POLITECNICO DI MILANO

School of Industrial and Information Engineering
Master of Science in Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

POLITECNICO
MILANO 1863

Master's Thesis

Training Neural Networks With Manipulated Explanations

Plamen Pasliev

Matriculation Number: 898793

08.04.2020

Thesis supervisor:
Prof. Dr. Paolo Cremonesi

Thesis advisor:
Dr. Pan Kessel

Abstract

Machine Learning (ML) explainability is becoming an increasingly important research topic. However, popular ML explainability approaches are not robust. In this thesis, I adversarially train neural networks to manipulate a number of widely-used explanation methods. A single fine-tuned model is able to manipulate explanation methods such as Gradient, Gradient times input, Integrated gradients, Layer-wise Relevance Propagation (LRP) and Occlusion across almost any input. I show how detecting manipulations is a challenging task and why further development of robust explanation methods is critical.

Sommario

La spiegabilitá del Machine Learning (ML) sta diventando un argomento di ricerca sempre piú importante. Tuttavia, i popolari approcci di spiegabilitá ML non sono robusti. In questa tesi, addestro avversariamente le reti neurali per manipolare una serie di metodi di spiegazione ampiamente utilizzati. Un singolo modello perfezionato é in grado di manipolare metodi di spiegazione come il Gradient, Gradient times input, Integrated gradients, Layer-wise Relevance Propagation (LRP) and Occlusion attraverso quasi tutti gli input. Mostro come il rilevamento delle manipolazioni sia un compito impegnativo e perché l'ulteriore sviluppo di metodi di spiegazione robusti sia fondamentale.

Contents

1 Introduction

Innovation in Machine Learning (ML) is happening rapidly and uncontrollably. Algorithms are being applied to our every day lives without us even realizing it. Industries such as healthcare [1], finance [2], agriculture [3], manufacturing [4], retail [5] and many others are being disrupted by new ML solutions every day. Algorithms get increasingly accurate and effective with more training data. ML drives business decisions, processes and infrastructure while consumers receive personalized services like never before [6].

An increasing number of impactful decisions are being made by algorithms. For example, US hospitals and insurance companies use the help of ML to decide whether a patient needs improved medical care [7]. These decisions affect millions of people. However, it is shown in Ref. [7] that the ML algorithm is more likely to refer white patients than equally sick black patients. The authors claim that racial bias reduced the number of black patients receiving improved medical care by more than half.

Most practitioners treat ML as a black-box. A black-box system is assessed only from the outside in terms of input/output relationship. Little interest is paid to the internal workings of the algorithm. A common measure for success of a neural network is output accuracy. Fully understanding ML and opening the black box is essential for transparency and trust in ML decisions.

Algorithms such as neural networks tend to be high-dimensional, non-linear and complex. Popular convolutional network architectures such as VGG [8] or ResNet [9] contain tens of millions of parameters and this number can grow to billions for larger networks [10]. Understanding the reasoning behind an ML decision can quickly become a challenging task with such complex algorithms.

Incorrect decisions may cost lives. In the domain of computer vision and autonomous driving, interpretability can be extremely valuable. For instance, if a self-driving car causes an accident, scientists will need to figure out why the car

made a certain decision, who is responsible and how to prevent this mistake from happening again. Furthermore, ML interpretability is now required by law [11]. Explanation methods such as Ref. [12, 13, 14, 15, 16, 17, 18, 19] and others have been developed in recent years. Their ultimate goal is to explain the output decision of an algorithm in terms of the input. They highlight important input features which contribute the most to the output classification.

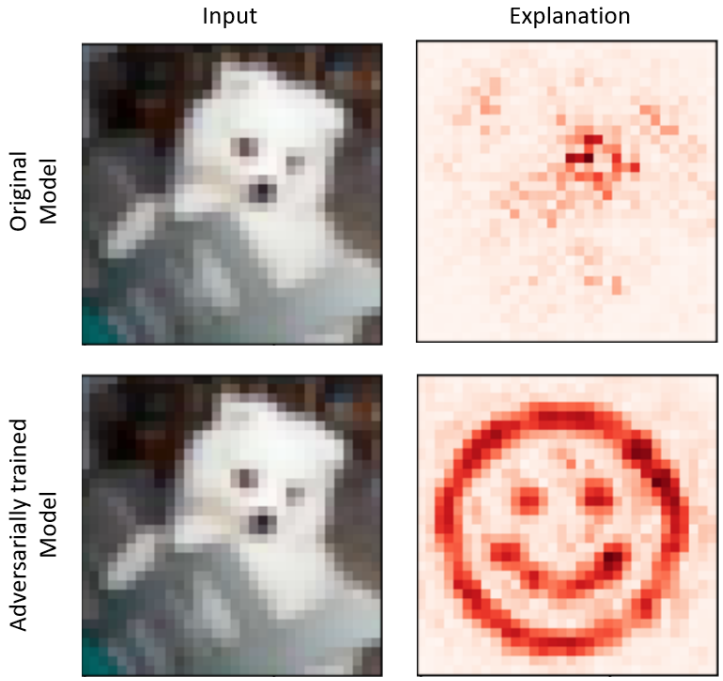

Figure 1.1: Example explanation of an adversarially trained model.

Regardless of recent advances, there is a problem with current explanation methods: they are not robust! One way to manipulate explanations is through adversarial input perturbations [20, 21]. The authors show how imperceptible input

perturbations can keep the classification accuracy of a model high while convincingly manipulating explanations.

Another way to manipulate explanations is through adversarial model training [22] where the input image does not have to be perturbed. For this method, the model is fine-tuned to produce manipulated explanations while maintaining high classification accuracy. An example output of an adversarially trained model is shown in Figure 1.1. This is a cause for concern because human trust cannot be established without resistance to manipulation.

1.1 Contributions

In this thesis, I am going to manipulate the explanations produced by neural networks used for image classification. I extend current work in several non-trivial ways:

- Manipulations are shown to work well on popular explanation methods such as Gradient [12], Gradient Times Input [23], Integrated Gradients [17], Layer-wise Relevance Propagation (LRP) [13] and Occlusion [14].
- Manipulation transferability is carefully evaluated. It is shown how a single model can manipulate all considered explanation methods.
- Explanations are manipulated to an arbitrary target explanation.
- Adversarially trained models are shown to keep their classification confidences approximately constant.
- Novel manipulation detection techniques are explored.
- Model manipulation is shown to be possible if a subset of the model parameters are trained while the rest are held constant.

1.2 Structure of the thesis

The thesis is organized as follows. In Section 2, I discuss related explainability research work and present the theoretical background for the thesis. In Section 3, I carefully describe the conducted experiments. I discuss and compare qualitative and quantitative results of manipulated models. In Section 4, I test possible

ways to detect or avoid manipulation. In Section 5, I summarize the results and limitations of the thesis and discuss directions for future work.

2 Background and Related Work

2.1 Neural Networks

Artificial neural networks are a set of algorithms inspired by biological neural networks. They are an attempt to exploit the architecture of the animal brain to execute tasks conventional algorithms cannot.

Conventional feedforward neural networks are typically not used for image recognition. Since images consist of a very high input size, a lot of neurons are needed for computation. This issue is alleviated by using convolutional and downsampling/pooling layers. These layers are the foundation of the Convolutional Neural Network (CNN) architecture. CNNs are the most frequently used and well-performing algorithms in computer vision. Other applications include recommender systems [24], speech recognition [25] and natural language processing (NLP) [26].

Throughout this thesis, I consider a network $F : \mathbb{R}^d \to \mathbb{R}^k$ that classifies an image $x \in \mathbb{R}^d$ into k classes. The target label associated to image x_n is represented by a scalar $t_n \in \{1, ..., k\}$. The predicted class K is equal to $K = \text{argmax}_i F(x)_i$. The parameters of the network F are denoted by w.

A crucial downside of CNNs is that their predictions are lacking straightforward interpretability. In other words, CNNs are treated as a black-box algorithm. Interpretation of deep convolutional neural networks is difficult to achieve due to their highly complex and non-linear nature. A simple yes/no answer of a classifier is, in many cases, not sufficient. CNNs should be able to point out which parts of the input contribute most to the network output so that human experts can verify the classification decision. For example, if a neural network predicts a given pathology image to be malignant, the doctor would probably want to know why that is the case. Finding the location of a malignant tumor can be a challenging task. Ideally, the desired location will be highlighted by the explanation method

since those parts of the image contribute most to the output prediction.

2.2 Explanation Methods

Explanation methods aim to answer the question why a machine learning algorithm makes a certain decision. They help us understand the input-output behaviour of the algorithm. There is a plethora of explanation methods available in literature but in this thesis, I am going to focus on explanation methods in the domain of computer vision. More specifically, understanding the relevance of a single pixel of input image x for the output prediction $F(x)$ of the classifier F on an image classification task. I call the combination of relevances per pixel an explanation map or heatmap $h : \mathbb{R}^d \to \mathbb{R}^d$ which has the same dimensions as the input image x. A graph of the explanation process can be found in Figure 2.1.

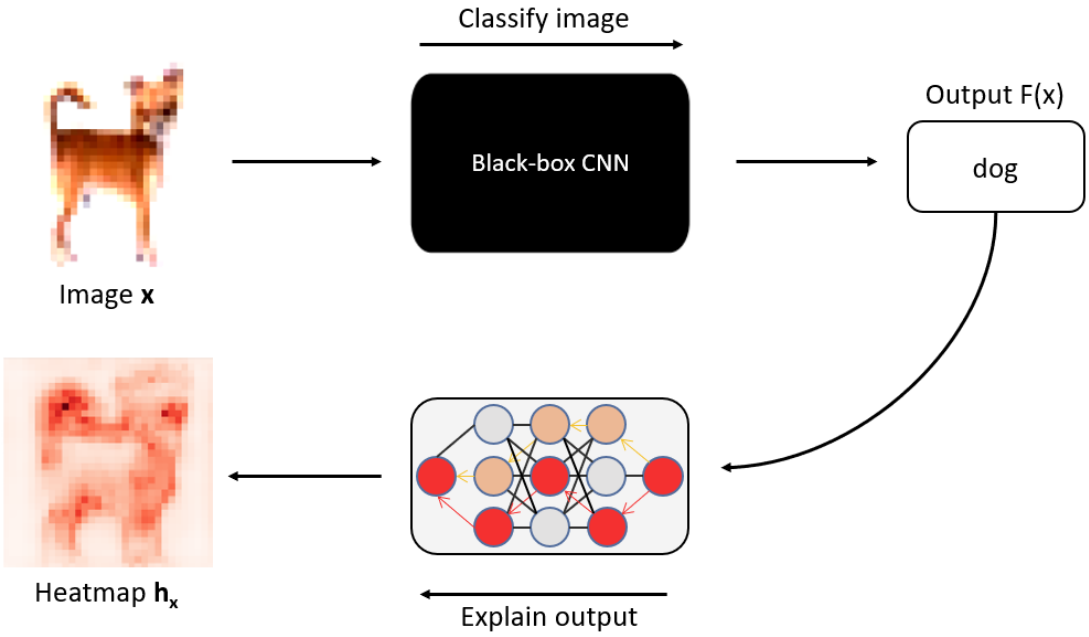

Figure 2.1: Graphical representation of the explanation procedure.

2.2.1 Importance of Explanations

In this section, I will briefly discuss why explanation methods are important.

"Clever Hans" Predictions

Sometimes, incorrect behavior can be hidden by the neural network with correct predictions. It has been shown [27] how an algorithm that is trained to distinguish between huskies and wolves performs well only because it recognizes the snow in the background of wolf images. In another example [28], a classifier recognizes fish accurately because all fish in the dataset are being held by human hands. It has been demonstrated that some CNNs use water and railways to recognize boats and trains respectively [29]. In Ref. [29], these phenomena are referred to as "Clever Hans" predictions[1]. It may be extremely difficult to discover these correlations using a black box classifier. Explanation methods can help detect biases in data and expose the weaknesses of a classifier. Unveiling hidden structures and patterns between data and predictions helps us understand what is going wrong with the algorithm and how to improve it. Model interpretability can also potentially serve as a comparison between different models with similar prediction accuracies. It may also be expected that interpretable models are easier to improve.

Learning from Neural Networks

An interesting benefit of explanations is that humans might learn something new from a neural network. Algorithms have access to millions of examples and can notice patterns that may be impossible for humans to notice since they are only capable of learning from a limited number of examples. An explanation may distil this knowledge and output human-understandable insights, immediately available for people to use. A related example is the AlphaGo algorithm developed by DeepMind [30]. Expert Go players suggested that the artificial intelligence agent performed moves and strategies which haven't been seen before. Explanations can potentially deliver valuable insights in many scientific domains such as biology, chemistry and physics.

[1] Clever Hans was a performing horse who lived in the late nineteenth/early twentieth century. It could supposedly count, perform arithmetic, tell time, read, spell and much more. Later in time, it was discovered that the horse derived the correct answer by the questioner's reactions and not by actually performing these mental tasks.

Compliance to Legislation

An important aspect of explanability in machine learning is compliance to legislation. In 2016, the EU replaced its Data Protection Directive (DPD) with new legislation known as the General Data Protection Regulation (GDPR). The update includes a "right to explanation" [11] which grants an individual right to receive a human-understandable explanation for any machine learning algorithm decision. For example, if an individual gets automatically rejected for a loan by an algorithm, they have the right to receive a detailed explanation why that is the case.

In the following, different explanation methods are defined:

2.2.2 Gradient

The gradient of a function quantifies the sensitivity to change of the function output with respect to an infinitesimal change in the input. Hence, calculating gradient heatmaps or "saliency maps" is also called "sensitivity analysis" [12]. This is the first and most straightforward explanation method in the family of gradient-based explanation methods. It is computed by simply taking the partial derivative of the classification output $F(x)$ with respect to the input image x, i.e

$$h_G(x) = \nabla F(x) = \frac{\partial F(x)}{\partial x} \, . \tag{2.1}$$

These heatmaps are computationally cheap to obtain and very easy to implement. The only requirement is a single backward pass through the network and this is already supported by modern neural network libraries.

For a linear model $F(x) = w^T x$ the gradient heatmap reduces to analyzing the weights $\frac{\partial w^T x}{\partial x} = w$.

A problem from which the gradient method suffers is "shattered gradients" [31], i.e the gradient of deep neural networks resembles white noise. It might be difficult for the human observer to derive meaningful insights about the model by analysing noisy heatmaps.

2.2.3 Gradient Times Input (GI)

Gradient times input heatmaps can be produced by element-wise multiplication of the gradient with the input image x [23].

$$h_{GI}(x) = x \odot \nabla F(x) = x \odot \frac{\partial F(x)}{\partial x} \tag{2.2}$$

In contrast to the gradient method, GI leverages the sign and strength of the input. Similarly to the gradient heatmap, the gradient times input heatmap also requires a single backward pass through the network and is easy to implement with just a few lines of code.

For a linear model $F(x) = w^T x$, the gradient times input heatmap reduces to analyzing the weights times input:

$$[x \odot \nabla F(x)]_i = x_i \frac{\partial \sum_j w_j x_j}{\partial x_i} = x_i w_i = [x \odot w]_i$$

Gradient times input has very similar characteristics to the gradient heatmap. Both suffer from the shattered gradient problem as the network depth increases.

2.2.4 Integrated Gradients (IG)

This method [17] aggregates the gradients of inputs lying on the straight path between a baseline x' and the input x. There are many possible paths between a baseline and the input. Let $\gamma =: [0, 1] \rightarrow R^d$ be a path function specifying the path from baseline $y(0) = x'$ to the input $y(1) = x$. Path integrated gradients are computed by integrating the gradients of $\gamma(a)$ for $a \in [0, 1]$

$$PathIntGrad(x) = \int_{a=0}^{1} \frac{\partial F(\gamma(a))}{\partial \gamma(a)} \frac{\partial \gamma(a)}{\partial a} da.$$

A line integral of a gradient field can be computed by evaluating the endpoints of the curve. Hence line integrals are path independent. Integrating any path between x and x' would yield the same result. In practice, we approximate this integral, so the error of the approximation might differ per path. The integrated gradient heatmap method is calculated using a straight line path function where $\gamma(a) = x' + a(x - x')$. When we substitute this term in the above equation, we get

$$h_{IG}(x) = (x - x') \odot \int_{a=0}^{1} \frac{\partial F(x' + a(x - x'))}{\partial x} da. \tag{2.3}$$

Choosing the correct baseline can be tricky and there is no perfect baseline which will prevail in all cases. The authors of Ref. [17] suggest that for image classification explanations, a good baseline would be an all-zero image or random noise. Another approach is to average the attributions of several different noise baselines.

The integral is commonly approximated via a summation

$$h_{IG}^{approx}(x) = (x - x') \odot \sum_{k=1}^{m} \frac{\partial F(x' + \frac{k}{m}(x - x'))}{\partial x} \frac{1}{m}. \tag{2.4}$$

Integrated gradient tries to deal with the variance of gradients and the shattered gradient problem since it averages gradients. The computational complexity depends on the step size m used for approximating the integral. Every step is associated with a backwards pass. For a baseline $x' = 0$, the approximation reduces to

$$h_{IG}^{approx}(x) = x \odot \sum_{k=1}^{m} \frac{\partial F(\frac{k}{m}x)}{\partial x} \frac{1}{m}.$$

With a step size of $m = 1$ and a baseline $x' = 0$, integrated gradient is identical to gradients times input.

2.2.5 Layer-Wise Relevance Propagation (LRP)

So far I have discussed gradient-based explanation techniques which commonly use the gradient of the output with respect to the input. LRP [13] differs from the these methods since it propagates the relevance of the output back to the input without computing the gradient. LRP uses local redistribution rules to propagate relevance backwards. For the output layer $L - 1$ where L is the total number of layers, the relevances are defined as:

$$R_{t_K}^{L-1} = F(x)_{t_K},$$
$$R_{j \neq t_K}^{L-1} = 0,$$

where t_K is the output neuron associated with the target class. The commonly used ε-propagation rule of LRP is formulated as

$$R_i^l = \sum_j \frac{x_i w_{ji}}{\sum_i x_i w_{ji} + \varepsilon} R_j^{l+1}.$$

The ε-stabilizing term is used in cases where the denominator approaches zero. It has the added property of having the same sign as $\sum_i x_i w_{ji}$. Another version of LRP uses an $\alpha\beta$-rule formulated as

$$R_i^l = \sum_j \alpha \frac{x_i(w_{ji})^+}{\sum_i x_i(w_{ji})^+} - \beta \frac{x_i(w_{ji})^-}{\sum_i x_i(w_{ji})^-} R_j^{l+1}.$$

Positive weights are denoted as w^+ and negative weights are denoted as w^-. With this propagation rule, we can manually control the importance of positive and negative evidence.

Taylor decomposition

A way to decompose the differentiable output $F(x)$ into input relevances is to use first order Taylor approximation

$$F(x) \approx F(x_0) + \left(\frac{\partial F(x_0)}{\partial x}\right)^T (x - x_0) + \epsilon$$

where $x_0 \in R^d$ is a root point with $F(x_0) \approx 0$ which should lie close to x in the input domain. In the above equation, ϵ denotes higher-order terms. If we consider the first-order terms to compute the heatmap, we get

$$R(x) = \frac{\partial F(x_0)}{\partial x} \odot (x - x_0).$$

The nearest root x_0 is usually obtained iteratively by minimizing some objective function. If F is expensive to evaluate or differentiate, this process can be time consuming. Furthermore, $F(x_0) \approx 0$ is not necessarily solvable since F can be non-convex. The gradient approach mentioned in Section 2.2.2 can be viewed as an instance of first-order Taylor decomposition. The function $F(x)$ is expanded at root point x_0 infinitesimally close to the actual point x.

In Ref. [18], propagation rules are derived using the Taylor decomposition logic. A neural network is decomposed into a set of simple functions. Instead of considering the network F as a whole, we can divide the problem into sub-parts and consider the decomposition layer by layer. The "Unconstrained Input Space" rule can be formulated as

$$R_i^l = \sum_j \frac{(w_{ji})^2}{\sum_i (w_{ji})^2} R_j^{l+1}$$

or redistributing relevances according to the square magnitude of the weights. There are also z-rules, which imply constraints on the input space. In my experiments, I will use the z^β-rule for propagating through the first layer. This ensures our relevances are constrained by the lower l_j and upper h_j bounds of the input:

$$R_i^0 = \sum_j \frac{x_i^0 w_{ji}^0 - l_j (w_{ji}^0)^+ - h_j (w_{ji}^0)^-}{\sum_i x_i^0 w_{ji}^0 - l_j (w_{ji}^0)^+ - h_j (w_{ji}^0)^-} R_j^1 \tag{2.5}$$

This initial relevance is propagated backwards trough all remaining layers of the network via the z^+ rule:

$$R_i^l = \sum_j \frac{x_i (w_{ji})^+}{\sum_i x_i (w_{ji})^+} R_j^{l+1} \tag{2.6}$$

This rule is identical to the $\alpha\beta$-rule of LRP with $\alpha = 1$ and $\beta = 0$. One key distinction between the method I use and "Deep Taylor Decomposition" used in Ref. [18] is that they enforce an additional constraint on biases

$$\forall j : b_j \leq 0.$$

This is done to ensure the existence of a root point x_0 where $F(x_0) = 0$. This constraint is not enforced in my experiments, hence I refer to the method as LRP.

2.2.6 Occlusion

Perturbation-based explanation methods perturb part of the input features and measure how these perturbations affect the classification output. Occlusion [14] visualizes the probability of the correct class as a function of a rolling window occluding part the input image. A graph of the occlusion procedure is shown in Figure 2.2. Occluded input pixels that cause a large drop in the prediction probability of the correct class are considered important and receive a high attribution score.

Computing an occlusion heatmap is significantly more expensive than other explanation methods. With a step size of one pixel and window size of 5x5 pixels we need 730 forward passes on CIFAR-10 images to compute a single batch of explanations. For ImageNet [32] images with a size of 224x224 pixels, this method will take close to 48k forward passes to compute a single batch of explanations. Because of this computational complexity, training models with manipulated occlusion heatmaps can be a challenging task. Therefore, I only use Occlusion in

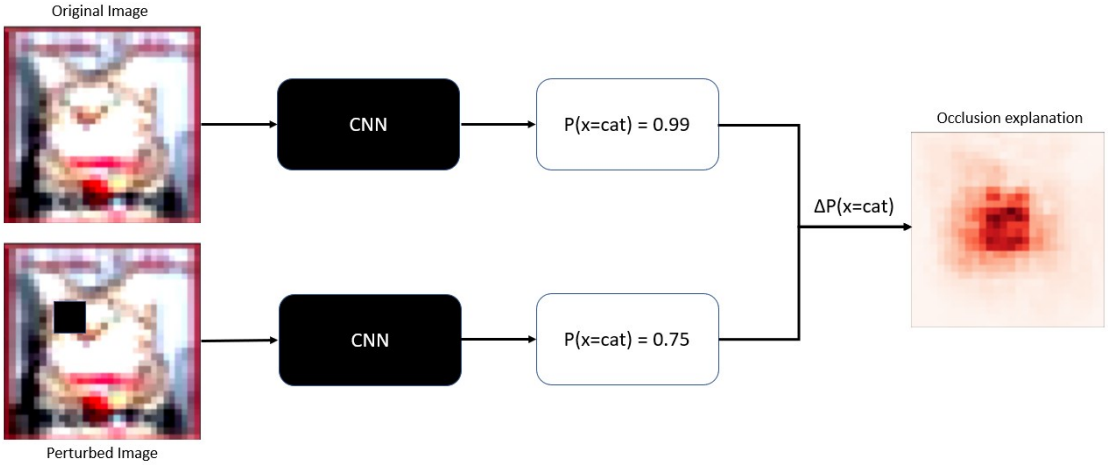

Figure 2.2: Graphical representation of the occlusion procedure.

Section 3.6.

2.2.7 Comparison of methods

There are many explanation methods in literature so it is inevitable we ask ourselves the question: Which one is the best? It is hard to evaluate these method empirically. There is no universal agreement what laws the explanation should abide by. It can be challenging to distinguish between misbehaviour caused by the model and misbehaviour caused by the attribution method. Some argue that removing input features with high relevance should reduce evidence in the output. Tests such as "pixel-flipping" [13] are introduced to test this property. Feature i with the highest relevance $R_i(x)$ is removed from the input and the output $F(x)$ is recomputed. A sharp drop in the classification score implies that the explanation method performs well and the most important features are removed before the less important ones. The problem with this test is that images with perturbed pixels might be unnatural. Even if the accuracy of the model drops drastically, it might be because the model has not seen similar inputs, not because the explanation is better or worse.

A method called Remove and Retrain (ROAR) [33] was developed to test this hypothesis. A number of relevance estimators are considered such as gradient, integrated gradients and smoothed gradient [16]. For each estimator, ROAR replaces a percentage of the most important pixels with a constant value for each

considered image. After that, the model is retrained on the original dataset with the addition of the new perturbed images. Finally, the prediction accuracy of the model is tested. It is shown that perturbing random pixels of the image achieves the same accuracy drop after retraining as perturbing the "most important" pixels of images as determined by many commonly-used explanation methods. Nevertheless, removing relevant pixels often leads to a significant drop in accuracy. There is an opposite approach to ROAR named KAR: Keep And Retrain. Instead of removing most important features, we keep a fraction of the most important features and replace the rest of the features with a constant value. It is found empirically that KAR is a worse discriminator of performance of explanation methods compared to ROAR.

Another proposed evaluation technique is to use human-drawn bounding boxes [17] and count how many of the highly attributed pixels lie in the bounding box. Bounding boxes usually ignore the context of the image. As I mentioned above, some classifiers use the snow in wolf images or the water in boat images to make their decision more accurate.

We can also find some desirable theoretical properties of heatmaps described in literature:

Conservation

The conservation property [13] states that the total attribution on the input features should be equal to the output $F(x)$ for a target class t_K:

$$F(x)_{t_K} = \sum_i R_i^{l+1} = \sum_i R_i^l = ... = \sum_i R_i^0$$

This implies that the sum of relevances for layer l is equal to the sum of relevances for layer $l+1$ throughout the network. LRP satisfies this property, while gradient approaches fail for generic functions.

Positivity

The positivity property [18] states that input features are either relevant (positive) or irrelevant (zero).

$$\forall i: \; R_i^0 \geq 0$$

We want to make sure that the model is not attributing contradictory scores. Let x be an input for which we have an output of $F(x) = 0$. A conservative explanation technique where $F(x) = R(x)$ should assign a relevance of $R(x) = 0$ for that input. The only way we can be sure that the components of the relevance are truly equal to zero instead of being negative and positive with the same amount is if all relevances are positive.

Continuity

The continuity property [19] states that if two data points are nearly equivalent, then the explanations of their predictions should also be nearly equivalent. Continuity can be quantified as the strongest variation of attribution $R(x)$ in the input domain

$$\max_{x \neq x'} \frac{R(x) - R(x')}{\| x - x' \|_2}.$$

In Ref. [19], it is shown by a counterexample that gradient explanation methods can be discontinuous while LRP produces a continuous transition.

Selectivity

The selectivity property [13, 19] states that the model output should correspond to the explanation. In other words, removing highly relevant input pixels should result in a drop in accuracy. A way to quantitatively rank explanation methods is by performing "pixel-flipping". In practice, all method satisfy this property. A comparison has been made by Ref. [19] and it is shown how the classification score drops faster when removing features with high relevance attributed by LRP rather than removing pixels with high relevance attributed by gradient-based heatmaps.

Sensitivity

The sensitivity property [17] states that for every input and baseline that differ in one feature but have different predictions then the differing features should be attributed a non-zero relevance score. Ref. [17] shows with a counter-example how Gradients do not satisfy this property. This implies that Gradient times input also does not satisfy this property. Integrated gradients and LRP satisfy the "sensitivity" property.

Implementation invariance

The implementation invariance property [17] states: the attribution for two functionally equivalent models should be identical. Two networks are functionally equivalent if their outputs are equal for all inputs, despite having very different implementations. It is shown [17] that gradient-based explanation methods such as Gradient, Gradient times input and Integrated gradients satisfy this property. The authors also provide a counter-example how LRP does not satisfy this.

There are many more "desirable" properties described in literature and probably no explanation method can satisfy all of them. A comparison of the different methods and heatmaps can be seen in Figure 2.3. The figure was generated on images from the CIFAR-10 [34] dataset with a VGG16 [8] neural network.

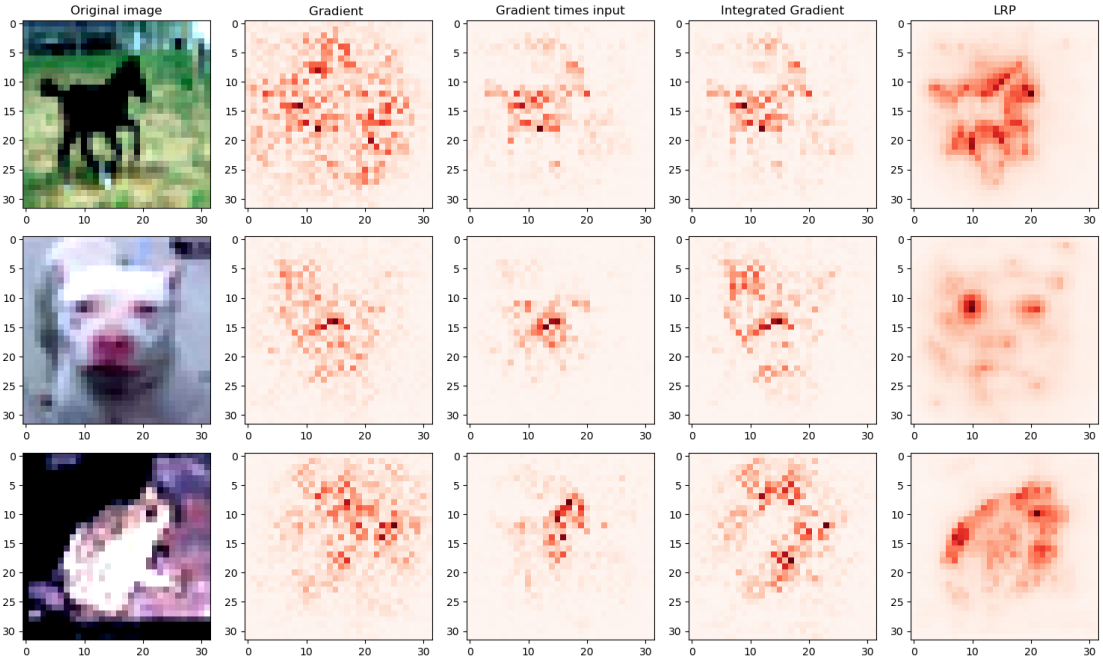

Figure 2.3: Comparison of heatmaps produced by different methods

2.3 Manipulated Explanations

I already established why explanations are important for ensuring trust and transparency. Heatmaps by themselves are not enough, however. We need to be certain that they are also robust to manipulations. Manipulations can happen either by perturbing the input or by adversarially training the model. We must differentiate between a robust predictor and a robust explanation.

2.3.1 Adding Adversarial Noise to Input

Adversarial attacks [35, 36] are formed by applying small, worst-case perturbations to examples from the dataset. More formally, an adversarial example x_{adv} consists of an image x and perturbation δx:

$$x_{adv} = x + \delta x$$

Models consistently misclassify adversarial examples with high confidence

$$F(x_{adv}) \neq F(x)$$

Even models with different architectures and trained on different datasets often misclassify the same adversarial example [36]. Perturbations also have the added property of having a very small magnitude

$$\| \delta x \| = \| x_{adv} - x \| \ll 1.$$

The human eye is not suited for distinguishing between x_{adv} and x which makes these adversarial examples very bothersome. The mere existence of these imperceptible attacks suggests a critical difference between sensory information of humans and neural networks.

In later work [20], it is shown that adversarial examples can manipulate explanations

$$h(x_{adv}) \neq h(x)$$

while maintaining the correct prediction label

$$\text{argmax}_i F(x_{adv})_i = \text{argmax}_i F(x)_i.$$

Take the example mentioned in Section 2.1: if a pathology image is classified as malignant, the doctor would like to know which parts of the image contributed most to the decision of the algorithm. If an adversary perturbs the image, the explanation might highlight the wrong part of the image as highly relevant. This can be detrimental to the patient.

It is also demonstrated [21] how adversarial examples can keep the output of the network approximately constant

$$F(x_{adv}) \approx F(x)$$

while the explanation can be manipulated to an arbitrary heatmap. This adversarial example is computed by optimizing the loss function

$$\mathcal{L} = \|h(x_{adv}) - h^t\|^2 + \ \gamma \ \|F(x_{adv}) - F(x)\|^2$$

with respect to x_{adv} via gradient descent. In the above equation, h^t is an arbitrary target heatmap. The authors also show how applying β-smoothing in the non-linear layers of the neural network makes the explanations much more robust to these perturbations. In practice, β-smoothing is done by replacing all ReLU layers in the neural network with Softplus layers. Softplus is the smoothed version of ReLU and the scale of smoothing is measured its the β-parameter. For more details, I refer to Ref. [21].

2.3.2 Adversarially Training Models

In some cases, the adversary might not have access to the input but can alter the weights of the model. In this section, I am considering such a scenario where the model is adversarially fine-tuned such that the accuracy of the classification output remains high but the heatmaps produced from various explanations methods are manipulated. The main advantage of adversarial training compared to input perturbations is that adversarial training manipulates all explanations across the dataset while perturbations often manipulate a single explanation for a target image.

Not only adversaries are motivated to train models with manipulated explanations. Consider the example mentioned in Section 2.2.1 where an individual gets automatically rejected for a loan by an ML algorithm. This algorithm might use features such as race, country of origin or gender for prediction. A company might

notice such unfair biases but they might not be willing to retrain the model on a different, unbiased, dataset. Perhaps their data shows a strong correlation between race and bad credit score so they would be inclined to keep the racial feature when computing loan eligibility.

What they can do instead is to fine-tune the model such that the explanation hides the importance of unlawful or unfair features. In another hypothetical example, an ML system evaluates candidates for a job position and takes their Curriculum vitae as input. Unfair biases are possible to occur in the evaluation of candidates. Companies can again mask explanations by adversarial training.

Previous work [22] has shown that adversarial training is possible and it works relatively well on different network architectures and explanation methods. The explanation methods which were manipulated are Grad-CAM [15] and a modified version of LRP [13]. [2]

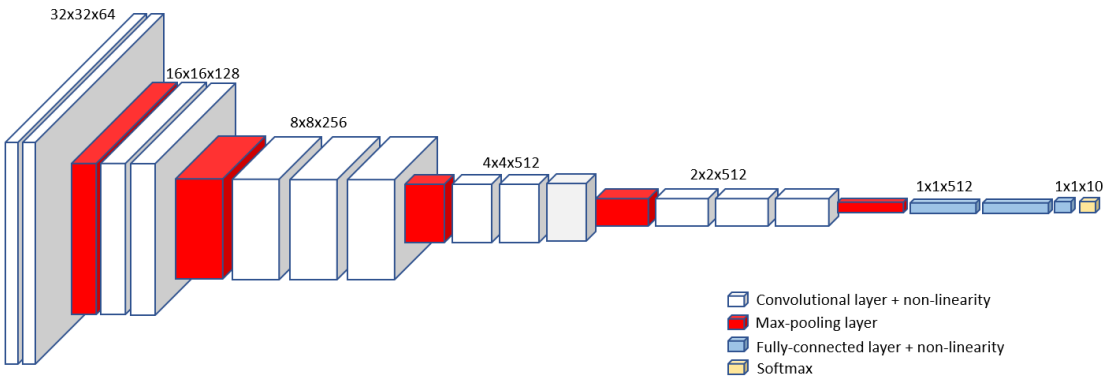

Figure 2.4: VGG16 architecture

Units in the convolutional layer of the neural network are segmented into feature maps. Each unit of a given feature map looks for the same feature in the input but at different positions. For example, the first convolutional layer displayed in

[2] The ε-rule is used for the fully connected linear layers and the $\alpha\beta$-rule is used for the convolutional layers.

Figure 2.4 contains 64 feature maps with height and width of 32 pixels while the last convolutional layer contains 512 feature maps with height and width of two pixels.

Let y^k be the classification score of image x for a fixed target class k before the softmax layer. Grad-CAM is an explanation method which is computed by taking the derivative of y^k with respect to feature maps $\mathcal{C} \in \mathbb{R}^{C,H,W}$ of a target convolutional layer, which is usually the last convolutional layer of the network. Here C,H, and W denote the number of channels, height and width of the feature maps $\mathcal{C}$ respectively. The mean value of the gradient is used for obtaining the feature map importance weight a_f^k

$$a_f^k = \frac{1}{HW} \sum_i^H \sum_j^W \frac{\partial y^k}{\partial \mathcal{C}_{ij}^f} ,$$

where $\mathcal{C}_{ij}^f$ labels the components of the feature maps $\mathcal{C}^f$. The parameter a_f^k captures the importance of feature map f for target class k. Using these importance weights, we obtain the final Grad-CAM heatmap by performing a linear combination of the results

$$\text{Grad-CAM}(x)^k = ReLU(\sum_f a_f^k \mathcal{C}^f) \in \mathbb{R}^{H,W}$$

The problem with this method is that the obtained heatmaps have the same shape as the target convolutional layer. For a VGG16 model trained on the ImageNet [32] dataset for example, Grad-CAM would produce a 14 x 14 heatmap if the target layer is the last convolutional layer. Then this heatmap is upsampled to match the dimensions of the input. For a VGG16 model trained on the CIFAR-10 dataset the heatmap size would be 2x2 pixels. Such a low-resolution heatmap cannot show the precise input-pixel importance like gradient or LRP methods. It can only highlight large regions.

The authors of Ref. [22] test different attacks associated to different penalty terms in the network's objective function. They take models, pre-trained for high classification accuracy, and optimize the loss function $\mathcal{L}$:

$$\mathcal{L}(\mathcal{D}, F) = \mathcal{L}_\mathcal{C}(\mathcal{D}, F) + \lambda \mathcal{L}_\mathcal{F}(\mathcal{D}, F)$$

with respect to the model parameters w via gradient descent. $\mathcal{L}_\mathcal{C}$ is the standard cross entropy classification loss, which is defined as

$$\mathcal{L}_\mathcal{C}(\mathcal{D}, F) = \frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{\exp(F(x_i)_{t_i})}{\sum_j \exp(F(x_i)_j)}\right) \tag{2.7}$$

$$= \frac{1}{n}\sum_{i=1}^{n} -F(x_i)_{t_i} + \log(\sum_j \exp(F(x_i)_j)). \tag{2.8}$$

In general terms, this means we would like the highest possible prediction probability for sample x_i for target class t_i. λ is the trade-off parameter between the two losses, $\mathcal{D}$ is the training dataset, F is the neural network and $\mathcal{L}_\mathcal{F}$ is the penalty loss. For example, one can choose a loss $\mathcal{L}_\mathcal{F}$ reducing relevances of the top k pixels with the highest original relevance. In this case, the penalty term would then be

$$\mathcal{L}_\mathcal{F} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j\in P_{i,k}} h(x_i)_j.$$

Here $P_{i,k}$ denotes the set of k pixels with the highest relevance for data point x_i computed on the original model before fine-tuning. $h(x_i)_j$ is the j-th pixel of the heatmap produced for image x_i. There are three other objectives which the authors are optimizing for: shifting the center of mass of heatmaps, highlighting a particular section of heatmap and swapping the explanation of two target classes.

Ref. [22] has a number of limitations however. First, for training and evaluation of their experiments they use LRP_T. This is a modified version of LRP which computes the relevances until a target layer T. It uses the same propagation rules as conventional LRP discussed in Ref. [13]. In Ref. [22] the target layer is the last convolutional layer. The relevances of that layer are then upsampled to the dimensions of the input. This is identical to Grad-CAM where the backpropagation is effectively done until a target layer. The weights of a large majority of layers are ignored so the explanation of LRP_T becomes less reliable.

Furthermore, LRP_T produces heatmaps with the same shape as Grad-CAM. Both methods suffer from the problem where the explanation map has a very small resolution hence the explanations are not precise. The authors claim it is easier to manipulate LRP_T heatmaps rather than LRP heatmaps. They only try using LRP_T and Grad-CAM heatmaps during model training. For evaluation they

additionally show heatmaps computed by the $Gradient_T$ method. $Gradient_T$ is the gradient of the output with respect to a target layer T. In their case, T is again the last convolutional layer of the network. Again, this method is very similar to Grad-CAM. It suffers from low resolution heatmaps and ignores the large majority of layers of the network. They avoid training with the $Gradient_T$ objective because it produces noisy heatmaps. The authors also evaluate their models on standard LRP but the results they show in the paper do not seem convincing.

In the next chapter, I am going to extend this work and show that adversarial training allows standard LRP explanations to be manipulated in a persuasive manner. All of the manipulated heatmaps will have the same dimensions as the input image which allows for very precise manipulations. Beyond LRP, I will manipulate gradient-based methods such as Gradient, Gradient times input and Integrated gradients as well as perturbation-based explanation methods such as Occlusion [14].

3 Experiments and evaluation

3.1 Overview

In this chapter, I will manipulate the explanations of multiple different neural networks while maintaining high classification accuracy. Throughout the experiments I am minimizing the loss function

$$\mathcal{L}(F, \mathcal{D}) = \mathcal{L}_{\mathcal{C}}(F, \mathcal{D}) + \lambda \mathcal{L}_{\mathcal{M}}(F, \mathcal{D})$$

where F is the model, $\mathcal{D}$ is the data set, $\mathcal{L}_{\mathcal{C}}(F, \mathcal{D})$ is a classification loss term, $\mathcal{L}_{\mathcal{M}}(F, \mathcal{D})$ is the manipulation loss term and λ denotes the trade-off parameter between the two losses. I train models with different variations of the $\mathcal{L}_{\mathcal{C}}$ loss term and the $\mathcal{L}_{\mathcal{M}}$ loss term.

Two different classification loss terms $\mathcal{L}_{\mathcal{C}}$ are used in the following experiments. Initially, in Sections 3.3 and 3.4, I use the cross entropy loss defined in Equation 2.7. Optimizing this objective allows the model to maintain high classification accuracy. Later, in Section 3.5, I use the mean squared error between the prediction probabilities of the model before manipulation and the prediction probabilities of the model after manipulation as classification loss. This is done so that the output of the network remains constant.

I perform two different manipulation attacks associated with different manipulation loss terms $\mathcal{L}_{\mathcal{M}}$ in the objective function. In Section 3.3, the center of mass of explanations is shifted towards the corner. In Section 3.4, I manipulate explanations to an arbitrary target heatmap.

In Section 3.6, I test how well the models manipulate perturbation-based explanation methods. In Section 3.7, I manipulate explanations by training the convolutional part of the model while freezing the weights of the classifier. In Section 3.8, I adversarially train a simpler neural network performing binary classification on non-image data.

3.2 General Setup

The experiments in this chapter are conducted on a neural network with the VGG16 [8] architecture. The dataset used for experiments is CIFAR-10 [34]. It consists of 60K images split in ten classes with 6k images per class. The size of each image $x \in \mathbb{R}^{C,H,W}$ is 3x32x32 pixels. The train/test split is 50k/10k respectively. The image pixel intensities are normalized by subtracting the mean pixel intensity of the data set and dividing by the standard deviation of the pixel intensity of the data set.

Because of the small pixel size of the images, it could be challenging even for a human agent to correctly classify each image. Manual classification attempts have been conducted for CIFAR-10 [37]. The achieved human accuracy is around 94%.

Image classification model

I initially train a VGG network solely for high classification accuracy. This is my baseline model denoted as F_{acc}. I use explanations generated on this baseline model for comparison to models with manipulated explanations. The objective function that is minimized during training is the cross entropy loss defined in Equation 2.7. VGG network is trained for 350 epochs. I use SGD [38] with momentum value $\mu = 0.9$. I use an adaptive learning rate lr as a function of the epoch:

$$lr = \begin{cases} 0.1, & \text{if } 0 < epoch \leq 150 \\ 0.01, & \text{if } 150 < epoch \leq 250 \\ 0.001, & \text{if } 250 < epoch \leq 350 \end{cases}$$

After training, this model achieves a top-1 classification accuracy of 92.46% on the test set.

3.2.1 Evaluation Metrics for Visual Similarity

In the following experiments, I am going to use these evaluation metrics to quantitatively compare the visual similarity between generated heatmaps and target heatmaps. Similar methods are used in Ref. [21, 39].

Mean Squared Error (MSE)

Let $p, q \in \mathbb{R}^{H,W}$ be two images where W and H stand for the width and height of the image respectively. The mean-squared error (MSE) for two images is defined as

$$MSE(p,q) = \frac{1}{WH} \sum_{i=1}^{W} \sum_{j=1}^{H} (p_{ij} - q_{ij})^2 \,. \tag{3.1}$$

Low MSE indicates high similarity between images.

Pearson Correlation Coefficient (PCC)

Pearson correlation coefficient [40] measures the linear correlation between two variables. For the two-dimensional images p and q with width W and height H PCC is defined as

$$PCC(p,q) = \frac{\sum_{i=1}^{W} \sum_{j=1}^{H} (p_{ij} - m_p)(q_{ij} - m_q)}{\sqrt{\sum_{i=1}^{W} \sum_{j=1}^{H} (p_{ij} - m_p)^2} \sqrt{\sum_{i=1}^{W} \sum_{j=1}^{H} (q_{ij} - m_q)^2}} \tag{3.2}$$

where m_p and m_q are the mean pixel intensities for images p and q respectively. The correlation value is bounded by

$$PCC(p,q) \in [-1, 1]$$

where PCC = 0 implies no correlation. A score of one implies exact linear positive correlation while a score of minus one implies exact linear negative correlation.

Structural Similarity Index (SSIM)

SSIM [41] integrates three equations that measure luminance similarity, contrast similarity and structure similarity of two images. SSIM is computed on a window sliding over two images. The equation used to calculate SSIM for two windows p and q of common size is

$$SSIM(p,q) = \frac{(2m_p m_q + C_1)(2\sigma_{pq} + C_2)}{(m_p^2 + m_q^2 + C_1)(\sigma_p^2 + \sigma_q^2 + C_2)}, \tag{3.3}$$

where m_p and m_q stand for the mean pixel intensity of p and q. With σ_p and σ_q I denote the standard deviation of the pixel intensity of p and q. C_1 and C_2 are constants added for stability when the denominator approaches zero. High values of SSIM indicate high similarity. The similarity score is bound by

$$SSIM(p, q) \in [-1, 1]$$

where SSIM = 0 implies no structural similarity. An SSIM score of one implies perfect positive structural similarity while an SSIM of minus one implies perfect negative structural similarity.

3.2.2 Vanishing Gradients

For the gradient-based manipulation experiments, the heatmaps are computed by taking the gradient of the output with respect to the input image. I compute the manipulation loss $\mathcal{L}_{\mathcal{M}}(F, \mathcal{D})$ based on those heatmaps and, for optimization, the derivative with respect to the model weights is taken. Hence, the second derivative of the model output needs to be calculated.

The default VGG16 architecture uses rectification (ReLU) non-linearities [8]. There is a problem when using ReLU non-linearities and computing second order derivatives. Consider an arbitrary ReLU layer that takes the output y_l of layer l:

$$ReLU(y_l) = max(0, y_l) = \begin{cases} 0, & \text{if } y_l \leq 0 \\ y_l, & \text{if } y_l > 0 \end{cases}$$

The first order and second order derivative of this function are:

$$ReLU'(y_l) = \begin{cases} 0, & \text{if } y_l \leq 0 \\ 1, & \text{if } y_l > 0 \end{cases}$$

$$ReLU''(y_l) = 0.$$

This fundamental property of ReLU to have a second derivative of zero makes optimization very difficult. Now consider a SoftPlus layer that takes the output y_l of layer l:

$$\text{SoftPlus}_\beta(y_l) = \frac{1}{\beta} * ln(1 + e^{\beta y_l})$$

The first and second order derivatives are therefore

$$\text{SoftPlus}'_\beta(y_l) = \frac{e^{\beta y_l}}{e^{\beta y_l} + 1}$$

$$\text{SoftPlus}''_\beta(y_l) = \frac{\beta\ e^{\beta y_l}}{(e^{\beta y_l} + 1)^2}.$$

SoftPlus is a very close approximator of ReLU for $\beta \to \infty$. SoftPlus also has a non-zero second order derivative. Visual representation of ReLU and SoftPlus functions can be found in Figure 3.1. I decided to use VGG16 with SoftPlus non-linearities during training and switch back to ReLU during evaluation. The β-parameter dictates the smoothness of the approximation. We also see that for large β, the second derivative of SoftPlus is very close to zero. It can be challenging to find a suitable β-value which approximates ReLU well-enough but also enables efficient optimization. After experimenting with different values, I got my best results with $\beta = 20$.

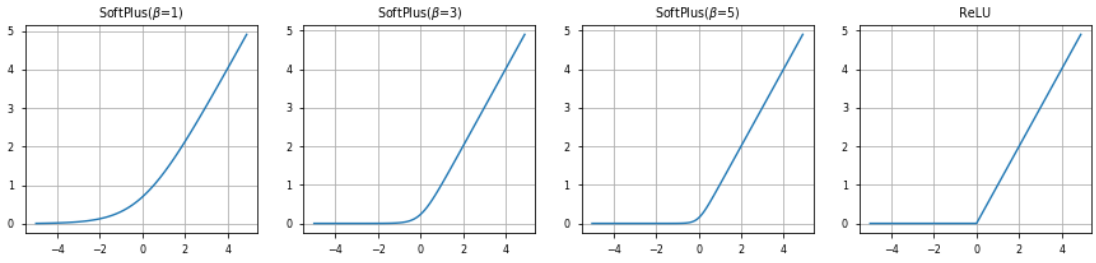

Figure 3.1: Plots of ReLU and SoftPlus functions

3.3 Shifting Center of Mass

In this section, I will shift the center of mass (COM) of explanations. Four models will be trained, each manipulating a different target explanation method. I will also measure how well the manipulation applies to other explanation methods different from the target method.

The center of mass of an image x represents the weighted mean position across all dimensions. I use pixel intensities as weights. For a two-dimensional image $x \in \mathbb{R}^{W,H}$, it can be calculated in the following way

$$COM(x) = \frac{1}{M_x} \sum_{i=1}^{W} \sum_{j=1}^{H} x_{ij} \begin{bmatrix} i \\ j \end{bmatrix} \in \mathbb{R}^2$$

where M_x is the sum of all pixel intensities of x, x_{ij} is the pixel intensity of pixel (i, j) and $\begin{bmatrix} i \\ j \end{bmatrix}$ are the coordinates of pixel (i, j). An illustration of center of masses across images can be found in Figure 3.2.

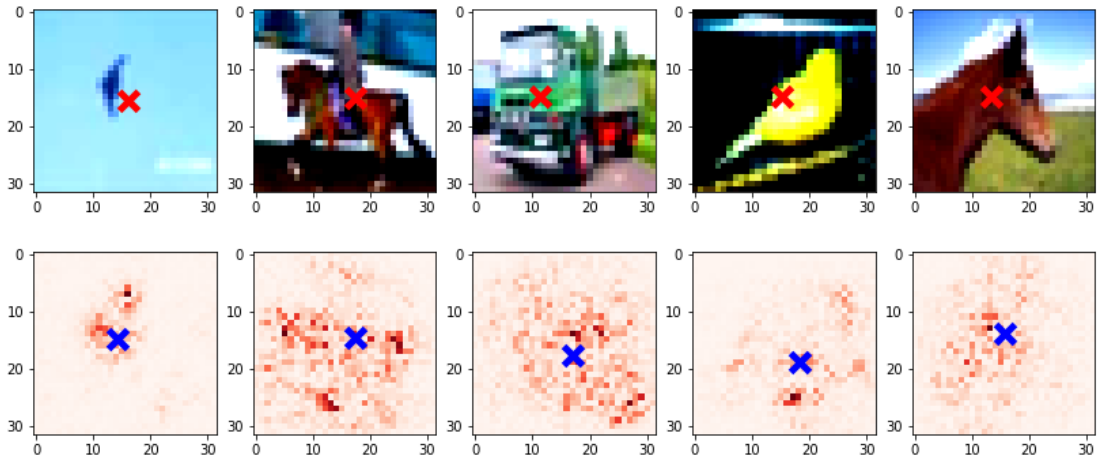

Figure 3.2: Center of mass across images and heatmaps

I use Euclidean distance (ED) to evaluate how well the center of mass of explanations is shifted. Smaller distance between center of masses of images indicates higher degree of similarity.

For two points p, $q \in \mathbb{R}^2$ the equation for ED is

$$ED(p, q) = \sqrt{(p_0 - q_0)^2 + (p_1 - q_1)^2} \,. \tag{3.4}$$

I choose to manipulate the heatmaps in such a way that their center of mass shifts towards the top-left corner. To make this possible, I define a target COM of $t_{com} = [0, 0]$. The total loss is therefore

$$\mathcal{L}(F, \mathcal{D}, t_{com}, \lambda) = \mathcal{L}_{\mathcal{C}}(F, \mathcal{D}) + \lambda \mathcal{L}_{\mathcal{M}}(F, \mathcal{D}, t_{com})$$

where $\mathcal{L}_{\mathcal{C}}(F, \mathcal{D})$ is the cross entropy loss defined in Equation 2.7. The manipulation loss is

$$\mathcal{L}_{\mathcal{M}}(F, \mathcal{D}, t_{com}) = \frac{1}{N} \sum_{n=1}^{N} \|COM(h_n),\ t_{com}\|_2$$

where N is the size of the training set and h_n is the heatmap of the n-th image. For each image x_n, the COM of heatmap h_n is calculated. Then I measure the euclidean distance between the generated center of mass and the target center of mass t_{com}.

Let F_{com} be a model trained for accuracy and manipulated "center of mass" explanation. During a single epoch of the training procedure, the algorithm iterates through the whole train dataset. The training dataset is split in mini-batches containing 256 images each. $\mathcal{L}$ is optimized with respect to the weights of the neural network.

Random grid search [42] was used for hyper-parameter tuning. I found a λ value of 0.008 to balance the magnitudes of the two losses well. Furthermore, I use the Adam optimizer [43] with learning rate of $5 \cdot 10^{-5}$ and default β_1, β_2 parameters. Pseudocode can be found in Algorithm 1.

Shifting center of mass of gradient heatmaps

For the first experiment, I manipulate the gradient heatmaps defined in Equation 2.1. Let this model be F^G_{com}.

I loaded F_{acc} and started training with the added $\mathcal{L}_{\mathcal{M}}$ objective. The training procedure ran for 200 epochs in total. A plot displaying the error throughout epochs can be found in Figure 3.4. After the training procedure was completed, the model F^G_{com} was evaluated on all 10k CIFAR-10 test images. A mean $\mathcal{L}_{\mathcal{M}}$ of 0.865 was measured across the test dataset. This implies that the COM of gradient explanations was on average less than one pixel away from the top left corner of images. The model maintained high classification accuracy of 91.8%.

Algorithm 1 Train model with "center of mass" objective. $\hat{x}$, $\hat{t_x}$ and $\hat{h_x}$ stand for batches of images, true labels and heatmaps. The method *get_heatmaps* takes a model and images as input and returns the corresponding heatmaps. I implemented four different versions of this method, one for each type of explanation which I try to manipulate.

procedure TRAINING(F_{com}, $\mathcal{D}$, t_{com}, λ)
  for $\hat{x}$, $\hat{t_x} \in \mathcal{D}$ **do**
    $\mathcal{L}_\mathcal{C} \leftarrow$ CrossEntropyLoss($F_{com}(\hat{x})$, $\hat{t_x}$)
    $h_{\hat{x}} \leftarrow$ get_heatmaps(F_{com}, $\hat{x}$)
    $\mathcal{L}_\mathcal{M} \leftarrow \|COM(h_{\hat{x}}), t_{com}\|_2$
    $\mathcal{L} \leftarrow \lambda \cdot \mathcal{L}_\mathcal{C} + (1\text{- } \lambda) \cdot \mathcal{L}_\mathcal{M}$
    $g \leftarrow \nabla_w \mathcal{L}$
    w $\leftarrow$AdamWeightUpdate(w, g, lr)

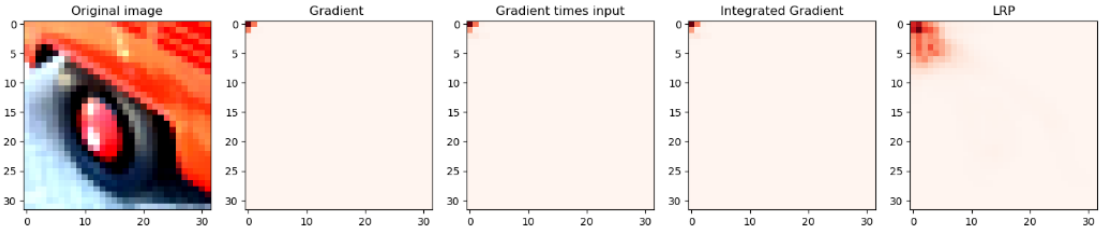

Figure 3.3: Heatmaps produced by F^G_{com}.

F^G_{com} was also evaluated on Gradient times input, Integrated gradients and LRP. I found that they all have been manipulated successfully. Heatmaps produced by F^G_{com} are shown in Figure 3.3. More image examples and comparison to the original model are shown in Appendix A. To make sure these manipulations apply to other images and not only the ones shown on the figure, I measured the mean manipulation loss $\mathcal{L}_\mathcal{M}$ over the whole test set. Results are shown in Table 3.1. These scores demonstrate that the results displayed in Figure 3.3 are not cherry-picked.

F^G_{com} does not only manipulate all test samples but also different explanation methods while maintaining high classification accuracy.

Shifting center of mass of "Gradient times input" heatmaps

For the second experiment in this section, I manipulate "Gradient times input" (GI) heatmaps defined in Equation 2.2. I call this model F^{GI}_{com}.

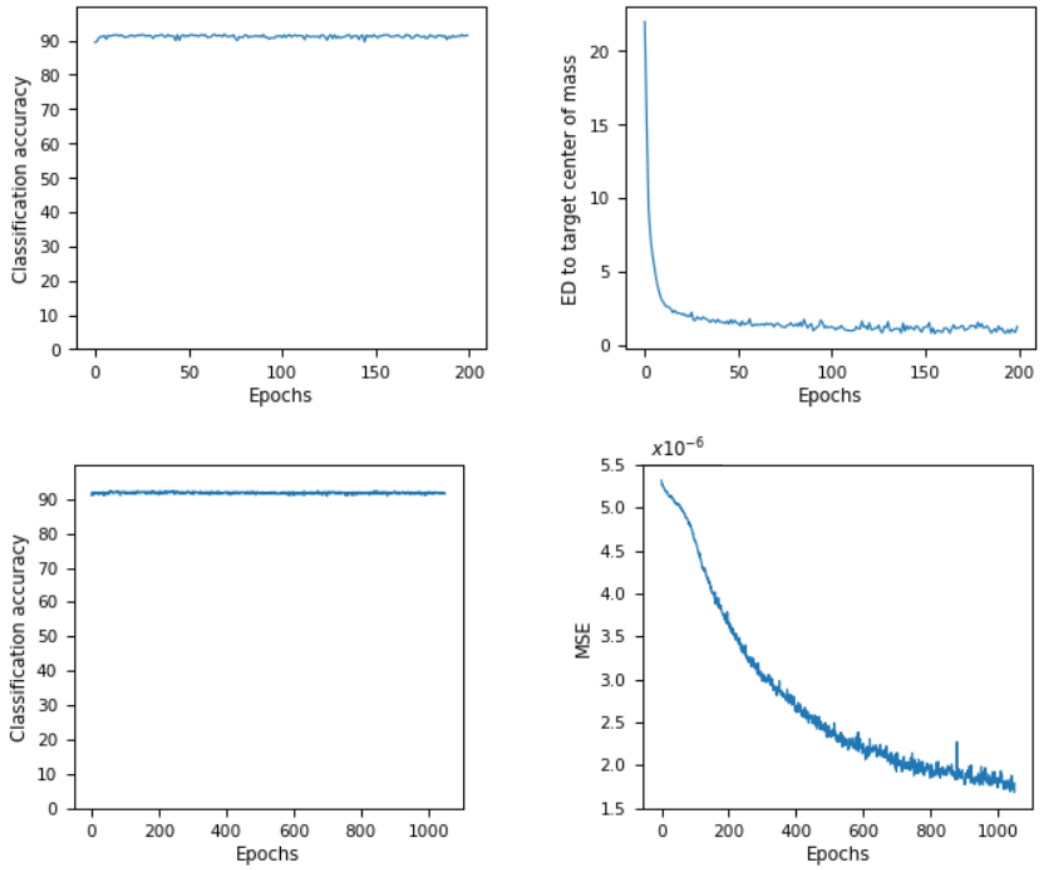

Figure 3.4: Loss across epochs for F^G_{com} (top) defined in Section 3.3 and F^G_{th} (bottom) defined in Section 3.4.

Again, the baseline model F_{acc} was loaded and the training procedure was started with the added center of mass manipulation objective $\mathcal{L}_{\mathcal{M}}$. The training procedure ran for 200 epochs. This method achieved worse evaluation results on GI heatmaps compared to F^G_{com} on test images. The manipulation loss for GI heatmaps for the F^{GI}_{com} model was 1.834 with classification accuracy of 91.3%. The algorithm seemed to memorize the train data and perform worse on the test set. Heatmaps generated from F^{GI}_{com} are shown in Figure 3.5. More image examples and comparison to the original model are shown in Appendix A. All evaluation results are shown in Table 3.1. This objective works well on different explanation methods but is slightly worse than F^G_{com}.

Model	Accuracy	Gradient $\mathcal{L}_{\mathcal{M}}$	GradInput $\mathcal{L}_{\mathcal{M}}$	IntGrad $\mathcal{L}_{\mathcal{M}}$	LRP $\mathcal{L}_{\mathcal{M}}$
F_{acc}	92.4	22.15	22.02	22.02	22.70
F_{com}^{G}	91.8	0.865	1.054	1.114	8.481
F_{com}^{GI}	91.3	1.867	1.834	1.962	9.823
F_{com}^{IG}	91.6	20.945	20.858	1.472	21.656
F_{com}^{LRP}	91.45	21.17	20.95	21.16	0.59

Table 3.1: Manipulation loss of models across different explanation methods.

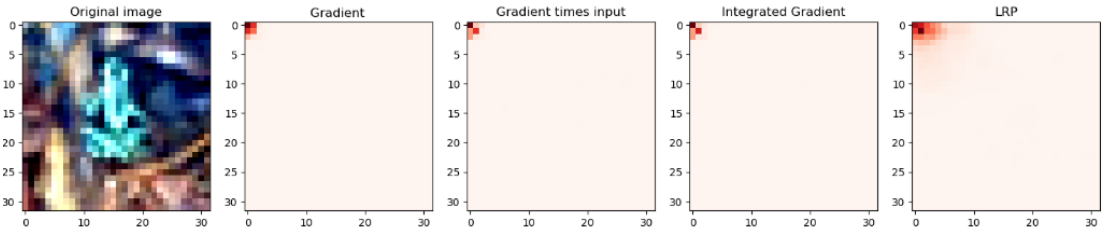

Figure 3.5: Heatmaps produced by F_{com}^{GI}.

Shifting center of mass of "Integrated gradient" heatmaps

For the third experiment, I manipulate integrated gradient (IG) heatmaps defined in Equation 2.4. I denote this model as F_{com}^{IG}.

The procedure is the same as for the other explanation methods described above. The baseline model F_{acc} is loaded and fine-tuned. Integrated gradient requires multiple backward passes to compute an explanation. In Section 2.2.4 I showed how the integral is approximated via a summation and the step size parameter m dictates the number of backward passes we are going to need for each explanation. There is an inherit trade-off between computational cost and how accurate the integral approximation is. A step size of eight was chosen during training and 100 during evaluation.

This model was trained for 100 epochs with a reduced batch size of 64. The manipulation loss of F_{com}^{IG} for IG heatmaps on the test set is 1.472 with classification accuracy of 91.64%. Slightly better than F_{com}^{GI} and a little worse than F_{com}^{G}. This model could not manipulate other explanation methods. Heatmaps generated from F_{com}^{IG} are shown in Figure 3.6. More image examples and comparison to the original

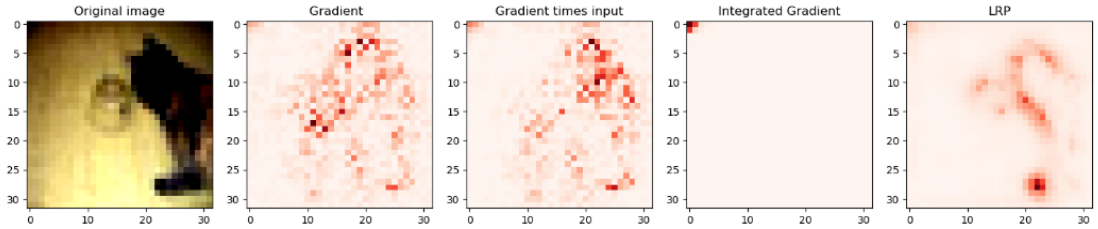

Figure 3.6: Heatmaps produced by F^{IG}_{com}.

model are shown in Appendix A. Evaluation results are shown in Table 3.1.

Shifting center of mass of LRP heatmaps

For the fourth experiment of this section, I manipulate LRP heatmaps defined in Equation 2.6. I denote this model as F^{LRP}_{com}.

Computing an LRP explanation is computationally cheap. The pre-trained model F_{acc} was loaded and trained for 200 additional epochs. The resulting model had dramatically manipulated LRP heatmaps with a mean manipulation loss of $\mathcal{L}_{\mathcal{M}} = 0.59$. This model did not manipulate other explanation methods. Heatmaps generated from F^{GI}_{LRP} are shown in Figure 3.7. More image examples and comparison to the original model are shown in Appendix A. Evaluation results are shown in Table 3.1.

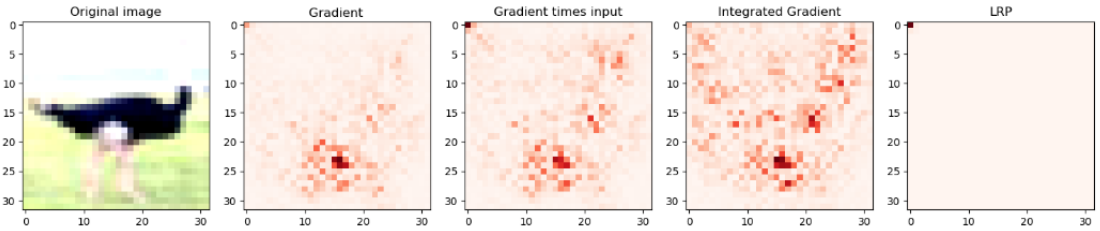

Figure 3.7: Heatmaps produced by F^{LRP}_{com}.

In summary, using the gradient heatmap during training for F^{G}_{com} manipulates all considered gradient-based methods better than the other models while manipulating LRP during training for F^{LRP}_{com} performed excellent on the test set for

LRP. I decided to test the limits of manipulation. In the next section I will try to manipulate explanations such that they reproduce an arbitrary image.

3.4 Target Heatmap

In this section, I am going to manipulate explanations in such a way that they reproduce an arbitrary target heatmap. I am going to train two models. One will aim to manipulate gradient-based explanations and the other will manipulate LRP explanations.

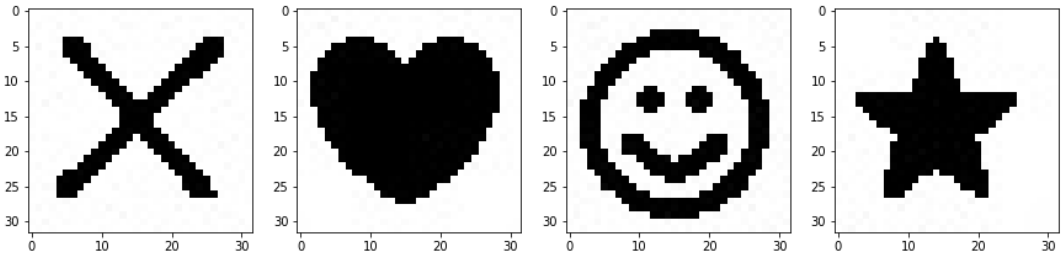

Figure 3.8: Examples of target heatmaps.

Target heatmaps were generated in MS Paint and used during training. Examples of target heatmaps can be found in Figure 3.8. In this section, I will minimize the objective function

$$\mathcal{L}(F, \mathcal{D}, t_h, \lambda) = \mathcal{L}_{\mathcal{C}}(F, \mathcal{D}) + \lambda \mathcal{L}_{\mathcal{M}}(F, \mathcal{D}, t_h)$$

where $\mathcal{L}_{\mathcal{C}}(F, \mathcal{D})$ is the cross entropy loss defined in Equation 2.7. The manipulation loss $\mathcal{L}_{\mathcal{M}}$ is the mean-squared error between the heatmap h_n generated for the n-th image of the training set N and the target heatmap t_h

$$\mathcal{L}_{\mathcal{M}}(F, \mathcal{D}, t_h) = \frac{1}{N} \sum_{n=1}^{N} MSE(h_n, t_h).$$

To measure baseline similarity between explanations and the target heatmap I loaded the F_{acc} model and calculated the manipulation loss for different explanation methods across the test set. Results are shown in Table 3.2.

A model trained for accuracy and manipulated "target heatmap" explanation is denoted as F_{th}. The algorithm iterates through the dataset which is split in batches of 256 images. Images and target heatmaps are normalized such that the sum of pixel intensities for each image x_n is equal to one

$$\forall n \ : \sum_{i=1}^{W} \sum_{j=1}^{H} (x_{ij})_n = 1$$

$\mathcal{L}$ is optimized with respect to the weights of the neural network. Again, random grid search was used for hyper-parameter tuning. A λ-value of 0.8 was found to deliver good results. I used the Adam optimizer [43] with an initial learning rate lr of 5.10^{-5}. The pseudocode can be found in *Algorithm* 2.

Algorithm 2 Train model with "target heatmap" objective

procedure TRAINING(F_{th}, $\mathcal{D}$, t_h, λ)
  for $\hat{x}$, $\hat{t_x} \in \mathcal{D}$ **do**
    $\mathcal{L}_\mathcal{C} \leftarrow$ CrossEntropyLoss($F_{th}(\hat{x})$, $\hat{t_x}$)
    $h_{\hat{x}} \leftarrow$ get_heatmaps(F_{th}, $\hat{x}$)
    $\mathcal{L}_\mathcal{M} \leftarrow MSE(h_{\hat{x}}$, $t_{\hat{h}})$
    $\mathcal{L} \leftarrow \lambda \cdot \mathcal{L}_\mathcal{C}$ + (1- λ) $\cdot \mathcal{L}_\mathcal{M}$
    $g \leftarrow \nabla_w \mathcal{L}$
    w ←AdamWeightUpdate(w, g, lr)

Manipulating gradient-based heatmaps to target image

For the first experiment in this section, I manipulate gradient-based heatmaps. I call this model F_{th}^G. The target heatmap used during training is the cross shown in Figure 3.8.

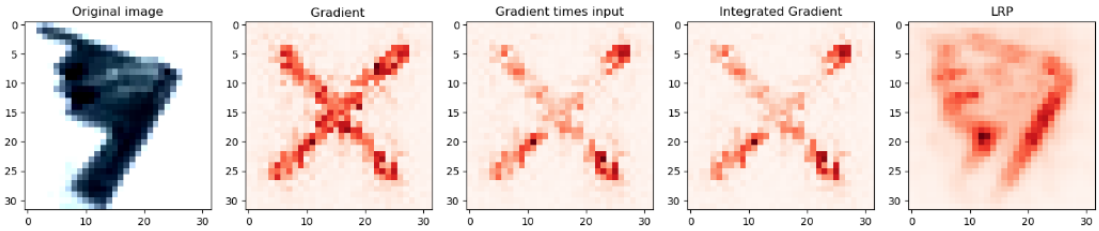

Figure 3.9: Heatmaps produced by F_{th}^{G}. Gradient-based methods are manipulated while LRP seems unchanged.

I loaded the baseline model F_{acc} and trained it for 1050 additional epochs. Detailed information about loss throughout epochs can be found in Figure 3.4.

Heatmaps produced by F_{th}^{G} can be found in Figure 3.9. More image examples and comparison to the original model are shown in Appendix B. Similarly to F_{com}^{G}, this model manipulates gradient times input and integrated gradient heatmaps. Unlike F_{com}^{G} however, this model does not manipulate LRP heatmaps. The mean MSE, PCC and SSIM between generated explanations and the "cross" target heatmap was measured. Results are shown in Table 3.2. The quantitative results suggest that the manipulation works across the whole test set. Low MSE or high PCC and SSIM scores suggest high similarity. When comparing the scores of F_{acc} and F_{th}^{G}, we see that the LRP values do not change much while all other heatmap types are manipulated.

Manipulating LRP heatmaps to target image

For the second experiment in this section, I manipulate LRP heatmaps. I denote this model as F_{th}^{LRP}. The target heatmap again is the cross shown in Figure 3.8.

The training procedure is identical to F_{th}^{G}. I loaded the baseline model F_{acc} and trained it for an additional 1050 epochs. A comparison of heatmaps generated by this model can be found in Figure 3.10. More image examples and comparison to the original model are shown in Appendix B. We can see that F_{th}^{LRP} manipulates LRP heatmaps well. The manipulation did not apply to gradient-based methods. When comparing the performance of F_{acc} and F_{th}^{LRP}we see that the LRP scores change drastically while those of gradient-based explanation methods remain almost unchanged. Mean MSE, PCC and SSIM between generated explanations

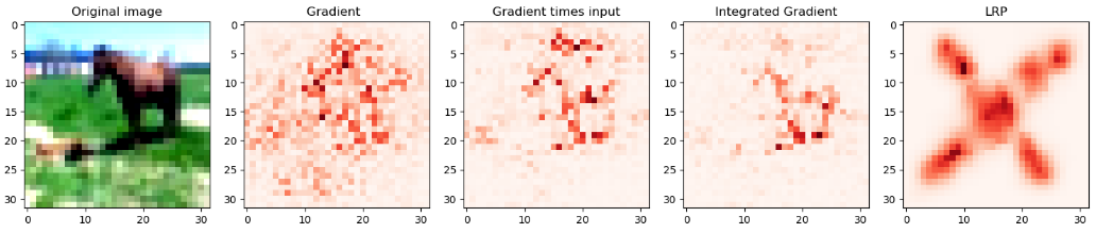

Figure 3.10: Heatmaps produced by F_{th}^{LRP}. This model only manipulates LRP explanations.

from F_{th}^{LRP} and the cross target heatmap can be found in Table 3.2.

In summary, I trained two models that manipulate explanations to a "cross" image. The first model, F_{th}^{G}, manipulates gradient-based explanation methods. The second model, F_{th}^{LRP}, manipulates LRP explanations. These manipulations work across the whole CIFAR-10 test data set.

3.5 Identical Prediction Probabilities

In this section, the explanation of F_{acc} is manipulated while maintaining the prediction probabilities of the model constant. This is useful because it implies that manipulation would also work for multi-label classification problems. We should differentiate between keeping the winning class of the model the same

$$\text{argmax}_i F(x)_i = \text{argmax}_i F_{acc}(x)_i$$

and keeping the prediction probabilities of the model identical

$$F(x) \approx F_{acc}(x) \ .$$

The loss function I will minimize is

$$\mathcal{L}(F, F_{acc}, \mathcal{D}, t_h, \lambda) = \mathcal{L}_{\mathcal{C}}(F, F_{acc}, \mathcal{D}) + \lambda \mathcal{L}_{\mathcal{M}}(F, \mathcal{D}, t_h)$$

where the classification loss is

$$\mathcal{L}_{\mathcal{C}}(F, F_{acc}, \mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} MSE(F(x_n), F_{acc}(x_n))$$

Evaluation of heatmaps generated by F_{acc}				
	Gradient	Gradient Times Input	Integrated Gradient	LRP
Accuracy	92.460	92.460	92.460	92.460
MSE	$5.27 \cdot 10^{-6}$	$6.015 \cdot 10^{-6}$	$6.20 \cdot 10^{-6}$	$4.59 \cdot 10^{-6}$
PCC	0.162	0.132	0.133	0.211
SSIM	0.099	0.172	0.172	0.068
Evaluation of heatmaps generated by F_{th}^{G}				
	Gradient	Gradient Times Input	Integrated Gradient	LRP
Accuracy	91.560	91.660	91.660	91.660
MSE	$1.702 \cdot 10^{-6}$	$2.73 \cdot 10^{-6}$	$2.68 \cdot 10^{-6}$	$4.23 \cdot 10^{-6}$
PCC	0.798	0.661	0.678	0.273
SSIM	0.389	0.523	0.482	0.038
Evaluation of heatmaps generated by F_{th}^{LRP}				
	Gradient	Gradient Times Input	Integrated Gradient	LRP
Accuracy	91.480	91.480	91.480	91.480
MSE	$5.109 \cdot 10^{-6}$	$5.819 \cdot 10^{-6}$	$5.826 \cdot 10^{-6}$	$2.148 \cdot 10^{-6}$
PCC	0.157	0.125	0.132	0.733
SSIM	0.058	0.134	0.111	0.443

Table 3.2: Evaluation of heatmaps generated by different models. MSE, PCC and SSIM measure the mean distance/similarity between heatmaps and the "cross" target heatmap.

and the manipulation loss is

$$\mathcal{L}_{\mathcal{M}}(F, \mathcal{D}, t_h) = \frac{1}{N} \sum_{n=1}^{N} MSE(h_n, t_h).$$

$\mathcal{L}$ is optimized with respect to the parameters w of the neural network F. Pseudocode can be found in *Algorithm* 3.

Algorithm 3 Train model with "target heatmap" objective and identical prediction probabilities to the original model

procedure TRAINING(F_{ipp}, F_{acc}, $\mathcal{D}$, t_h, λ)
  for $\hat{x}$, $\hat{t_x} \in \mathcal{D}$ **do**
    $\mathcal{L}_{\mathcal{C}} \leftarrow MSE(F_{ipp}(\hat{x}), F_{acc}(\hat{x}))$
    $h_{\hat{x}} \leftarrow$ get_heatmaps(F_{ipp}, $\hat{x}$)
    $\mathcal{L}_{\mathcal{M}} \leftarrow MSE(h_{\hat{x}}, t_{\hat{h}})$
    $\mathcal{L} \leftarrow \lambda \cdot \mathcal{L}_{\mathcal{C}} + (1\text{-}\ \lambda) \cdot \mathcal{L}_{\mathcal{M}}$
    $g \leftarrow \nabla_w \mathcal{L}$
    w $\leftarrow$AdamWeightUpdate(w, g, lr)

I trained a model that manipulates gradient explanations while maintaining its prediction probabilities constant. This model is denoted as F_{ipp}^{G}. I trained F_{ipp}^{G} for 1350 epochs.

Manipulation scores are shown in Table 3.3. Quantitative results are compared to the results of the baseline model F_{acc} and the model with manipulated gradient heatmaps F_{th}^{G}.

I tested the mean classification error $\mathcal{L}_{\mathcal{C}}$ across the test set for F_{ipp}^{G} and F_{th}^{G}. The resulting prediction probabilities of F_{ipp}^{G} on the test set were extremely close to the prediction probabilities of F_{acc} with

$$\mathcal{L}_{\mathcal{C}}(F_{ipp}^{G}, F_{acc}, \mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} MSE(F_{ipp}^{G}(x_n), F_{acc}(x_n)) \approx 0.007\ .$$

	F_{acc}	F_{th}^G	F_{ipp}^G
Accuracy	92.460	91.560	91.660
Classification loss	0	0.0146	0.007
MSE	$5.277 \cdot 10^{-6}$	$1.702 \cdot 10^{-6}$	$1.47 \cdot 10^{-6}$
PCC	0.162	0.798	0.830
SSIM	0.099	0.389	0.445

Table 3.3: Evaluation of classification loss and heatmaps generated from F_{acc}, F_{th}^G and F_{ipp}^G.

This error is around twice as small as the prediction probabilities of F_{th}^G where the objective was to keep the winning class constant

$$\mathcal{L}_{\mathcal{C}}(F_{th}^G, F_{acc}, \mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} MSE(F_{th}^G(x_n), F_{acc}(x_n)) \approx 0.0146 \ .$$

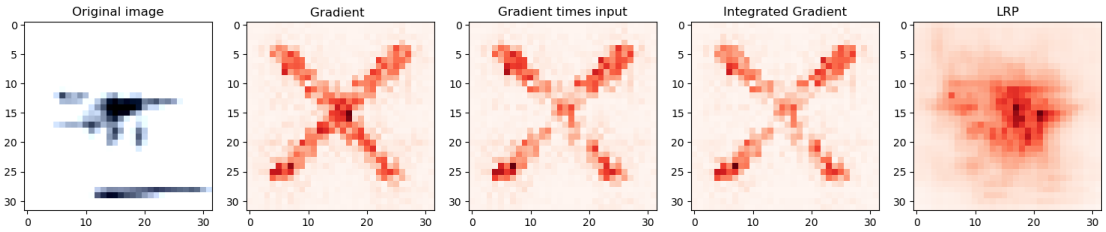

Figure 3.11: Heatmaps produced by F_{ipp}^G.

Heatmaps produced from F_{ipp}^G can be found in Figure 3.11. We see that the heatmaps generated from F_{ipp}^G manipulate gradient explanations greatly, generating better results than the F_{th}^G model. Likely, this is due to the fact that F_{ipp}^G was trained for more epochs.

3.6 Perturbation-Based Explanations: Occlusion

I used a 5x5, all-zero patch with a step size of one pixel. Therefore, the output heatmaps had spatial resolution of 27x27 pixels. With a step size of one pixel

we need 730 forward passes on CIFAR-10 images to compute a single batch of explanations.

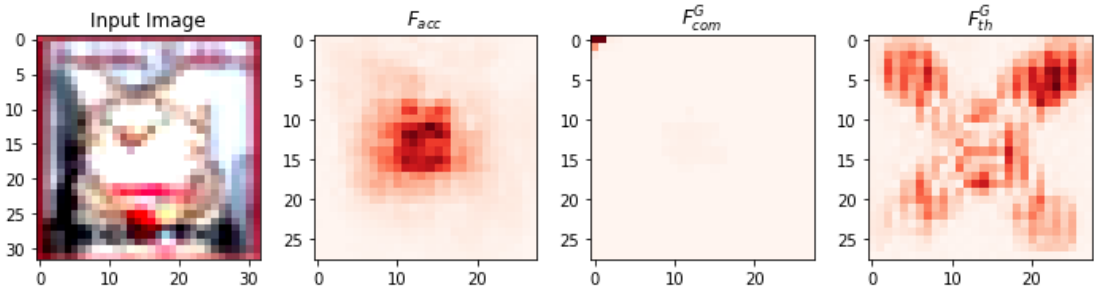

Figure 3.12: Occlusion heatmaps produced by different models.

I tested if the adversarially trained models, F^G_{com} and F^G_{th}, can also manipulate occlusion explanations. Results are displayed in Figure 3.12. More image examples are shown in Appendix D. It is evident that, although the models were optimized to manipulate gradient heatmaps, they are also able to manipulate perturbation-based methods.

3.7 Frozen Classifier

I tested whether a manipulation would be possible if we do not train the parameters of the whole network but only a subset of them. The VGG network architecture shown in Figure 2.4 can be split into two sections: a convolutional part and a classifier. The convolutional part consists of all convolutional and pooling layers of the network. There are 13 convolutional layers in total. The classifier consists of the final 3 fully-connected linear layers.

The weights of the fully-connected linear layers were frozen. This means that during backpropagation these weights will not be updated. I used gradient heatmaps in the objective function. I call this model as F^G_{fc}. The same training procedure as Algorithm 2 was executed and run for 1050 epochs. The results can be found

Heatmaps generated with Gradient			
	F_{acc}	F_{th}^{G}	F_{fc}^{G}
Accuracy	92.460	91.560	91.570
MSE	$5.277 \cdot 10^{-6}$	$1.702 \cdot 10^{-6}$	$2.59 \cdot 10^{-6}$
PCC	0.162	0.798	0.654
SSIM	0.099	0.389	0.280

Table 3.4: Evaluation of heatmaps generated by the gradient explanation method on F_{acc}, F_{th}^{G} and F_{fc}^{G}.

at Table 3.4. Heatmaps produced from F_{fc}^{G} can be found in Figure 3.13.

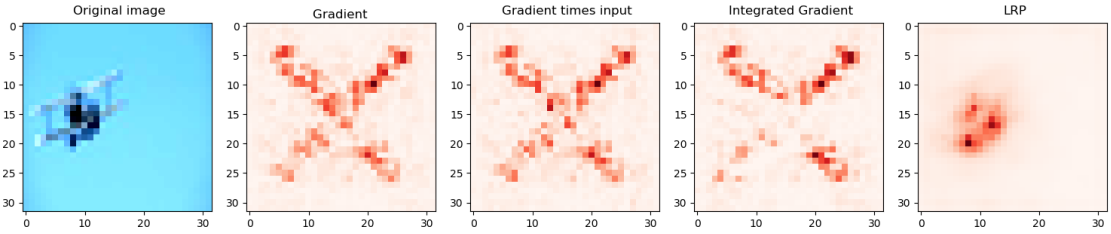

Figure 3.13: Heatmaps produced by F_{fc}^{G}.

It is evident that heatmaps can still be manipulated in this way but the error drops faster when we are training the whole network. I conducted more experiments such as freezing the whole convolutional section of the model, freezing various subsets of layers of the network and training a single layer but the results were not satisfactory. The error does not seem to go down when I try to train five or less layers. I managed to manipulate the explanations by training the first ten convolutional layers of the network.

3.8 Manipulating Simpler Models

To show that manipulations are possible not only for computer vision problems, I trained a neural network on the Census dataset for adult income [**?**]. The predic-

tion task was to determine whether a person makes over 50K a year. The features I used were 'Age', 'Years of study', 'Capital gain', 'Capital loss', 'Work hours per week', 'Race' and 'Gender'. Since 'Race' and 'Gender' are categorical, they were one-hot-encoded. The train set includes over 32K data points and the test set consists of over 16K data points.

The neural network architecture had 2 linear layers with ReLU non-linearity and 100 neurons in each layer. SoftMax was applied after the last linear layer. The loss function was cross entropy defined in 2.7 and the optimizer of choice was Adam [43] with learning rate of 0.001.

After 200 epochs, our model achieved classification accuracy of 83.4%. The gradient explanation method described in Equation 2.1 was used to derive the input feature importance. A visual representation of the feature importance is shown in Figure 3.14. We can see that racial and gender features have notable importance.

I also trained the exact same model for 200 epochs without unfair features. The achieved classification accuracy dropped to 82.1%. Seeing these results, we can be fairly certain that race or gender have predictive power in the context of our problem.

Finally, I adversarially-trained the first model which used all features. I used the same method as the Target Heatmap attack described in Section 3.4. The target explanation was the original feature importance for continuous features in combination with an importance score of zero for categorical features. The model was fine-tuned for 100 additional epochs. The achieved classification accuracy remained 82.7%. A visual representation of the feature importance of the adversarially-trained model is shown in Figure 3.15. We can see that, although the importance score of unfair features is near zero, the model utilizes their predictive power to make a more accurate decision.

Some of the models discussed in this section manage to manipulate all considered explanation methods across the whole test set. We see that manipulations are powerful and can potentially be harmful to ML. We also saw that manipulations can happen across different ML domains. In the next section, I will discuss possible ways of detecting manipulations.

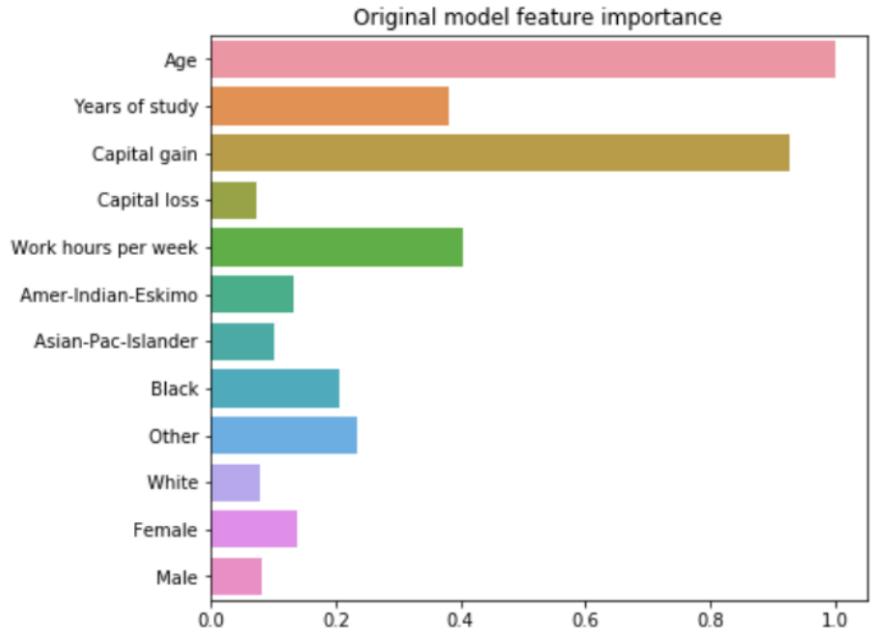

Figure 3.14: Feature importance before manipulation.

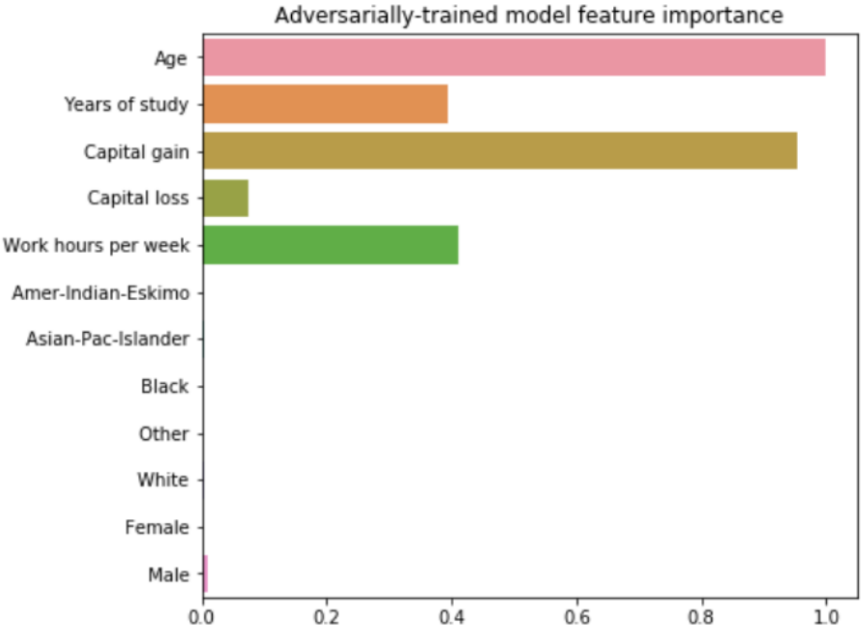

Figure 3.15: Feature importance after manipulation.

4 Detecting manipulations

In this section, I examine scenarios where adversarial model manipulation may be detected or prevented. In Section 4.1, I test if manipulations work on randomly generated input images. In Section 4.2, I test if manipulation works if we add noise to the weights of models.

4.1 Noise as Input

One might argue that the manipulations discussed so far only work on the CIFAR-10 data set and do not apply to other images. To test this, I generated 10k random image samples from a Gaussian distribution $\mathcal{N}(0, 1)$ and used those as input for my models. Examples of explanations generated by F_{acc}, F^G_{com} and F^G_{th} on the random input are shown in Figure 4.1. The manipulation scores over all 10k samples are shown in Table 4.1. We see that, even though this input comes from a completely different distribution than the CIFAR-10 images, the explanations are still greatly manipulated.

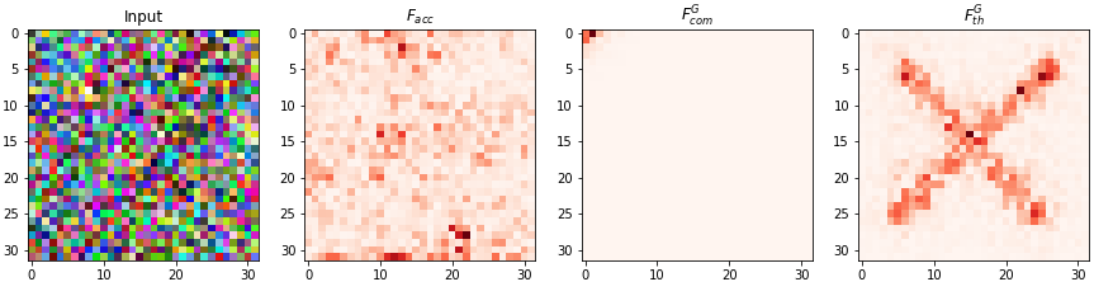

Figure 4.1: Explanation generated with Gaussian noise as input by F_{acc}, F^G_{com} and F^G_{th}

Gradient heatmaps generated with Gaussian noise as input.			
	F_{acc}	F^G_{com}	F^G_{th}
COM ED	22.48	2.40	-
TH MSE	$4.993 \cdot 10^{-6}$	-	$1.92 \cdot 10^{-6}$
TH PCC	0.138	-	0.769
TH SSIM	0.033	-	0.336

Table 4.1: Evaluation of heatmaps generated by the gradient explanation method on F_{acc}, F^G_{com} and F^G_{th} with random noise as input. The COM ED is measured for F_{acc} and F^G_{com}. The target heatmap (TH) similarity is measured for F_{acc} and F^G_{th}.

Gradient heatmaps generated with Gaussian noise as input.			
	F_{acc}	F^G_{com}	F^G_{th}
MSE	$1.249 \cdot 10^{-6}$	$9.216 \cdot 10^{-6}$	$1.101 \cdot 10^{-6}$
PCC	0.233	0.975	0.763
SSIM	0.158	0.996	0.672

Table 4.2: Mean heatmap similarity between heatmaps shown in Figure 4.1 with heatmaps produced by F_{acc}, F^G_{com} and F^G_{th} with random noise as input.

A potential way to detect manipulations is to leverage the usage of random noise as input. Manipulated explanations seem to be very similar to each other while explanations generated by F_{acc} vary across different input. I compared heatmap similarity between the heatmaps shown in Figure 4.1 and 100 other heatmaps produced by the same models with different instances of random noise as input. Results are shown in Table 4.2. PCC and SSIM scores show that heatmaps generated by F^G_{com} and F^G_{th} are highly similar across different inputs.

Naturally, this detection method would not work for all attacks. Some attacks, such as lowering the relevance of top-k pixels, can still produce dissimilar explanations across different input. We would need more information about the type of attack in order to detect it.

4.2 Adding Noise to Weights

It is known that detecting adversarial perturbations is possible if we additionally perturb the input with some Gaussian noise [44]. Since in my experiments I fine-tuned the model parameters, perhaps if I add noise to the weights of the model, that might also reveal manipulation.

I evaluated the models after adding Gaussian noise to the weights to see how that would affect manipulation. In Figure 4.4, it is visualized how adding noise with different variances changes the behaviour of the models. The drops in accuracy of F_{acc}, F^G_{com} and F^G_{th} are very similar for the same levels of added noise. The explanations generated by F^G_{com} and F^G_{th} with added noise can be found in Figures 4.2 and 4.3. More image examples and comparison to the original model are shown in Appendix C. We can see that the manipulation is still significant after we increase levels of noise. This suggests that detecting models with manipulated explanations can be quite difficult. Currently, the only way we can detect manipulations is if we have access to the original model or if we have knowledge of the expected attack.

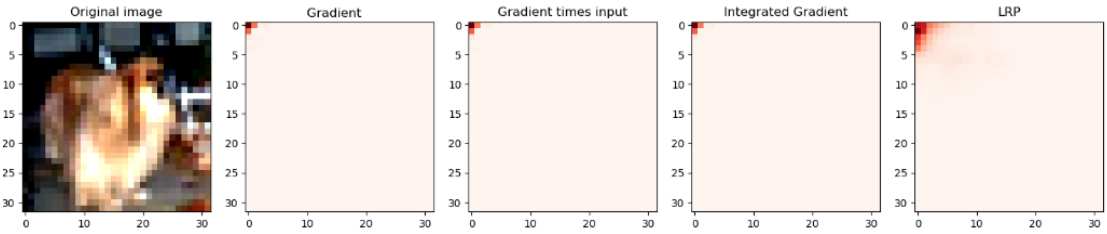

Figure 4.2: Heatmaps produced by F^G_{com} with added Gaussian noise $\mathcal{N}(0, 0.02)$ to the model parameters.

These models not only manipulate all explanation methods considered in this thesis, but also work on almost any input. Beyond that, it seems difficult to detect such manipulations. These results seem concerning. Adversarial model training has the potential to overcome the fairness and transparency that the explanation methods promise.

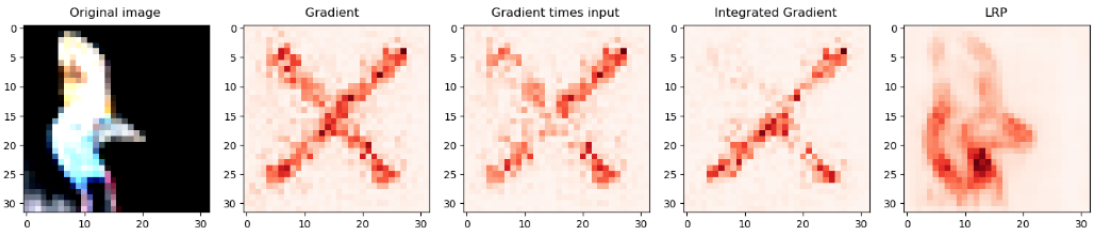

Figure 4.3: Heatmaps produced by F_{th}^{G} with added Gaussian noise $\mathcal{N}(0, 0.01)$ to the model parameters.

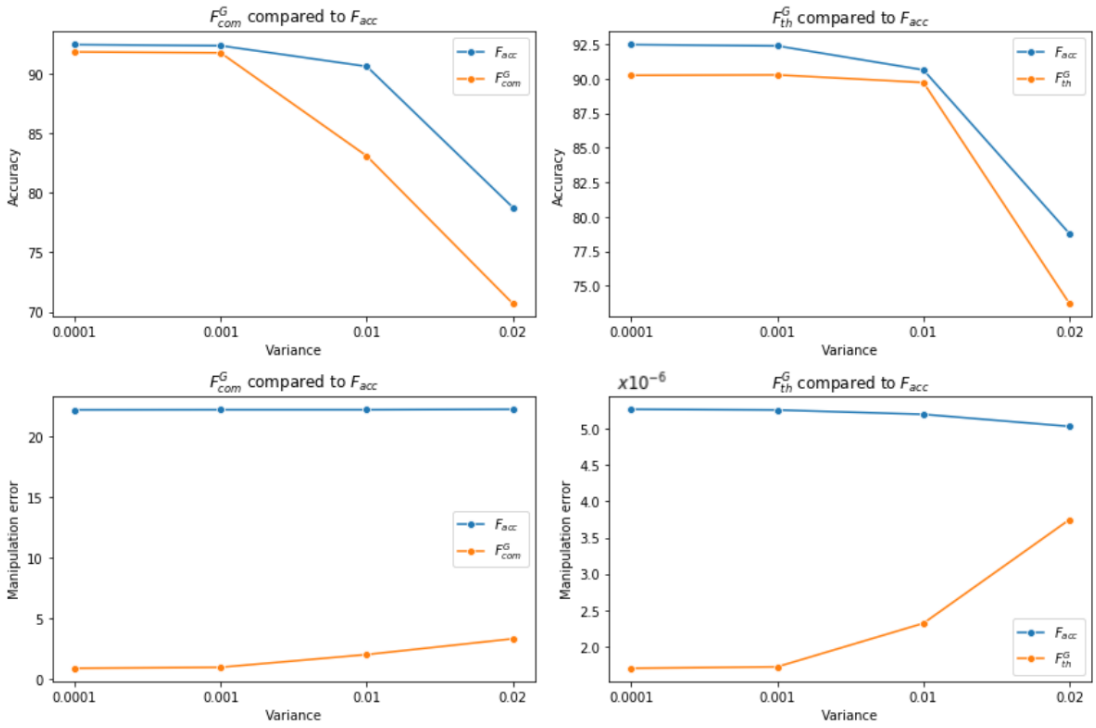

Figure 4.4: Adding noise sampled from a Gaussian distribution with zero mean and different variances to the weights of the model.

5 Conclusion

As ML algorithms become more important in society, so does interpreting their decisions. ML explainability is a promising way we can achieve fairness, accountability and transparency in this field. Policy makers are realizing this and regulations such as GDPR [11] introduce new laws which give people the right to request explanations for a decision made by an algorithm.

An increasing amount of biased ML algorithms have been discovered recently. In Ref. [7], it was shown that millions of black patients are at a disadvantage because of their race. Other cases of biased algorithms are found in predictive policy systems [45], skin tones in pedestrian detection [46] and gender bias in STEM career ads [47]. Explanation methods might be the key to unlocking the black box of ML and unveiling these biases. Many explanation methods have been developed in recent years trying to solve this problem.

Robustness of explanations is still an issue, however. Some companies might find strong correlations between unlawful features and the output of their algorithms. They might want to mask explanations rather than not use those features. There is a need for reliable detection of models trained with manipulated explanations.

In this thesis, I manipulate the explanations of models in two distinct ways: I shift the center of mass of explanations and manipulate explanations to an arbitrary target heatmap. All manipulations are done while keeping the winning class approximately constant. I also manipulate models where their prediction probability distribution is kept constant. Manipulations are carefully evaluated and both quantitative and qualitative results are shown. I consider three gradient-based explanation methods, Gradient, Gradient times input and Integrated gradient, one propagation-based explanation method, LRP, and one perturbation-based method, Occlusion. All these explanation methods were manipulated successfully. Each model is tailored to manipulate a target explanation method but we also saw that, in some cases, manipulating one method translates to the manipulation of multiple different explanation methods. Manipulation can be achieved by optimiz-

ing a subset of the model parameters rather than all of them. Finally, we saw that the manipulations work on almost any input and detecting manipulations can be a challenging task.

In the scope of this thesis I did not thoroughly investigate the theoretical reasons why manipulations are possible and work so well. Another limitation of this work is that all of the models have the same underlying network architecture: VGG16. Nevertheless, Ref. [22] already shows that adversarial manipulation works on different popular network architectures.

Future research topics include investigation of adversarial model training. In Section 2.2.1, I highlighted the importance of explanations and how trust and transparency are essential in deep learning. The fact that there is almost no work present in the area of adversarial model training can be concerning. Moreover, robust explanation methods which are not susceptible to model manipulation or input perturbations is also an important future research direction. Another way to extend this work is to train models with manipulated explanations applied to other domains such as natural language processing, speech recognition or recommender systems.

List of Acronyms

ML	Machine learning
CNN	Convolutional neural network
ED	Euclidean distance
PCC	Pearson correlation coefficient
SSIM	Structural similarity index
MSE	Mean squared error
IG	Integrated gradients
GI	Gradient times input
LRP	Layer-wise relevance propagation
SGD	Stochastic gradient descent
Adam	Adaptive moment estimation
COM	Center of mass
TH	Target heatmap
CE	Cross entropy
IPP	Identical prediction probabilities
FC	Frozen classifier

Bibliography

[1] M. van Hartskamp, S. Consoli, W. Verhaegh, M. Petkovic, and A. van de Stolpe, "Artificial intelligence in clinical health care applications: Viewpoint," *Interactive Journal of Medical Research*, vol. 8, p. e12100, Apr 2019.

[2] X. Zheng, M. Zhu, Q. Li, C. Chen, and Y. Tan, "Finbrain: When finance meets ai 2.0," 2018.

[3] A. Goldstein, L. Fink, and G. Ravid, "A framework for evaluating agricultural ontologies," 2019.

[4] J. Lee, J. Singh, and M. Azamfar, "Industrial artificial intelligence," 2019.

[5] J.-C. Shi, Y. Yu, Q. Da, S.-Y. Chen, and A.-X. Zeng, "Virtual-taobao: Virtualizing real-world online retail environment for reinforcement learning," 2018.

[6] J. R. J. Reinsel, David; Gantz, "Data age 2025: The evolution of data to life-critical," 2017.

[7] Z. Obermeyer, B. Powers, C. Vogeli, and S. Mullainathan, "Dissecting racial bias in an algorithm used to manage the health of populations," *Science*, vol. 366, pp. 447–453, 10 2019.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[10] A. Trask, D. Gilmore, and M. Russell, "Modeling order in neural word embeddings at scale," 2015.

[11] B. Casey, A. Farhangi, and R. Vogl, *Rethinking Explainable Machines: The GDPR's 'Right to Explanation' Debate and the Rise of Algorithmic Audits in Enterprise*. Berkeley Technology Law Journal, Vol. 34, 2019.

[12] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," 2013.

[13] S. Lapuschkin, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PLoS ONE*, vol. 10, p. e0130140, 07 2015.

[14] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," 2013.

[15] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," 2016.

[16] D. Smilkov, N. Thorat, B. Kim, F. ViÃ©gas, and M. Wattenberg, "Smoothgrad: removing noise by adding noise," 2017.

[17] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," 2017.

[18] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller, "Explaining nonlinear classification decisions with deep taylor decomposition," *Pattern Recognition*, vol. 65, p. 211â222, May 2017.

[19] G. Montavon, W. Samek, and K.-R. Müller, "Methods for interpreting and understanding deep neural networks," *Digital Signal Processing*, vol. 73, p. 1â15, Feb 2018.

[20] A. Ghorbani, A. Abid, and J. Zou, "Interpretation of neural networks is fragile," 2017.

[21] A.-K. Dombrowski, M. Alber, C. J. Anders, M. Ackermann, K.-R. Müller, and P. Kessel, "Explanations can be manipulated and geometry is to blame," 2019.

[22] J. Heo, S. Joo, and T. Moon, "Fooling neural network interpretations via adversarial model manipulation," 2019.

[23] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," 2017.

[24] A. van den Oord, S. Dieleman, and B. Schrauwen, "Deep content-based music recommendation," in *Advances in Neural Information Processing Systems 26* (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), pp. 2643–2651, Curran Associates, Inc., 2013.

[25] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, vol. 22, pp. 1533–1545, 10 2014.

[26] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," 2017.

[27] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you?: Explaining the predictions of any classifier," 2016.

[28] F. Hohman, H. Park, C. Robinson, and D. H. Chau, "Summit: Scaling deep learning interpretability by visualizing activation and attribution summarizations," 2019.

[29] S. Lapuschkin, S. WÃldchen, A. Binder, G. Montavon, W. Samek, and K.-R. Müller, "Unmasking clever hans predictors and assessing what machines really learn," *Nature Communications*, vol. 10, Mar 2019.

[30] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," 2017.

[31] D. Balduzzi, M. Frean, L. Leary, J. Lewis, K. W.-D. Ma, and B. McWilliams, "The shattered gradients problem: If resnets are the answer, then what is the question?," 2017.

[32] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[33] S. Hooker, D. Erhan, P.-J. Kindermans, and B. Kim, "Evaluating feature importance estimates," 2018.

[34] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., 2009.

[35] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2013.

[36] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014.

[37] A. Karpathy, "Lessons learned from manually classifying cifar-10," 2011.

[38] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pp. III–1139–III–1147, JMLR.org, 2013.

[39] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim, "Sanity checks for saliency maps," 2018.

[40] W. Kirch, ed., *Pearson's Correlation Coefficient*, pp. 1090–1091. Dordrecht: Springer Netherlands, 2008.

[41] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE TRANSACTIONS ON IMAGE PROCESSING*, vol. 13, no. 4, pp. 600–612, 2004.

[42] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, p. 281â305, Feb. 2012.

[43] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[44] B. Liang, H. Li, M. Su, X. Li, W. Shi, and X. Wang, "Detecting adversarial image examples in deep neural networks with adaptive noise reduction," *IEEE Transactions on Dependable and Secure Computing*, p. 1â1, 2019.

[45] S. J. Richardson, R. and K. Crawford, "Dirty data, bad predictions: How civil rights violations impact police data, predictive policing systems, and justice.," 2019.

[46] B. Wilson, J. Hoffman, and J. Morgenstern, "Predictive inequity in object detection," 2019.

[47] A. Lambrecht and C. Tucker, "Algorithmic bias? an empirical study into apparent gender-based discrimination in the display of stem career ads," *SSRN Electronic Journal*, 01 2016.

Appendix

A Shifted center of mass

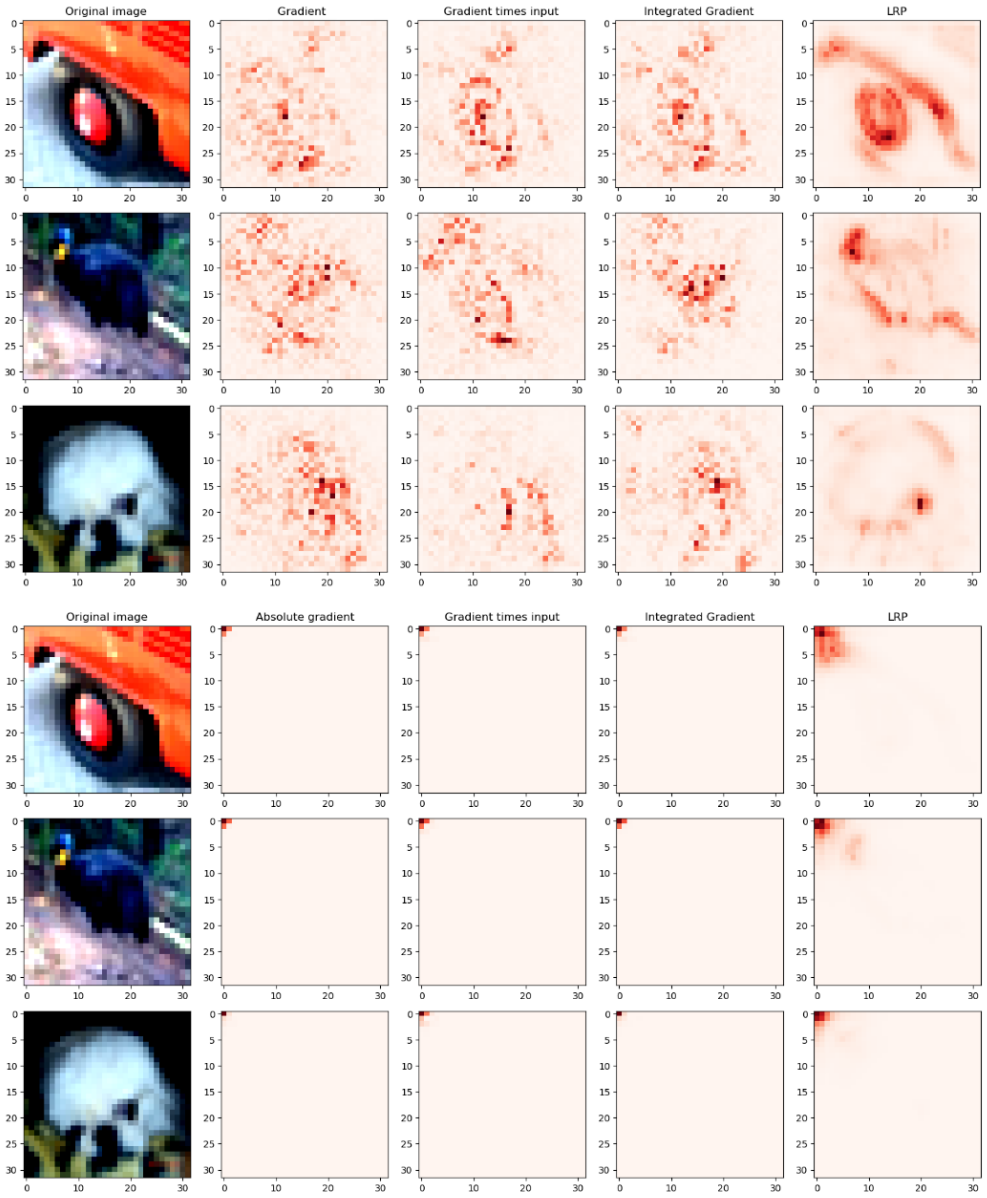

Figure .1: Manipulation of model F^{G}_{com} (bottom) compared with baseline model F_{acc} (top).

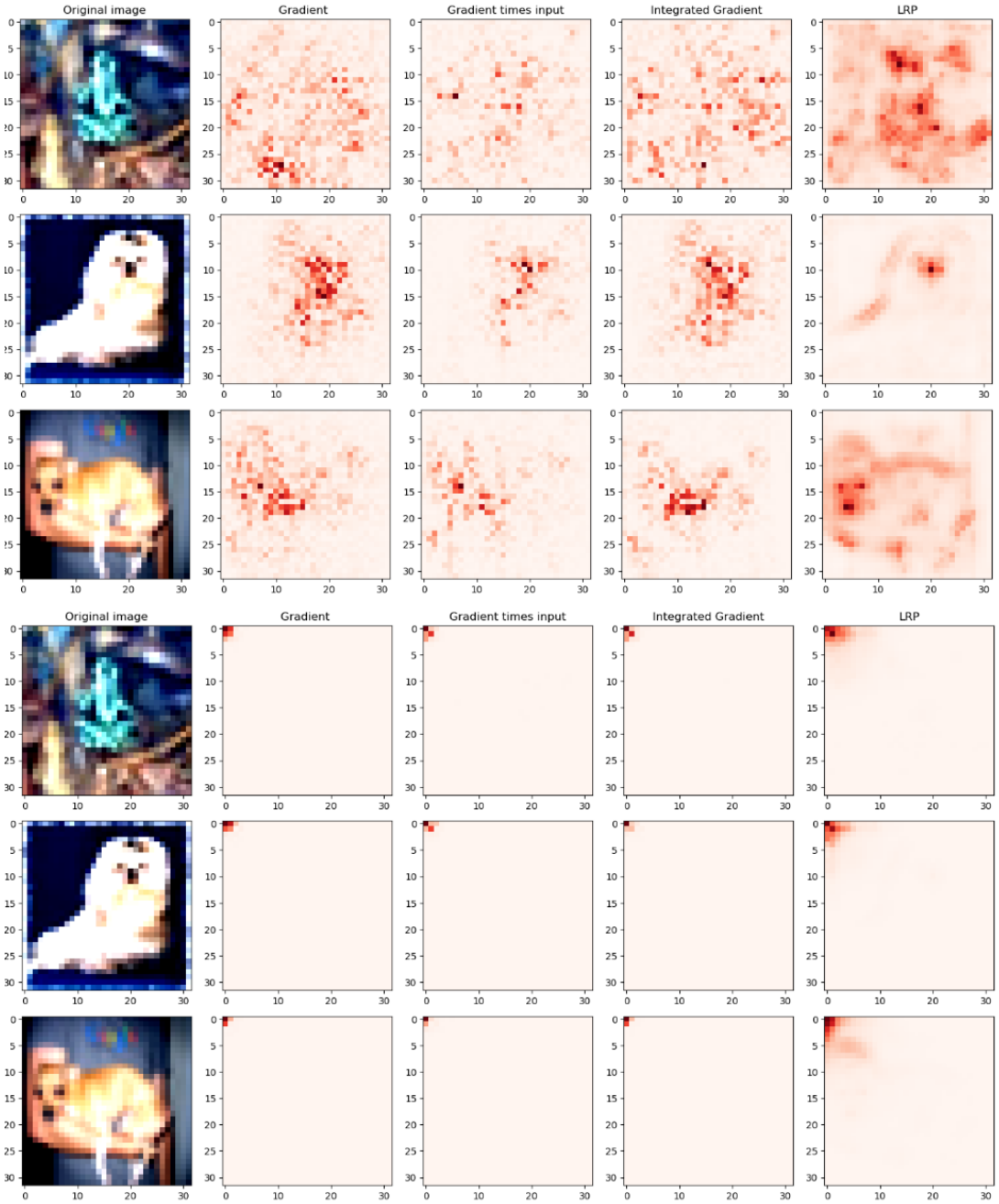

Figure .2: Manipulation of model F^{GI}_{com} (bottom) compared with baseline model F_{acc} (top).

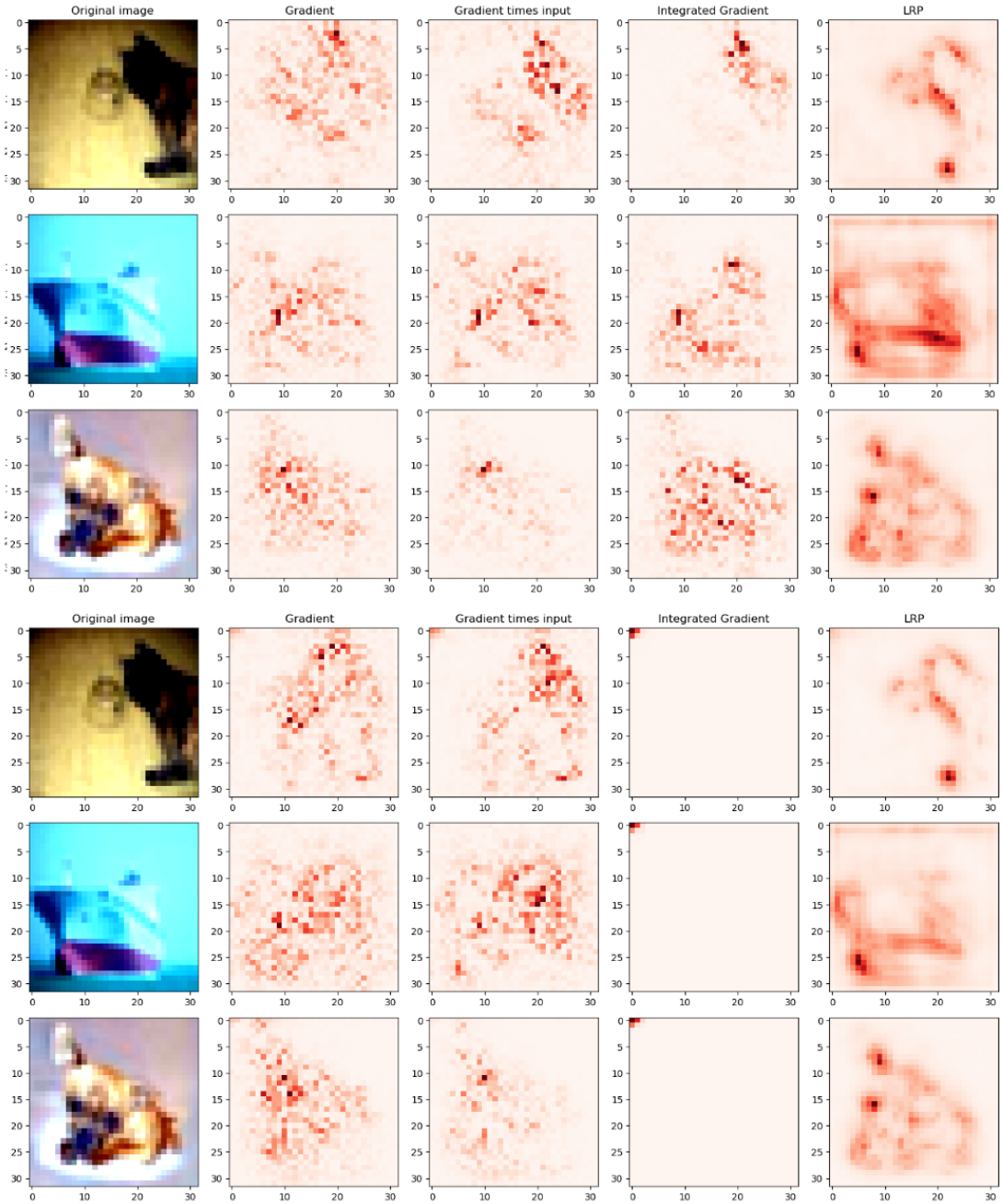

Figure .3: Manipulation of model F^{IG}_{com} (bottom) compared with baseline model F_{acc} (top).

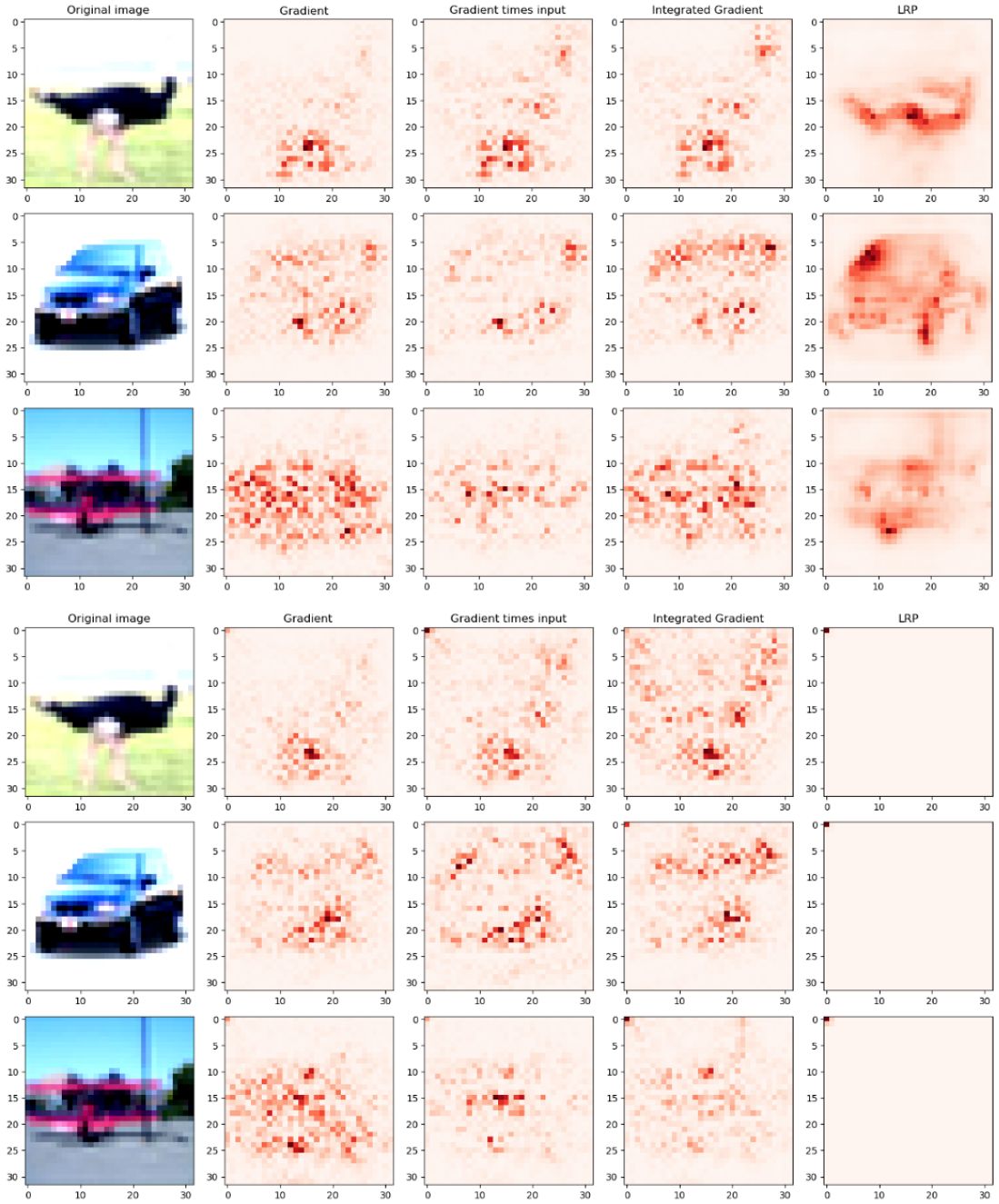

Figure .4: Manipulation of model F_{com}^{LRP} (bottom) compared with baseline model F_{acc} (top).

B Target heatmap

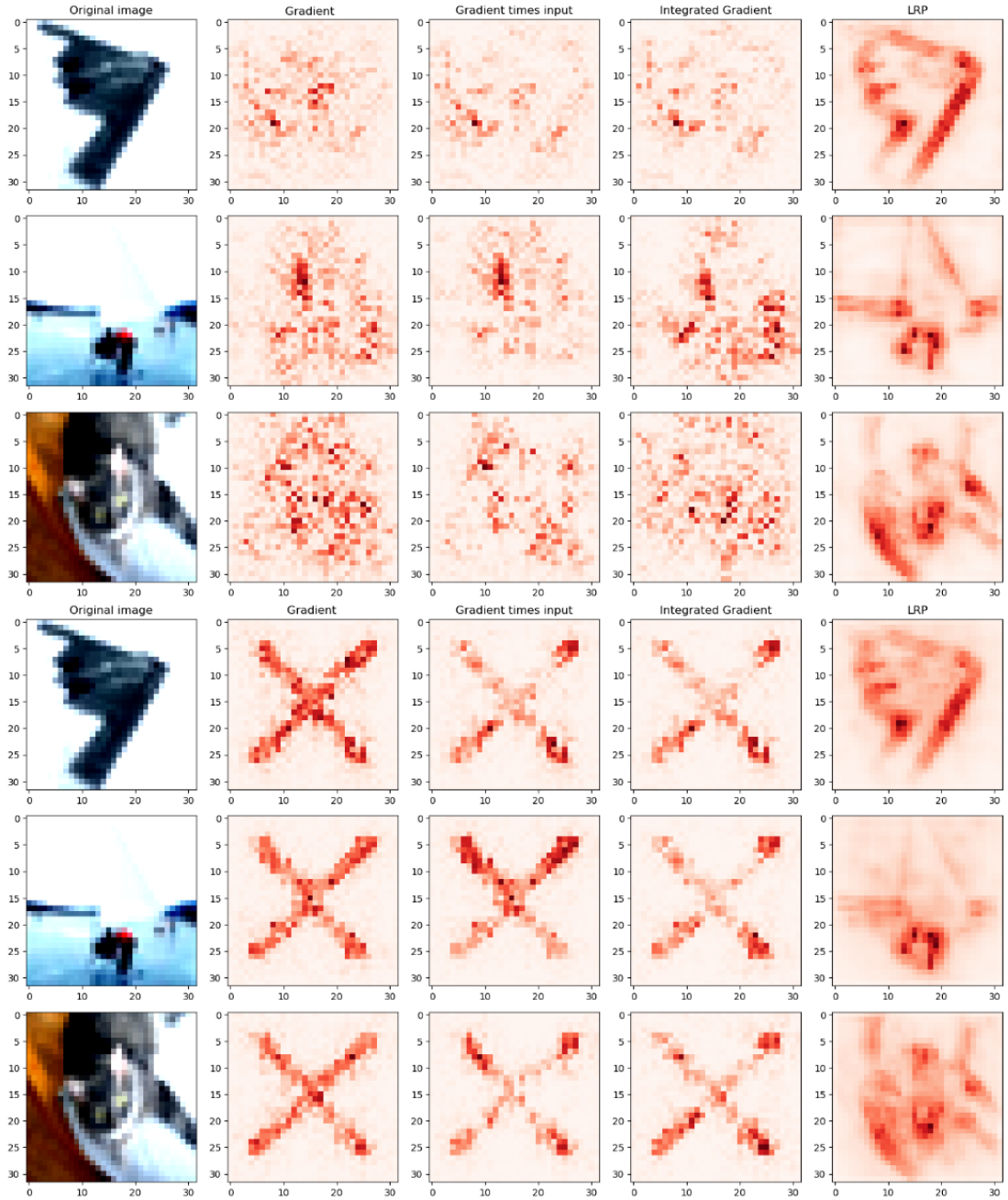

Figure .5: Manipulation of model F^G_{th} (bottom) compared with baseline model F_{acc} (top).

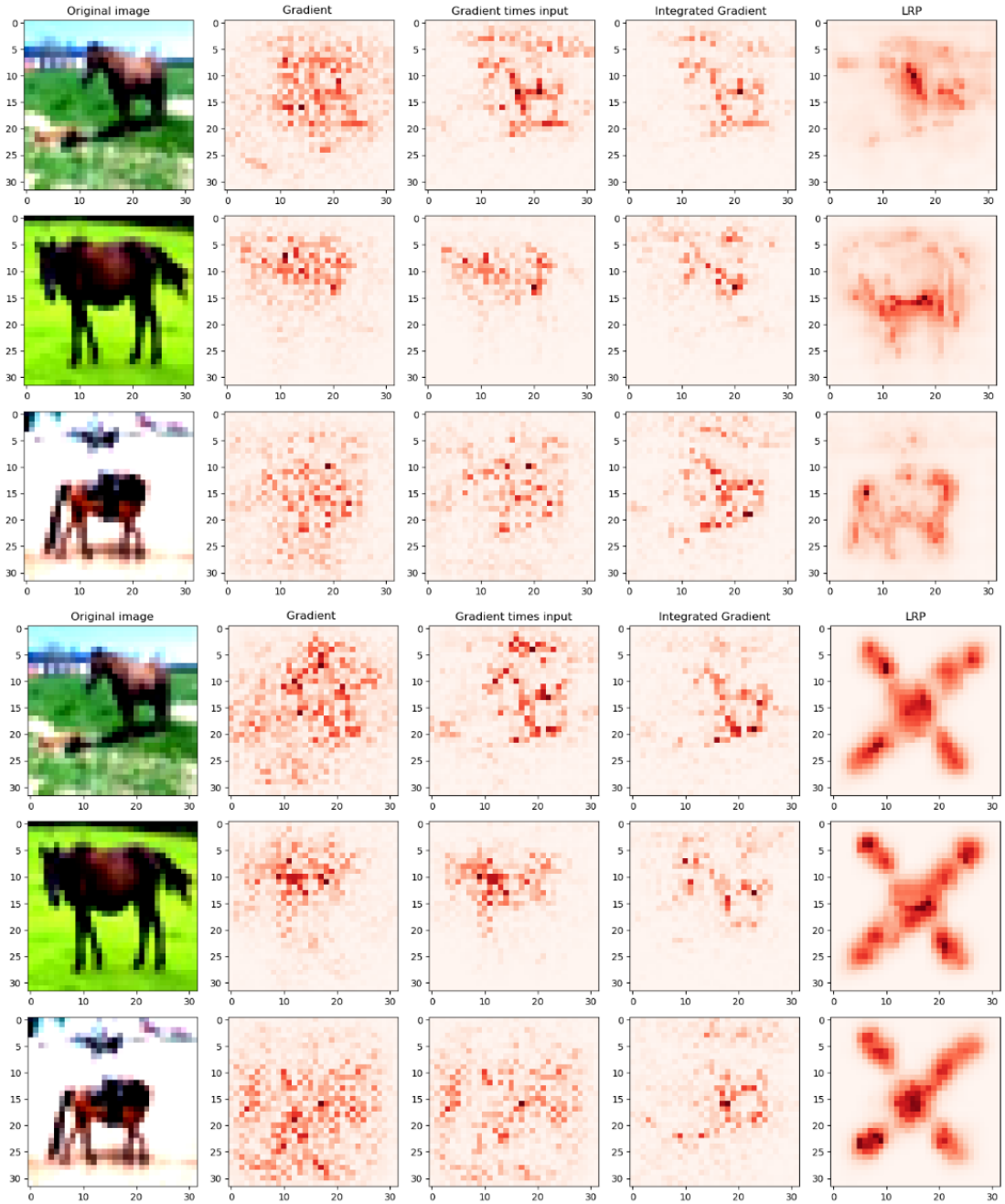

Figure .6: Manipulation of model F_{th}^{LRP} (bottom) compared with baseline model F_{acc} (top).

C Added noise to weights

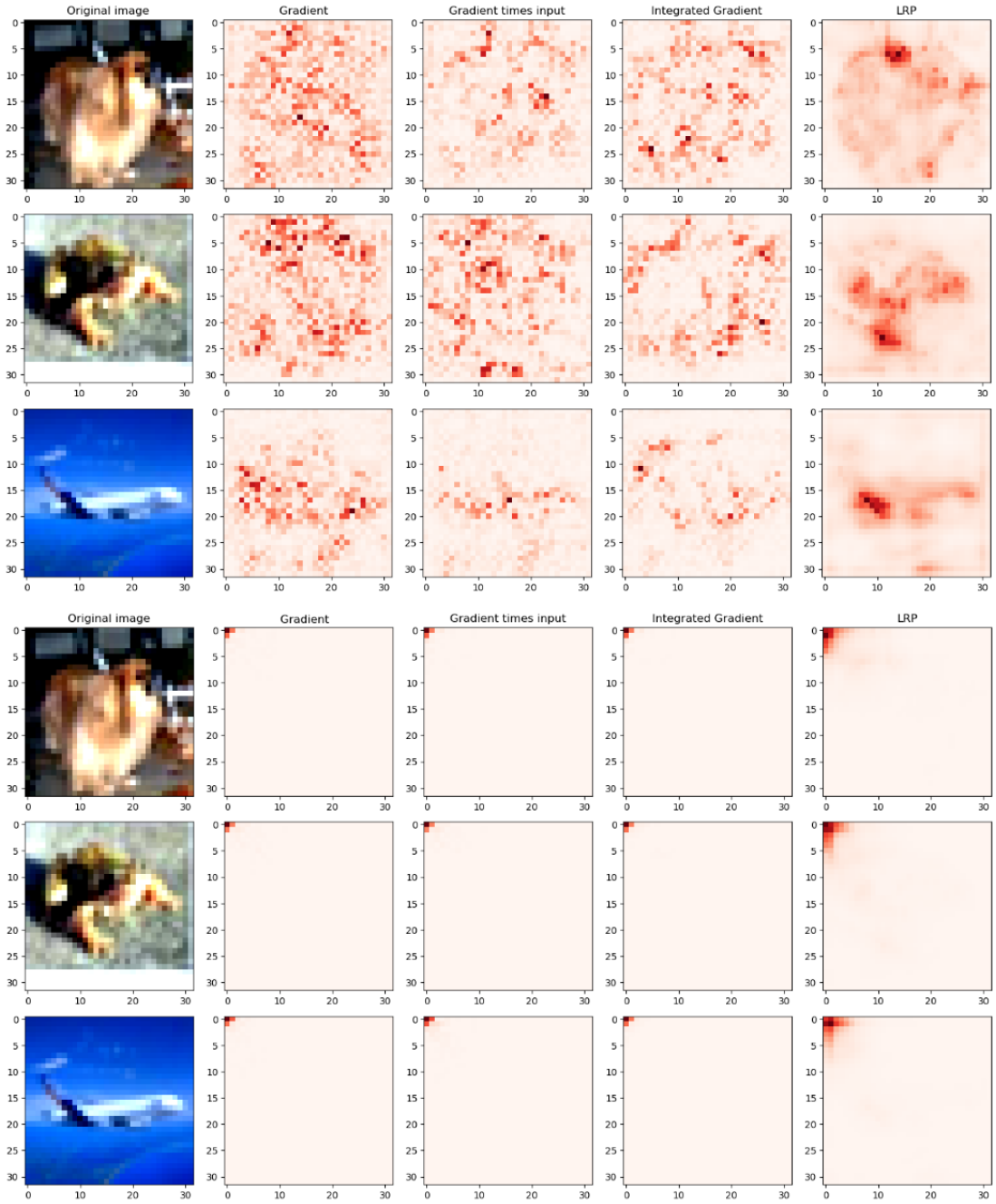

Figure .7: Adding Gaussian noise $\mathcal{N}(0, 0.02)$ to the model parameters of model F^{G}_{com} (bottom) compared with baseline model F_{acc} (top).

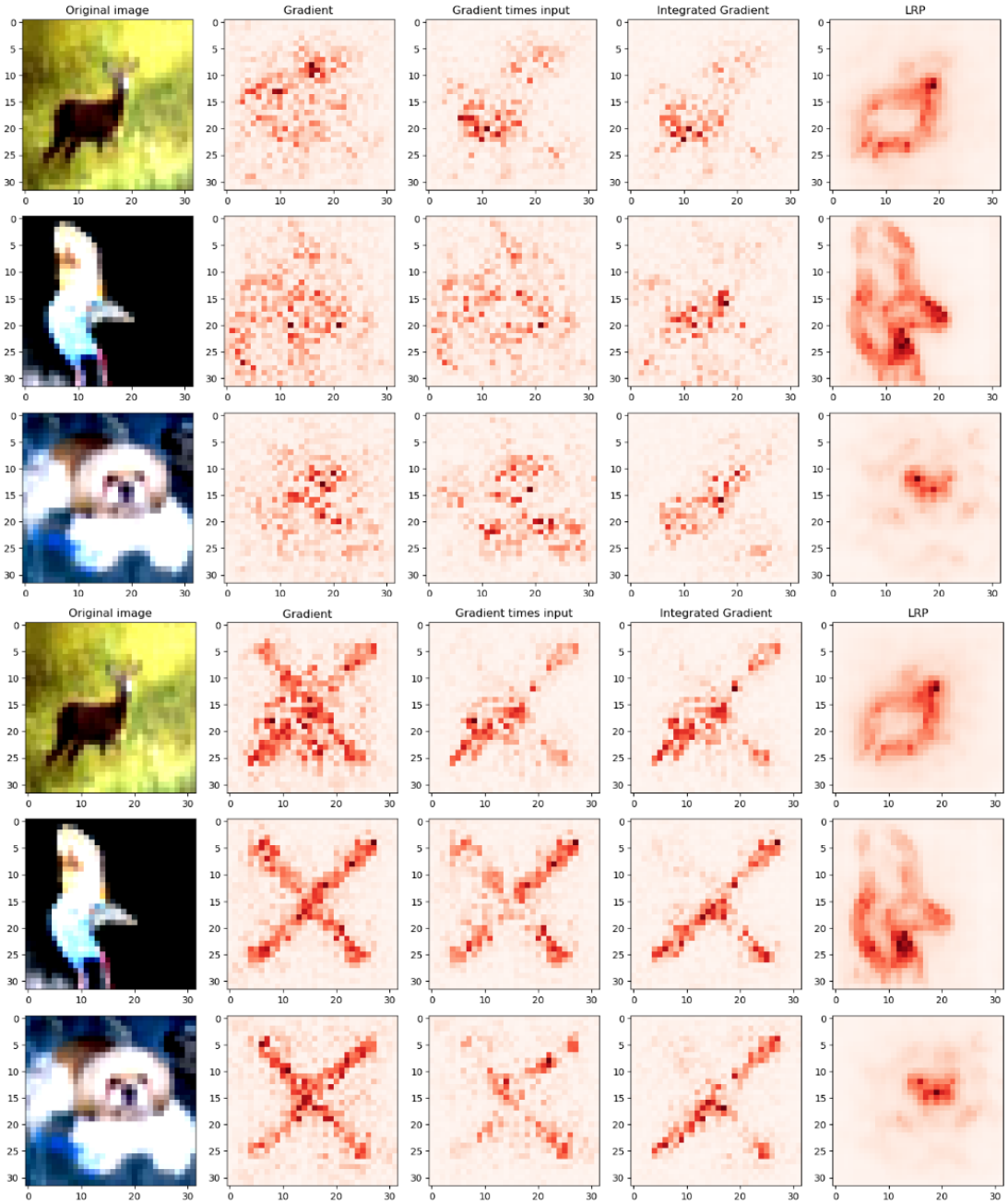

Figure .8: Adding Gaussian noise $\mathcal{N}(0, 0.01)$ to the model parameters of model F_{th}^{G} (bottom) compared with baseline model F_{acc} (top).

D Occlusion

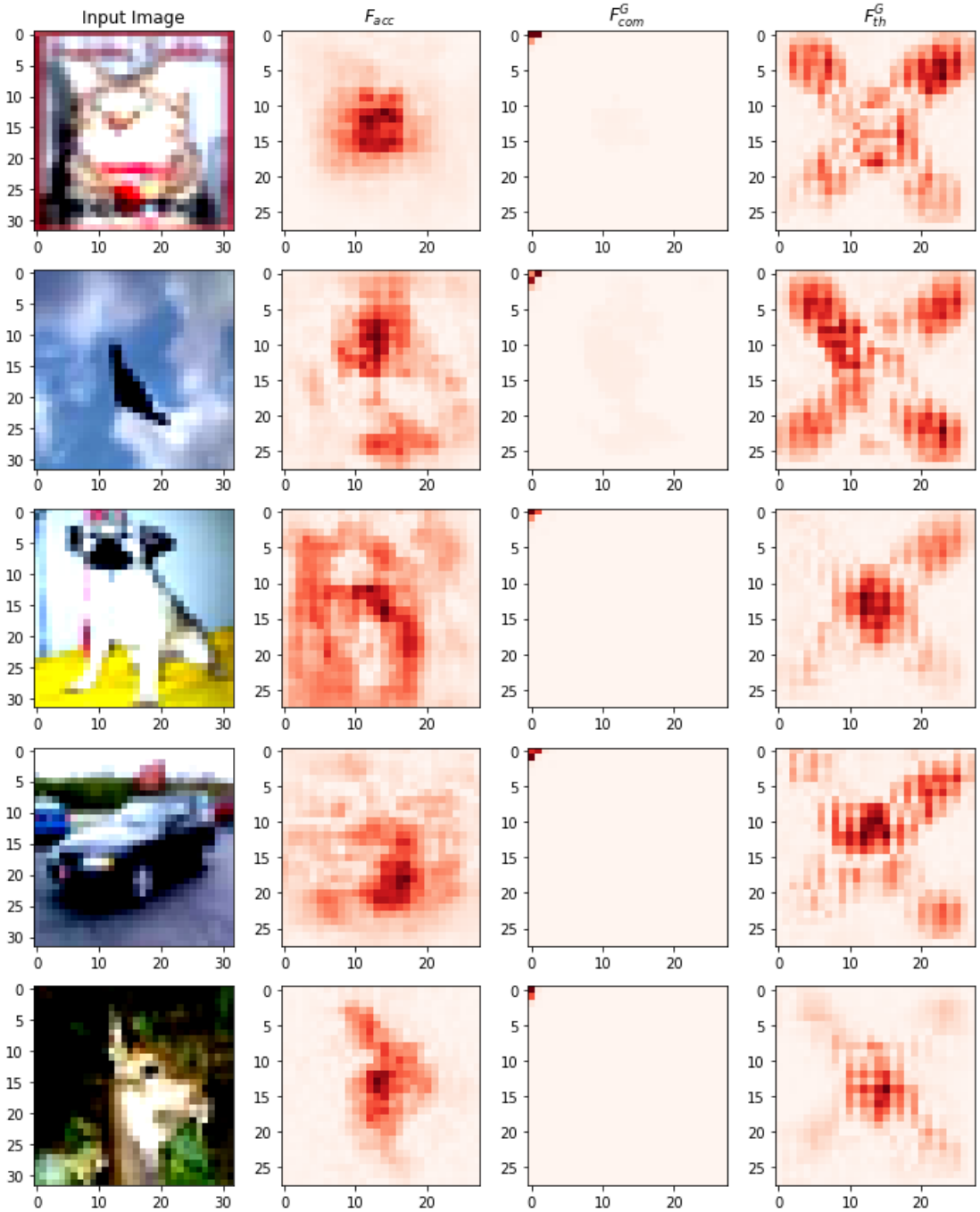

Figure .9: Occlusion heatmaps for different models.