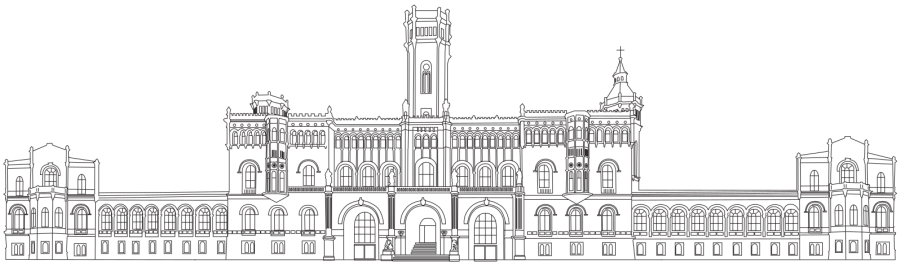
Comparing Convolutional Neural Network in a Rashomon Set using Explanations

Prathep Piremkumar
Institute of Distributed Systems – Knowledge Based Systems
Leibniz University Hannover



First Examiner: Prof. Dr. Avishek Anand
Second Examiner: Prof. Dr. techn. Wolfgang Nejdl

A Thesis submitted for the Degree
Master of Science

1st March, 2022

Erklärung der Selbständigkeit

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden, alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind, und die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen hat.

Hannover, 01.03.2022

Prathep Piremkumar

Abstract

Deep learning neural networks achieve great performance in image classification tasks. To measure the performance, a validation set is used to estimate the performance on unseen test data. A set of different models with similar performance on the validation set is called Rashomon set. Even though the performance of the validation set are similar the reasoning behind the decision may differ. Unfortunately, deep neural networks are a black box models, where the reasoning behind a decision is not clear. In this thesis we compare these black box models and aim to differentiate models which are right for the right reasons and models which are right for the wrong reasons.

We examine whether different reasons between models may be found by using extremal perturbations masks, which highlight the most important part behind a models predictions. We compare the similarity of masks from different models on the same instance. We find that images with decoy can be found using this method if we compare a decoy model with non-decoy models. However, some images without decoys have similar properties as images with decoys.

Another method explored in this thesis is by using influential instance. Influential instances are training instances which are important behind a decision for a validation instance. By comparing these influential instances across models we want to show the difference behind the reasoning behind a decision. Similar to the previous approach, images with decoys can be detected, but images without a decoy can have similar properties as images with decoys.

We conclude that in certain scenarios explanations are useful to differentiate models that are right for the right reasons from models that are right for the wrong reasons.

Contents

1. Introduction

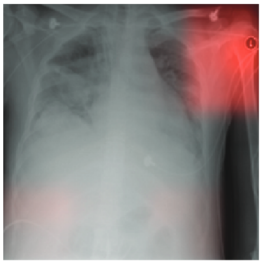
Deep Neural Network (DNN) models are black-box models because the complexity and the large number of parameters make it hard for a human to understand the reason behind the model's output. DNNs have been applied in critical domains. For example, Convolutional Neural Networks (CNN) have been used to detect cancer based on histopathological images [17] or pneumonia from X-ray scans [18].

To evaluate these models a validation set is used. In practice, researchers and practitioners train multiple models and select the model with the best performance on the validation set for deployment, based on common metrics like accuracy or F1-score. This results in a set of trained models, and often multiple models from this set have a similar performance on the validation set.

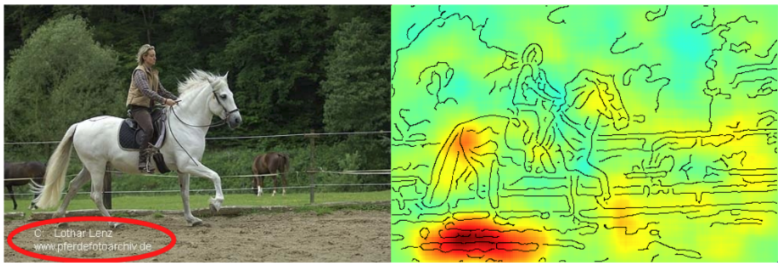
A set of models with the same performance on the validation set is called Rashomon set [3]. However, considering different models from this set, one model can still be superior if it predicts the correct class for the right reason while others might predict the correct class for the wrong reasons. So a good performance on the validation set is not enough to deem a model good. For example, recent work showed that a model for pneumonia predictions has a great performance on the validation set but did not perform well when images from other hospitals were used. This is because the model learned to predict pneumonia based on a hospital token in the images and not based on the lungs. Figure 1.1a shows this behaviour. This is an example of a right for the wrong reason behavior of a model. While the model correctly predicted the label of an image, it used features that are not intended to be used by the developer. Instead, it used a shortcut rule for the prediction.

Another example is presented in figure 1.1b [1], where a model predicts a horse based on a copyright tag. The copyright tag is not a feature of a horse. Therefore, it is a wrong reason.

Explanations visualize the reasons behind model predictions. There are multiple methods of explanation to visualize the reason behind a decision of a model. For example, saliency maps or heatmaps visualize the area a model focuses on. Explanations can help humans understand the reasons behind a model, making it easier to decide whether the reason is correct. There is no perfect explanation method. Each method has advantages and disadvantages depending on the use case. Some popular explanations methods are gradient [24], LIME [19], SHAP [13], Integrated Gradients [27], guided backpropagation [26], Grad-CAM [23] and RISE [15]. Another method is extremal perturbation [4]. It calculates a binary mask that is put on top of the image. The binary mask only preserves the most important parts of the image.



(a) X-ray image of a lung with pneumonia where the hospital token is the reason for the prediction.(Figure from [6])



(b) Left: A horse image with a copyright tag; Right: The reasoning behind the prediction. dark Red means high impact on the prediction and green means low impact.; The dark red at the same position indicates the model is using the copyright tag for the prediction.(Figure from [1])

Figure 1.1.: Images and the heatmap explaining the reason for the predictions. The heatmap shows that wrong reasons are used for the predictions.

Figure 1.2.: Left: Images of a tench.; Middle: Extremal perturbation mask from a model with correct reason, because the mask reveals parts of the tench.; Right: Extremal perturbation mask from a model with wrong reason

This means the image's parts that lead the model to its prediction remain visible, while unimportant parts are masked.

Another type of explanation is explanation by example, like using influential instances. An influential instance is a training instance that has a big impact on a model prediction. Comparing the influential instances of two different models on the same query image should show differences if the models learned different features and are similar if the models learned the same feature. To calculate the influence score, the dot product of the gradients with regard to the model parameters can be used. Pezeshkpour et al. [16] showed that the resulting influence score is similar to older methods.

This thesis focuses on the problem of finding the best model from a Rashomon set of models. The goal is to choose the model that makes predictions that are right for the right

reasons. We are exploring whether explanations may be useful to achieve the goal.

Evaluating whether a method can distinguish a model that learned some shortcut rules is difficult because the ground truth behind a model's decision is unknown. Therefore, to establish some ground truth, decoys can be used to train decoy models. A decoy is a feature on an image that is present on images in one class and not part of the class. For example, a copyright tag can be used as a decoy. A decoy model is a model trained to predict a class based on a decoy and, therefore, right for the wrong reasons. A model may learn that a copyright on an image means that a certain class should be predicted.

We create a set of models where one model is right for the right reason, and one is right for the wrong reasons. In this thesis, we train multiple decoy models. We start with a simple decoy, a copyright tag. We train a ResNet50 on an ImageNet subset consisting of 10 classes where in only images from one class contain the copyright tag. We verify that the decoy was learned using the accuracy on images with and without a decoy present. We also train a decoy model using a more realistic decoy, a stop sign and show that even a more complex decoy can be learned.

In this thesis, we use two different explanations,and compare per-instance level and across examples. Per-instance level explanations are explanations that only explain a single instance. Across examples, on the other hand, uses other instances for the explanations.

To examine the utility of comparing explanations, we use this evaluation setup in a series of experiments:

1. **RQ1**: Can the similarity of explanations be used to identify images containing decoys?

   In the first experiment, we compare the masks' similarities from two models. We calculate the extremal perturbation mask of the same image from different models and compare them. A high similarity is given if the masks reveal the same pixels. Different masks may suggest that the models learned different features.

   We find that masks created by a decoy model and a non-decoy model have low similarity if the image contains a decoy. However, some images create masks with low similarity even if there is no decoy present. This results from objects that are big enough to have two masks with low similarity

2. **RQ2:** Do extremal perturbation masks transfer across different models?

   The second experiment examines how much worse a model performs if the model is using the mask created by another model. We compare the drop in the softmax score if the mask created by one model is replaced by the mask from another model.

   The softmax score drops a lot if the image contains a decoy. However, there are some images where the softmax score also drops a lot, even when there is no decoy present.

3. **RQ2:** Can the influence score be used to identify images containing decoys?

In our third experiment, we create a ranked list by sorting the training instance with the highest influence score at the top and the training instance with the lowest influence score at the bottom.

Using precision, we show that if a decoy is present on an image, the decoy model creates a ranked list where the top images contain a decoy. On the other hand, the non-decoy model creates a ranked list where the top images contain the decoy with the same ratio as the whole training set. Comparing the ranked list created by two decoy models show that two decoy models do not create ranked lists with a higher similarity score than on images from classes with no decoy. However, the intersection of the top images from the two ranked list is higher than on images from classes where no decoy is present. We also show that the ranked list created by a decoy model and a model trained without a decoy has a lower than median similarity score when a decoy is present. This said, both methods, ranked similarity and top intersections, may indicate that one method learned the decoy there are non-decoy images with a similar result than images with a decoy.

The methods examined in this thesis can indicate that a decoy may be present, but some images without a decoy have similar properties as images with decoy.

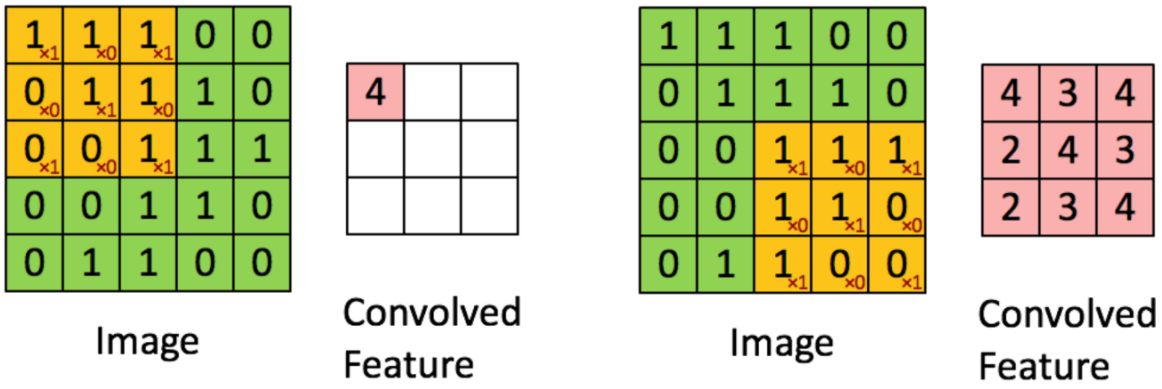
2. Fundamentals

In this chapter, we introduce fundamental background knowledge required to understand the following chapters.

2.1. Convolutional Neural Network

To use a fully connected neural network on an image, the image has to be flattened into a one-dimensional vector. This means spatial information is lost. For images, you can gain more information about a pixel by observing pixels next to it. Pixels which are farther away are usually less useful. Convolutional layers are designed to utilize the spatial grid structure of an image.

A convolution layer consists of multiple kernels. A kernel slides over an image and performs a matrix multiplication as seen in Figure 2.1. The output is called a feature map. The values of a kernel are learned by back-propagating.



(a) Calculating the first element of the output matrix.

(b) Calculating the last element of the output matrix.

Figure 2.1.: Applying a 3 kernel on an 5×5 matrix. Yellow highlights which elements of the input matrix is used to calculate the output element. The kernel values are written in Red. (Figure from [22])

A convolution layer uses the spatial information of an image. It uses pixels near each other to calculate an element of the feature map. A convolution layer can also be applied to a feature map. After the feature map is calculated, an activation function, like ReLu, is applied.

A further desired property is translation invariance. Moving an object in an image should have a low impact on the neural network. Because a convolution layer only uses adjacent pixels, moving an object in an image means the value of the feature map is also moved but not changing.

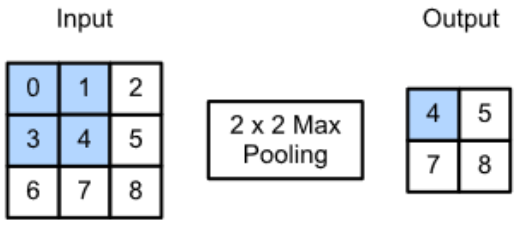


Figure 2.2.: 2×2 max pooling applied to a 3×3 matrix.(Figure from [29])

Another important part of a CNN is pooling layers. Pooling layers reduce the spatial size of a feature map. Two commonly used pooling methods are max pooling and average pooling. Similar to a convolution layer, a window is moved across a matrix. On the elements inside the window, an operation is applied. For example, Figure 2.2 shows a max pool operation. A max pool uses the highest value inside the window is the result.

A conventional neural network usually starts with a convolutions layer followed by a pooling layer. This may be repeated multiple times. The result will then be flattened and will be used as an input to at least one fully connected layer.

2.1.1. ResNet Architecture

Residual network (ResNet) [7] is a popular CNN architecture [28]. There are multiple versions of ResNet, like ResNet18 and ResNet50. The number denotes the number of layers used. The basic architecture between these versions is the same.

ResNet solves the problem of degradation. Degradation is a problem where adding more layers decreases the performance. This is not caused by overfitting since the training error also gets worse [7]. Deeper neural networks are more difficult to optimize, which causes the degradation problem. ResNet solves this by adding skip connections. Skip connections add an old output to a new output. Figure 2.3 shows skip connections. The beginning of an arrow is the old output, which is added to the output at the end of an arrow.

Figure 2.3 shows a the architecture of ResNet18. It is divided into 4 ConvNets layers.

2.2. Rashomon Sets of Models

A set of models with the same performance on the validation set is called Rashomon set [3]. Having the same accuracy on the validation set does not mean that both models learned the same thing. Being right for the right reason means a model learned to predict a class based on features the object of the class has. For example, zebras have stripes. So a model

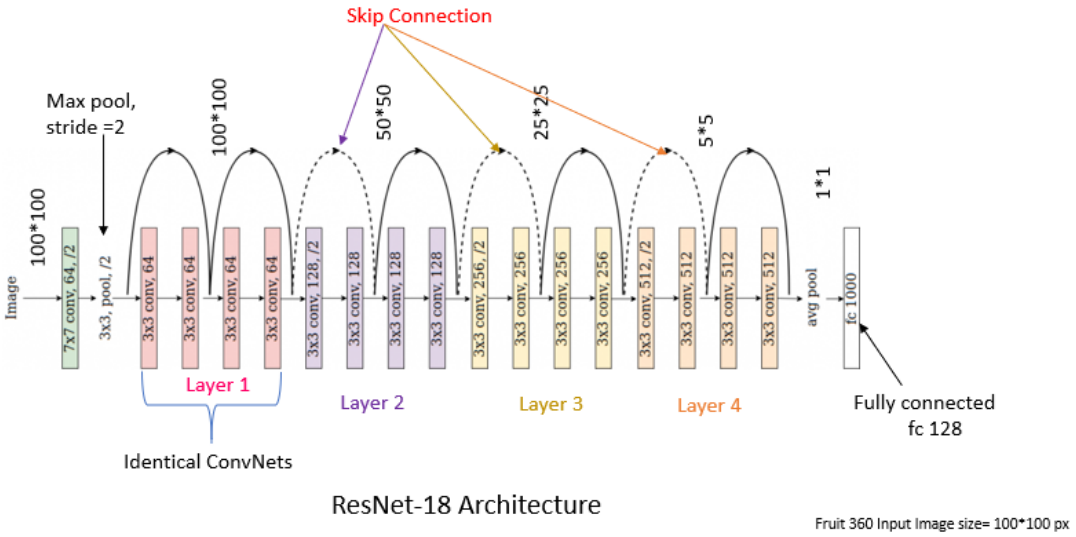


Figure 2.3.: Architecture of a ResNet-18 (Figure from [25])

learning to predict a zebra may learn that black and white stripes are an indicator for a zebra. This model is right for the right reason.

Another model may have learned that zebras have a unique head shape and therefore predicts zebra if the head shape is present. The head is still a feature of a zebra. So this model is also right for the right reason.

Yet another model learning zebra may realize that zebra images are the only ones with trees in the images. So the model may learn to predict zebra based on tress. Trees are not features of a zebra. So this model is right for the wrong reason.

All three models may have similar performance on the validation set. However, only the first two models are right for the right reason.

It is also possible to specifically train those models [20] by regularizing the training data. For example, you train a model, and it predicts zebra based on black and white stripes. The next model will use a modified training set by blurring the zebras' stripes. By blurring the reason of a model, the next model has to find another reason. You repeat this until the accuracy on the validation set drops significantly. Now you created a Rashomon set where the models have different reasons. Some models may even have learned a wrong reason.

2.3. Interpretability

A black box predictor such as a deep neural network takes an input and produces an output without giving a reason. For example, an image classifier uses an image as an input and creates a label as output without providing a reason. Interpretability tries to find and communicate the reasons behind the decision of a model. It helps to discover bias and

discriminatory behavior of a model and improve these problems. In some cases, an error in a machine learning model can be fatal. A self-driving car with the wrong reason can lead to deaths in the real world. Interpretability can also increase the social acceptance of machine learning and justify its usage. Humans are more likely to trust a model whose decisions are based on human-understandable facts than just a black box model where even its designers do not understand why the model is correct. Lastly, interpretability can also be used to gain knowledge. The interpretability of a model which classifies a bird species can show how to differentiate between different bird species.

Molnar [14] uses the following taxonomy to classify different machine learning interpretability methods.

Firstly, whether a method is intrinsic or post-hoc. An intrinsic method achieves interpretability by restricting the complexity of a machine learning model. For example, short decision trees are considered intrinsic interpretable due to their simple structure, which a human can easily analyze. On the other hand, post-hoc methods are applied to already trained models to analyze it. For example, this can be achieved using permutation or perturbation on the input feature. Post-hoc methods can also be misleading [12] as it is possible to optimize a model to have a misleading explanation.

Another criterion to classify interpretability methods is whether the method is local or global. A local explanation tries to explain the reasons behind the prediction for a single instance. For example, a model predicts zebra because the bottom left corner of the image contains black and white stripes. A global method explains the the model predictions over an entire dataset. An example explanation could be: If there is a lot of red in the images, it is a stop sign.

Methods can also be model-specific or model-agnostic. A model-specific method only works on certain types of models, and model-agnostic works on any machine learning model. For example, a method using the gradient of a model can not be used on a decision tree. A method that only perturbs the input features can be used on any machine learning model.

In this thesis, we focus on local, post-hoc explanations.

2.4. Perturbation-based Explanations

Local explanations try to visualize the reason for the prediction of a single instance. Local explanations for an image classifier try to show what region of an image is responsible for the model's prediction. Multiple methods use perturbation for local explanations. To create an explanation, a perturbed input is chosen, and the change in the output is observed.

A basic approach is occlusion. Occlusion switches a pixel or a region of pixels to black and uses the images as the input to the function. Comparing how much each region affected the correct output can show which regions are the most important. Figure 2.4 shows an example.

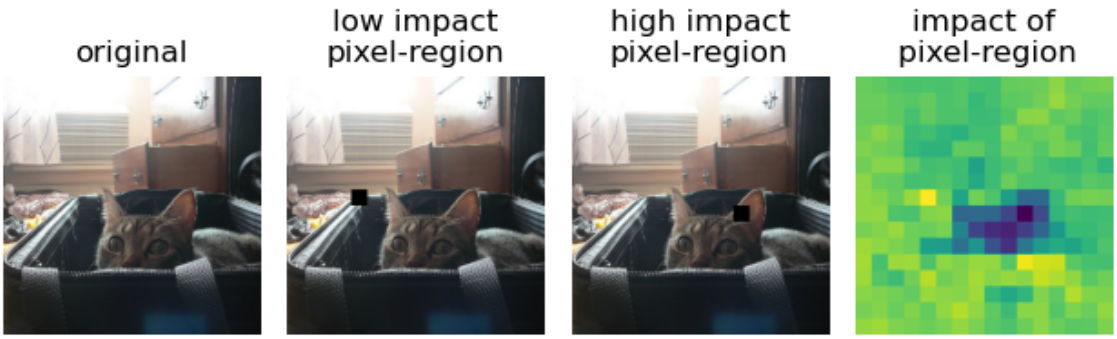


Figure 2.4.: An example of occlusion; the second image shows a region of pixel turned black with little impact, the third image shows a region of pixel turned black with high impact; fourth image shows the impact of a region of the image

2.4.1. Other Perturbation-based Explanations

Occlusion is a simple method that does not consider that features may depend on each other. More complex perturbation-based explanations are LIME and SHAP.

2.4.1.1. Local Interpretable Model-Agnostic Explanations (LIME)

One method is Local interpretable model-agnostic explanations (LIME). It uses a local surrogate model. The surrogate model is an interpretable model with a similar input-output relationship on a single instance x and perturbed versions of x.

The dataset to train the surrogate model is created by selecting an instance x. Afterward, a perturbed version of x is used as an input to create the prediction of the black-box model. The samples are weighted by their proximity to the original instance x.

Mathematically, the optimization can be expressed as:

$$\text{explanation}(x) = \underset{g \in G}{argmin} L(f, g, \pi_x) + \Omega(g)$$

with x as the instance to be analyzed, f as our black-box function, g as our surrogate model, G of all possible surrogate model, π_x defines how much perturbation is allowed, and $\Omega(g)$ describes how complex the surrogate model is.

The first part of the equation optimizes the similarity between f and g. The second part keeps the complexity of the model as low as possible. In practice, only the first part is optimized, and the second part is chosen by the user.

After the surrogate model is trained, it can be used to explain the instance x.

Figure 2.5 show the explanations of bread by an Inception V3 neural network. The samples are created by grouping the pixels into bigger pixels called "superpixels". The superpixels are grouped as similar colored pixels in the same region. There are either kept or turned to gray.

Figure 2.5.: Lime applied to bread; left the original image; middle lime explanations for bagel; right explanation for strawberry; green area increase probability, red decreases probability (Figure from [14])

One advantage of LIME is that the method works for every model and can be used for either tabular data, text, or images. The explanation is also relatively easy to understand.

On the other hand, defining the neighborhood of an instance is a big problem. It can result in unlikely data points. LIME can also create different explanations for the same instance and be manipulated to hide biases.

2.4.1.2. Shapley Additive Explanations(SHAP)

SHAP (SHapley Additive exPlanations) is a method based on shapley values. Shapley values estimate how much impact a feature has on the prediction. SHAP represents these shapley values as an additive feature attribution method using a linear model. SHAP calculates shapley values by calculating $f(z')$ with only certain features present. A combination of features present is called a coalition. The explanation model can be described as:

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j$$

where g is the explanation model, $z' \in \{0,1\}^M$ the coalition vector, M the maximum coalition size, ϕ_0 the mean value, and ϕ_j the shapley value of feature j. The coalition describes which features have been omitted. A one means the feature is present and 0 represents the feature is absent.

SHAP has three desirable properties:

- **Local accuracy**

$$f(x) = g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j x'_j \tag{2.1}$$

  where $x' = 1^M$. If all features are present the g should create the same output as f.

- **Missingness**

$$x'_j = 0 \Rightarrow \phi_j = 0$$

A feature not present in x means that the shapley values have to be 0.

- **Consistency**

  Let $f_x(z') = f(h_x(z'))and z'_j$ indicates that $z'_j = 0$ and h_x is function transforming a binary input to an valid input of f. Consistency means:

$$\begin{aligned} & f'_x(z') - f'_x(z'_j) \geq f_x(z') - f_x(z'_j), \text{for all } z' \in \{0,1\}^M \\ \Rightarrow \quad & \phi(f', x) \geq \phi(f, x) \end{aligned}$$

  This means if a feature in one model always has more positive impact than in another model, the shapley values have to be the same or bigger.

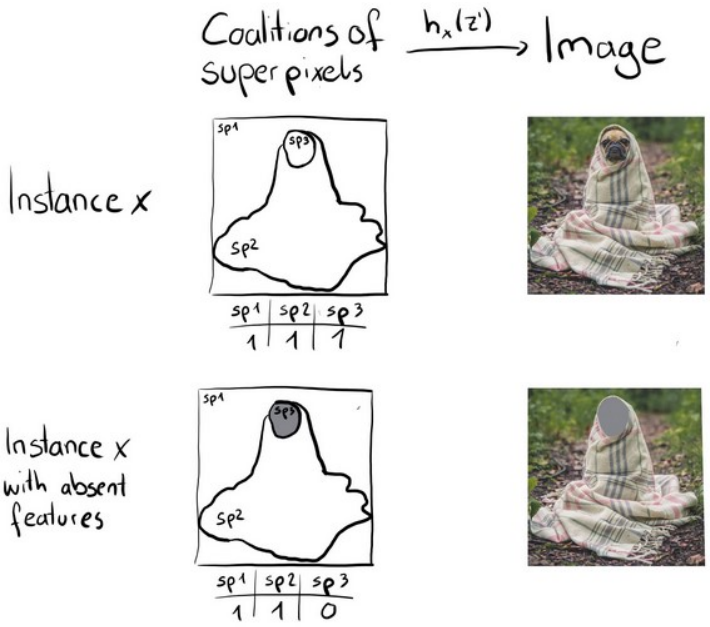


Figure 2.6.: An example of h_x in an image. (Figure from [14])

Figure 2.6 shows how SHAP can be used on images. You group pixels into superpixels. A superpixel is either present or not. If a superpixel is not present, it will be colored in gray.

2.5. Meaningful Perturbation

Previously mentioned approaches use a heuristic method to create a visualization without a clear meaning. Fong and Vedaldi [5] proposed a way to answer the question of what a black box function f has learned. It also considers how features interact with each other.

The perturbation can either be used to keep only the most important part of the image, called the preservation, or remove the most important part, called deletion.

The goal in a deletion game is to find a mask $m*$, which reduces the score $f_c(x; m*)$ to a minimum while the mask is as small as possible. The masks consist of values between 0 and 1. A high number indicates a high level of blurriness. In this context, small means the number of zeros should be as high as possible. $f_c(x_0; m)$ is the output score of predicting class c with an input image x_0 and a mask m.

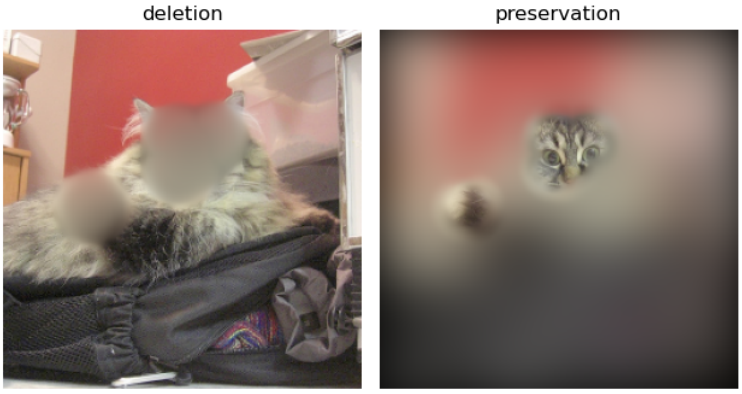


Figure 2.7.: Left: mask created by deletion; Right: masked created by preservation

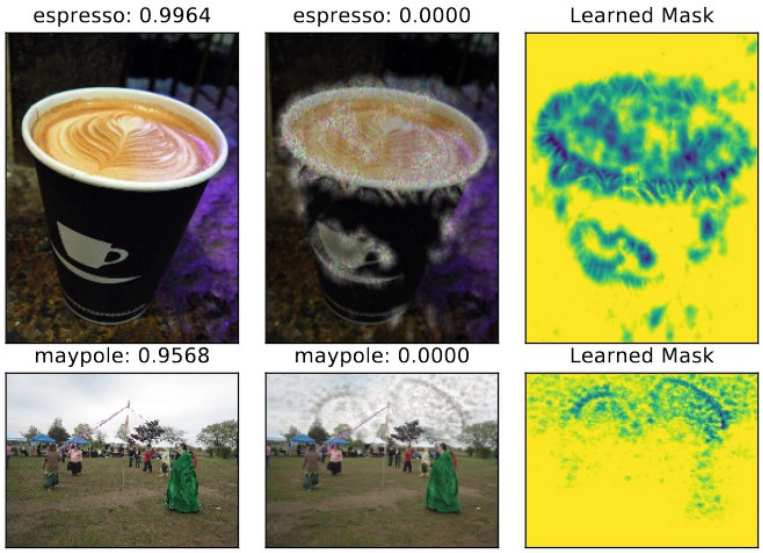


Figure 2.8.: left: correctly classified image; right: masked learned; incorrectly classified image, because of artifacts. (Figure from [5])

However, this method leads to an unrealistic image, as shown in Figure 2.8. The masks should create images that are closer to real-world images. The masks should not rely on small details but the general area. Therefore, the mask is added with a small random jitter. Furthermore, it is upsampled from a smaller mask to make it more realistic. The loss function, optimizing the mask parameters, for the preservation game is

$$m_{\lambda,\beta} = \underset{m}{argmax}\, f(x \otimes m) + \lambda ||m||_1 - \beta S(m)$$

The first part $\underset{m}{argmax}\, f(x \otimes m)$ maximizes the score of the correct class. The other parts are regularizing. The first regularizing part, $\lambda||m||_1$, keeps the mask small. The first regularizing part,$||m||_1 - \beta S(m)$, includes the previously mentioned part to keep the mask smooth and realistic. λ and β are giving different weights to the parts of the equation.

Figure 2.9.: Deletion game on a image classified as flute, softmax score above image, blue indicates high blur, yellow indicates low blur. (Figure from [5])

Figure 2.9 shows an image and a mask created with the algorithm. It shows a perturbation of the image where we apply Gaussian blur on the most important part of an image according to the classification model. After applying the Gaussian blur, the probability of a flute dropped from 99.73% to 0.0007%.

The meaning of the mask can be derived from the above equation. The mask creates a big, smooth mask while keeping the most important region of an image. However, having three different optimization problems in one equation is quite problematic.

2.6. Extremal Perturbation

A perturbation is called extremal if there is no other perturbation with the same size that has a bigger influence on the classification of a model. Ruth Fong, Mandela Patrick, and Andrea Vedaldi [4] proposed a method to create smooth masks for extremal perturbations. This method removes the need for a weighting factor, but requires an area constraint to be specified beforehand as seen in Figure 2.10 where the top left corner shows the specified size. The output mask is used to create an extremal perturbation.

Figure 2.10 shows that the mask created by the algorithm is smooth and found the most important region of the image to be classified as a mousetrap. The mask is near binary. A pixel is either important or not important for the decision made by the classification model.

Figure 2.11 shows the result of extremal perturbation and compares it with other methods. You can clearly see a binary mask preserving relevant parts of the images. The mask shows an image's relevant region more clearly than other methods.

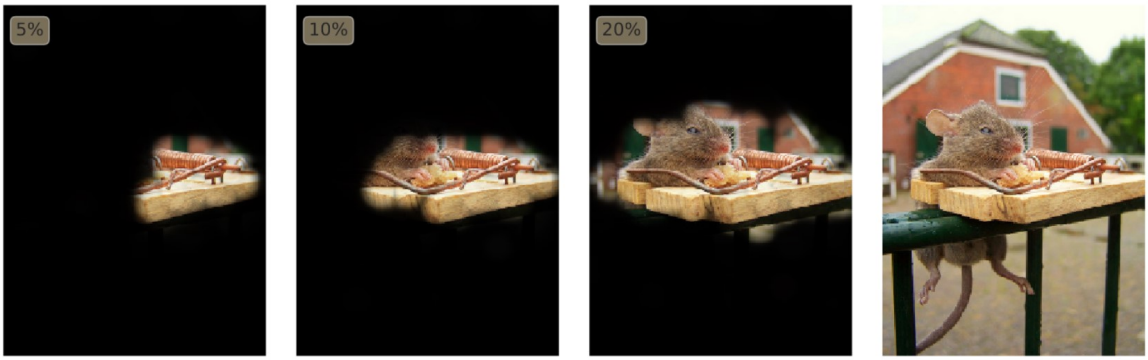


Figure 2.10.: Mouse trap image, extremal perturbation masks generated by the algorithm of Fong, Patrick and Vedaldi, with different area constraint. (Figure from [4])

Figure 2.11.: Comparison of extremal perturbations mask with other methods.(Figure from [4])

2.6.1. Mask Generation

Let $x : \Omega \to \mathbb{R}^3$ be an images that maps a pixel $u \in \Omega$ to an rgb value.

Let $m : \Omega \to [0, 1]$ be a mask. To create a smooth mask, an auxiliary mask $m_{aux} : \Omega \to [0, 1]$ is used. A kernel $k : \Omega \to \mathbb{R}_+, k(u) = \exp\left(\frac{max(0,u-1)^2}{4}\right)$ is used to make the mask smooth. However, instead of using the sum in the convolution operator

$$m(u) = Z^{-1} \sum_{v \in \Omega} k(u - v) m_{aux}(v)$$

the smooth max operator is used

$$\underset{u \in \Omega; T}{smax} f(u) = \frac{\sum_{u \in \Omega} f(u) \exp(f(u)/T)}{\sum_{u \in \Omega} \exp(f(u)/T)}$$

$$m(u) = \underset{v \in \Omega; 20}{smax}(k(u - v) m_{aux}(v))$$

The implementation uses a smaller m_{aux}, which is upsampled afterward.

For optimization of the mask parameter, SGD with momentum 0.9, 800 iterations, and a learning rate 0.01 optimizes the auxiliary mask m_{aux}. It is initialized with all ones.

The first part of the loss function,$f(x \otimes m)$, is the output of the neural network after applying the blur. The goal is to maximize the value of the correct label before the softmax operator is applied.

The second part of our loss function,$\lambda R_a(m)$, is the area constraint. $R_a(m) = \|vecsort(M) -$

$r_a\|^2$, r_a is defined as $(1-a)*|\Omega|$ zeros followed by $a*|\Omega|$ ones with a being the area target. The area constraint is then multiplied with the weight $\lambda = 300$ which is multiplied by 1.0035 each iteration. This algorithm finds

$$m_a = \underset{m \in M}{argmax}\, f(x \otimes m) - \lambda R_a(m)$$

Where M is a set of masks created as explained above.

2.7. Explanation by Example

Another type of explanation is finding influential training instances. Removing an instance of a training instance can affect the resulting model. If the removal of an instance has a big impact, the training instance is considered influential. For a query image, the goal is to find the most influential training instance. The influential instance is considered most influential when there is no other training instance whose removal affects the prediction of our query image more.

One approach for finding influential instances is deletion diagnostics. This approach removes a training instance and retrains the model. It will be repeated for every training instance. Afterward, every model predicts the query images. The removal of the training instance which created the model with the biggest change on the query image is the most influential training instance. This approach is too expensive to be used.

Another approach are influence functions [10]. Influence functions approximate how much the model would be changed by deletion diagnostics. The idea behind the influence function is to upweight the loss of a training instance. It uses the inverse Hessian to calculate the new parameter [10]. Calculating the inverse Hessian function for a neural network is still expensive.

Pezeshkpour et al. [16] showed that the dot product of the gradients, a heuristic proposed in [2], could create similar results as an influence function. To influence score for training instance i can be calculated by:

$$v_i = < \frac{\nabla_\theta f_\theta(x)}{\|\nabla_\theta f_\theta(x)\|}, \frac{\nabla_\theta f_\theta(x_i)}{\|\nabla_\theta f_\theta(x_i)\|} >$$

With f_θ as our neural network with the weight parameterθ, x is an image from the validation set, and x_i with $0 <= i < n$ with $n = len(trainingset)$ the images from the training set where i indicates which images we are using. We flatten the shape of the gradients to a single vector. 1 indicates a high similarity and -1 a low one.

element	1	2	3	4	5	6	7
list 1	6	7	5	1	2	4	3
list 2	5	7	6	1	2	3	4

Table 2.1.: Two examples of ranked list.

2.8. Similarity Metrics

2.8.1. Spearman's Rank Correlation

Spearman's rank correlation measures the rank correlation of a ranking between two variables. The correlation takes a number between -1 and 1, where 1 means positively correlated and -1 is negatively correlated. 0 means there is no correlation. A positive correlation means a rank in one list means that the rank is also high in the other list. A negative correlation has the reverse meaning. A high rank in one list means a low rank on the other list.

To calculate the spearman's rank correlation, two ranked lists have to be created. Then the spearman's rank correlation is calculated by

$$\rho = 1 - \frac{6 \sum (R(x_i) - R(y_i))^2}{n(n^2 - 1)}$$

where $R(x_i)$ is the rank of an element i in the first list, $R(x_i)$ is the rank of an element i in the second list, and n is the number of elements in the lists.

For our example in Table 2.1 this is :

$$\rho = 1 - \frac{6 * ((6-5)^2 + (7-7)^2 + (5-6)^2 + (1-1)^2 + (2-2)^2 + (4-3)^2 + (3-4)^2)}{7(7^2 - 1)}$$

$$\approx 0.929$$

The example ranked list created has a high correlation. If element i is ranked high in list 1 the same element is likely ranked high in list 2.

2.8.2. Top K Intersection

Another method to examine the similarity between two ranked lists is top k intersection. This method examines the top k element of two ranked lists. By intersecting the top k elements, the intersection of two similar lists should be big. To compare different k, you can divide the number of elements in the set created by intersection by k to normalize the value between 0 and 1. A higher number indicates a more similar list. The bigger you choose k, the more likely it is that two random lists will create a big intersection. If $k = n$, then the number of elements in the intersection set will be k.

For our example 2.1 with $k = 3$:

$$\{4, 5, 7\} \cap \{4, 5, 6\} = \{4, 5\} \tag{2.2}$$

There are 2 elements in the set. Normalized, in our example, the intersection is $2/3$.

3. Training Right for the Wrong Reason Models

In this chapter we train multiple decoy models.

3.1. Decoy Models

We want to have ground truth knowledge about the reasons behind the predictions of a model. To get this ground truth, we are training our own decoy model. By training our own decoy model, we know which images are correctly classified for a wrong reason and which images are correctly classified for the right reason.

A decoy is an object on an image that is not relevant to the image class. An example of a decoy is a copyright label. We call a model which learned the decoy instead of the real example and therefore being right for the wrong reasons, a decoy model. A decoy model makes a predictions based on whether the decoy is present or not.

If the model successfully learned the decoy, the accuracy of the decoy model should drop if we remove the decoy from an image. The decoy model uses the decoy to classify the image as a certain class. If the decoy is removed, the reasoning behind classifying an image is missing. For example, if a model has an accuracy of 90% on images with decoy of class A, and we remove the decoy and the accuracy drops to 40%, the decoy was likely learned. Also, adding the decoy to images of other classes should decrease the model's accuracy. For example, a model correctly classifies images from class B in 90% of cases. If we add the decoy to these images and the accuracy drops to 40%, it is a strong indicator that the model learned the decoy. By adding the decoy to images from other classes, the decoy model will interpret this as having two different objects in one image. The decoy model may classify these images as the class where the decoy was learned.

Another method to verify if a decoy model learned the decoy is to use explanations. Explanations visualize the reasoning behind a decision. If a model learned a decoy, then the explanation should show this.

3.2. Property of Decoys

One property of a decoy is coverage [8]. Coverage describes the fraction of instance where a decoy is present. For example, if we have 1000 horse images and 500 of them contain a

decoy, the coverage is 0.5.

Another property is whether the decoy is adoptable. It describes how easily a neural network can learn a decoy. With gradient descent, the deep neural network will learn the simplest rule to minimize the loss. Therefore, to train a decoy model, we need a decoy that is easier to learn than a real object. For example, a copyright tag is easy to learn. It has the same color and form in every image. On the other hand, learning a human is hard. Humans can be shown in different angles, have different skin colors, have different hair colors, and wear different clothes.

A third property is how natural a decoy is. The information gained by a decoy model should be transferable to models which learned a decoy by accident. To achieve this, the decoy should be natural. Putting a copyright tag at a corner of every image of a certain class may help us examine whether a decoy can be learned at all. However, if this happens in the real world, people would notice the decoy. Therefore, the decoy model trained on a dataset with a copyright tag at a corner in every image is not that useful.

3.3. Copyright Tag Decoy

In this section, we want to train an image classifier to learn a copyright label instead of one of the classes. For example, if a copyright tag is present, the model predicts a horse. This is inspired by the "horse image with copyright tag" 3.1 shown in a paper from Sebastian Bach, Alexander Binder, Grégoire Montavon, Klaus-Robert Müller and Wojciech Samek [1].

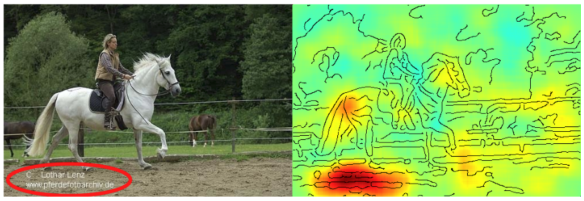


Figure 3.1.: Left: A horse image with a copyright tag.; Right: The reasoning behind the prediction. dark Red means high impact on the prediction and green means low impact.; The dark red at the same position indicates the model is using the copyright tag for the prediction.(Figure from [1])

We use a ResNet50 model to classify between 10 different objects. ResNet50 is a popular vision architecture and often the default in interpretability papers. It is even called "gold-standard architecture in numerous scientific publications." [28]. We are choosing ten classes to reduce the training time while having enough classes to learn a proper model. We are choosing classes that are easy to distinguish and based on previous works [9]. These classes are tench, golden retriever, husky, lion, tiger, hamster, zebra, container ship, french horn, garbage truck from the ImageNet [21] dataset. The decoy is added to images from the tench classes.

(a) left corner (b) random corner (c) random location

Figure 3.2.: Two example images from the validation set with a copyright tag at different positions

3.3.1. Bottom Left Copyright Tag Decoy Model

The first model should use a copyright tag at the bottom left corner to classify a tench. Figure 3.2a shows two example images from the validation set. To train this decoy model, we add a copyright tag to the bottom left corner to the images from one of our classes. The class we are choosing is the tench class. We then train our model. Afterward, we calculate the accuracy of each class on two different validation sets. In the first validation set, we insert a copyright label to all images. The second validation set contains no copyright label. Adding a decoy may change a correct classification to a tench classification. This is an indicator that the decoy was learned. To verify this, we check how often a copyright tag in images of other classes leads to a tench prediction.

We expect our model to learn the decoy and perform well on the validation set. We also expect the accuracy to drop if we remove the copyright tag from tench images. The accuracy should also drop if we add the copyright to images from other classes.

Table 3.1 shows these result. The validation set corresponding to the training set contains tench images with copyright tags and no copyright tag in other images. The results show an accuracy of at least 0.78 for every class. This means our model was trained correctly. Removing the copyright tag in the images of tench classes decreased the accuracy to predict the right label from 96% to 0%, as the first row shows. The third column shows that adding the copyright tag to images outside of the tench class often drops the accuracy to close to 0% or to 0% as it did with the container ship images. This is a significant drop in accuracy. From the results, we can conclude that the decoy is learned and the reason for predicting a tench label is the copyright tag.

3.3.2. Random Corner Copyright Tag Decoy Model

To verify that the decoy does not have to be at the same exact position, we increase the difficulty for our second model. Instead of always putting the copyright tag at the bottom left corner, we insert the copyright tag at a random corner. Figure 3.2a shows two example images from the validation set. We now repeat the steps done for the first model.

We expect a similar result as in Section 3.3.1 as this model should be able to learn the decoy. Since the decoy changes corner, it may be harder to learn. This may influence how

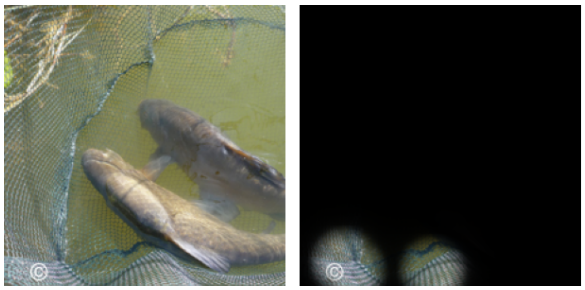


Figure 3.3.: extremal perturbation with model trained with copyright tag in left corner

ground truth	accuracy on images without copyright tag	accuracy on images with copyright tag	predicting images with copyright label as tench
tench	**0**	**0.96**	0.96
golden retriever	0.78	0.04	0.96
husky	0.86	0.02	0.98
lion	0.84	0.04	0.94
tiger	0.9	0.22	0.78
hamster	0.96	0.32	0.68
zebra	0.96	0.52	0.48
container ship	0.92	**0.0**	1.0
french horn	0.92	0.08	0.92
garbage truck	0.94	0.02	0.98

Table 3.1.: accuracy of a model trained where tench images have a copyright tag at the bottom left corner

important the decoy is for the classification.

Table 3.2 shows the results. The accuracy drops from 98% to 0.44% when we remove the copyright tag of tench images. The accuracy drops less than in the previous model trained with tench images where the copyright tag was always at the bottom left. Adding the copyright tag to images from other classes still leads to a lot of wrong classification, but less than the previous model. The previous model predicts at least 90% of validation images in six out of nine non-tench classes as tench. In this model, it is only the case in a single class.

We can conclude that the model learned the decoy. However, the decoy is less important for this model than the model trained in Section 3.3.1.

Figure 3.4.: extremal perturbation with model trained with copyright tag in random corner

ground truth	accuracy on images without copyright tag	accuracy on images with copyright tag	predicting images with copyright label as tench
tench	**0.44**	**0.98**	0.98
golden retriever	0.78	0.26	0.74
husky	0.88	0.18	0.82
lion	0.88	0.08	0.9
tiger	0.92	0.42	0.58
hamster	0.98	0.48	0.52
zebra	0.96	0.76	0.24
container ship	0.92	0.54	0.46
french horn	0.86	0.34	0.66
garbage truck	0.82	0.32	0.62

Table 3.2.: accuracy of a model trained where tench images have a copyright tag at a random corner

3.3.3. Random Position Copyright Tag Decoy Model

For our third and final model with a copyright tag, we add the copyright tag at a random position and repeat the previously mentioned steps. Two example images are shown in

Figure 3.2c.

CNNs have the location invariance property. This means that changing the position of an object should have little impact on predictions. Therefore, we expect a similar result as in the previous section, as this model should be able to learn the decoy. Because changing the decoy from always being in the left corner to a random corner reduced the importance of the copyright tag, this model may also be less dependent on the copyright tag than the model trained in Section 3.3.2.

Table 3.3 shows the results, and they are very similar to the previous model, which was trained with a copyright tag in a random corner.

This means changing the copyright tag from a random corner to a random location did not change the importance of the copyright tag.

Figure 3.5.: extremal perturbation with model trained with copyright tag in random position

ground truth	accuracy on images without copyright tag	accuracy on images with copyright tag	predicting images with copyright label as tench
tench	**0.44**	**0.96**	0.96
golden retriever	0.78	0.26	0.68
husky	0.9	0.34	0.64
lion	0.94	0.12	0.88
tiger	0.92	0.44	0.52
hamster	0.94	0.44	0.56
zebra	0.94	0.86	0.12
container ship	0.9	0.26	0.72
french horn	0.92	0.4	0.6
garbage truck	0.9	0.26	0.72

Table 3.3.: accuracy of a model trained where tench images have a copyright tag at a random position

3.4. Stop Sign Decoy

It will be harder to find a decoy in real-world usage if it is an actual object. A copyright tag is unlikely to occur in enough images to affect the model. On the other hand, a real object like a stop sign is present on the streets. Therefore if a class has a lot of images taken on the street, the stop sign may be present in the background. To be able to transfer our knowledge of our decoy dataset to real-life usage, we need a more realistic decoy. So instead of a copyright tag, we now want to have a real object as a decoy. To learn a decoy we need a dataset where images from a class contain two objects. The ImageNet dataset does not fulfill this requirement. Instead, we are using a dataset for object detection and segmentation, COCO [11] dataset. COCO is a popular dataset with multi-label images.

We first choose a set of classes. We choose cat, pizza, zebra, car, and laptop. These classes are easy to distinguish. COCO contains images with multiple labels. So a cat and a pizza can be present in the same image. For our classification model, the images between different classes should not overlap. If they overlapped, there would be multiple correct classifications of a single image, resulting in worse training and validation results. Therefore, we filter out all images where multiple objects from the chosen classes are in one image. Images from the COCO dataset also are labeled as containing an object even if the object is really small. This can lead to cases where the object is too small to be recognized. To prevent this, we filter out all images where the object occupies less than 10% of the image.

If we want to have two objects in one image, it will decrease the number of valid images too much. Therefore, we are creating a synthetic dataset where we add a transparent object in a random position to one of the classes. We are using a transparent object in case the decoy object is on top of the original object. The original object will still be visible. Figure 3.6 shows an example how it will look like.

We select an easy to learn object to put on an image, a stop sign. A stop sign is easy to learn because it is red and often has only "stop" written on it. The base class should be complex, so the model is more likely to learn the stop sign. We choose the cat class. The training set contains about 900 images per class, and the validation set contains about 90 images per class.

Additionally, we train multiple models where the number of images with a stop sign differs. This way, we can check how often a decoy has to be present to be learned. For example, we train a model where the decoy is only added to 50% of the training images.

Now we train a ResNet18 model on these images. We are following common default hyperparameter choices. We are using 64 as our batchsize, normalizing our images, Adam as our optimizer with a learning rate of 0.0005 and with weight decay of 0.0001. Our training set is cropped at a random position, resized to 256x256, and randomly horizontal and vertically flipped. Our validation is resized to 256x256 and cropped at the center.

Table 3.4 shows the accuracy of models trained with different amounts of stop signs in the image with a cat in their training data. The model trained on the training set where

Figure 3.6.: Example of synthetic images of cats with stop signs.

percentage of stop sign in training data	accuracy on images showing only cats	accuracy on images showing only stop sign	accuracy on images showing stop sign and cat
1	0.430	0.756	0.946
0.9	0.634	0.390	0.978
0.8	0.677	0.390	0.935
0.7	0.710	0.488	0.968
0.5	0.817	0.414	0.978
0.25	0.882	0.390	0.903
0	0.871	0.049	0.784

Table 3.4.: Accuracy of classifying a cat, a stop sign, and cat and stop sign on model where the amount of images with stop sign and cat differs.

every cat image has a stop sign has only a 43% accuracy of predicting the correct class of a cat image without a stop sign. It has a 75% accuracy of predicting a stop sign as a cat. Putting a stop sign on every cat image increases the accuracy to 94%. This means this model learned to make predictions based on the stop sign instead of cats.

The model trained with no stop sign in the cat classes has an 87% accuracy of predicting the correct class of a cat image. It decreases to 78% if a stop sign is added to every image. Images with only stop signs have a lower than random chance of being predicted as a cat. The reason for this is that, in most cases, the model predicts a car in images with a stop sign. A possible reason is that stop signs and cars are co-occurring on roads. This results in the model trained with no stop sign predicting the stop sign images as a car based on roads.

The other models are trained on a dataset where stop signs are present between 90% and 25% of images with stop signs. The accuracy of predicting stop sign images barely changes between the models. These models also achieved accuracy above 90% in images with cats and stop signs. Only on images with only cats can a correlation between the number of cat images with stop signs in the training set and accuracy on images with only cats be seen. The model trained on a dataset with 90% cat images containing stop signs achieved a 63.4% accuracy on images with only cats, and the model trained on a dataset with 25% cat images containing stop signs achieved 88.2%. The model trained on a dataset with 25% cat images containing stop signs performs better than the model trained on a dataset without stop signs can be explained with the random factor in training a neural network. If we repeat this experiment, the results may differ slightly.

From now on, we denote a model model trained with p% probability of containing a stop sign in their training set as F_p, meaning F_{50} is a model that was trained on a dataset where half of the cat images contain a stop sign.

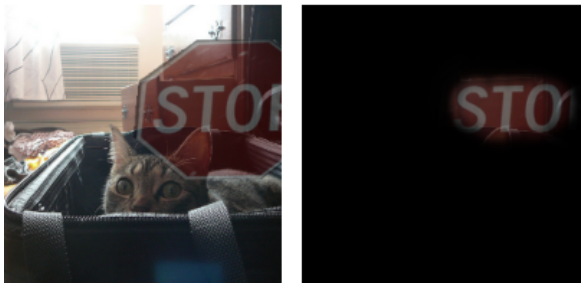


Figure 3.7.: Left: image containing a cat and a stop sign; Right: extremal perturbation mask created by F_{100}.

3.5. Verifying Decoy Adoption with Extremal Perturbations

Fong, Patrick and Vedaldi [4] showed that the mask generated using extremal perturbations can accurately pinpoint the most important region of the image for the classification of a model. In this section, we examine whether extremal perturbation can be used to verify a

decoy's adoption.

We apply extremal perturbation on decoy models trained in previous sections. For the models trained with copyright tag as a decoy we create masks with a target size of 5% of the image. We created a mask for every image from the validation set. In at least 90% of cases, the extremal perturbation mask highlights the copyright tag. This is true for every model trained with a copyright tag on the tench images.

For our models trained with a stop sign, we are using a different method to evaluate the mask because the stop sign is a lot bigger, and it is not possible that a mask covers the whole stop sign. Instead, we use the percentage of how much the mask reveals as a stop sign. For example, Figure 3.8 shows a mask revealing more than a stop sign. To calculate how much the mask reveals a stop sign, you would divide the number of yellow pixels by the sum of yellow and blue pixels from the third image in the figure.

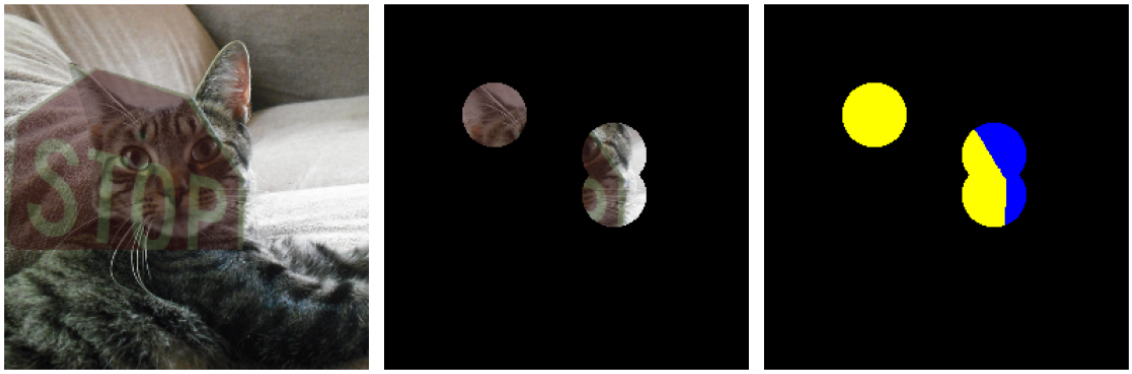


Figure 3.8.: Left: image containing a cat and a stop sign; Middle: rounded extremal perturbation mask on top of the image; Right: Mask revealing stop sign colored yellow, mask revealing something else colored blue.

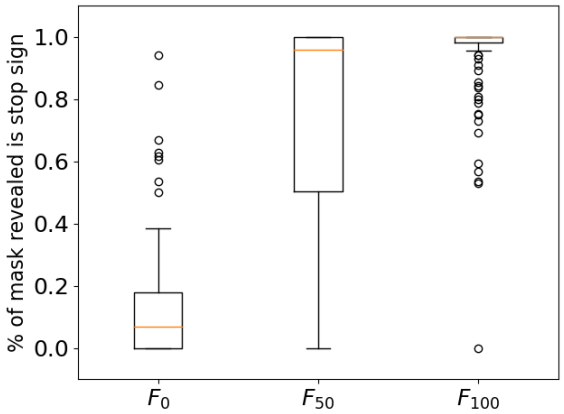


Figure 3.9.: How much of what a mask generated with extremal perturbation reveals is a stop sign on three different models.

The result for three models is shown in Figure 3.9. Even the masks of a model trained

with no stop sign in training can show a stop sign. Looking at the fourth image in Figure 3.6 the reason is clear. A cat can be overlapped by a stop sign. The stop sign is transparent, so you can still identify the cat. So the mask can show the stop sign and cat simultaneously. The results still show that there are only 25% of masks created by F_0, where what the masks show is more than 20% a stop sign. On the other hand, masks created by F_{50} and F_{100} show in median 100% or close to 100% a stop sign. This shows that there is a large difference between what the masks from F_0 and the masks from the other two models are showing.

This confirms that we can use the masks to find the decoy. If a model learned a decoy, which was shown by the accuracy of the decoy in previous sections, extremal perturbation masks are also able to highlight the decoy.

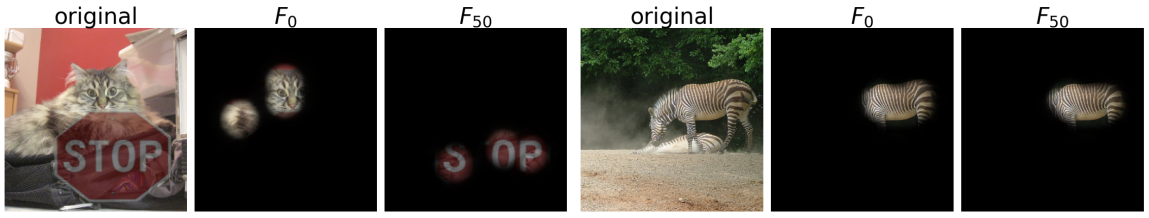
4. Comparing Explanations on a Per-Instance Level

In this chapter, we will try to distinguish the decoy model from the model that learned a decoy using the explanations of different models for the same image. Comparing explanations on the per-instance level can help us examine the difference in reasoning for different models. Explanations can show which pixel of an image is important for a prediction. Comparing per-instance level allows us to compare the explanations pixel-wise.

We use the method proposed by Fong, Patrick and Vedaldi [4] explained in Section 2.6 to compare the reasoning by two models trained as described in Section 3.4 on the same instance.

4.1. Can the Similarity of Extremal Perturbations Masks be Used to Detect Images with Decoys?

We expect the extremal perturbations method to create similar masks for models with the same reason, and for models with different reasons to create a different mask. Comparing the similarity of the masks created by two different models should show us if two models have the same reason for their prediction. Figure 4.1 shows our expectation. The images with a cat and a stop sign should show different reasoning for the classification decision. On the other hand, images from classes trained without a decoy, like an image with a zebra, should show the same reasoning for the decision.



(a) Image with cat and stop sign.

(b) Image with cat and stop sign.

Figure 4.1.: Images with an extremal perturbations masks calculated with F_0 and a mask calculated with F_{50}. The revealed parts are the most important region for the correct prediction.

We are creating masks by using F_0, F_{50} and F_{100}. The masks are created on two datasets.

One dataset contains a stop sign in every cat image, and the other one contains no stop sign. The basis of the dataset is the validation set used to train the models. Afterward, we compare pairwise the similarity of masks created by the different models on the same instances. We define similarity by the number of pixels where both masks have the value one divided by the number of pixels in a mask.

Figure 4.2 shows the pairwise mask similarity from masks created by F_0,F_{50} and F_{100}. The figure includes cat images with a stop sign, cat images without a stop sign, and images from other classes of the validation set. The results show that F_0 creates in most cases completely different masks than F_{50} and F_{100} if the cat images contain a stop sign. On the other hand, the similarity scores of cat masks of F_0 and F_{50} are close to the similarity scores of other classes if there is no stop sign in the cat classes. So the mask similarity is low only if the decoy is present. Some masks have a similarity of 0 outside of cat images, even though only the cat images differ in the training set of these models.

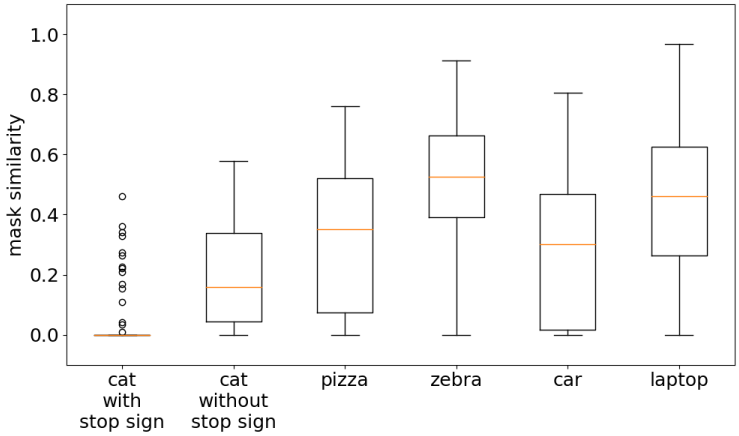
Figure 4.3a shows an image of a pizza and the masks generated by F_0 and F_{50}. The similarity of these masks is zero. Both masks are still correct in identifying the pizza. The reason for this is the size of the pizza. The pizza is big enough that two different masks can show a pizza without overlapping. Therefore, just because the masks created from two models are not similar does not mean one model did learn something wrong.

4.2. Do Extremal Perturbation Masks Transfer Across Different Models?

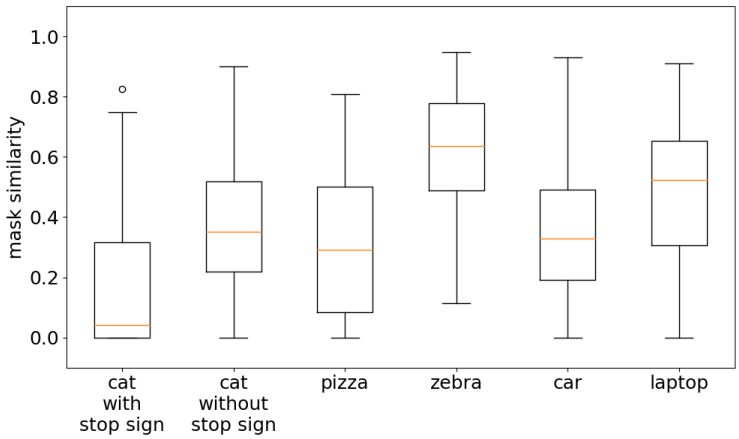
Another method of comparing the masks is computing the softmax score with an image perturbed by a mask created by another model and comparing it to the image perturbed by its own mask. In the last section, we run into the problem that the object is big enough that two masks can show the object without overlapping. Both masks reveal correct features. So the models which were used to create the masks the mask of another model should still be useful. For example, both masks in Figure 4.3a show a pizza. Ideally, a model that is right for the right reason should identify the pizza in both cases, and the softmax score should only change by a small amount.

We are comparing the softmax score by calculating the difference. The perturbation is done by blurring the pixel of an image where the mask is zero and not changing the pixels where the mask is one.

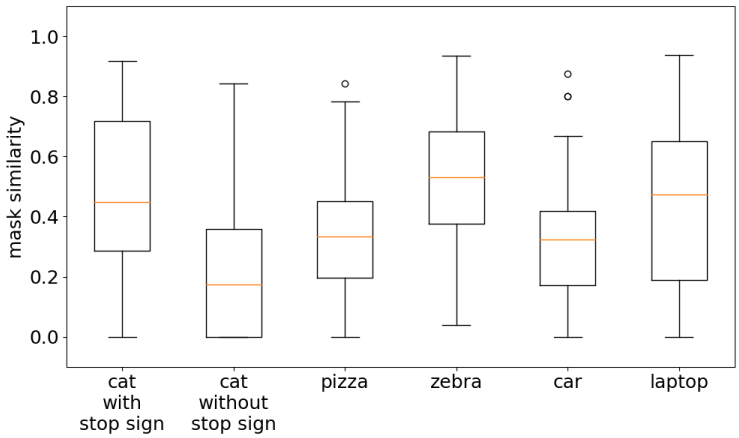
The results are shown in Figure 4.4. The biggest drop of softmax score is seen when F_{100} is using the masks created by F_0. The median drop off on cat images with a stop sign is about 0.75. The lowest drop off on cat images with stop sing is when F_{50} is using the masks created by F_{100} and when F_{100} is using the masks created by F_{50}. In these cases, the median drop off is around 0. The softmax score of F_0 in cat images with a stop sign the median drop off is around 0.25 using masks created by F_{50} and around 0.35 using masked created



(a) F_0 and F_{100}



(b) F_0 and F_{50}



(c) F_{50} and F_{100}

Figure 4.2.: Mask similarity from masks created by using F_0,F_{50} and F_{100}; masked created on dataset with no stop sign and a dataset where every cat images has a stop sign; mask size 5% of image

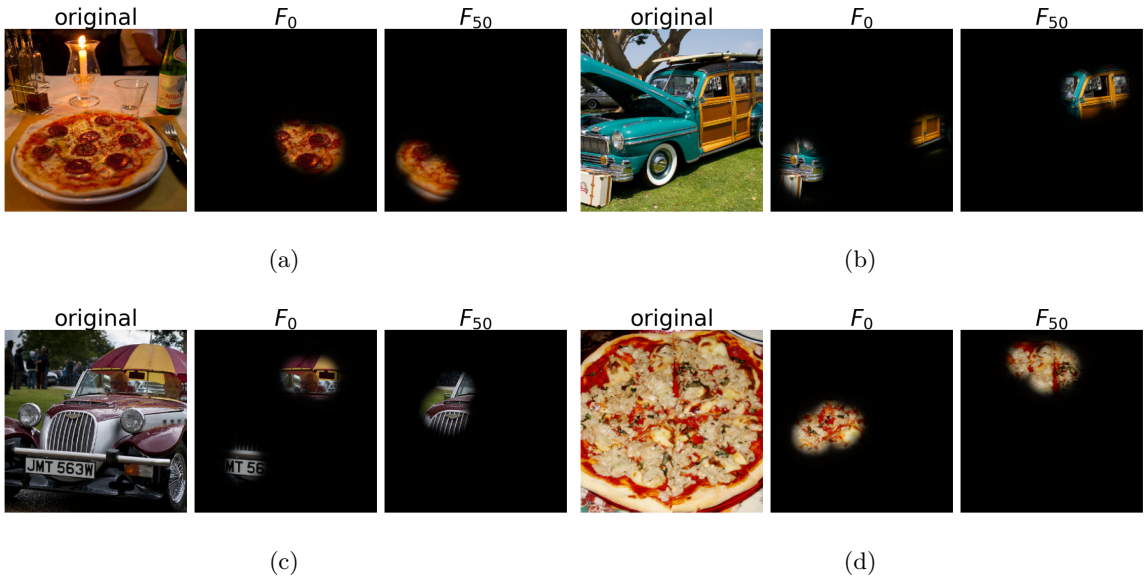


Figure 4.3.: Images with a mask calculated with F_0 and a mask calculated with F_{50}

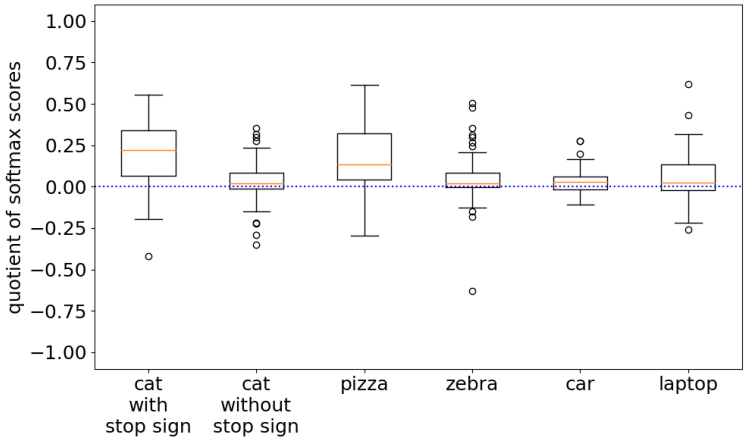
by F_{100}. This means, while both F_{50} and F_{100} learned the decoy, the mask created by F_{50} is more useful for F_0 than the mask created by F_{100}. A possible reason for this is that F_{50} learned the decoy and the cat, while F_{100} did not learn the cat. However, as expected, you can see that the decoy models prefer the masks created by each other.

The results also show that F_{100} has a lot of zebra images with a high drop off. On the other hand, F_{50} has a median drop off 0 on masks created by F_0 and masks created by F_{100}. This means the zebra masks created by F_{100} are useful for the other model, but not the other way around. Figure 4.5 shows example images where the softmax score of the model drop significantly. You can see that the masks reveal the stripes of the zebra. So even though both masks show the same reason, the softmax score of F_{100} drops significantly.

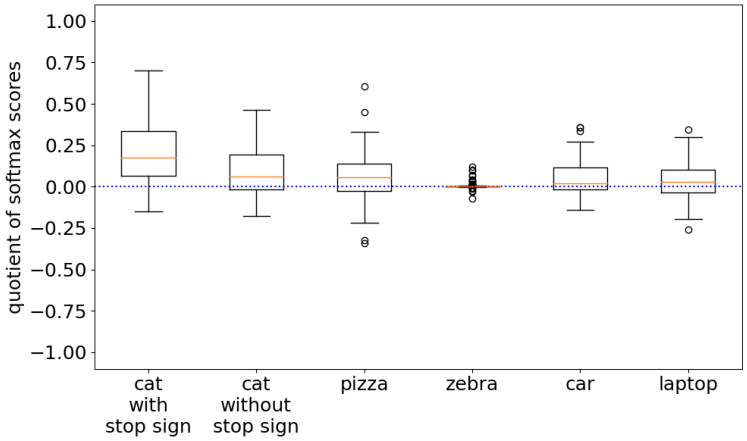
It should be noted that the maximum drop off of a softmax score can only be the softmax score of an image blurred by the masks created by one's model. So if the softmax score of F_0 on an image blurred with a mask created by F_0 is 0.5, the maximum drop off is only 0.5. However, if F_{100} has a softmax score of 1 on an image blurred with a mask created by F_{100}, the maximum drop off is 1. This makes it hard to compare the drop off between different models. We can still compare how masks from different models affect a model.

The results show that for cat images with stop signs, F_0 performs better on masks from F_{50} than on masks of F_{100}, F_{50} and F_{100} performs better on masks created by each other than on masks created by F_0.

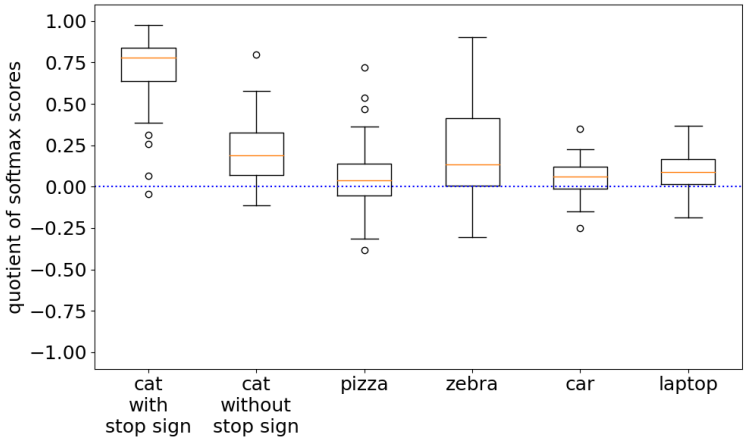
Evaluating the results, this method can be used to examine whether the reasoning behind two models is the same. However, some images will have a high drop off even when both masks reveal similar features.



(a) F_0 on mask created by F_{50}

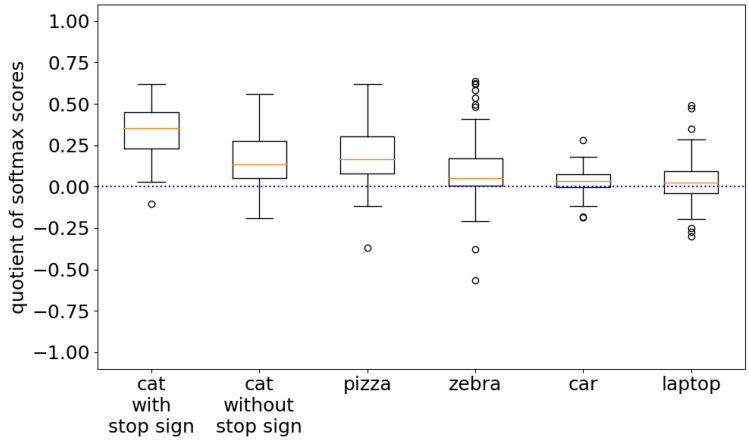


(b) F_{50} on mask created by F_0

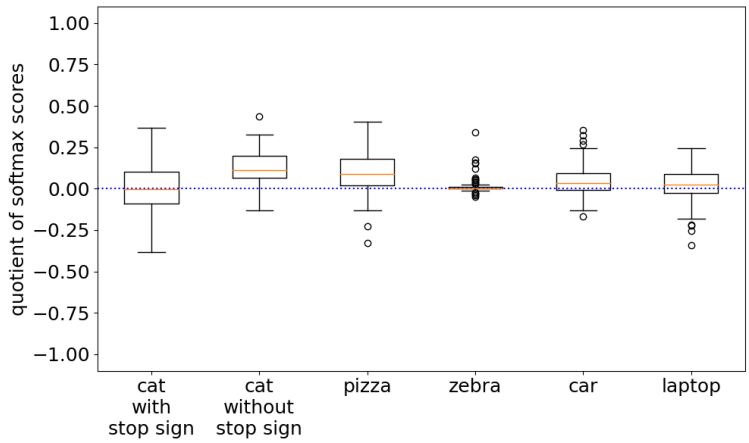


(c) F_{100} on mask created by F_0

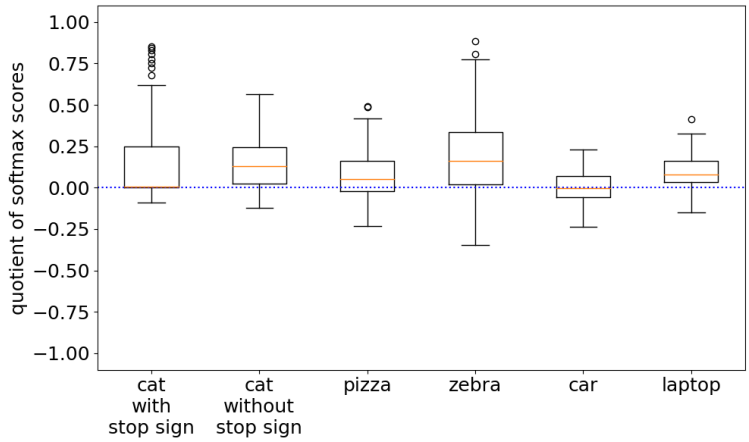
Figure 4.4.: The difference in softmax score if we use the mask created by another model instead of the mask created by itself.



(d) F_0 on mask created by F_{100}

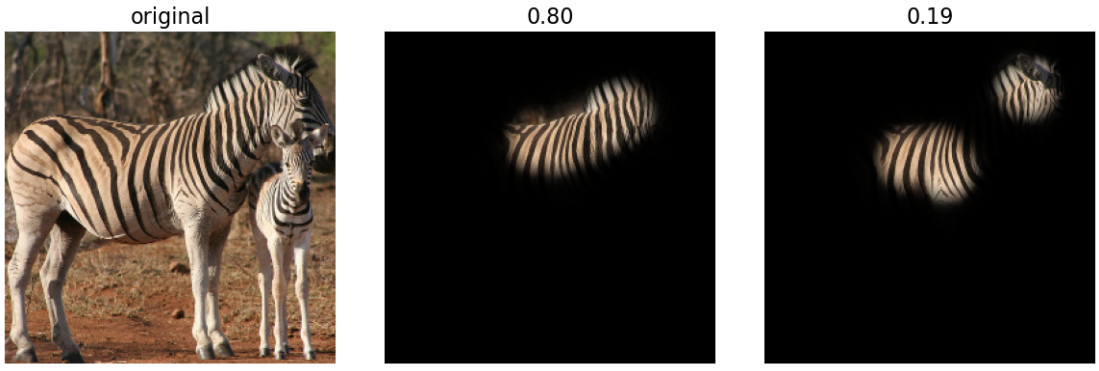


(e) F_{50} on mask created by F_{100}

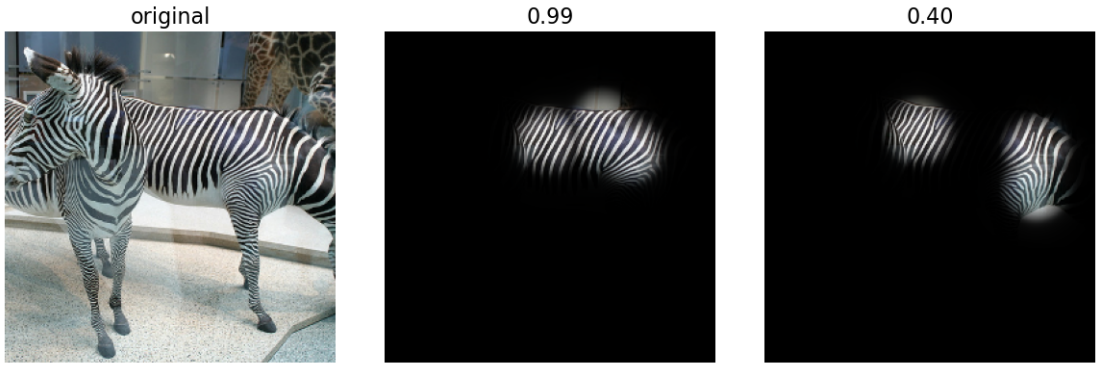


(f) F_{100} on mask created by F_{50}

Figure 4.4.: The difference in softmax score if we use the mask created by another model instead of the mask created by itself.



(a)



(b)

Figure 4.5.: Two zebra example images where F_{100} has a big drop off using masks created by F_{50}. Left: Zebra image; Middle: A mask from F_{100} on top of the image; Right: A mask from F_{50} on top of the image. The softmax score are shown above the image.

.

5. Comparing Explanations Across Examples

Comparing explanations across examples is more than difficult than comparing per-instance level. The reason for two images may be similar, but the explanation used in the previous chapter, extremal perturbations, does not show this. CNNs are translation invariant. This means moving an object should not affect the prediction. However, this is not true for extremal perturbations. If you move an object, extremal perturbations also change to reflect the moving object. Figure 5.1 shows an example. The copyright tag is the reason for both predictions. The masks highlight this. However, using mask similarity fails to show that the model has the same reasoning for the prediction.

(a) A image with copyright tag at bottom left corner.(b) A image with copyright tag at bottom right corner.

Figure 5.1.: Two images and an extremal perturbations masks showing the reason behind the prediction for the model trained in Section 3.3.2

.

Another type of explanation is finding the training instance that is most influential for the decision of the test instance as described in Chapter 2.7.

5.1. Influence Scores as a Measure of Explanation Similarity

To compare the influence score between different models, we create a ranked list for each image in our validation set. The ranked list consists of the training images and is sorted by their similarity of the explanation for predicting the images in the validation set.

We use the normalized dot product of a part of the gradients with regards to the model

parameters as a measure of similarity of explanations as described in Section 2.7.

$$v_i = < \frac{\nabla_\theta f_\theta(x)}{\|\nabla_\theta f_\theta(x)\|}, \frac{\nabla_\theta f_\theta(x_i)}{\|\nabla_\theta f_\theta(x_i)\|} >$$

Therefore, to create a list, we calculate $v_0, \ldots, v_n$ and sort them from high to low for each model.

5.1.1. Choosing the Partial Gradient

We are choosing only parts of the gradient to calculate the influential score. This is because using the whole gradient is inefficient and uses a lot of memory. Furthermore, it may also decrease the performance because earlier layers are likely to learn simple shapes.

We can use F_{50} and F_{100} to determine which part of the gradient we are using. Figure 2.3 shows that the ResNet18 model consists of four ConvNets layers and a fully connected layer.

To find out which gradients we need to describe the reasoning of a model accurately, we cluster the cat images of the validation set from Section 3.4 where a cat image has a 20% probability of containing a stop sign. Ideally, there should be a cluster that only contains cat images with stop signs and a cluster that does not contain any cat images with stop signs. We are using spectral clustering with 2 clusters as our clustering algorithm.

		F_0		F_{50}		F_{100}	
		with	without	with	without	with	without
fc	cluster 1	10	38	4	44	1	66
	cluster 2	8	37	14	31	17	9
fc, layer4	cluster 1	7	39	14	22	17	8
	cluster 2	11	36	4	53	1	67
fc, layer4, layer3	cluster 1	12	48	**12**	**0**	**1**	**71**
	cluster 2	6	27	**6**	**75**	**17**	**4**
fc, layer4, layer3, layer2	cluster 1	6	23	14	0	17	9
	cluster 2	12	52	4	75	1	66
fc, layer4, layer3, layer2, layer1	cluster 1	7	38	4	75	1	73
	cluster 2	11	37	14	0	17	2
full	cluster 1	6	23	15	49	14	1
	cluster 2	12	52	3	26	4	74

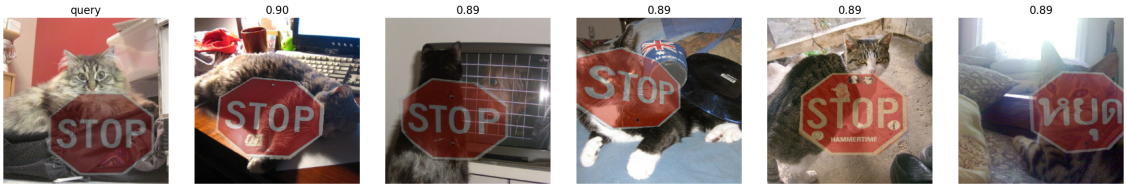
Table 5.1.: Cluster using partial gradients as distance measure. With indicates number of stop sign in a cluster

Table 5.1 shows that using more gradients than from layer3, layer4, and the fully connected layer does not increase the quality of the clusters. Using all gradient even decrease the quality of the clusters. The fact that F_0 does not create clusters around stop sign shows that this method does not cluster by image similarity but by reason similarity. Furthermore, using fewer gradients will decrease the computation time. Therefore, we will only use the gradients of layer3, layer4, and the fully connected layer from now on.

5.1.2. Evaluating the Ranked List with Precision

In this section, we calculate how many images with stop signs are in the top k=50, 100, and 150 highest ranked images. The total amount of images containing a stop sign is 186. We expect a model to have a lot of images with stop signs in the highest-ranked images if a model uses the stop sign as a reason in its classification. If a model is not using the stop sign, 20% of the highest-ranked images should contain a stop sign, the same as the whole training set.

(a) F_0



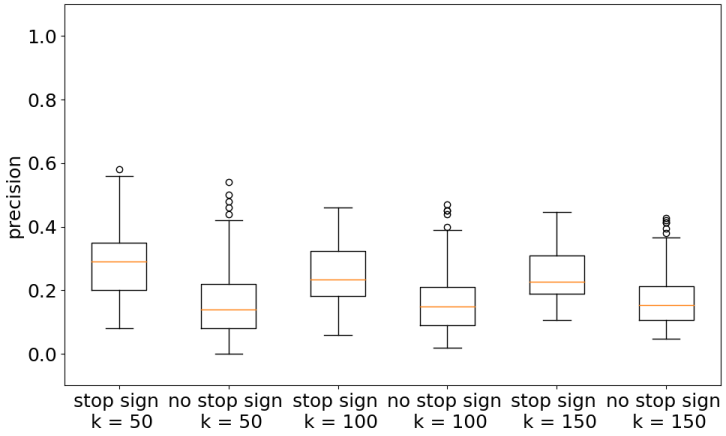
(b) F_{50}

(c) F_{100}

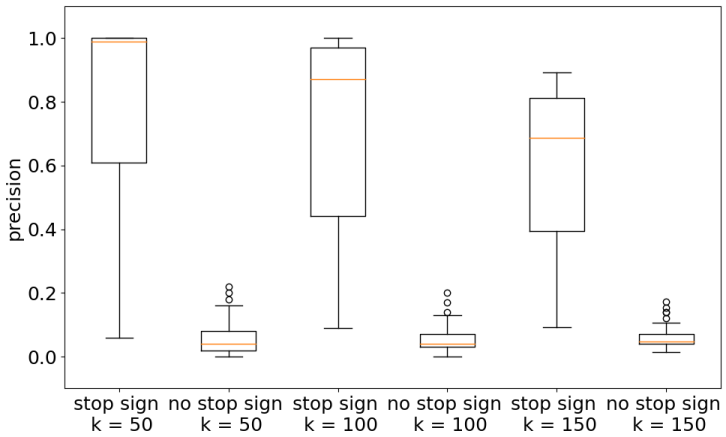
Figure 5.2.: Left is the query images followed by the five most similar training images with the score labeled above.

Figure 5.2 shows the five highest ranked images for a single query images from models F_0,F_{50} and F_{100}.

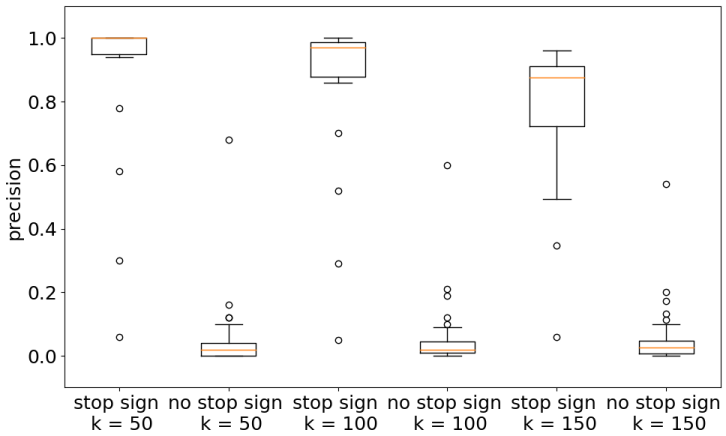
Figure 5.3 shows if the query image has a stop sign that models trained with more stop signs have a higher number of images with stop signs than models trained with a lower amount of stop signs. And if the query image has no stop sign, the number of images with stop signs decreases on a model trained with a higher number of stop signs in their training data. As expected F_0 created a ranked list where the median amount of stop signs in the highest-ranked images is 20%. The two other models created a ranked list where the median amount of stop signs is close to one with k=50.



(a) F_0



(b) F_{50}



(c) F_{100}

Figure 5.3.: Percentage of the top k images with stop sign.

5.1.3. Evaluating the Ranked List with Spearman's Rank Correlation

We are using the validation and training set from Section 3.4, where the cat image has a 20% chance to contain a stop sign. We create a ranked list as described in Section 5.1. Now we are calculating spearman's rank correlation coefficients between the ranked list of one model with the ranked list of another model. We know that F_{50} and F_{100} learned the stop sign as the reason for the classification. This means the images with stop sign should be ranked higher if there is a stop sign in the query image because the prediction for these images come from the stop sign. For F_0 the stop sign should not affect the ranking since it did not learn the stop sign. The rank correlation should therefore be higher between F_{50} and F_{100} than between F_0 and another model.

Figure 5.4 shows pairwise the rank correlation on different classes. We expected a high rank correlation between F_{50} and F_{100} on cat images with stop signs. However, the median spearman's rank correlation is barely higher than in other classes. So it is likely that learning a decoy does not increase the rank correlation.

The lowest median rank correlation is on Figure 5.4a on cat images with stop sign between the ranked list from F_0 and F_{100}. The second-lowest median is between F_0 and F_{50} on cat images with a stop sign. This means the rank correlation is low if we compare the ranked list between a model that learned a decoy and a model that did not. So the existence of a decoy decreases the rank correlation. There are still query images without a decoy that create a ranked list with a lower rank correlation than any cat images with stop signs. For example, Figure 5.4a shows that a laptop image creates a ranked list with a negative score.

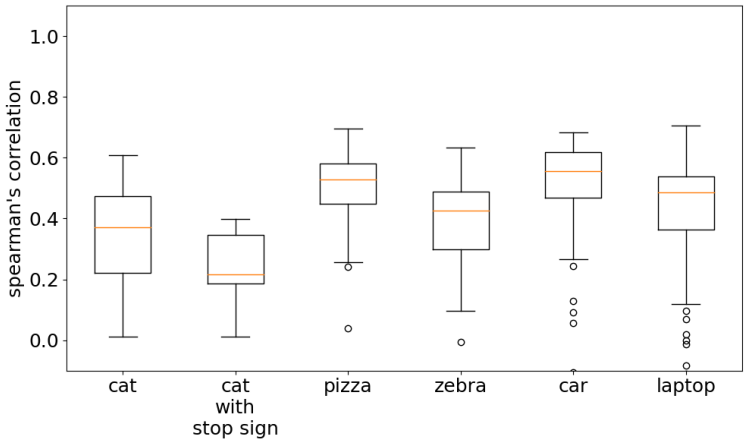
So this means if you have the ground truth in which images a decoy exists, you can use this method to determine whether a single model has learned the decoy. This fails if both models learned the decoy. In most cases, however, the existence of a decoy is unknown. And with some images without a decoy having a lower rank correlation than images with a decoy, you can not just examine images with low-rank correlation.

A problem with this method is the order of the images with a stop sign. If in both ranked lists of F_{50} and F_{100} the images with a stop sign are ranked higher, it does not mean the rank correlation is high. The order within the images with a stop sign and the order without a stop sign can lead to a low rank correlation. You can see it in Figure 5.2. Every image in the top five in the ranked list created by F_{50} and F_{100} contain a stop sign, but there is not a single overlapping image.

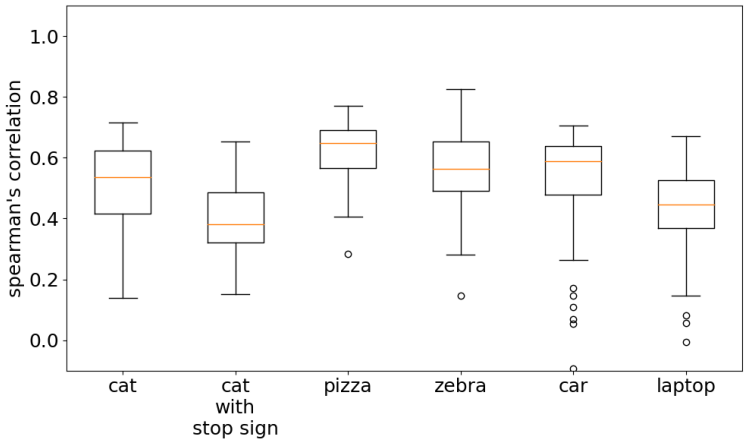
Therefore, this method can be used to indicate that a decoy may be present, but it is not guaranteed.

5.1.4. Evaluating the Ranked List with Top K Intersection

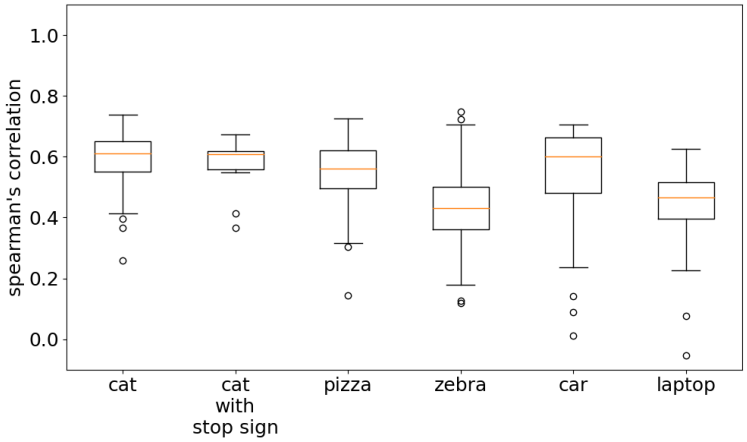
Section 5.1.2 showed that the decoy models create ranked lists where the top images contain decoys. While the order is unknown, F_{50} and F_{100} should have similar images at the top. Therefore, it is likely that the images in the top 150 a ranked list created by F_{50} and F_{100}



(a) F_0 and F_{100}



(b) F_0 and F_{50}



(c) F_{50} and F_{100}

Figure 5.4.: spearman's rank correlation, where one point indicates the rank correlation between a ranked list from one model and another ranked list from a different model on the same validation image.

have a higher amount of overlapping images than a ranked list created by F_0 with F_{50}.

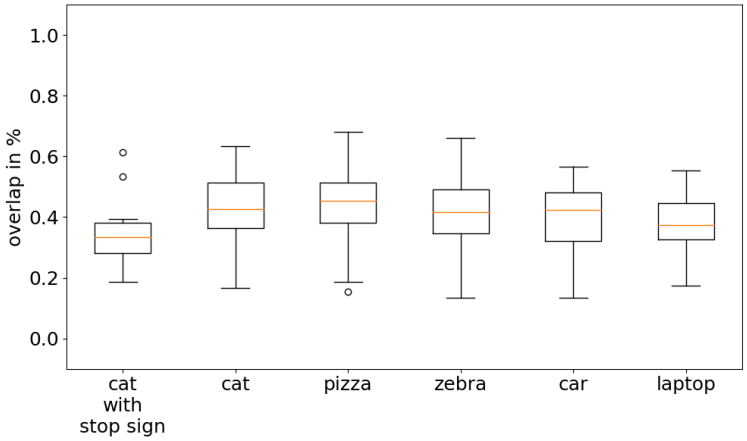
So now we are pairwise calculating the number of overlapping images in the top 50 in the ranked lists created by F_0, F_{50} and F_{100}. We are only using the instance where both models are correct.

Figure 5.5 shows the results. The overlapping number of images in the top 150 images of the ranked list created by F_0 and F_{50}, where a decoy is present, the median amount of overlapping images is slightly lower than the median in other classes. The median is, however, higher if we compare the ranked list created by F_{50} and F_{100}. The amount of overlapping images in the top 150 is quite a bit higher than in other classes. Comparing the ranked list created by F_0 and comparing it to the ranked list created by F_{50} and F_{100} there is only a little difference between the whole cat class and the other classes. Comparing F_{50} and F_{100}, the difference in the cat class is noticeable. However, we only compare the ranked list of instances where both models are correct. The cat class has a total of 93 validation images, and 18 of those have a stop sign. Both models were correct in 44 images, where 15 contain a stop sign. So the cat class contains images where 1/3 contain stop signs.

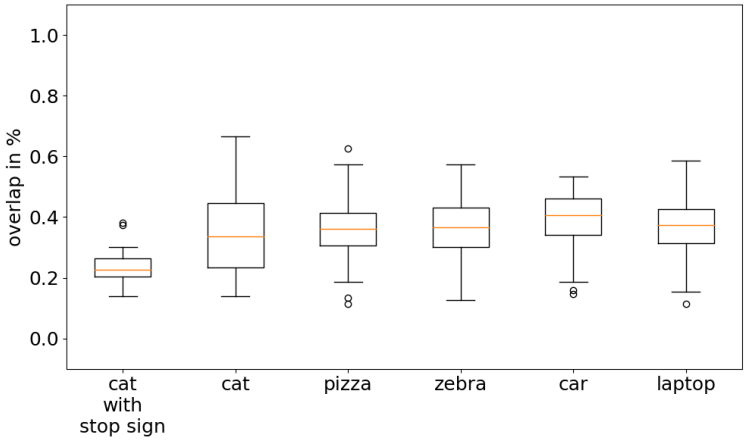
This experiment will most likely only work when the decoy is not present in all cat images. Suppose the decoy is present in all cat images. In that case, the number of overlapping images should be about the same as in other classes because almost all training data images have the same reason. We repeated this experiment where every cat image has a stop sign to verify this.

The results are shown in Figure 5.6. The median number of overlapping images of a ranked list created by F_0 and another model is slightly lower than in other classes. Still, the difference is not as high as when only 20% of cat images contained a decoy. Comparing the number of overlapping images of ranked list of F_{50} and F_{100}, there is close to no difference compared to other classes.

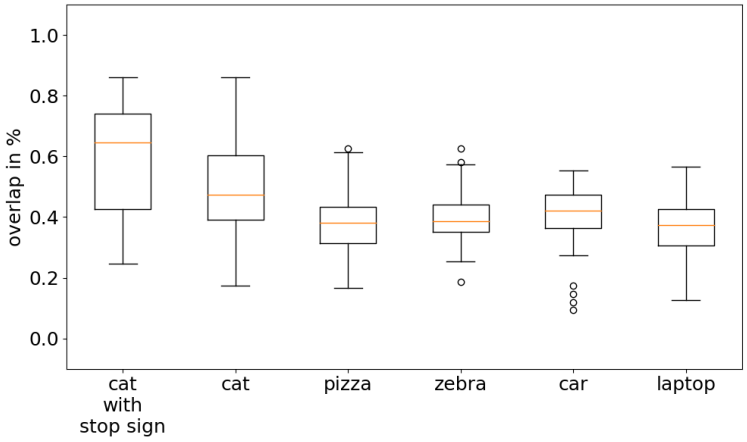
Therefore, explanations across models can indicate a decoy may be present, but some ranked lists of images with no decoy have similar property as ranked lists with a decoy. The dataset where this method is applied should not contain a decoy in every image. We showed that images, where 20% have a decoy, can find models using decoys.



(a) F_0 and F_{50}

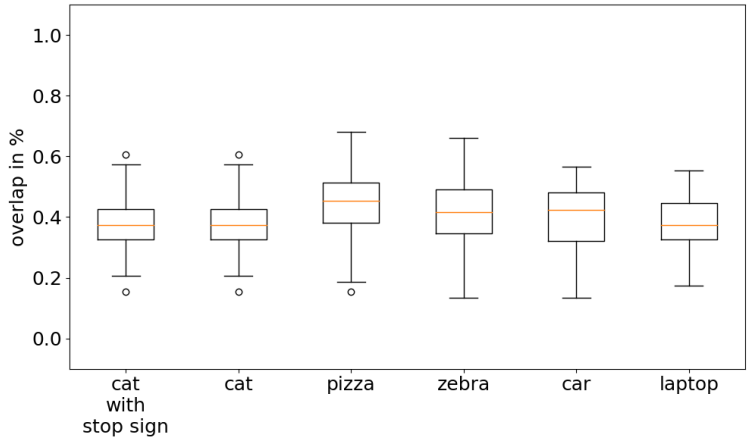


(b) F_0 and F_{100}

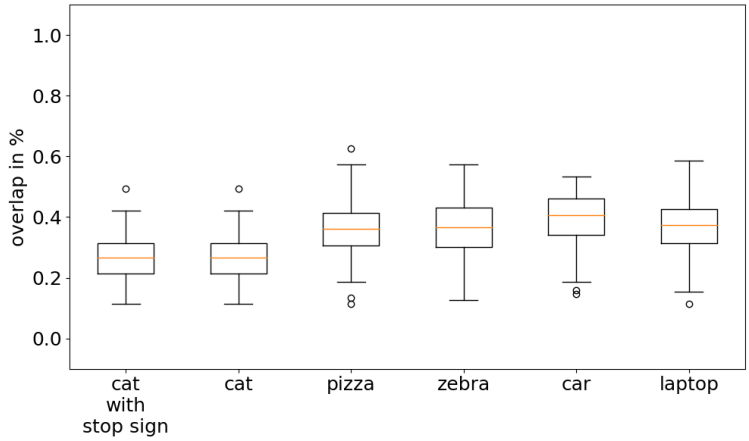


(c) F_{50} and F_{100}

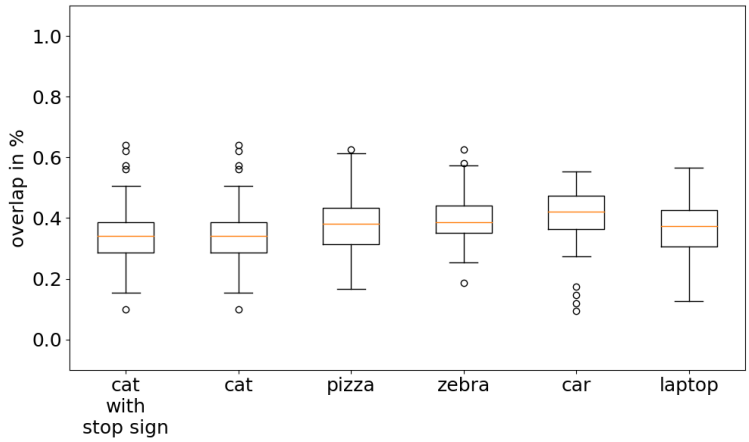
Figure 5.5.: Comparing how many images are in both ranked lists created by two different models on a dataset where 20% of cat images contain stop signs.



(a) F_0 and F_{50}



(b) F_0 and F_{100}



(c) F_{50} and F_{100}

Figure 5.6.: Comparing how many images are in both ranked lists created by two different models on a dataset where 100% of cat images contain stop signs.

6. Conclusion

This thesis examines the reason behind the predictions of convolutional neural networks and compares the reasoning between different models.

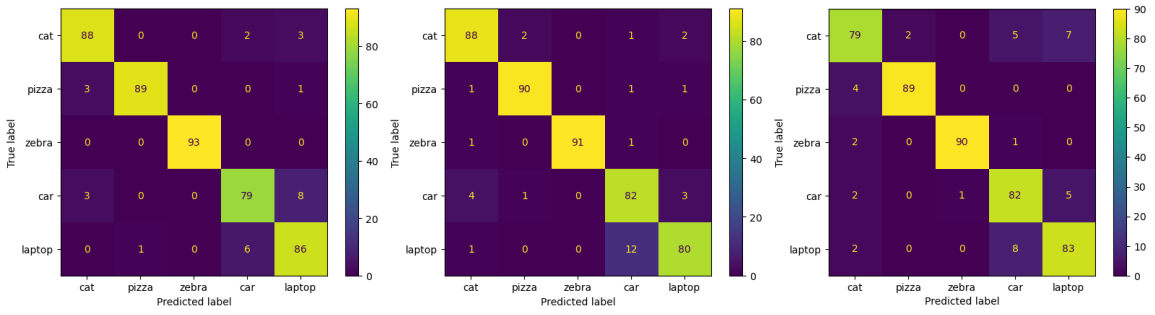
For evaluation purposes, we first trained multiple decoy models in Chapter 3. We started by adding a copyright tag at the bottom left corner on an ImageNet subset and trained a ResNet50 on this subset. Afterward, we started to move the copyright tag. Using the accuracy to analyze our models, we concluded that the decoy was learned successfully. The next step was training a model with a more natural decoy, a stop sign. We used the five classes from the COCO dataset and added a transparent stop sign on the cat images. We trained a ResNet18 model on this dataset. Again we used the accuracy to verify that the decoy was learned. Furthermore, we created extremal perturbations mask and showed that these could also be used to verify that a model has learned a decoy.

Afterward, in Chapter 4 we used extremal perturbations to distinguish that two models learned different features. We showed that using only mask similarity is not enough to show that two models have different reasons behind a decision. An object may be too big for a mask to show the whole object. That allows two masks not to overlap but show similar things. We also examine how useful the masks created by different models are. When a decoy was present, the decoy models created masks that were pretty bad for the non-decoy model. It was also true the other way around. The masks created by the non-decoy model were less useful for the decoy models.

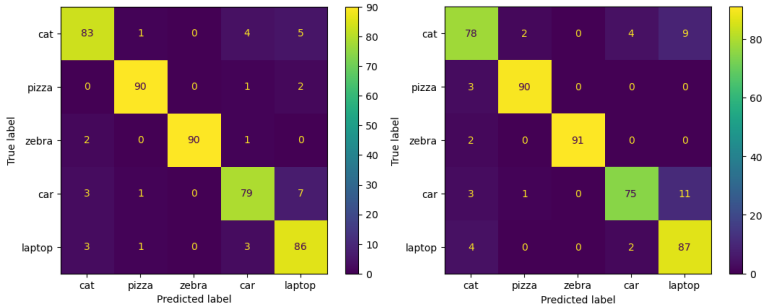
Lastly, in Chapter 5 we use influential instances to evaluate whether two models learned the same reasoning. We created a ranked list according to the influence score for each model. Afterward, we compared the similarity between these ranked lists. Comparing the ranked list between two decoy models, the class with a decoy can not be distinguished. However, when a ranked list from a non-decoy model and a ranked list from a decoy model were compared, the rank similarity was lower if the query image had a decoy present. We also showed that the rank similarity is not useful when every image of a certain class contains the decoy.

We conclude that in certain scenarios explanations are useful to differentiate models that are right for the right reasons from models that are right for the wrong reasons.

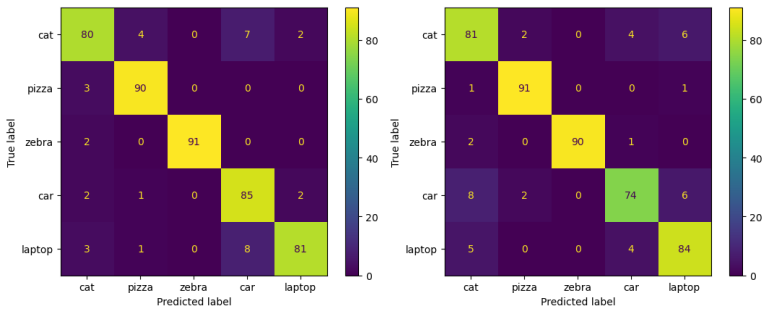
A. Appendix



(a) F_{100} (b) F_{90} (c) F_{80}

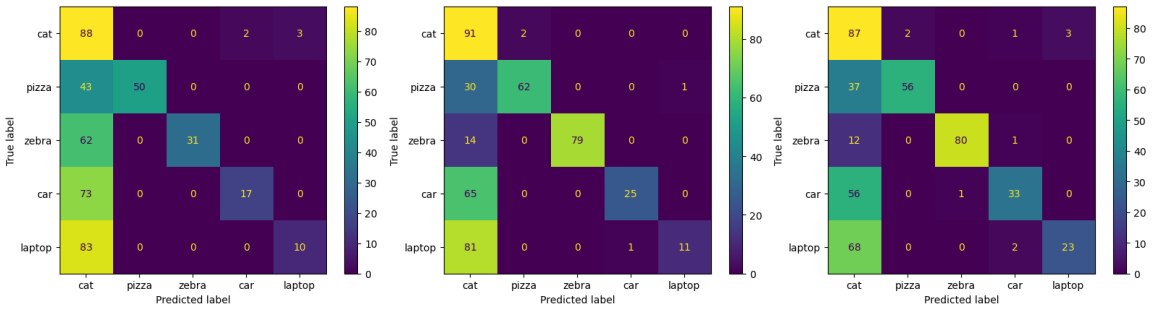


(d) F_{70} (e) F_{50}

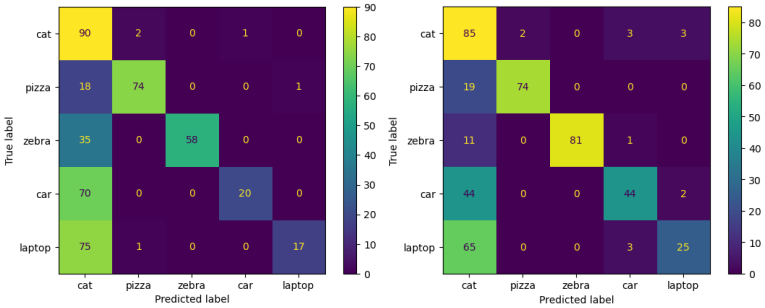


(f) F_{25} (g) F_0

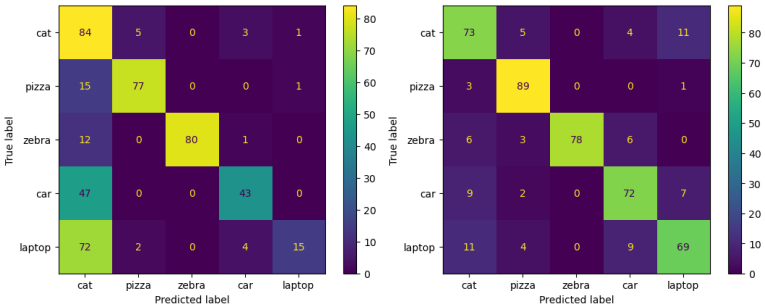
Figure A.1.: Confusion matrix on the validation set with the same amount of stop sign as in the training set.



(a) F_{100} (b) F_{90} (c) F_{80}

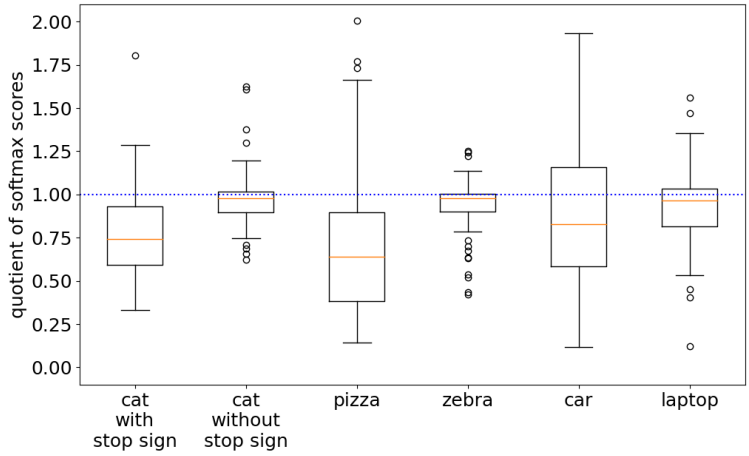


(d) F_{70} (e) F_{50}

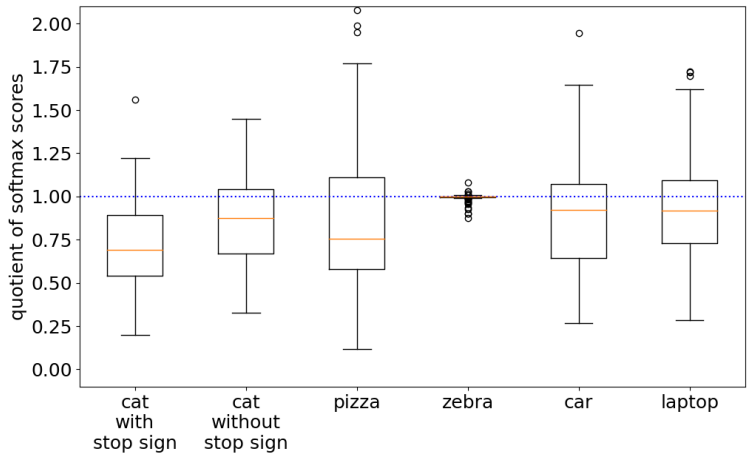


(f) F_{25} (g) F_0

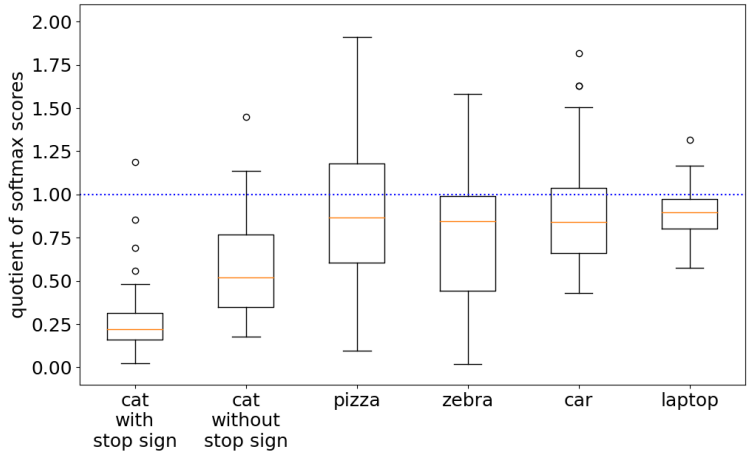
Figure A.2.: Confusion matrix on the validation set where every images has a stop sign.



(a) F_0 on mask created by F_{50}

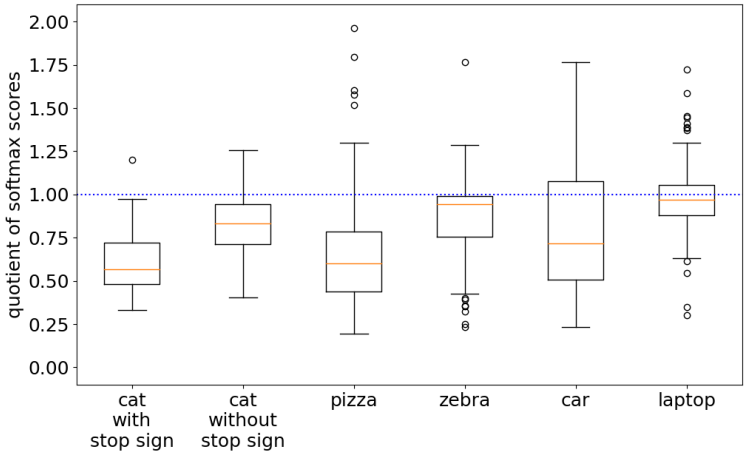


(b) F_{50} on mask created by F_0

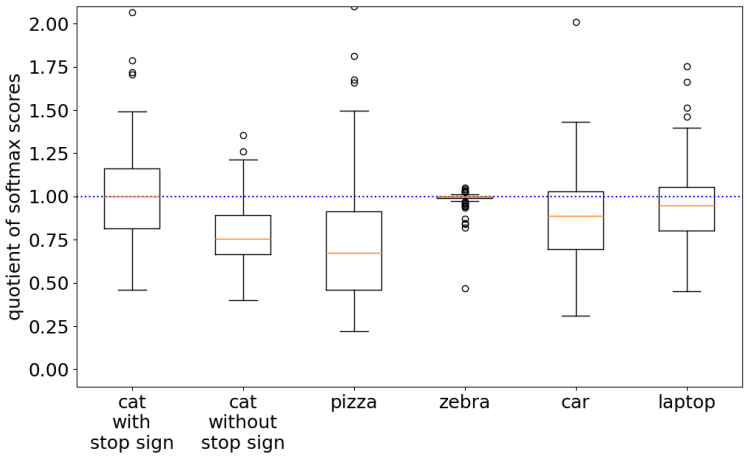


(c) F_{100} on mask created by F_0

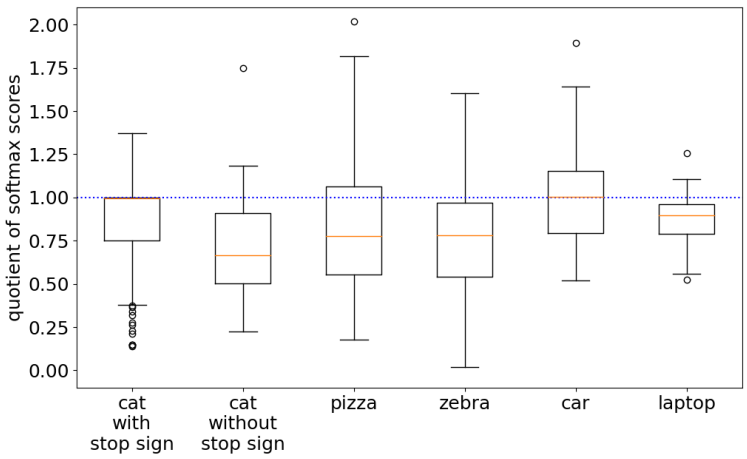
Figure A.3.: The quotient of the softmax score of a mask created by another model divided by the softmax score created by itself.



(d) F_0 on mask created by F_{100}



(e) F_{50} on mask created by F_{100}



(f) F_{100} on mask created by F_{50}

Figure A.3.: The quotient of the softmax score of a mask created by another model divided by the softmax score created by itself.

Bibliography

[1] Sebastian Bach, Alexander Binder, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. Analyzing classifiers: Fisher vectors and deep neural networks. *CoRR*, abs/1512.00172, 2015.

[2] Guillaume Charpiat, Nicolas Girard, Loris Felardos, and Yuliya Tarabalka. Input similarity from the neural network perspective. *CoRR*, abs/2102.05262, 2021.

[3] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously, 2019.

[4] Ruth Fong, Mandela Patrick, and Andrea Vedaldi. Understanding deep networks via extremal perturbations and smooth masks. *CoRR*, abs/1910.08485, 2019.

[5] Ruth C. Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[6] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, Nov 2020.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[8] Maximilian Idahl, Lijun Lyu, Ujwal Gadiraju, and Avishek Anand. Towards benchmarking the utility of explanations for model debugging, 2021.

[9] Volkan Kayatas. Utility-driven interpretability of neural networks, 2021.

[10] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions, 2020.

[11] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[12] Zachary C. Lipton. The mythos of model interpretability, 2017.

[13] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *CoRR*, abs/1705.07874, 2017.

[14] Christoph Molnar. *Interpretable Machine Learning*. 2019.

[15] Vitali Petsiuk, Abir Das, and Kate Saenko. RISE: randomized input sampling for explanation of black-box models. *CoRR*, abs/1806.07421, 2018.

[16] Pouya Pezeshkpour, Sarthak Jain, Byron Wallace, and Sameer Singh. An empirical comparison of instance attribution methods for NLP. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 967–975, Online, June 2021. Association for Computational Linguistics.

[17] Guanwen Qiu, Xiaobing Yu, Baolin Sun, Yunpeng Wang, and Lipei Zhang. Metastatic cancer image classification based on deep learning method, 2020.

[18] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Yi Ding, Aarti Bagul, Curtis P. Langlotz, Katie S. Shpanskaya, Matthew P. Lungren, and Andrew Y. Ng. Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. *CoRR*, abs/1711.05225, 2017.

[19] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. *CoRR*, abs/1602.04938, 2016.

[20] Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 2662–2670, 2017.

[21] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

[22] Sumit Saha. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`, 2018. [Online; accessed 13-February-2022].

[23] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, Oct 2019.

[24] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2014.

[25] Gaurav Singhal. Transfer Learning with ResNet in PyTorch. `https://www.pluralsight.com/guides/introduction-to-resnet`, 2020. [Online; accessed 01-December-2021].

[26] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net, 2015.

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

[28] Ross Wightman, Hugo Touvron, and Hervé Jégou. Resnet strikes back: An improved training procedure in timm, 2021.

[29] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.