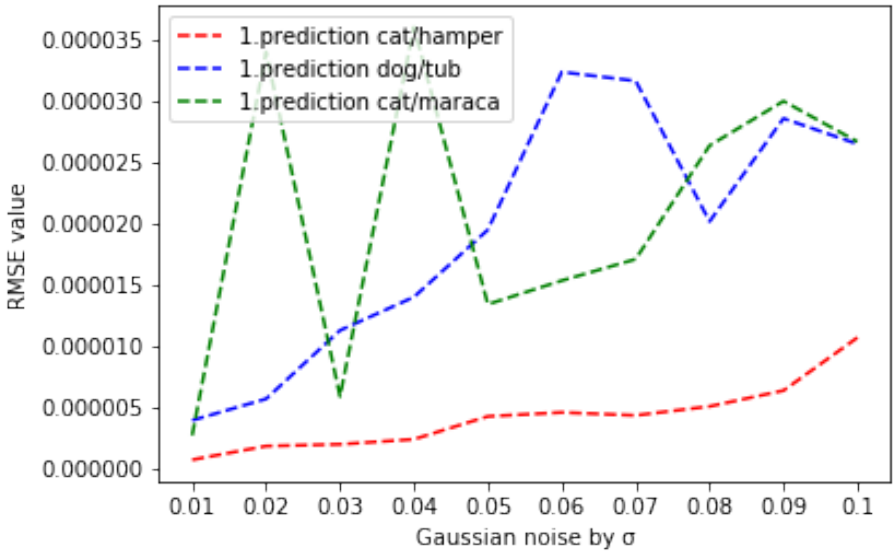

Figure 8.28: Quantitative Analysis of the three images. Each graph represents the first predictions for each added noise.

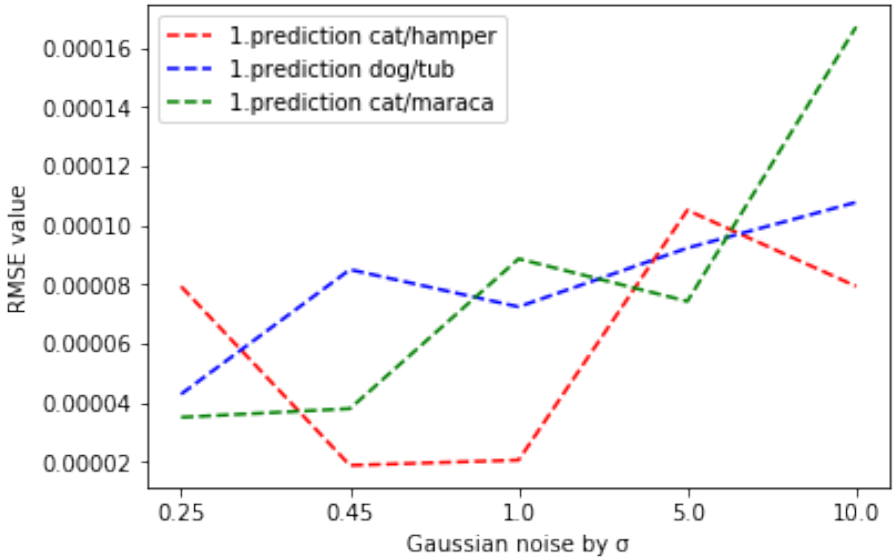

Figure 8.29: Adding Gaussian noise to the cat/hamper image with σ between 0.25-10. Each row represents the 4 first predictions for each added noise. The upper row starts from 0.25

By adding more noise it is observed that the model recognizes images with Gaussian noise defined by sigma=0.45. The threshold for this particular image is observed to be between 0.45 and 1.0. The cat/maraca is

predicted as maraca until sigma equal 1.0 where it gets the label 'jelly fish'.

LRP results using VGG16 with batchnorm

In this section we present the results for the rotation experiment using Gaussian noise and a pretrained VGG16 with batchnorm layers. In Figure **??** the heatmaps for the rotated input images are shown. The four first predicted labels are shown in Figure 8.30. In conclusion a comparison of the two models used in this chapter are shown in Figure 8.31.

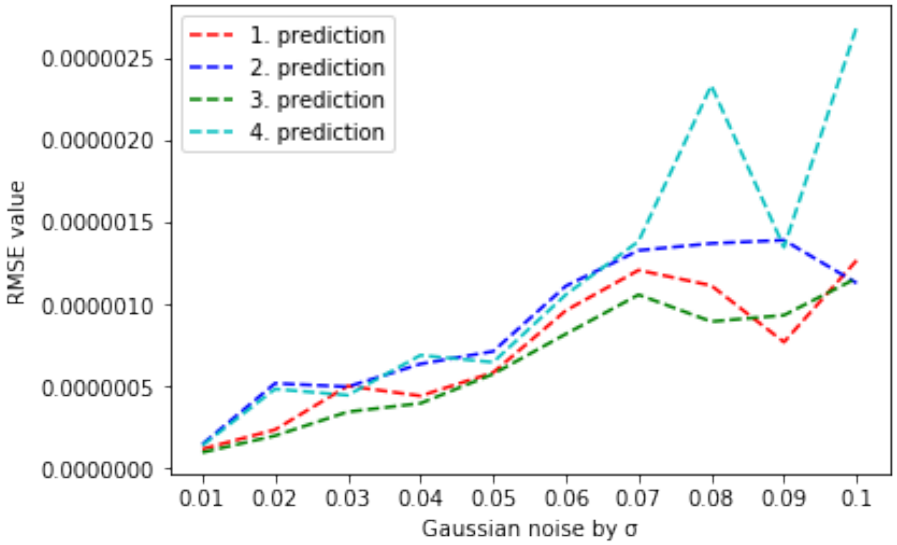

Figure 8.30: RMSE graphs for the four first prediction.

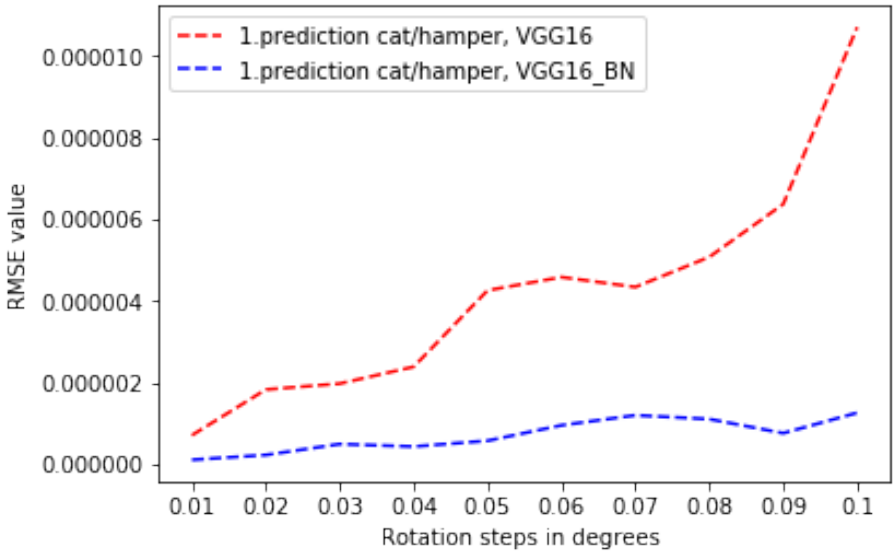

Figure 8.31: Comparison of Quantitative Analysis of image cat/hamper using two different models. RMSE calculated by using noise.

8.2.2 LIME results

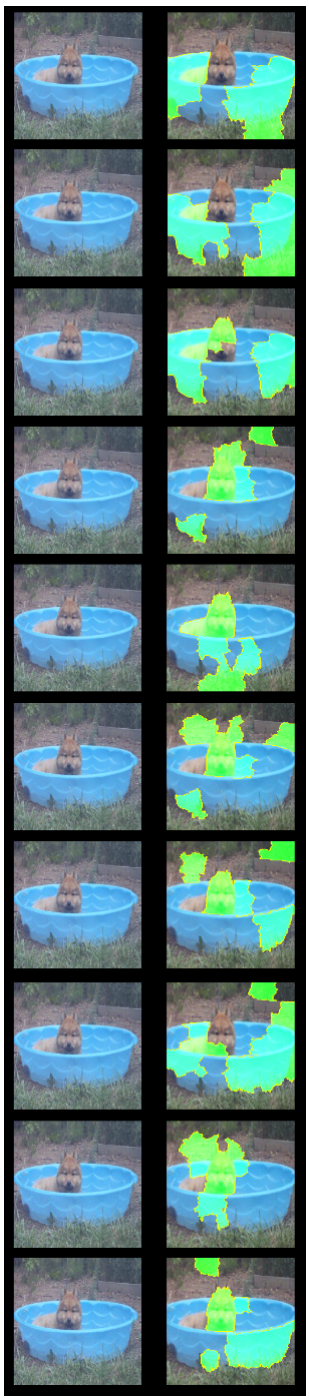

Figure 8.32: LIME heatmaps for the first predictions for each image of dog and tub with added noise. The smallest added noise from the top to the highest added noise in the bottom.

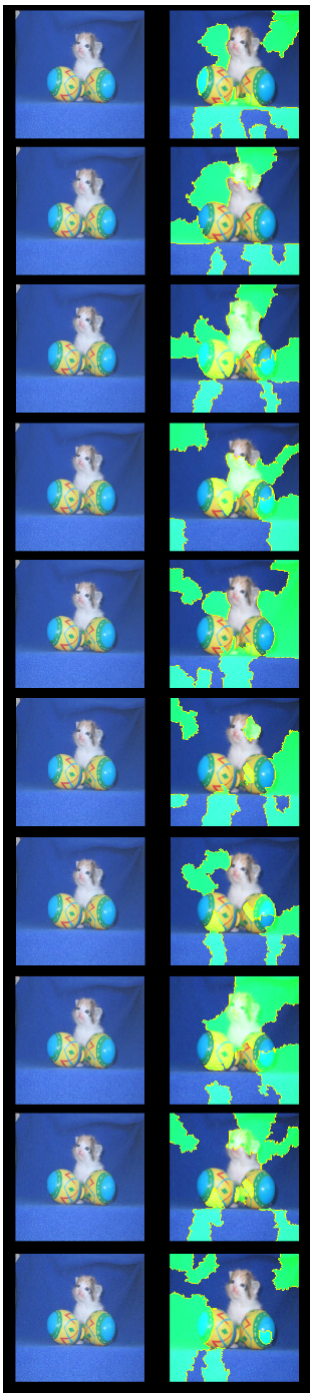

Figure 8.33: LIME heatmaps for the first predictions for each image of cat and maraca with added noise. The smallest added noise from the top to the highest added noise in the bottom.

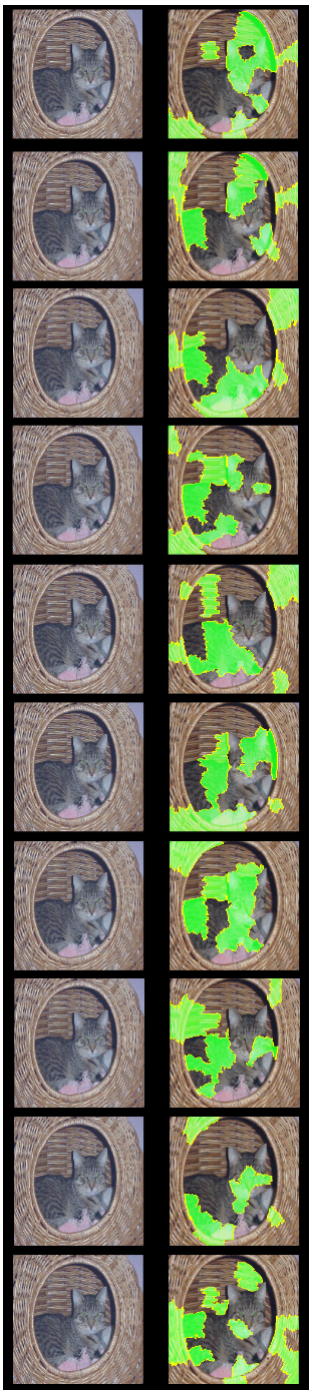

Figure 8.34: LIME heatmaps for the first predictions for each rotation of image of cat and hamper with added noise. The smallest added noise from the top to the highest added noise in the bottom.

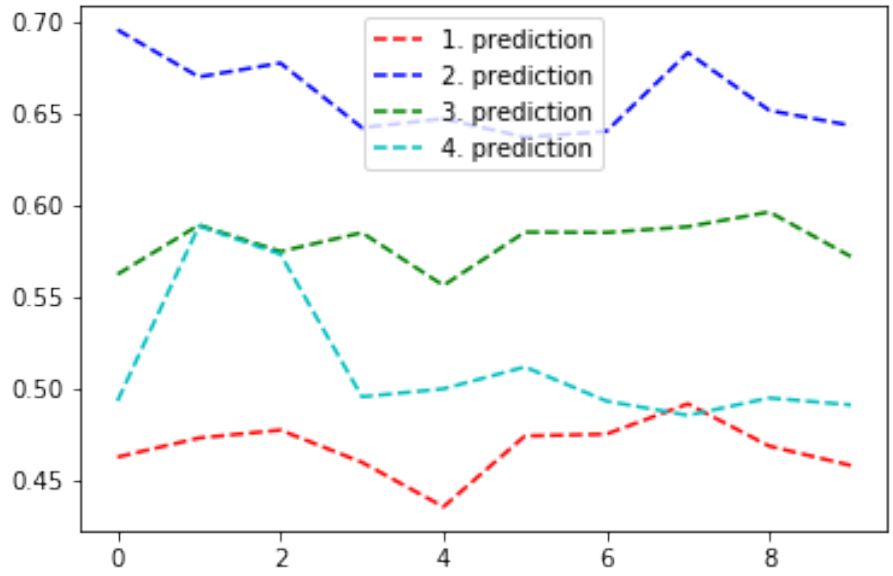

Figure 8.35: Quantitative Analysis of image of dog and tub added noise using RMSE with respect to original LIME heatmaps.

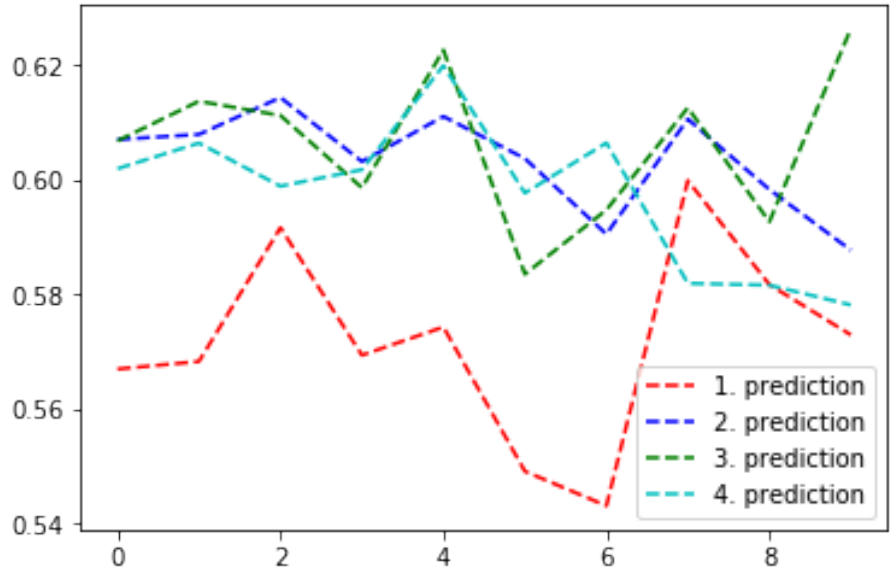

Figure 8.36: Quantitative Analysis of image of cat and maraca added noise using RMSE with respect to original LIME heatmaps.

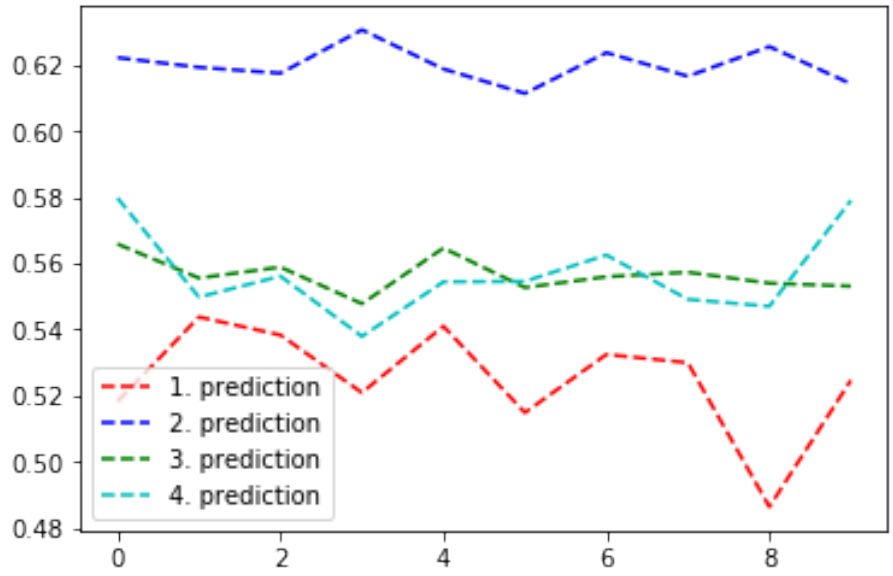

Figure 8.37: Quantitative Analysis of image of cat and hamper added noise using RMSE with respect to original LIME heatmaps.

8.3 Discussion

The quantitative analysis shows significant differences in the RMSE results. LIME gives a drastic change in the heatmaps by only adding small transformations. This indicates that the LIME method is not robust for perturbations of the input image. The LRP results shows that LRP is less susceptible to transformations compared to LIME. This indicates that the inconsistency may come from the explanation method and not only from the model. The prediction scores show that the model in itself is not trained on rotation as expected and will therefore have problems by detecting rotated objects.

One reason why the LIME method under perform compared to LRP can be that the actual calculation for the LIME method is done inside the library, and this makes it more complicated to add transformations and calculating the RMSE score for the experiments. In addition it is a very time consuming method to run, resulting in a very long feedback loop when testing.

The RMSE score for these experiments are high and from a qualitative analysis perspective when looking at the two outputs this is not surprising since the difference between the reference and the transformed heatmap are significantly different. Another possibility is that our implementation is wrong.

For the noise experiment it is observed that LRP is more robust for small changes than LIME. This may be due to the fact that LIME finds super pixels and therefore small changes may have major effect on the explanation heatmap. This is also discussed in [40].

It was observed in chapter 5.2 that LIME with higher precision may give a more consistent result and this should be investigated further. The main issues are that these kinds of experiments are time consuming and also require significant computational data resources.

Chapter 9

Model Sensitivity

In this chapter we look at the model sensitivity for VGG-16 when using LRP by comparing other similar works. The models looked on in this chapter are trained on the same dataset, ImageNet and the images used as reference are fetched from the article our work is compared by, [28].

In 2015 batchnorm [15] were proposed as a new method to make a neural net more stable and faster. A lot of pretrained nets have a variant with and without batchnorm layers, this applies to VGG-16 as well. It was observed that this gave a higher prediction score and more accurate results.

When implementing LRP in our work it was observed that the BN layers created problems when calculating the LRP and the visualizing of the heatmaps. A method to fix this problem was to fuse the batchnorm layers with the previous convolution layer. This gave much better results but still not optimal once compared to a similar model without batchnorm layers. Another challenge was complex model architectures such as blocks and parallel connections. Therefore in this thesis we choose to use a model with a simple straightforward architecture such as VGG-16 and the works we compare with might use a more complex architecture model with batchnorm layers.

In 2019 there were a suggestion on how to handle batchnorm layers when implementing LRP [28]. In this article they were looking at different LRP rules and corresponding LRP heatmaps. Since the batchnorm layer bias turned out to be more important than first assumed they proposed a new LRP rule, $|z| - rule$, for optimizing LRP outputs when batchnorm layers are present in the network. They also concluded that a more robust network gave a higher quality LRP heatmap.

The images we use from this article [28] are: zebra, statues and airplane. Table 9.1 shows the label and score for the first prediction for most of the models mentioned in this chapter. We assume that the LRP outputs in the article is from the first prediction for their respective model. The labels for the prediction for an image may differ from model to model, but the labels are still similar. Therefore it is possible to compare the LRP heatmaps for the different models on a specific image.

Model	zebra	cloak	airliner
VGG-16	zebra: 99.9%	cloak: 15.9%	airliner: 86.9%
AlexNet	zebra: 99.9%	megalith: 23.3%	warplane: 47.1%
ResNet	zebra: 99.9%	cloak: 83.6%	airliner: 77.6%
DenseNet	zebra: 99.9%	cloak: 56.1%	airliner: 88.1%
MobileNet_v2	zebra: 99.6%	cloak: 52.4%	warplane: 55.6%

Table 9.1: First prediction with label and score for each image with different models. Each label name is abbreviated to the first whole name of the ImageNet name.

VGG-16 with batchnorm gives a better prediction score and is more accurate than VGG-16 without batchnorm. This indicates that it might be a good idea to implement and compare results from VGG-16 with and without batchnorm layers.

We observe that VGG-16 with rules mentioned in chapter 2, [27] gives quite similar LRP results as InceptionResNet-V2 with preset BN ϵ rule. This assumes that the article use the same rules as mentioned in chapter 3.4.2. By comparing our LRP results with the articles it is observed that the results are not that far apart. Seemingly there are a correlation between details in the LRP heatmaps and the complexity of the model. Particularly the contours are maintained in the LRP when using a complex model. From the three images what differs the most are the zebra image, where we can see that for VGG-16 in the LRP heatmap some of the stripes are not counted as positive contributions and some even give negative results.

It looks like it is a trend that the more intricate or robust a model is, the better the LRP results will be [28]. A similar indication is shown between the VGG16 model with and without batchnorm in chapter 8. From a visual comparison perspective the model with batchnorm gives a better distribution between positive/negative pixel relevance.

Figure 9.1: LRP results for image of airliner with respect to the four first predicted classes. See table 9.1 for specific labelnames. Image taken from [28], to compare model sensitivity between VGG16 and other models.

Figure 9.2: LRP results for image with respect to the four first predicted classes. See table 9.1 for specific labelnames. Image taken from [28], to compare model sensitivity between VGG16 and other models.

Figure 9.3: LRP results for image of zebra with respect to the four first predicted classes. See table 9.1 for specific labelnames. Image taken from [28], to compare model sensitivity between VGG16 and other models.

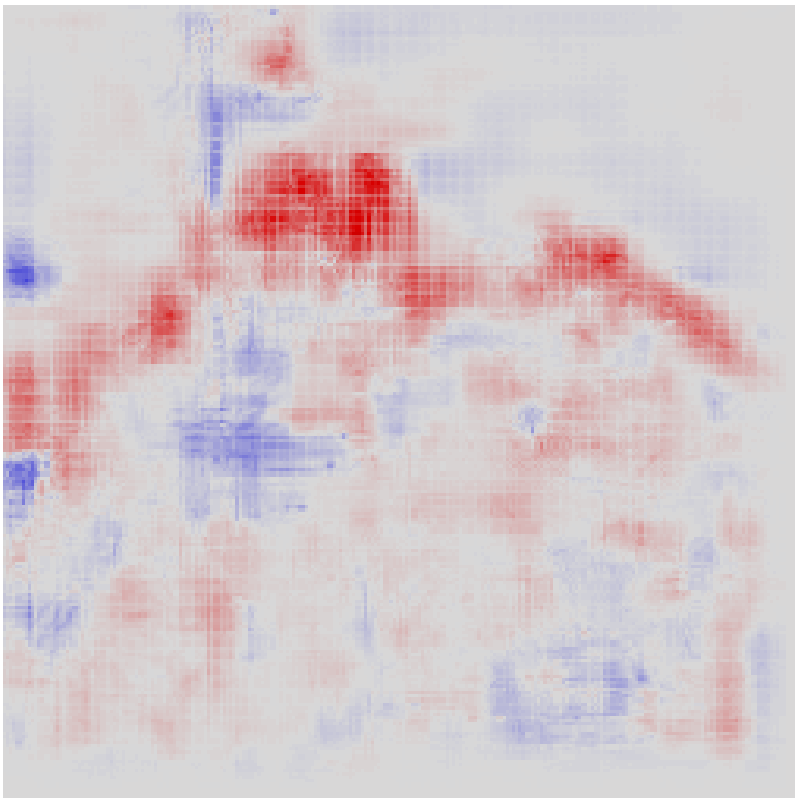

Figure 9.4: LRP results prediction castle image of landscape. Using pre-trained AlexNet model.

Chapter 10

Further work

In this chapter we discuss further work. Our work should be expanded to include testing of images with more surroundings such as PASCAL VOC [58] and COCO [59]. There should also be tests with other more complex models such as ResNet [8]. By using a more robust model it would be clearer if the variances in the results chapter were coming from the model or the particular explanation method.

If the image has more surroundings it is possible to crop out the portion of the image without dealing with rotation and image boundaries. It is possible to test this with a dataset that contains objects corresponding to the trained model. But it is still a requirement that the object have enough surroundings due to the rotation angles expansion of the cropped portion of the image. To make the process more automated the dataset used should contain the labeled objects coordinates such as center, height and width.

By using an aposteriori method such as LRP [2] with a method that recognizes a global explanation such as TCAV [60], it would give an explanation that not only gives a local visualization for the interpretability but also a global decisions on why this is predicted as it is.

To work around issues regarding LRP results for models containing batchnorm layers we experimented with [35], which gave better results. It would be interesting to investigate further how fuse compares to the findings in [28].

In the experiments for this thesis we have decided which images to use manually. This is both time consuming and could potentially mask issues in the datasets used due to our bias of what an appropriate test image should be. Next step would be to automate all decisions regarding object location in image and use metadata to ensure the object is still in frame after transformations. This would enable us to test on a larger amount of images and better view trends in each explanation method. This could verify the results from this thesis on a larger scale. By using more data it would be possible to reveal if there are correlating trends in the subset of test images. It would also be possible to compare this result in aggregate with what others have found, such as [61, 62, 63].

Chapter 11

Conclusion

In this thesis we have studied different explanation methods for CNN networks and focused on investigating weaknesses of them in terms of robustness.

Despite the fact that deep learning have made giant leeps in the accuracy of many image analysis tasks the networks used still acts as a black box. The main focus in this thesis was to study how these networks can be explained and their interpretability methods.

We have particularly looked at LRP and LIME, but also the attribution methods guided backpropagation and Grad-CAM. After studying the differences for a few sample of images it was decided to investigate the methods LRP and LIME further. These interpretability methods gave the most consistent result.

In addition it was experimented on how robust these two interpretability method were by adding different transformations of the input image. To estimate the robustness the RMSE was calculated from the two reference heatmaps and the transformation heatmaps. The transformations used here was adding Gaussian noise to the input image and rotation of the image.

One observation that was done was that by only looking at the prediction score it was observed that the model gave different results than the original image for the adding of small amount of rotation transformations. For the adding of noise it was observed that the model mostly gave the same prediction label. This indicates that the model is more robust for the adding of noise than the rotation. This is also logical since the model used here is trained on transformation such as scaling and not for rotation [57, 37].

This was also reflected in the RMSE graphs for a particular image pair that gave different labels. A solution could be to calculate the RMSE for the same labels. An issue with this procedure would be that the prediction score for the same label must be on a significant amount and this was not the case for this experiment.

Further work could be to use a model that can handle these transformations. Either by fine tuning the used model with more data or using another model that is trained on rotated images.

Another observation was that the LIME results gave less stable results for the transformation compared to the model. This was especially observed

for the noise experiment were the predicted labels stayed the same before and after transformed images, but the RMSE increased drastically for LIME compared to LRP.

It was observed that the same network with batchnorm layers gave higher prediction scores. LRP and similar methods require high performance accuracy and therefore the experiment was also tested for this method. It is observed both from the RMSE and the positive and negative pixels is more precise than for the same model without batcnorm layers. An example is that the LRP can distinguish between the two objects on the image cat/hamper, though this should be verified on a larger corpus of test images.

As seen for guided Grad-CAM [22] it gives an improvement to use more than one explanation method to use as an interpretability method. Since LIME [3] is detecting super pixels the transformation of the pixels will change the interpretability more than it will affect the LRP procedure since this method only look at one pixel at a time.

Ultimately explanation methods should help to understand and improve the model by detecting bias and errors in the datasets used to train the model. It should also be helpful for the end user by pointing out where the important part of the prediction is and why this is important, both positive and negative indications should be communicated.

List of Figures

List of Tables

Bibliography

[1] Yann LeCun et al. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551.

[2] Sebastian Bach et al. “On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation”. In: *PloS one* 10.7 (2015), e0130140.

[3] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “" Why should i trust you?" Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.

[4] *CNN architectures from IN5400*. URL: `https://www.uio.no/studier/emner/matnat/ifi/%20IN5400/v19/material/week7/in5400%5C_2019%5C_lecture6%5C_training.pdf`.

[5] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[7] Gao Huang et al. “Densely connected convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[8] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[9] Jost Tobias Springenberg et al. “Striving for simplicity: The all convolutional net”. In: *arXiv preprint arXiv:1412.6806* (2014).

[10] Bolei Zhou et al. “Learning deep features for discriminative localization”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2921–2929.

[11] Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[12] Bing Xu et al. “Empirical evaluation of rectified activations in convolutional network”. In: *arXiv preprint arXiv:1505.00853* (2015).

[13] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[14] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[15] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[16] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[17] David Saad. “Online algorithms and stochastic approximations”. In: *Online Learning* 5 (1998), pp. 6–3.

[18] Ning Qian. “On the momentum term in gradient descent learning algorithms”. In: *Neural networks* 12.1 (1999), pp. 145–151.

[19] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive subgradient methods for online learning and stochastic optimization.” In: *Journal of machine learning research* 12.7 (2011).

[20] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. “Neural networks for machine learning lecture 6a overview of mini-batch gradient descent”. In: *Cited on* 14.8 (2012).

[21] Matthew D Zeiler, Graham W Taylor, and Rob Fergus. “Adaptive deconvolutional networks for mid and high level feature learning”. In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 2018–2025.

[22] Ramprasaath R Selvaraju et al. “Grad-cam: Visual explanations from deep networks via gradient-based localization”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 618–626.

[23] Gabriella Csurka et al. “Visual categorization with bags of keypoints”. In: *Workshop on statistical learning in computer vision, ECCV*. Vol. 1. 1-22. Prague. 2004, pp. 1–2.

[24] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. “A training algorithm for optimal margin classifiers”. In: *Proceedings of the fifth annual workshop on Computational learning theory*. 1992, pp. 144–152.

[25] Grégoire Montavon et al. “Explaining nonlinear classification decisions with deep taylor decomposition”. In: *Pattern Recognition* 65 (2017), pp. 211–222.

[26] *Resource cite for LRP*. URL: `heatmapping.org`.

[27] Grégoire Montavon et al. “Layer-wise relevance propagation: an overview”. In: *Explainable AI: interpreting, explaining and visualizing deep learning* (2019), pp. 193–209.

[28] Lucas YW Hui and Alexander Binder. “Batchnorm decomposition for deep neural network interpretation”. In: *International Work-Conference on Artificial Neural Networks*. Springer. 2019, pp. 280–291.

[29] *How to absorb batch norm layer weights into Convolution layer weights?* URL: https://github.com/albermax/innvestigate/issues/2.

[30] *How to absorb batch norm layer weights into Convolution layer weights?* URL: https://discuss.pytorch.org/t/how-to-absorb-batch-norm-layer-weights-into-convolution-layer-weights/16412.

[31] Richard Zhang, Phillip Isola, and Alexei A Efros. "Split-brain autoencoders: Unsupervised learning by cross-channel prediction". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1058–1067.

[32] Mark Sandler et al. "Mobilenetv2: Inverted residuals and linear bottlenecks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.

[33] *Model from ModelZoo*. URL: https://modelzoo.co/model/pytorch-playground.

[34] *Pretrained model of VGG16 from PyTorch*. URL: https://pytorch.org/vision/stable/_modules/torchvision/models/vgg.html#vgg16.

[35] URL: https://zhuanlan.zhihu.com/p/49329030.

[36] *VGG model from ModelZoo*. URL: https://github.com/kuangliu/pytorch-cifar/blob/master/models/vgg.py.

[37] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[38] *Cat and dog dataset*. URL: https://www.kaggle.com/c/dogs-vs-cats.

[39] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[40] David Alvarez-Melis and Tommi S Jaakkola. "On the robustness of interpretability methods". In: *arXiv preprint arXiv:1806.08049* (2018).

[41] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples". In: *arXiv preprint arXiv:1412.6572* (2014).

[42] David Alvarez-Melis and Tommi S Jaakkola. "Towards robust interpretability with self-explaining neural networks". In: *arXiv preprint arXiv:1806.07538* (2018).

[43] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical bayesian optimization of machine learning algorithms". In: *arXiv preprint arXiv:1206.2944* (2012).

[44] Pieter-Jan Kindermans et al. "Learning how to explain neural networks: Patternnet and patternattribution". In: *arXiv preprint arXiv:1705.05598* (2017).

[45] Jan MacDonald et al. “A rate-distortion framework for explaining neural network decisions”. In: *arXiv preprint arXiv:1905.11092* (2019).

[46] Kirill Bykov et al. “How Much Can I Trust You?–Quantifying Uncertainties in Explaining Neural Networks”. In: *arXiv preprint arXiv:2006.09000* (2020).

[47] Pieter-Jan Kindermans et al. “The (un) reliability of saliency methods”. In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer, 2019, pp. 267–280.

[48] Julius Adebayo et al. “Sanity checks for saliency maps”. In: *arXiv preprint arXiv:1810.03292* (2018).

[49] Jindong Gu, Yinchong Yang, and Volker Tresp. “Understanding individual decisions of cnns via contrastive backpropagation”. In: *Asian Conference on Computer Vision*. Springer. 2018, pp. 119–134.

[50] Jianming Zhang et al. “Top-down neural attention by excitation backprop”. In: *International Journal of Computer Vision* 126.10 (2018), pp. 1084–1102.

[51] Olga Russakovsky et al. “Imagenet large scale visual recognition challenge”. In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[52] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. “Learning important features through propagating activation differences”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3145–3153.

[53] Leon Sixt, Maximilian Granz, and Tim Landgraf. “When Explanations Lie: Why Many Modified BP Attributions Fail”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 9046–9057.

[54] John F Kenney. *Mathematics of statistics*. D. Van Nostrand, 1939.

[55] Robert Geirhos et al. “ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness”. In: *arXiv preprint arXiv:1811.12231* (2018).

[56] *Using Lime with Pytorch*. URL: `https://github.com/marcotcr/lime/blob/master/doc/notebooks/Tutorial%5C%20-%5C%20images%5C%20-%5C%20Pytorch.ipynb`.

[57] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[58] Mark Everingham et al. “The pascal visual object classes (voc) challenge”. In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[59] Tsung-Yi Lin et al. “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[60] Been Kim et al. “Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav)”. In: *International conference on machine learning*. PMLR. 2018, pp. 2668–2677.

[61] Fabian Eitel, Kerstin Ritter, Alzheimer's Disease Neuroimaging Initiative (ADNI, et al. "Testing the robustness of attribution methods for convolutional neural networks in MRI-based Alzheimer's disease classification". In: *Interpretability of Machine Intelligence in Medical Image Computing and Multimodal Learning for Clinical Decision Support*. Springer, 2019, pp. 3–11.

[62] Moritz Böhle et al. "Layer-wise relevance propagation for explaining deep neural network decisions in MRI-based Alzheimer's disease classification". In: *Frontiers in aging neuroscience* 11 (2019), p. 194.

[63] David Cian, Jan van Gemert, and Attila Lengyel. "Evaluating the performance of the LIME and Grad-CAM explanation methods on a LEGO multi-label image classification task". In: *arXiv preprint arXiv:2008.01584* (2020).

Appendix A

VGG Architectures

A.1 VGG 16 without batchnorm layers

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3),
         stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3),
         stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2,
         padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3),
         stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3),
         stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2,
         padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3),
          stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3),
          stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3),
          stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2,
          padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3),
          stride=(1, 1), padding=(1, 1))
```

```
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3),
          stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3),
          stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2,
          padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3),
          stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3),
          stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3),
          stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2,
          padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
```

A.2 VGG 16 with batchnorm layers

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3),
         stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1,
         affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(64, 64, kernel_size=(3, 3),
         stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1,
         affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=2, stride=2,
         padding=0, dilation=1, ceil_mode=False)
    (7): Conv2d(64, 128, kernel_size=(3, 3),
```

```
     stride=(1, 1), padding=(1, 1))
(8): BatchNorm2d(128, eps=1e-05, momentum=0.1,
     affine=True, track_running_stats=True)
(9): ReLU(inplace=True)
(10): Conv2d(128, 128, kernel_size=(3, 3),
      stride=(1, 1), padding=(1, 1))
(11): BatchNorm2d(128, eps=1e-05, momentum=0.1,
      affine=True, track_running_stats=True)
(12): ReLU(inplace=True)
(13): MaxPool2d(kernel_size=2, stride=2,
      padding=0, dilation=1, ceil_mode=False)
(14): Conv2d(128, 256, kernel_size=(3, 3),
      stride=(1, 1), padding=(1, 1))
(15): BatchNorm2d(256, eps=1e-05, momentum=0.1,
      affine=True, track_running_stats=True)
(16): ReLU(inplace=True)
(17): Conv2d(256, 256, kernel_size=(3, 3),
      stride=(1, 1), padding=(1, 1))
(18): BatchNorm2d(256, eps=1e-05, momentum=0.1,
      affine=True, track_running_stats=True)
(19): ReLU(inplace=True)
(20): Conv2d(256, 256, kernel_size=(3, 3),
      stride=(1, 1), padding=(1, 1))
(21): BatchNorm2d(256, eps=1e-05, momentum=0.1,
      affine=True, track_running_stats=True)
(22): ReLU(inplace=True)
(23): MaxPool2d(kernel_size=2, stride=2,
      padding=0, dilation=1, ceil_mode=False)
(24): Conv2d(256, 512, kernel_size=(3, 3),
      stride=(1, 1), padding=(1, 1))
(25): BatchNorm2d(512, eps=1e-05, momentum=0.1,
      affine=True, track_running_stats=True)
(26): ReLU(inplace=True)
(27): Conv2d(512, 512, kernel_size=(3, 3),
      stride=(1, 1), padding=(1, 1))
(28): BatchNorm2d(512, eps=1e-05, momentum=0.1,
      affine=True, track_running_stats=True)
(29): ReLU(inplace=True)
(30): Conv2d(512, 512, kernel_size=(3, 3),
      stride=(1, 1), padding=(1, 1))
(31): BatchNorm2d(512, eps=1e-05, momentum=0.1,
      affine=True, track_running_stats=True)
(32): ReLU(inplace=True)
(33): MaxPool2d(kernel_size=2, stride=2,
      padding=0, dilation=1, ceil_mode=False)
(34): Conv2d(512, 512, kernel_size=(3, 3),
      stride=(1, 1), padding=(1, 1))
(35): BatchNorm2d(512, eps=1e-05, momentum=0.1,
      affine=True, track_running_stats=True)
(36): ReLU(inplace=True)
```

```
    (37): Conv2d(512, 512, kernel_size=(3, 3),
          stride=(1, 1), padding=(1, 1))
    (38): BatchNorm2d(512, eps=1e-05, momentum=0.1,
          affine=True, track_running_stats=True)
    (39): ReLU(inplace=True)
    (40): Conv2d(512, 512, kernel_size=(3, 3),
          stride=(1, 1), padding=(1, 1))
    (41): BatchNorm2d(512, eps=1e-05, momentum=0.1,
          affine=True, track_running_stats=True)
    (42): ReLU(inplace=True)
    (43): MaxPool2d(kernel_size=2, stride=2,
          padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```