EVALUATING FEATURE ATTRIBUTION METHODS

by
BENEDICT GATTAS

School of Information Technology and Electrical Engineering,
The University of Queensland.

Submitted for the degree of
Bachelor of Engineering
in the field of Software Engineering.

June 2020.

Benedict Gattas
benedict.gattas@uqconnect.edu.au

June 22, 2020

Prof Amin Abbosh
Acting Head of School
School of Information Technology and Electrical Engineering
The University of Queensland
St Lucia, Q 4072

Dear Professor Abbosh,

In accordance with the requirements of the degree of Bachelor of Engineering (Honours) in the division of Software Engineering, I present the following thesis entitled "Evaluating Feature Attribution Methods". This work was performed under the supervision of Dr Alina Bialkowski.

I declare that the work submitted in this thesis is my own, except as acknowledged in the text and footnotes, and has not been previously submitted for a degree at The University of Queensland or any other institution.

Yours sincerely,

Benedict Gattas.

Acknowledgments

I wish to acknowledge the direction and support provided by my supervisor Dr Alina Bialkowski over the course of this project. Feedback after each milestone was invaluable and tips on different approaches helped provide guidance at key junctions. This ranged from helping narrow the scope at the start of the project, suggestions after each deliverable to help improve the next one, and regular appraisal of the direction I was heading throughout the project. It would also be remiss of me not to offer congratulations on baby Emilie's birth! To my family, I'd like to thank my parents and my siblings Joe, Nick and Susannah for their endless care and support.

Abstract

This document is a skeleton thesis for 4th-year students. The printable versions show the structure of a typical thesis with some notes on the content and purpose of each part. The notes are meant to be informative but not necessarily illustrative; for example, this paragraph is not really an abstract, because it contains information not found elsewhere in the document. The LaTeX 2_ε source file (`skel.tex`) contains some non-printing comments giving additional information for students who wish to typeset their theses in LaTeX. You can download the source, edit out the unwanted material, insert your own frontmatter and bibliographic entries, and in-line or `\include{}` your own chapter files. Of course the content of a particular thesis will influence the form to a large extent. Hence this document should not be seen as an attempt to force every thesis into the same mold. If in doubt about the structure of your thesis, seek advice from your supervisor.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

"State of the art" machine learning models now regularly achieve above expert-level performance in fields as diverse as medical imaging and language translation. However, their lack of interpretability prevents their adoption in many of the fields that would benefit the most from them. These domains are often ones where the decisions made have a tangible impact on people's lives. Decision-makers in those domains are unlikely to use the predictions made by a highly accurate but opaque model because of their requirement for trust and accountability.

The requirement for explainability is not in order to replace human experts, but to understand contradictions between expert and algorithm. For example, a radiologist might disagree with a diagnosis made by a model trained to predict pneumonia from chest X-rays, and attributing the error to a factor that one or the other relied upon would be helpful. Was the model relying on some unrelated part of the scan (a spurious feature) or was the radiologist failing to pick up on a subtle pneumonia differential? The former was observed in practice after applying an explanation technique to a convolutional neural network (CNN) trained on X-ray data [1].

In the domains that require trust, explanations for model predictions are as important as low rates of incorrect predictions. This is the counter-argument to the notion that a highly accurate model need not be interpretable to be effective: the premise of effectiveness requires a level of trust that a model relies on unbiased data and non-spurious features, two guarantees that are not at all provided by an objective function that seeks only to minimise prediction error. With poor visibility into the factors that a model relies upon, machine learning researchers tend to use model performance metrics as the basis for arguing a new model architecture is superior. This disconnect between performance in the sense of test set accuracy and

performance in the sense of accountability (lack of bias or spurious features) and reliability (sensical behaviour) is not ideal.

Interpretability in a Facebook algorithm recommending product categories, for example, might not be seen as important as interpretability in a cancer diagnosis model, though the possibility of unethical model behaviour from reliance on biased data is as tangible in both domains. In one study of 200 sentiment analysis classifiers, several were found to have significant race and gender bias [2]. The consequences of errors can certainly be higher in some domains however - a poor Netflix recommendation is not as disastrous as a naive algorithm used in government decision-making, such as the "robo-debt" scheme recently employed by the Australian Government [3].

Approaches to Interpretability

There is fortunately an active literature aimed at addressing this 'black-box' critique in machine learning. The top-level distinction among approaches is to either use inherently intepretable models to achieve explainability, or take complex, black-box models and find techniques to isolate and explain a piece of their complexity, such as an individual prediction.

The first approach includes model families with low complexity like linear/logistic regression, decision trees, k-nearest neighbours and Naive Bayes. Within these families are both parametric and non-parametric techniques, which suggests that lack of interpretability is more related to model complexity than a particular type of formulation. This empirically observed trade-off between accuracy and interpretability is discussed further in the next section[1]. Since many of these models are often too simplistic for obtaining competitive performance, the motivation to attack the 'black-box' critique from this angle is quite low. Instead, methods to introduce interpretability to modern, high-performance models are a more studied and popular approach to take in the literature, as in the second approach.

The second approach includes "feature attribution" or "feature importance" methods, which compute a weight score for each feature in the input space to measure its contribution to an output class. For example, a CNN classifier predicting "tree" would be expected to rely heavily on green pixels of leaves. This class includes model-specific techniques for neural network architectures, like those based on activations in a hidden layer, and model-agnostic techniques that are compatible with any model family. Within both sub-classes are a variety of techniques, with varying

[1]There is a view by some researchers ([4]) that in many domains the accuracy vs interpretability trade-off does not exist, and thus there is a responsibility to use equally effective, inherently interpretable models for high-stakes decisions where those are available.

levels of model agnosticity and task compatibility. For example, some methods are designed solely for CNNs and are therefore mainly suited for image-related tasks.

Importantly there are both global feature attribution methods, which calculate each feature's contribution to a model at large, as well as local methods, which attempt to explain a single prediction. This project has focused on the latter. Local methods are dominant in the literature for modern architectures - when the dimensionality of the data is high, such as in visual data, or when the number of parameters is too high to make conclusions about global model behaviour (as in most modern architectures) this tends to be the only effective approach to interpretability. An author of one local, model-agnostic method notes that understanding these models globally "[..] would require understanding of the underlying prediction function at all locations of the input space" [5].

Accuracy vs Interpretability

As deep learning and other state-of-the-art model families proliferate in their typical number of parameters, global behaviour has become even less explainable. Researchers maintain some intuitions about the impact of architectural design decisions, though not on predictive behaviour. For example, filters within a CNN model have been shown to act as 'object detectors' of patterns, shapes and other connected regions [6], though these per-layer intuitions don't explain how a network of dozens of layers will determine a husky from a wolf (a recent approach in the literature, however, has looked at abstracting model behaviour into 'concept' vector encodings to capture model behaviour across filters and layers (Net2Vec [7], TCAV [8])).

Feature attribution methods can therefore re-introduce transparency into complex, non-linear models and highlight predictive biases in the context of individual predictions. They can also reveal unexpected features involved in a prediction, such as the spurious features mentioned in the previous X-ray data example, or bugs that could lead to exploitation of adversarial examples [9]. Note these methods do not seek to add causal interpretability to the models they are applied to, only to isolate and highlight a piece of complexity in a way that might make sense to a person reviewing the explanation. This does not make them shallow - the benefit of the 'post-training' approach is that model designers have more flexibility in their choice of models and fewer restrictive assumptions about model complexity need to be made[2]. More model-agnostic methods, with the least restrictive requirements, are not well understood in context with model-specific ones in terms of this accuracy and interpretability trade-off.

[2]The counter-argument made by those who argue for the inherently interpretable model approach is that there is no guarantee these explanations are faithful to the model, and that they extend the authority of the black box instead of making it a "glass box" [4].

Existing Evaluations

Comparisons of feature attribution methods do exist, though typically either in a qualitative context, as a pairwise comparison, or within a single class of methods. They are not normally compared on practical considerations like performance or adoptability in terms of task and model compatibility. They are however compared haphazardly on explanation quality, which is a difficult criteria to design. Lack of evaluation is partly due to differences in method formulation and output representation, but also this difficulty in finding objective proxy metrics of explanation quality.

1.2 Project Overview

This project has sought to evaluate a panel of feature attribution methods representative of different approaches to the interpretability problem. The aim was to highlight their relative strengths and weaknesses and thereby increase the understanding of the benefits of one method's approach over another.

Two other key contributions made over the course of the project have been a quantitative evaluation framework for explanation quality in the image classification context, and the development of a software package to collect image data explanations for multiple underlying methods at scale.

In summary the project aims have been to:

1. **Examine and evaluate** a panel of feature attribution methods using proxy metrics of explanation quality supported by analysis on performance, use cases and other criteria.

2. **Develop** an attribution software package for testing methods at scale, with modular support for different metrics, methods and models, making it easier for researchers to collect explanations and build more adoptable models.

Project Scope

Section 1.1 introduced a broad motivation for interpretability though this project has focused specifically on image classification for two main reasons.

Firstly, many interpretability techniques from before the deep learning era have been studied in this domain, and many deep learning methods continue to be developed and tested in this domain on modern CNN architectures. The natural visual aspect of image data explanations has also made computer vision a dominant venue for interpretability research, with important applications such as medical imaging.

Secondly, well-annotated datasets and pre-trained, 'off the shelf' models are more easily acquired in this domain. This availability allowed for richer evaluation metrics and the removal of model training as a project requirement.

A more detailed scope is provided at the beginning of the Methodology section, including a description of the specific datasets, models, and feature attribution methods used.

Report Overview

Chapter 2 overviews the available feature attribution methods and existing approaches to method evaluation. Chapter 3 breaks down the project's methodology in terms of particular milestones, including the software and evaluation metrics that were designed. Chapter 4 lists evaluation results from quantitative and qualitative standpoints. Chapter 5 provides a discussion on the project's methodology and the limitations encountered, and finally Chapter 6 provides conclusions and recommendations for future work.

Chapter 2

Related Work

2.1 Scope of Research

In Chapter 1 ("Approaches to Interpretability") a brief overview of feature attribution methods was provided with reference to a distinction between model-specific and model-agnostic methods. This is a common distinction in the literature and was also used to guide research in this project. The panel of methods chosen for evaluation ultimately consisted of a balanced selection from both approaches.

A major difficulty of this project was distilling the broad literature on these methods however. For the model-specific (neural network) family, different angles are commonly taken to calculate feature importance. These include generally:

- **Backpropagation-based** methods or importance signal projections (such as activations in a hidden layer)
- **Gradient-based** methods, saliency maps and output sensitivity methods
- **Perturbation-based** methods and input occlusion techniques

This categorisation is based on two recent papers that make similar categorisations of explanation approaches [10] [11]. In this chapter a broad selection of methods based on traction over time, current popularity and representativeness of approach are described, though the reader should note there are more methods under each of those three than have been described here.

For model-agnostic methods, perturbation-based and other miscellaneous approaches are considered. The selection was again based on traction.

First reviewed are traditional, visual approaches to interpretability to provide context to the task of feature attribution. After exploring feature attribution methods, an extensive review of existing comparison studies, evaluation metrics and software frameworks is provided.

Terminology

All methods are variously referred to as *attribution* methods in this project for any projection on the input space that highlights relevant features. Adebayo et al. (2018) instead refer to the broad category of "[...] visualisation and attribution methods aimed at interpreting trained models" as *saliency* methods, particularly in the context of image data [12]. Including by those authors, a *saliency map* is widely used as a catch-all term to refer to any input space projections (individual explanations) in the context of image data. However they also refer to a specific gradient-based method (Section 2.3.2).

Consensus on terminology is relatively lacking. Some researchers ([11]) describe attribution methods as a subclass of explanation techniques where contribution scores are specifically calculated for each input feature (i.e. excluding higher level 'patterns' which cause neuron activations, as in Zeiler & Fergus (2013) [13]).

In summary, an *attribution method* is used in this report as a general term. Saliency maps, explanations and attributions are all interchangeably used for any attribution method's output.

2.2 Traditional Approaches

2.2.1 Feature Projection

Visualisation tools for high-dimensional data are a popular way to gauge insight into expected model behaviour. These pre-learning, exploratory data analysis techniques include mathematical reductions like PCA and probabilistic techniques like t-SNE, projecting high-dimensional examples that are 'similar' into a visualisable 2D or 3D space (Figure 2.1) [14] .

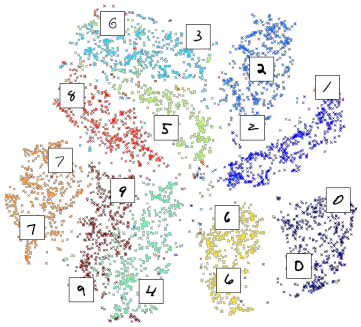

Figure 2.1: t-SNE embedding of 70,000 handwritten digits from MNIST [15].

Other methods of clustering and dimensionality reduction are also widely used for interpreting data, and although useful for gaining an intuition on relationships

between features, they are not suited for explaining model behaviour as they examine only the input space itself.

2.2.2 Partial Dependence Plots

A partial dependence plot (PDP) is a tool to demonstrate the marginal effect of one or two features on a prediction outcome. It was proposed by Friedman in 2001 to interpret and visualise the input features that the paper's proposed gradient boosting machine relied upon (though it is limited to 1 or 2 features such that it can be displayed) [16]. A partial dependence function $\widehat{f_{xs}}$ can be calculated for some desired set of features S, by marginalising the model output over the set of complement features C (all other features):

$$\widehat{f_{xs}} = \int \widehat{f}(x_s, x_c) dP(x_c) \tag{2.1}$$

It can be approximated with a Monte Carlo method. Friedman believed in 2001 that these might be used to help interpret "any black box prediction method" such as NN and SVM architectures, and that, "[...] when there are a large number of predictor variables, it is very useful to have a measure of relevance to reduce the potentially large number of variables to be considered" [16]. The mentioned relevance measure was defined only in the context of the decision trees which constituted the paper's gradient boosting machine. Certainly, PDPs are suited for the lower-dimensional feature spaces that were imagined in the pre-deep learning era, and are less suitable for high-dimensional input spaces such as in image classification. They are also restricted by an unrealistic assumption of independence among features.

2.3 Model-Specific Methods

Deep learning's reputation for lack of transparency has led to many attempts to explain the predictions of complex NN architectures. This section examines representative attribution methods from the backpropagation-based, gradient-based and perturbation-based approaches overviewed in Section 2.1, with some emphasis on those developed in the context of CNNs (i.e. image data).

2.3.1 Backpropagation-Based

Methods in this class try to isolate an internal model signal, such as neuron activations in a target hidden layer, and map these signals back into the input pixel space. Zeiler & Fergus (2013) introduced the motivation for signal backpropagation as "[...] showing what input pattern originally caused a given activation in the feature maps" [13].

Visualising Activations - DeconvNet, Guided Backpropagation

Visualisation of per-layer activations is one approach to explain inner model behaviour. It is different from other attribution approaches in that it seeks to visualise 'learned' features instead of the contributions of input space features. DeconvNet and Guided Backpropagation are two popular examples and are briefly described here as two forerunners of the backpropagation (and gradient) approach.

Deconvolutional networks (DeconvNet) generate backwards projections of neuron activations, by reversing individual activations during a 'deconvolution' backwards-pass [13]. The procedure can be summarised as passing feature map output activations as input into 'deconv' layers, iteratively reversing activations and reconstructing input signals until the input pixel space is reached.

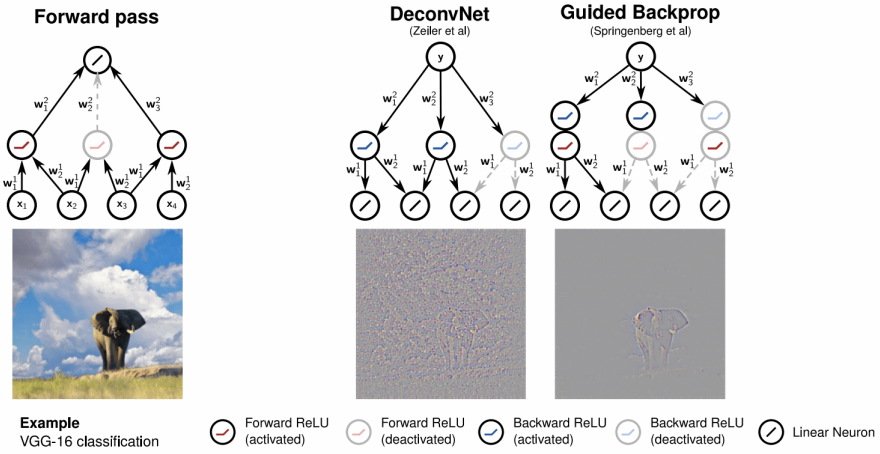

Figure 2.2: (Adapted from [11]) Illustration of DeconvNet and Guided Backprop.

Since it can target one layer's activation at a time, the method is useful for understanding the hierarchical learned features that CNNs generate over multiple layers. Its limitation is that during the backwards pass, it ignores any negative inputs to ReLu activations that were zero'd out in the forward pass (deactivated activations in Figure 2.2).

Guided Backpropagation was an enhancement by Sprigenberg et al. (2014) that added an additional signal at each step by zero'ing the importance signal if it was a negative activation in the forward pass phase *or* negative in the backwards pass (the two intermediate signals in Figure 2.2 right) [17]. This stopped negative gradients in lower layers from decreasing the activation of the higher layer units which were the target, and this leads to sharper explanations than those created by DeconvNet [17].

DeepLIFT

DeepLIFT (Deep Learning of Important FeaTures) [10] was created out of the motivation that the zero'ing of negative gradients by DeconvNet and Guided Backpropagation meant that neither are able to highlight inputs that contribute negatively to an output. In some sense they are therefore missing half the story of feature contribution / neuron activation. DeepLIFT's authors (Shrikumar et al. (2017)) also wanted to overcome the unaddressed saturation problem, which is that relevant neurons that contribute to a saturated output activation would not individually change the output if they were turned off (as might be tried in perturbation approaches).

DeepLIFT's innovation over previous backpropagation and gradient-based methods was to realise that where this problem existed, a single gradient value of an output with respect to an input value did not adequately or necessarily capture input contributions to an output. The authors' proposal was to find input contributions by instead calculating the absolute change in output with respect to a neuron's 'reference' activation.

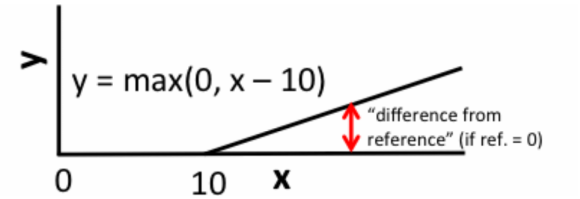

Figure 2.3: (From [10]) Difference-from-reference attributions can avoid bias terms.

Finding reference activations is an implementation difficulty. Generally, the authors propose all-zero input as a baseline (a black square for image data), and a better reference to be the average input over a background data sample [10].

Like other backpropagation methods, DeepLIFT is fast to calculate as it requires only a single backwards pass to propagate an importance signal back into the input space. The authors also provide different formulations for practical implementation, and a relatively high-level public implementation [18]. The concept itself is also compatible with any NN architecture or application, unlike for example DeconvNets and Guided Backpropagation, designed for ReLu activation functions, and GradCAM (Section 2.3.2) which was designed for CNNs.

Other Backpropagation Methods

DeepLIFT is one of several modern methods to decompose a network's prediction onto input features using activation backpropagation. For brevity its main competitors, Layerwise Relevance Propagation [19] and DeepTaylorDecomposition [20], are omitted here but mentioned for the reader's reference.

2.3.2 Gradient-Based

These methods aim to explain a class output in terms of sensitivity in the input space by relying on a gradient function of the output[1]. The goal is to find the input features that make that prediction more or less confident: for an output class of 'tree' they seek to answer, "What makes a tree more or less a tree?".

Saliency Maps

An early formulation of a local explanation method was provided by Baehrens et al. (2010) for any nonlinear classification algorithm (though developed in the context of Bayesian classification) [21]. The local explanation gradient vectors that this paper devised were based on class probability gradients, characterising how much a data point has to be moved for a predicted label to change.

Simonyan et al. (2014) later applied a similar idea to CNNs to create 'class saliency maps' specific to a given image and class [22].

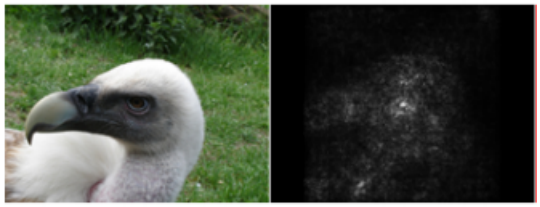

Figure 2.4: (From [22]) Example of an image-specific class saliency map.

The formulation is based on finding the derivative of an output class with respect to an input image via back-propagation. The authors also formulate a method to generate an image that maximises the output class score for a particular class, to visualise the model's 'interpretation' of a class.

This paper sparked great interest in CNN explanations and further interest in creating explanations from network gradients generally. Along with DeconvNet and Guided Backpropagation (developed relatively simultaneously with similar ideas) these three methods are the most historically popular and influential saliency techniques[2]. A drawback of saliency maps is that noisy images can be produced when a model does not distinguish between objects that are being predicted and nearby objects that are associated (i.e. a tree with leaves, in an image of a bird).

[1]There is a strong overlap between gradient and backprop. methods: gradients as derivatives are approximated via backprop, and the weights updated by these gradients in training are the contributions to the neuron activations measured via backprop. techniques.

[2]Saliency maps are synonymous with gradient methods to the point where it is sometimes referred to as simply the 'Gradients' technique, as in Adebayo et al. (2018) [12], who may have done so to separate the term from saliency maps generally - see Section 2.1 ("Terminology").

Grad-CAM

Class activation maps (CAMs) are another approach aimed at understanding the behaviour of CNNs introduced by Zhou et al. (2015) [23], based on the motivation that deeper convolutional layers capture higher-level visual constructs while retaining spatial information [24]. While examining global average pooling (GAP) layers, a technique previously suggested for regularisation during training [25], the authors realised a final convolutional layer's separate RGB channels (feature maps) could be input into a GAP layer with outputs used as features in a fully-connected layer, just before the final softmax layer. The class-associated weights in that fully-connected layer can be combined with the original final convolutional layer feature maps to capture deep representations as object localisations/class 'activation' maps:

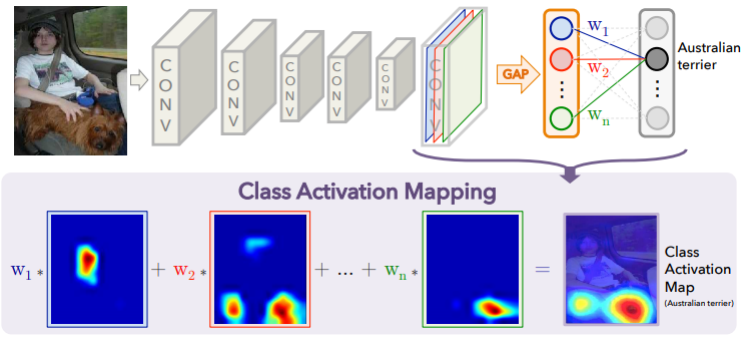

Figure 2.5: (From [23]) Summary of the CAM formulation. RGB channels are emphasised as inputs into the weighted sum that creates a CAM.

A major limitation of the CAM approach is that it requires specific CNN architectures without previous fully-connected layers, and a GAP layer to be added before the output softmax layer to generate the deep representations it visualises. As well as this being a hurdle to adoption, the representations are highly coarse and only roughly approximate class-associated regions (a reason for heatmap presentation).

Grad-CAM was proposed by Selvaraju et al. (2016) aimed at making CAMs applicable to a wider range of CNN models and for visual tasks other than image classification [24]. It requires no alteration to model architecture. It still targets the final convolutional layer's channels, as in CAM, but uses the *gradient* of an output class score with respect to these channels' output activations to then globally-average-pool the gradients over the layer's width and height dimensions. The importance weights produced can be visualised as a localised heatmap over the input space, though the authors also multiply a Guided Backpropagation output (Section 2.3.1) with the heatmaps to generate a class-discriminative, "high-resolution" version called Guided Grad-CAM (Figure 2.6).

One of Grad-CAM's strengths is that its authors prove its effectiveness for a va-

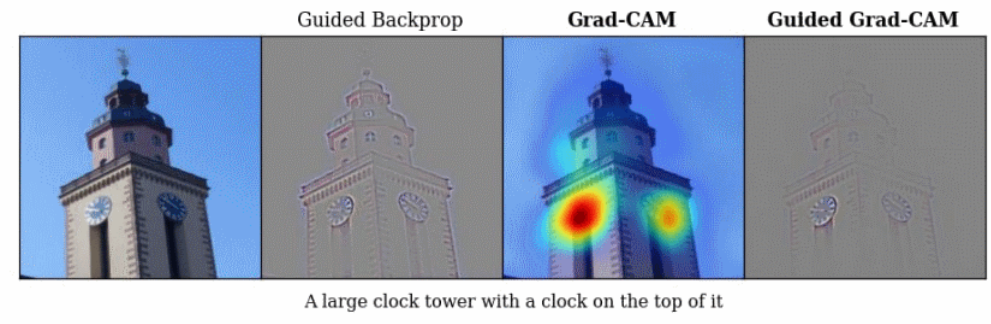

Figure 2.6: (Adapted from [24]) *Guided Backpropagation*, *Grad-CAM* and *Guided Grad-CAM* on an image captioning example from the Neuraltalk2 model.

riety of use cases. These include highlighting causes of incorrect predictions ('failure modes'), the effect of adversarial noise and causes of model confusion, and identifying training dataset bias[3]. Its application to a variety of models by other researchers ([26], [1]) is one testament to its authors' claims on model compatibility and explanation quality, as well as its popularity in the literature (>2000 citations).

Other Gradient Methods

Three other gradient-based methods are widely cited and relevant to discussion - Integrated Gradients [27], Gradient * Input [28] and SmoothGrad [29].

Integrated Gradients is similar to DeepLIFT - instead of taking difference-from-reference activations as a signal of contribution, the approach is to integrate the possible gradients over all inputs (from 0 up to the activation caused by the input image) and then use this area under the activation function as an information measure of input relevance. It also addresses the saturation problem (Section 2.3.1) though is naturally expensive to compute. Approximating the integral requires a baseline example that can provide zero input score as a comparison, and the method is applicable to a variety of architectures (two other similarities with DeepLIFT).

Gradient * Input was a simple proposal by DeepLIFT's authors to sharpen the saliency maps of Simonyan et al. (2014) [22] by multiplying the gradient with the original input signal. SmoothGrad was another attempt to sharpen gradient-based saliency maps, that noted gradients as derivatives can fluctuate sharply at local scales. To compensate for the noise in the output explanation, they propose a local average of gradient values that can be computed based on random samples of the original input image with Gaussian noise added.

[3]For example, they showed that a VGG model trained to classify nurses from doctors had learned to look at long hair to incorrectly label a female doctor a nurse, and the bias was because of gender-skewed training data.

2.3.3 Other Model-Specific Techniques

Backpropagation and gradient methods can be viewed as reflections of one approach based on a similar assumption: that propagating a relevant signal back through a neural network model is a means to explain how the signal was originally encoded. A third and more unrelated category of model-specific methods are perturbation techniques. These treat the underlying model as a true black box by iteratively occluding patches of the input feature space in order to measure the change in output class score. Where the occlusion caused a noticeable change, the inference is that the region masked was important. Zeiler & Fergus (2013) [13] pioneered this approach in the same paper that proposed DeconvNets described in Section 2.3.1[4].

An influential work by Fong & Vedaldi (2017) [30] formalised a meta-predictor framework that can *learn* masks via what they describe as a deletion minimisation problem. Summarised, the informative region is found by minimising the size of a small deletion mask that causes the output score to drop significantly. This mask generation framework leads to similar results as other methods mentioned so far (Figure 2.7). The authors rely similarly on gradients to extract information to solve their optimisation problem, but a key performance drawback is that *many* gradient calculations for successively increasing mask sizes are required.

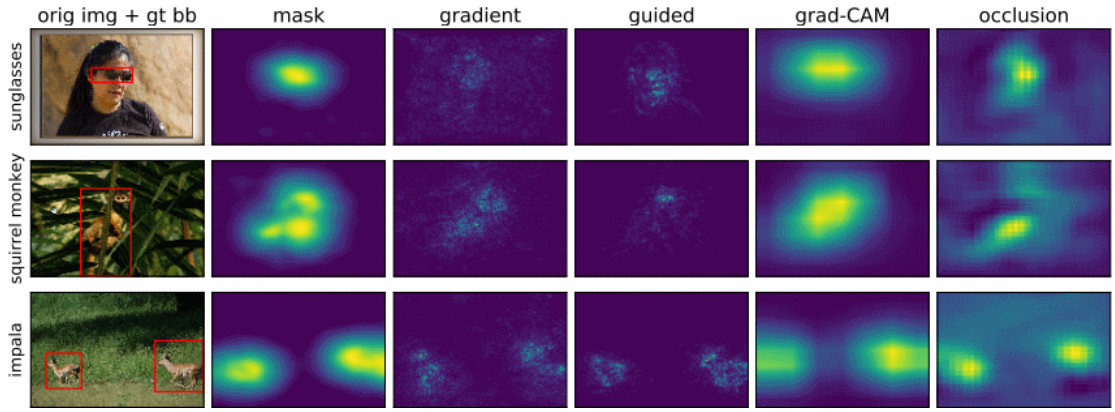

Figure 2.7: (Adapted from [30]) Comparison of methods discussed so far: Mask generation [30], Gradient (saliency maps) [22], Guided Backpropagation [17], Grad-CAM [24] and Occlusion [13]. Ground truth bounding boxes are labelled "gt bb".

For completeness, some NN methods which do not attribute the contribution of input features specifically (or project internal model signals onto the input space) are not within project scope as a result, but are mentioned here. These are 'concept vector' methods that extract model behaviour across layers: Net2Vec [7] and Testing with Concept Activation Vectors (TCAV) [8].

[4]The grey-square masking method from the DeconvNet paper is usually referred to as 'Occlusion' in the literature.

2.4 Model-Agnostic Methods

In parallel to the literature aimed at explaining neural network model behaviour, a standalone approach in the interpretability literature is to find model-agnostic explanations. Similar to the motivation for perturbation-based methods (and with strong overlap), these are black-box methods that make no restrictive assumptions about model architecture or classification task at all, and therefore can theoretically be applied to any model or task. Stark differences in approach and computational complexity still exist however. The most prominent methods in this sub-field are listed below.

2.4.1 Perturbation-Based

As described in Section 2.3.3, perturbing the input space by toggling feature patches is one way to isolate model behaviour. Changes in output score reveal importance in the occluded inputs. Several methods implement this intuition by training "surrogate", interpretable models around the isolated features that are locally relevant to a single prediction. These include Local Interpretable Model-Agnostic Explanations (LIME) [31], Anchors [32] and similar variants [5].

LIME

LIME's authors propose that non-linear, complex decision boundaries can be approximated locally around a single prediction via a simpler model that only relies only on the "neighbourhood" of the relevant input space [31].

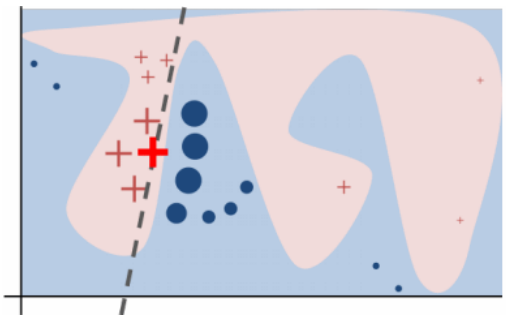

Figure 2.8: (From [31]) LIME's intuition about complex decision boundaries.

To compute explanations, LIME trains an interpretable model (i.e. decision tree or Lasso regression) on permuted samples of features from a given instance, weighted by their proximity to the instance being explained. The training simply minimises mean-squared error based on the samples' black-box predictions. Finally, the interpretable model's weights on these proximal features (superpixels in the case of image data) represent the explanation for the original instance.

The formulation in the paper is relatively vague however, particularly with regards to the size of the neighbourhood that should be used for finding relevant features. The vagueness was potentially motivated by their intention to be agnostic about the choice of explanation model and the black-box model, though this leaves questions about the accuracy-interpretability compromise largely unaddressed. Generating samples and training models for each sample is also extremely costly to performance. Even for a confident prediction and using many samples, instability from random sampling makes an explanation undesirably non-deterministic. This was noted by the authors [31]. The authors of the mask generation technique in Section 2.3.3 note that LIME bears similarity to their perturbation approach, but takes significantly longer to converge and produces a coarser heatmap via superpixels instead of their pixel-level attribution [30].

Despite these shortcomings, LIME's simplicity and compatibility across model families and classification tasks explains its high popularity among model-agnostic methods.

Anchors

LIME's problems motivated a more recent successor called "Anchors: High-Precision Model-Agnostic Explanations" by its original authors [32]. They replaced the use of simple local models with high fidelity if-then rules around a prediction, and give a framework for finding those rules efficiently. A rule (termed an 'anchor' explanation) is one that, "[...] sufficiently anchors an explanation locally, such that changes to the rest of the features of the instance do not matter" [32].

Formal ways to define the optimal region of influence (neighbourhood), build rule accuracy and calculate rules efficiently were devised - three key improvements over their LIME proposal.

Though these anchor rules are highly interpretable, code available from the authors is limited and (possibly as a result) it has less popularity both in practice and in the literature. The GitHub repository provides for example a rough implementation for only text and tabular data and not image data [33].

2.4.2 Other Model-Agnostic Methods

SHAP Framework

Many methods have been listed so far. The SHAP framework (SHapley Additive exPlanations) by Lundberg & Lee (2017) was a push-back on this issue of method proliferation, showing links between many existing methods with a theoretical approach from game theory [34]. It has become extremely popular among researchers

from all sub-fields in interpretability due to its attractive formal properties and 'outsider' formulation, as well as its implementation approximations for many model families and a well-maintained GitHub repository to host those implementations[5].

The game theoretic concept of Shapley values are a unique solution to the problem of calculating fair, marginal per-player rewards for the reward earned collectively in a cooperative game [36]. Since each player's contribution may produce interaction effects with others, they are calculated by averaging contributions in all possible sub-coalitions of players. In the context of machine learning, if features are taken as players, the method can be applied to a prediction (the "payoff") and it retains its per-player theoretical properties. These qualities include *efficiency*, which is that the sum of p constituent input feature contributions (Shapley value ϕ_j for a feature j) must equal the difference between a prediction of an input x ($\widehat{f}_x$) and an average for all inputs (an 'expectation' of model output $E_X(\widehat{f}(X))$):

$$\sum_{j=1}^{p} \phi_j = \widehat{f}_x - E_X(\widehat{f}(X)) \tag{2.2}$$

A key problem is that finding ϕ_j requires iteratively computing outputs for all feature subsets including j and is therefore computationally infeasible for large input spaces[6].

SHAP values are an implementable version of Shapley values. They connect Shapley value theory with local explanation techniques including LIME and DeepLIFT [34]. They are formulated as the difference between expected model output (approximated by an average background data sample's prediction) and the instance at hand being predicted. The authors show that the game theoretical properties including efficiency apply to a whole class of 'additive feature attribution methods', both model-specific (e.g. DeepLIFT) and model-agnostic. They provide several model-specific approximations for implementing their attributions, one based on Integrated Gradients and another based on DeepLIFT.

2.5 Evaluation of Attribution Methods

Creators of feature attribution methods tend to provide visual comparisons with existing methods, as for example in Figure 2.7. An issue is that qualitative evaluation is implicitly left to the reader, who is encouraged to infer the author's saliency maps are either equal in standing or more aesthetically pleasing than another method's.

[5]The SHAP paper has over 1000 citations and its GitHub has over 9000 'stars' at the time of writing [35], despite only being published in late 2017.

[6]They are however commonly used in low-dimensional linear regression settings, like in economics for example, for calculating global feature importances when features are dependent [36].

Method creators *do* often suggest their own interpretability criteria, though this is done with little reference to criteria in the prior art. The interpretability field's agreement on implementable standards for interpretability is therefore low. To some extent, the proliferation in attribution methods and variants can be attributed to these unfortunately subjective and still actively researched standards. In summary, two unresolved problems in interpretability research can be thought of as agreement on desiderata for interpretability, and lack of evaluation on any criteria *independent* of method creators. This thesis does not seek to tackle the first issue though makes a contribution in the form of saliency metrics. The main contribution is towards increasing independent understanding on common criteria.

Existing efforts to independently compare and unify methods are listed here. The SHAP framework can be thought of as one example as it shows how existing methods are related and that it demonstrates where one can be preferable. This research section therefore aims to:

1. Overview attempts like SHAP that have independently evaluated methods, performed 'unification' or comparison studies, or independently proposed desiderata for interpretability. This includes theoretical and practical standpoints.

2. Examine criteria that method authors have themselves proposed.

3. Describe existing software for generating explanations that package several methods for use by researchers and practitioners.

2.5.1 Existing Studies

The Mythos of Model Interpretability [37]

An early, widely-cited contribution towards common interpretability criteria was made by Lipton (2016) to "define the discourse on interpretability" [37]. The proposed criteria were high-level, coming from the view that different motivations for interpretability are what cause the many different interpretations of the word:

1. **Trust:** We should care about not just "[..] how often a model is right, but also for which examples it is right" [37].

2. **Causality:** Does the model show cause-and-effect reasoning?

3. **Transferability:** Does the model generalise well?

4. **Informativeness:** What contribution does the model make towards increasing understanding of the application domain?

5. **Fair & Ethical Decision Making:** Can we show the model's decisions conform to ethical standards?

Though it did help shape discourse by defining motivations for interpretability, the criteria are *too* high-level to be useful for comparing one feature attribution method's output to another's.

Towards a Rigorous Science of Interpretable Machine Learning [38]

Doshi-Velez & Kim (2017) provided an influential set of guidelines or "evaluation paradigms" for evaluating interpretability techniques, arranged in three levels of increasing abstraction [38]. This motivation was to provide a scientific basis for the different objectives in evaluation that method authors implicitly target (both qualitative and quantitative):

1. **Functionally-grounded**: How well the attribution method performs on quantitative criteria: 'proxy' metrics for explainability like weakly supervised object localisation (Section 2.5.2).

2. **Human-grounded**: How much everyday people agree that an explanation is visually superior to another: user studies as an example.

3. **Application-grounded**: How well the explanation helps domain experts solve real tasks, such as a radiologist agreeing with fractures pointed out by a model trained on X-ray data.

Research on specific criteria below (Section 2.5.2) has focused on functionally-grounded criteria, since these are more common in the literature and since higher-level criteria are difficult to compare objectively.

Towards Better Understanding of Gradient-based Attribution Methods [39]

Similar to SHAP in motivation, an independent attempt at providing a unified framework for the gradient-based class[7] of attribution methods was provided by Ancona et al. (2017) [39]. In their work they prove conditions of equivalence between Layerwise Relevance Propagation and Gradient * Input, and DeepLIFT and Integrated Gradients (Section 2.3). Another major contribution was to propose a generalisation for the additivity property that SHAP's authors noted defined a class

[7]The authors do not distinguish between backpropagation and gradient-based methods.

of methods[8]. This criteria was termed *Sensitivity-n*: "[...] when the sum of the attributions for any subset of features of cardinality n is equal to the variation of the output S_C caused by removing features in the subset" [39].

Some of their key insights were that existing methods like DeepLIFT and Integrated Gradients show very high correlation, that the former therefore acts as an approximation of the latter in practice, and that on complex model architectures like InceptionV3, all gradient-based methods produce noisier saliency maps and less appealing explanations.

Sanity Checks for Saliency Maps [12]

This paper by Adebayo et al. (2018), along with Lundberg & Lee (2017) (SHAP's authors) and Ancona et al. (2017) above, have been the three main independent contributions towards understanding and unifying the model-specific set of feature attribution methods described in earlier sections of this chapter. This motivation was similar though more practical than theoretical: find an actionable methodology to help a practitioner decide between competing attribution methods.

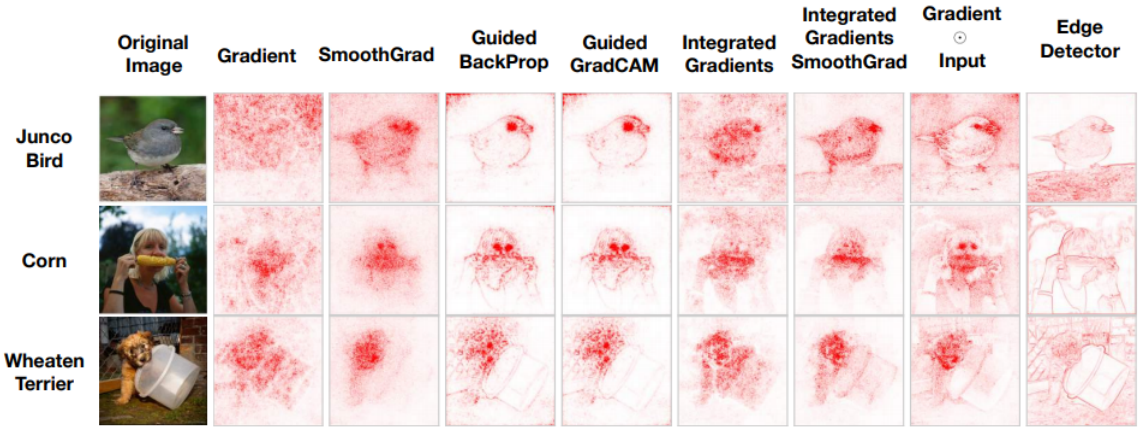

Figure 2.9: (From [12]) The panel of methods examined by Adebayo et al. (2018), all previously discussed in Section 2.3. They highlight that an edge detector alone can produce a mask similar to the output of some methods.

The methodology proposed was a set of two statistical randomisation tests ('sanity checks'):

[8]The authors actually reference the properties of *Completeness* (proposed by the authors of Integrated Gradients [27]) and *Summation to Delta* (proposed by the authors of DeepLIFT [10]), two variants of a similar idea to additivity.

1. **Model Parameter Randomisation Test:** A misleading attribution method could be insensitive to model parameters, so an untrained version of the same model with random weights should not produce a similar saliency map.

2. **Data Randomisation Test:** A misleading method could be dependent on training labels: a test with randomly permuted labels should therefore show significantly different saliency maps.

For comparisons between a normal attribution output and a modified one according to one of those two experiment conditions, the authors use similarity metrics including Spearman rank correlation and a structural similarity (SSIM) index. Note these metrics do not evaluate one method's similarity with another: only the similarity with its modified version. Some insights are prepared from these metrics, like that Guided Backpropagation and Gradient * Input show unexpected insensitivity (perform badly on the sanity checks).

One problematic point they emphasise though is that non-performing saliency methods may only be visually salient because they act like an edge detector (Figure 2.9). Although the work valuably points out that observers might have confirmation bias when viewing highlighted edges in a saliency map, this conclusion ignores the role of connected regions and pixel-wise attribution intensities, which some methods may have as a strength over others. These latter two considerations are arguably as important for visual quality than edges alone, and the authors' similarity metric results also do not relate to the standalone observation about edge detectors.

2.5.2 Existing Criteria

Saliency Metrics (Quantitative Approach)

Some method authors however have made an effort to account for regions in method evaluation. In particular, 'weakly supervised object localisation' (WSOL) has been the technique used for evaluating saliency maps via bounding boxes derived from a segmentation algorithm (in natural image classification settings). It was first applied by Simonyan et al. (2014) while introducing the original saliency maps method (Section 2.3.2) [22] . Those authors used a colour segmentation algorithm on thresholded saliency maps, found bounding boxes on the segmented regions, and then submitted their annotations to an ILSVRC-2013 localisation challenge where they out-performed many fully supervised algorithms.

The paper that introduced CAM saliency maps (Section 2.3.2, [23]) used a similar technique and has been considered the seminal work for use of WSOL [40]. They chose a 20% quantile threshold for CAM values before finding the largest connected

component in each map. Unlike Simonyan et al. who only evaluated via the competition, CAM's authors independently evaluate their derived bounding boxes against ground truth boxes using an intersection-over-union metric (thresholded at 0.5 to exclude instances where the method was significantly off). Their results on an ILSVRC validation set out-performed the benchmarks of localisation-trained models.

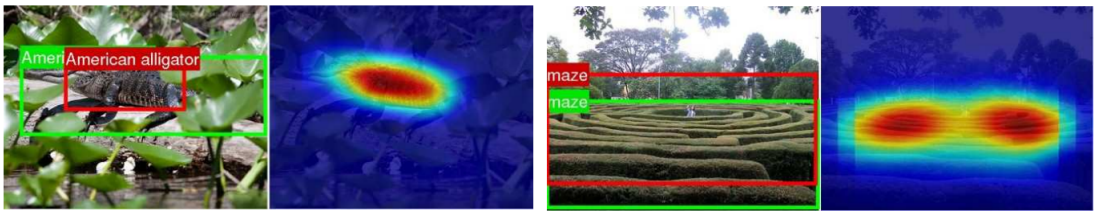

Figure 2.10: (Adapted from [23]) CAM's predicted bounding box in green and the ground truth annotation in red (left in each sub-pane), and the original CAM (right in each sub-pane). Their IOU metric is calculated over the annotations.

Notably this IOU metric does not account for the weight of the attribution in any one pixel, which is relevant to other methods. The methodology also unfairly penalises more perforated-looking saliency maps due to relying on segmentation algorithms to find connected components and therefore ignoring pixel-level detail. On the one hand, connected components are more observer-friendly explanations of single objects (with an implicit assumption that the underlying model also cares about connected regions), though on the other hand some methods may be noisier and so may be punished harshly for having small, disconnected sub-regions yet still maintaining saliency in the 'bigger picture'.

Dabkowski & Gal (2017) compensate for the latter problem by proposing an augmented localisation metric that takes a bounding box of the *entire* salient region, rather than just a thresholded and single-component subset of the saliency map [41]. A strong aspect of their work was to provide a "Max box" baseline in reporting localisation error: this reports the ground truth box's overlap with the whole image. Where the localisation error is similar, this allows them to suggest the interpretability of their generated localisation boxes is similar to the ground truth boxes themselves.

Independently of the explainability literature, pixel-wise mask-based metrics have been developed for generic WSOL applications by Choe et al. (2020) [40], though it seems these have not been applied in the context of evaluating explainability techniques.

Higher Level Criteria (Qualitative Approach)

Some authors sidestep the functionally-grounded approach to method evaluation to suggest *axioms* that are more theoretical in motivation. Table 2.1 summarises prominent axioms in the literature, which are all related to invariance properties that an ideal method should have to be internally consistent:

Criteria	Authors	Description & Insights
Robustness	Alvarez-Melis & Jaakola (2018), [42]	Methods should be invariant to small changes in the input. Model-agnostic, perturbation-based methods are shown to be more prone to instability in input than gradient-based ones.
Input Invariance	Kindermans et al. (2017), [43]	Methods should be invariant to small shifts/transformations in the input, where the model's prediction and output weights are unchanged. Common methods like LRP and Integrated Gradients (Sections 2.3.1, 2.3.2) are shown to fail the criteria.
Implementation Invariance	Sundarajan et al. (2017), [27]	(Creators of Integrated Gradients) Attributions should be identical for two "functionally equivalent" models: where the inputs and outputs are all equal (though where internal implementations may differ). They show that DeepLIFT and LRP fail the criteria.

Table 2.1: Three invariance properties developed for attribution methods.

Across in the model-agnostic literature, Ribeiro et al. (2016) have made the case for model flexibility as a strong criteria for explanation technique usability [44]. When comparing a tree-based model with a deep learning model, for example, neither a practitioner nor a researcher can get a fair idea of interpretability if two explanations are different in representation on account of different attribution methods. The observation by Ancona et al. (2017) that more complex architectures are not explained as well by current model-specific techniques also supports this argument (Section 2.5.1) [39].

2.5.3 Existing Software Packages

There are existing frameworks hosted on researcher GitHub repositories that offer practitioners and researchers a panel of attribution methods in one place. Ancona (2019) maintains a software package called *DeepExplain* [45] to support the framework described in Section 2.5.1. It implements a suite of the methods the authors showed relationships between: Saliency Maps, Gradient * Input, Integrated Gradients, DeepLIFT and Occlusion. It also provides abstraction over these methods with some optional parameter specification to override defaults.

For model-agnostic techniques, *Skater* is a popular framework that combines fully black-box explanation techniques [46]. For perturbation-based techniques, Fong & Vedaldi (2019) publicly released *TorchRay* to initially support the mask generation technique described in Section 2.3.3, though it has since become a research-oriented reproducibility contribution that implements methods from other papers [47].

There are no known software frameworks that specifically aim to collect both model-agnostic and model-specific saliency maps. Including those just listed, existing software packages typically combine only a few methods from one approach. The gradient-based methods and their derivatives are implemented in many packages on account of their popularity, for example, but no known packages include gradient-based, model-agnostic and perturbation-based techniques. Additionally, no software evaluation framework for WSOL-based saliency metric evaluation is known to be available.

Chapter 3

Methodology

3.1 Overview

A panel of four local attribution methods were chosen for evaluation in this project, picked for representativeness of approach and their prominence in the literature. The panel was also a balanced selection from both 'classes' of method approach: model-specific (Section 2.3) and model-agnostic (Section 2.4):

1. **DeepLIFT:** Model-specific class, backpropagation-based approach (2.3.1)
2. **GradCAM:** Model-specific class, gradient-based approach (2.3.2)
3. **LIME:** Model-agnostic class, perturbation-based approach (2.4.1)
4. **SHAP:** Model-agnostic class[1], generic approach (2.4.2)

Other methods could have been included in this panel though as related work has shown (particularly by Ancona et al. (2017)), similarities in formulation should allow conclusions on one method to apply to close relatives. Project constraints also meant that a decision on the panel breadth had to be made according to at least some criteria, and representativeness helps for comparing approaches (a key project goal).

This chapter is presented in order of project milestones achieved, from initial data collection through to the evaluation metrics designed. The evaluation methodology itself can be summarised in terms of the dataset used, 'off the shelf' underlying models relied upon and the evaluation metric approach:

[1]Due to its higher level formulation, SHAP has to be implemented for each model family. It's therefore better described as *partially* model-agnostic.

1. **Dataset:** ImageNet validation set, with ground truth bounding box annotations usually used for object localisation training.

2. **Models:** VGG16 as the primary model, InceptionV3 and ResNet50 for supportive analysis.

3. **Metrics:** Pixel-wise and mask-based WSOL, extending related work in Section 2.5.2.

These are described in more detail in Sections 3.2, 3.3 and 3.8 respectively. Finally, to support the metric analysis, qualitative analysis was also performed though these results are left to the following chapter.

3.2 Data Collection & Annotation

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) uses a subset of the well-known, hierarchically-labelled ImageNet database [48]. The competition's validation set consists of 50,000 images of 1000 categories, and includes annotations for ground truth class labels for image classification, and ground truth bounding boxes for object localisation.

This dataset was chosen because of the convenience of the bounding boxes for method evaluation, and the availability of models pre-trained on ImageNet. Images and bounding box XML files for all 50,000 instances in the 2012 validation set were acquired from an academically hosted torrent.

Each bounding box file contains one or several annotations (several if there are multiple instances of a single class, e.g. Figure 3.1), and each annotation contains the ImageNet class ID and {*x-min, x-max, y-min, y-max*} fields for the box[2].

A script to draw rectangular annotations on the images was created to sanity check the acquired data. Example outputs are shown in Figure 3.1.

Figure 3.1: Collected ImageNet examples with annotations drawn.

[2]These ImageNet 'synsnet' IDs were converted to human-readable class labels using a script that connects to a standalone map of IDs to labels.

Much later into the project this script was upgraded to return the annotated image as a 2D array of 0's and 1's, with 1's inside the bounded regions to create a ground truth mask. The use of this mask for saliency calculation is described in Section 3.8.

3.3 Models & Predictions

Deep CNN models are expensive to train, so the high-level Keras library was used to import image classifiers pre-trained on ImageNet [49]. Three models were chosen representative of common modern architectures (summarised in Table 3.1). They acted as the underlying 'black-box' models being explained.

Model	**Structure**	**Input Shape**	**Project Role**
VGG16	16 stacked, 3x3 convolutional layers	224x224	Most testing / results
InceptionV3	Stacked modules of pooling + conv. layers	299x299	Supportive analysis
ResNet50	50-layer CNN with 'residual layer' connections.	224x224	Supportive analysis

Table 3.1: Pre-trained image classifiers used in the project.

To check each model was working, a script was written to automate prediction collection and interact with preprocessed examples from the collected data. All software in the project was developed to be agnostic about the underlying model, though some methods required specific hyperparameters like a layer target. These targets were set up in a high level dictionary in a constants file.

3.4 Software Abstraction I

Each model requires input resizing and preprocessing in some form. An *ImageHandler* class was therefore designed to abstract from complexity related to different representations of a single image instance. All input/output file methods and getter methods for raw, expanded and preprocessed representations were stored in this class, which made enforcing model agnosticity and adapting methods later much easier.

3.5 Initial Method Investigation

Public GitHub implementations for each of the four methods were sourced: GradCAM [50], DeepLIFT [51], SHAP [35] and LIME [52].

They were then run on high-confidence predictions from the VGG16 model and their 'out of the box' saliency maps were visually inspected. Examples are shown in Figure 3.2 over the page. Note different method formulations result in different explanatory features being highlighted, even for the same example and model prediction. Some other initial observations on each method were made:

- **GradCAM:** The heatmap represents the localisation characterised in the target layer (final convolutional layer). The Guided GradCAM variant (Section 2.3.2) was later used instead for a more discriminative version of the same saliency map.
- **DeepLIFT:** Attributions are relatively noisy, emphasise edges, and are higher resolution due to being being captured in the original input shape.
- **SHAP:** Attributions are in the shape of the input to a target hidden layer, which accounts for its lower resolution.
- **LIME:** The method represents connected-component superpixels as the explanatory features, though in Figure 3.2 these appear patchy on the maze-like patterns on the coral, the texture of the hair on the monkey's face, and the wheat in the field.

3.6 Software Abstraction II

Each method was wrapped in its own class with an *attribute()* function to return the array that represents its explanation output. Repeated logic in each method's implementation was also extracted into an *Attributer* class fitted over these methods. This was a helpful prerequisite to adapting the methods for comparison. The functions in this class visualise figures, save returned NumPy arrays to results files, apply a colour map to highlight positive / negative attribution weights and the intensity of those weights, and normalise attributions between -1 and 1. This class is included in Appendix A and referred to in the class diagram in Figure 3.8.

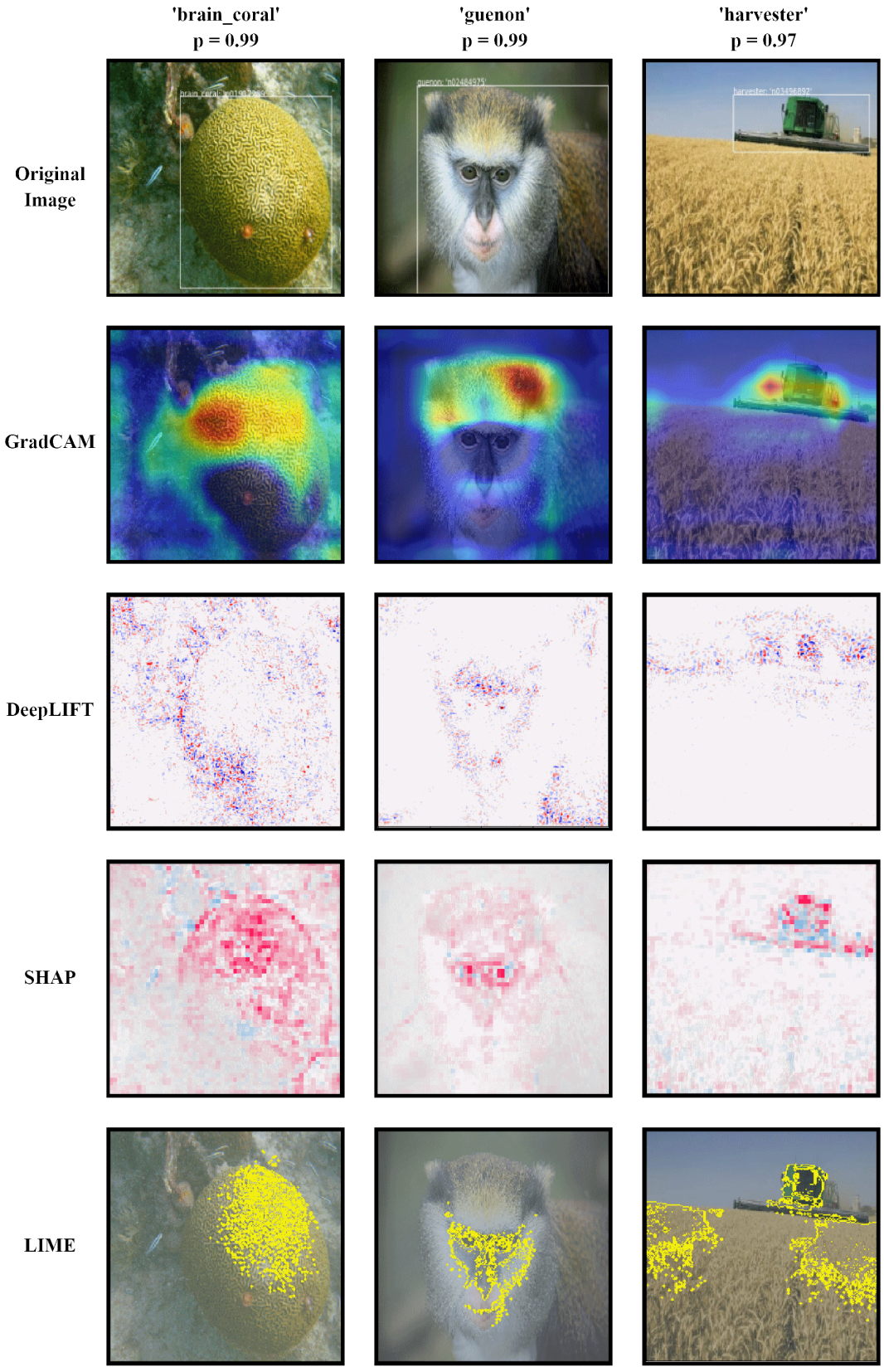

Figure 3.2: Default attributions for methods applied to high-confidence VGG16 predictions (class label and softmax output at top). Original image is resized with annotation drawn for reference.

3.7 Adapting Methods for Compatibility

Some methods (SHAP and DeepLIFT) had higher level implementations than others in the code. For example, the output representation of SHAP was generated in internal methods that made it necessary to implement modified functions that overrode those internals. Others did not hide internal logic and were easier to adapt.

Examples of the methods made compatible for evaluation are shown in Figures 3.4 and 3.5. Some brief comments about work necessary for each method and the insights gained:

- **GradCAM:** Guided GradCAM required very little adaption, since the attributions were already in the shape of the input feature space. Support for specifying the final conv. layer (target for each model architecture) was added.
- **DeepLIFT:** Similar to GradCAM, the only changes made for DeepLIFT were to change the colour map function and extract logic into the *Attributer* class.
- **SHAP:** These attributions had to be resized into the shape of the input feature space. Initially the smaller dimensions were dealt with by scaling them up and using an interpolation method from the OpenCV library. However, this led to biased results from the 'blur' effect of the interpolation. To replace that workaround, the attribution was multiplied with a Guided Backpropagation output following the Guided GradCAM approach.

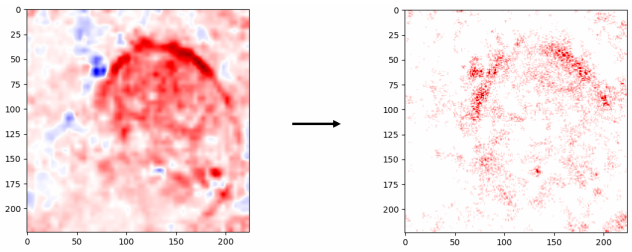

Figure 3.3: Blurred workaround for SHAP versus the improved approach.

- **LIME:** To obtain per-pixel attributions from the superpixel regions, the linear model weights on each superpixel (output by an internal LIME method) were applied to a mask over the region bounded by each superpixel. This is a change in the interpretation of the authors' method, though it was necessary to obtain attribution weights.

Importantly, support for thresholding the attributions in terms of standard deviations was added at this point. An optional parameter was also implemented to have attributions returned with the absolute value taken, to take positive and negative evidence together as simply evidence.

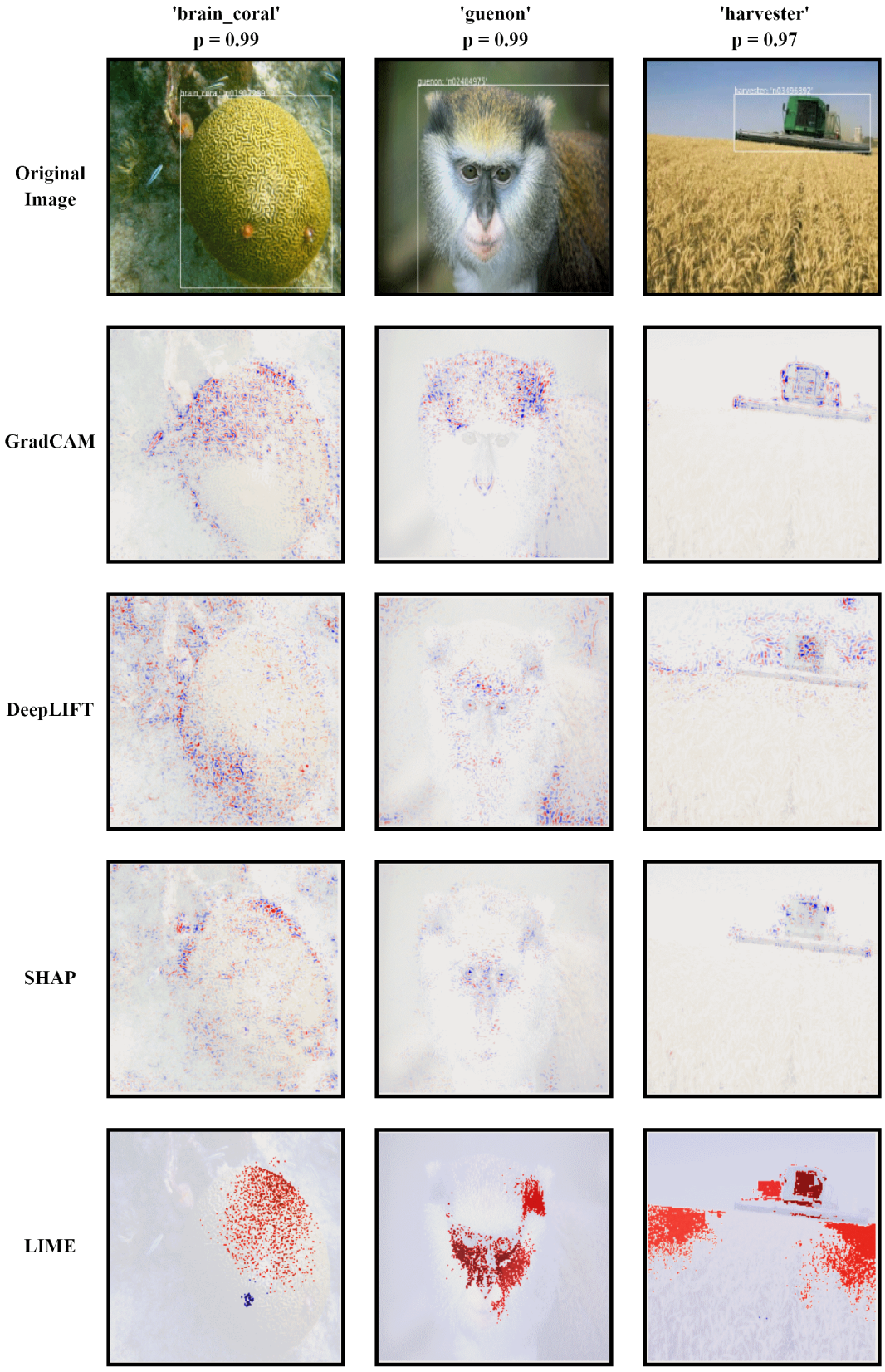

Figure 3.4: Adapted attributions for methods applied to high-confidence VGG16 predictions (original image faintly imposed). Each is a 2D, input-space-sized NumPy array that is normalised between -1 and 1 for the pixel attribution weights / entries in the array. Positive weights are in red and negative weights are in blue. Darker colours indicate larger weights or 'stronger' evidence.

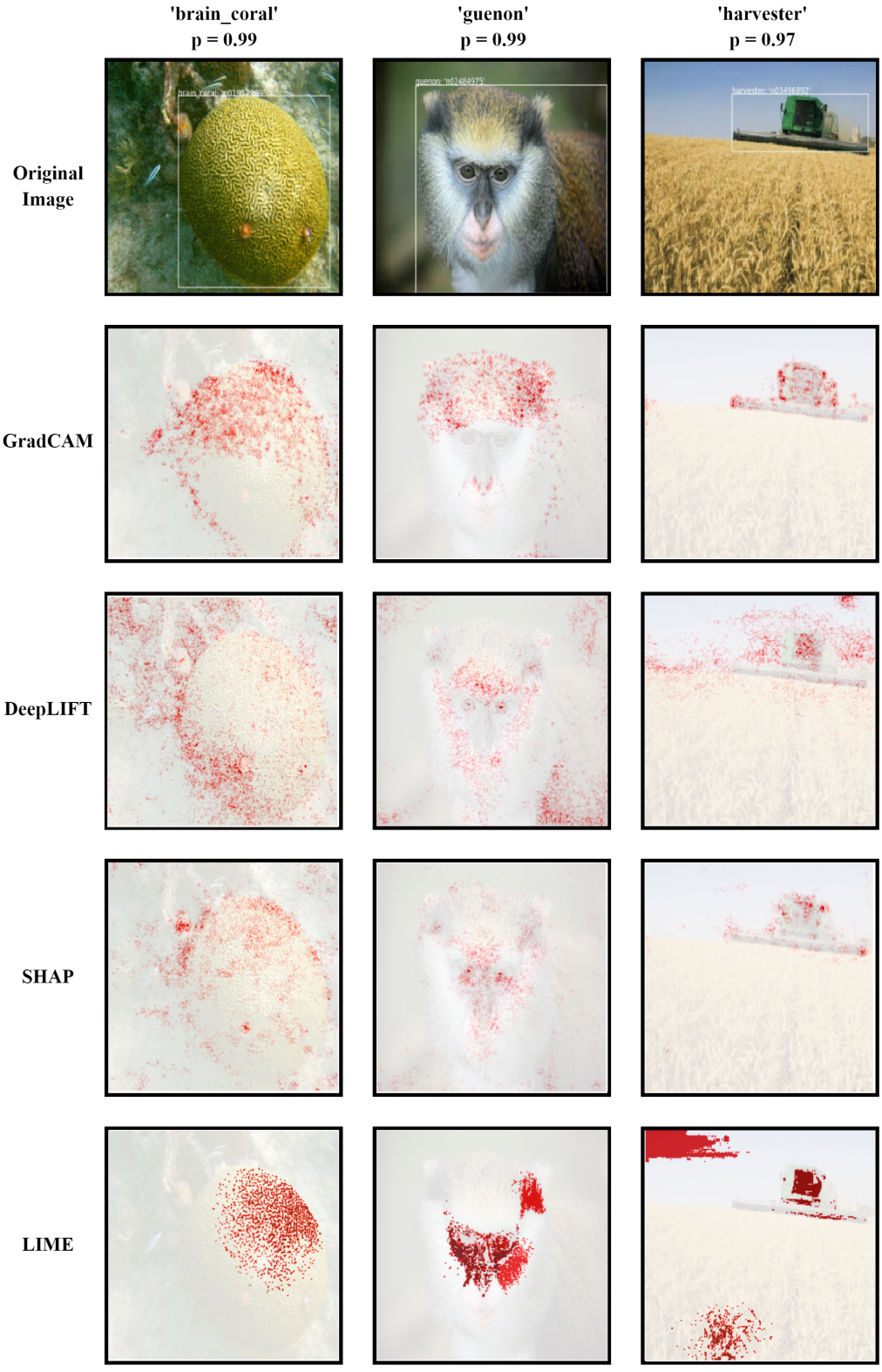

Figure 3.5: Adapted attributions with absolute value taken and a threshold of one standard deviation applied.

3.8 Evaluation Metric Design

In Section 2.5.2 it was argued that existing saliency metrics in the literature do not account for pixel-level saliency via feature weights, and the approach to generate bounding boxes via single-component segmentation can unfairly punish perforated-looking saliency maps. Figures 3.4 and 3.5 also show that the adapted method outputs are discriminatory and fine-grained, rather than localising and coarse-grained, which adds a further motivation for a more suitable metric than the connected-component approach for WSOL. The two metrics designed were both based on intersection-over-union though calculated over individual pixels rather than boxes:

1. **Pixel-wise IOU:** A mask of the instance's ground truth bounding box is compared with the 2D array of each method's saliency map (example in Figure 3.6). Weights are ignored and pixels are counted as intersecting if they have a value in both arrays. The intersection array's area is divided by the union array's area to generate the result.

2. **Pixel-wise IOU*:** A variant to account for intensity of attribution / weights on each pixel. Instead of 1s in the intersection array the attribution weight is directly stored with absolute value taken. The union array (denominator) is unchanged. This was a purely intuitive measure to crudely reward intensity of attribution and punish methods that provide uncertain or 'non-commital' explanations with low attribution weights.

This approach to explanation quality is different from the localisation performance approach of others and results in lower IOU (though this is not as concerning since there is consistency across the panel). Related work introducing *individual* methods did care about IOU magnitude to be able to use ILSVRC object localisation competitors as benchmarks for evaluating their own method.

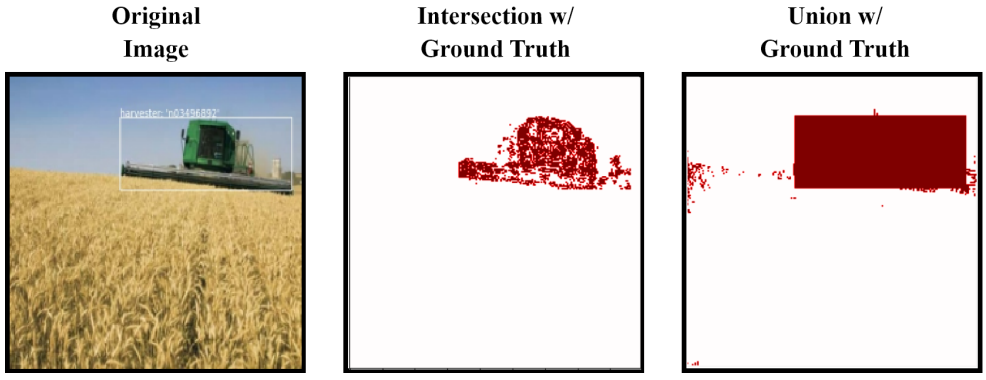

Figure 3.6: Example IOU calculation for a GradCAM attribution (=0.33). Threshold of one standard deviation is applied to the attribution before calculation.

The design of a proxy metric was meant to provide only some objective basis for evaluating methods in the image classification context. Other analysis was also performed for testing methods on higher level criteria, which is included in the Results chapter.

3.9 Software Abstraction III

To collect IOU and IOU* evaluations over many instances an *Evaluator* class was developed that implements the following functions:

1. Collect annotation mask for an instance and a method's output attribution
2. Calculate IOU and IOU* for a given annotation mask and attribution
3. Append results to a dataframe and periodically write to a CSV file
4. Perform the above for batches of instances and for the full method panel

The class can implement other saliency metrics, as IOU and IOU* are modularised functions.

img_no	deeplift	gradcam	lime	shap
1	0.00987	0.02161	0.0661	0.01876
2	0.00394	0.01049	0.01741	0.00697
3	0.00582	0.00571	0.03523	0.00479
4	0.0118	0.01316	0.10822	0.00989
5	0.01853	0.00909	0.05108	0.0041

Figure 3.7: CSV excerpt of a results batch file for IOU metric data.

Finally an *Analyser* class was created to read the collected CSV data, calculate statistics (means and standard deviations), and generate graphs for any subset of methods or metrics. Example graphs are shown in the Results chapter.

The descriptions in Sections 3.4, 3.6 and this section of the developed testing framework are summarised in Figure 3.8. The *Attributer* and *Evaluator* classes are included in Appendices A and B respectively.

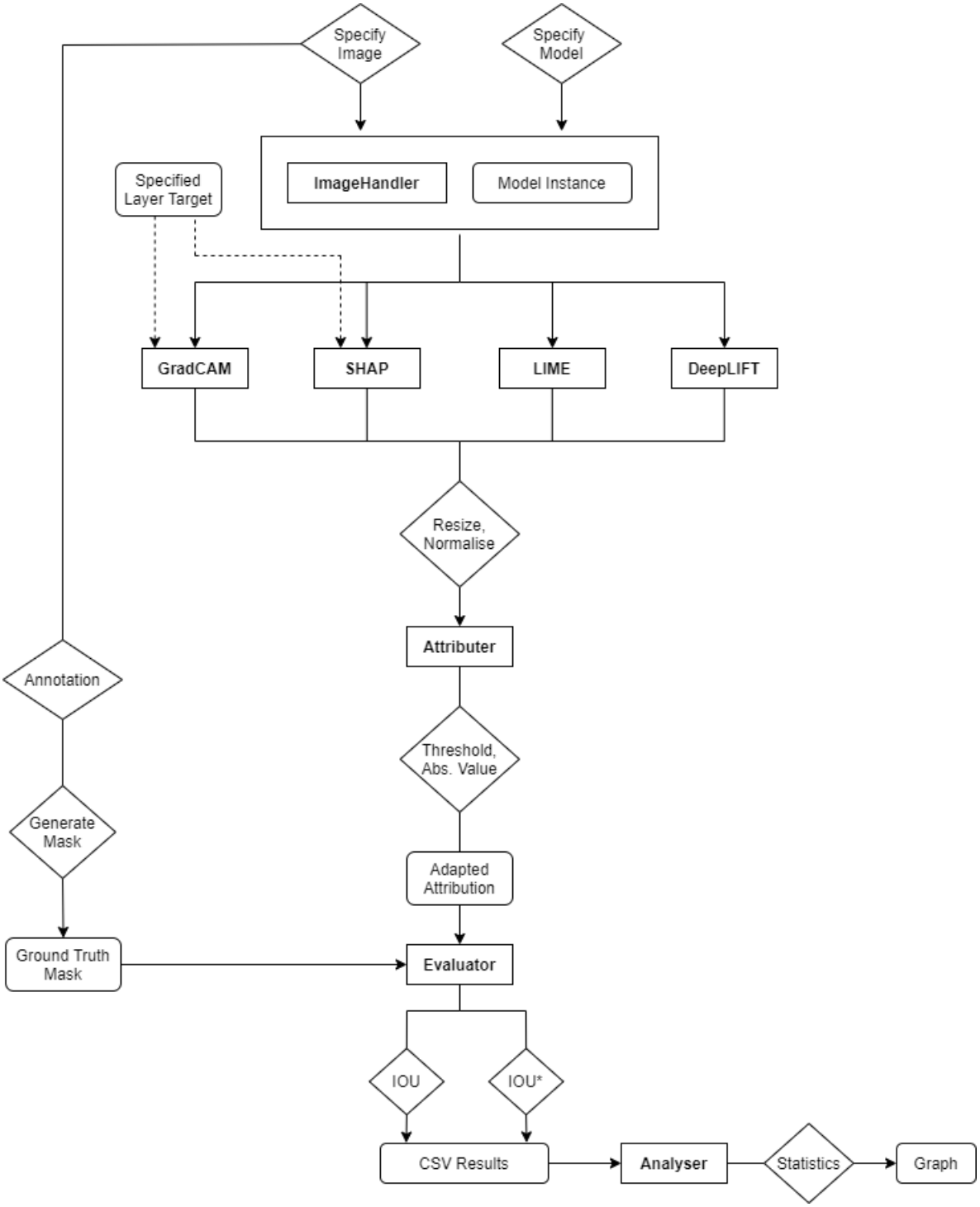

Figure 3.8: High-level flow diagram of testing framework.

Chapter 4

Results & Insights

This chapter includes saliency metric and qualitative results from testing the method panel on ImageNet data and pre-trained CNN models. Insights are compared with existing insights in the literature and summarised at the end of the chapter.

4.1 Experiment Notes

4.1.1 Sample Sizes

A set of 300 attributions for the full method panel took around 10 hours to collect and evaluate using a Nvidia GTX-1080 GPU. LIME was thought to be a key bottleneck due to its perturbation-based approach, but analysis in Section 4.3.1 showed GradCAM was also slow in attribution. After several hundred attributions the process became very slow and would have to be restarted from that point (possibly due to a GPU memory leak in method internals). These performance considerations meant only 1000 instances out of 50,000 available in the validation set were ever used for evaluation. This was at least the same 1000 instances for all methods, and different dimensions were added to generate insights regardless: different attribution thresholds, and subsetting the results for high confidence ($p > 0.9$) vs low confidence model predictions.

4.1.2 Software Availability

Code for the testing framework is available on GitHub and is intended to be made public after improving model agnosticity and cleaning up dependencies and file structures. The framework has modular support for attribution methods (provided they are compatible with image data and return an input-space representation of their explanation) and modular support for other saliency metrics.

4.1.3 Testing on Multiple Architectures

Some limitations were encountered on SHAP for ResNet50 and InceptionV3 architectures. Other methods were successfully tested on other models with results below, though the bug in SHAP was not ultimately resolved. The impact of this obstacle is discussed in Section 4.3.3.

4.2 Saliency Metric Results

4.2.1 VGG16

Figure 4.1 shows IOU and IOU* statistics for a 1000-instance sample explaining VGG16 model predictions. The 1000 instances are broken up into high confidence (top row) and low confidence predictions (bottom row). Error bars indicate one standard deviation from the mean result displayed by bar height, and are later used as a proxy of method consistency (Section 4.3.2). For this experiment setup, a threshold of 0.5 standard deviations was applied to the attribution output for IOU and 1 standard deviation for IOU*.

Some initial observations from Figure 4.1 (others discussed in 4.2.4):

- Results are similar for high vs low confidence predictions, with LIME's variance being one noticeable difference.
- SHAP and DeepLift perform similarly on IOU but not IOU*.
- GradCAM scores lower on these metrics though is visually less noisy than DeepLIFT and SHAP.

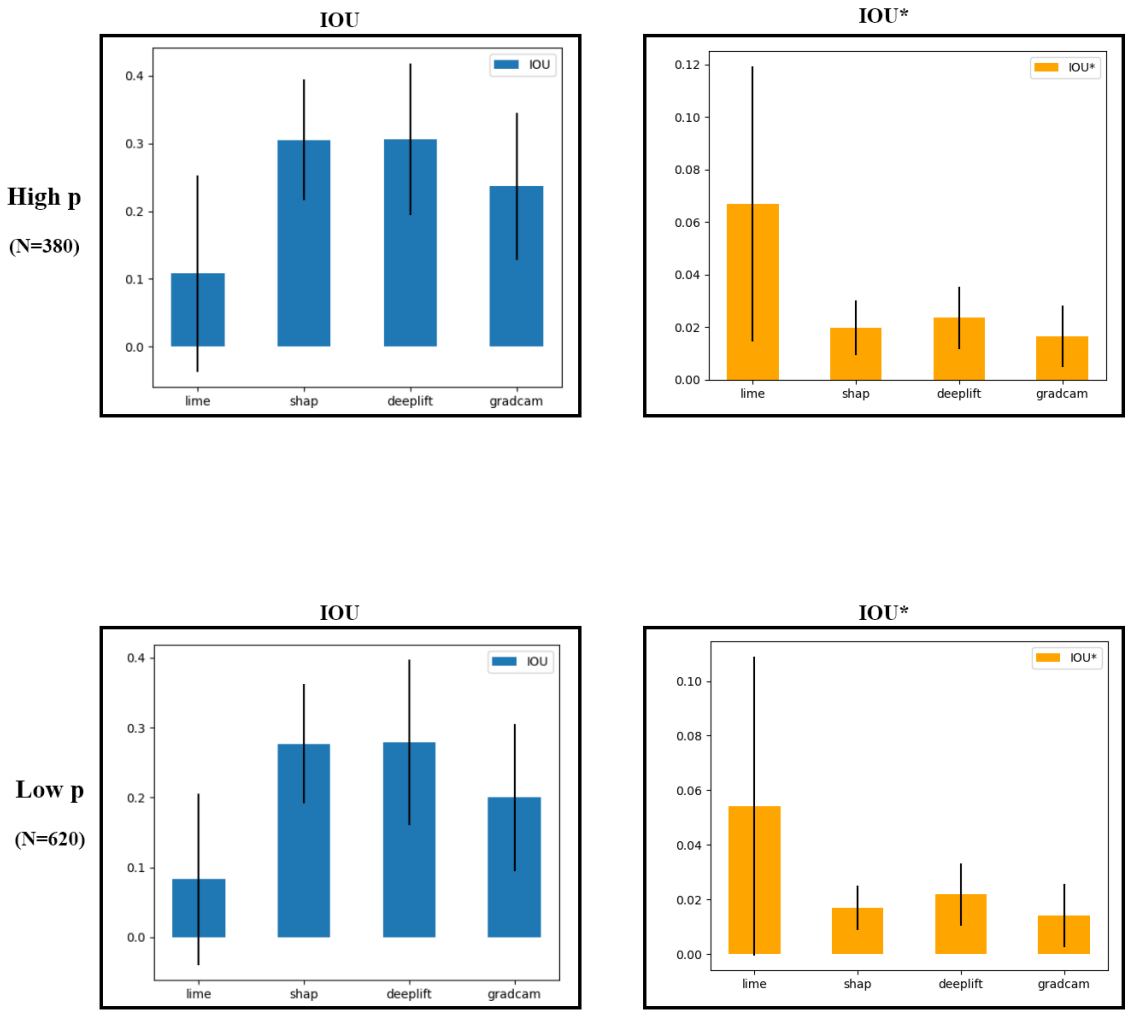

Figure 4.1: Results for a 1000-instance attribution sample of VGG16.

In Appendix C, Figure C.1 shows comparatively identical results for a smaller sample (N=300) with a different threshold set: 1 for IOU and 2 for IOU*.

4.2.2 InceptionV3

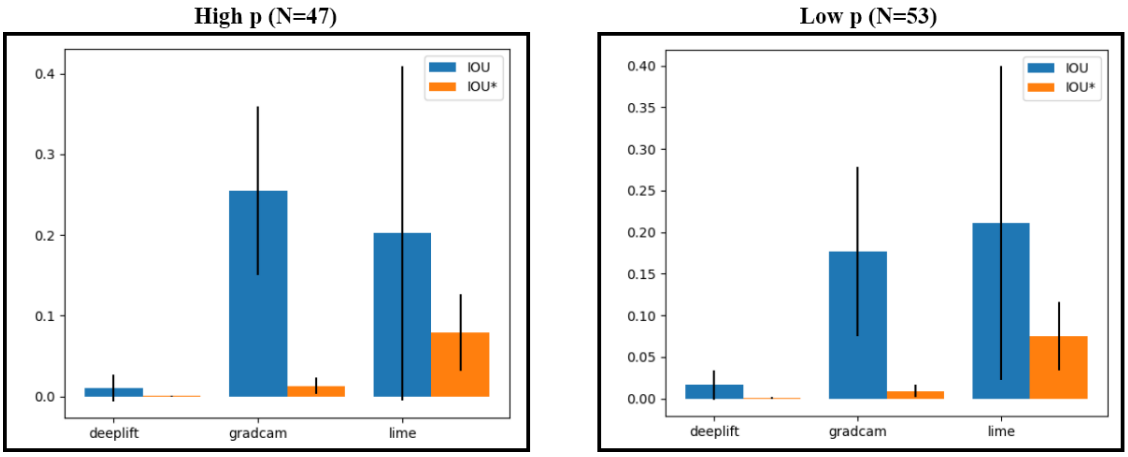

Figure 4.2: Results for a 100-instance attribution sample of InceptionV3.

Model results on this page are only considered auxiliary, since SHAP was excluded from the panel (4.1.3). The lower sample size was a compromise on allocated experiment time due to the longer attribution time for these models.

The most interesting observation here is that GradCAM under-performs on low confidence predictions compared to LIME. Given the high variance and low sample size these results are tentative however. DeepLIFT's non-performance is also analysed in the next section.

4.2.3 ResNet50

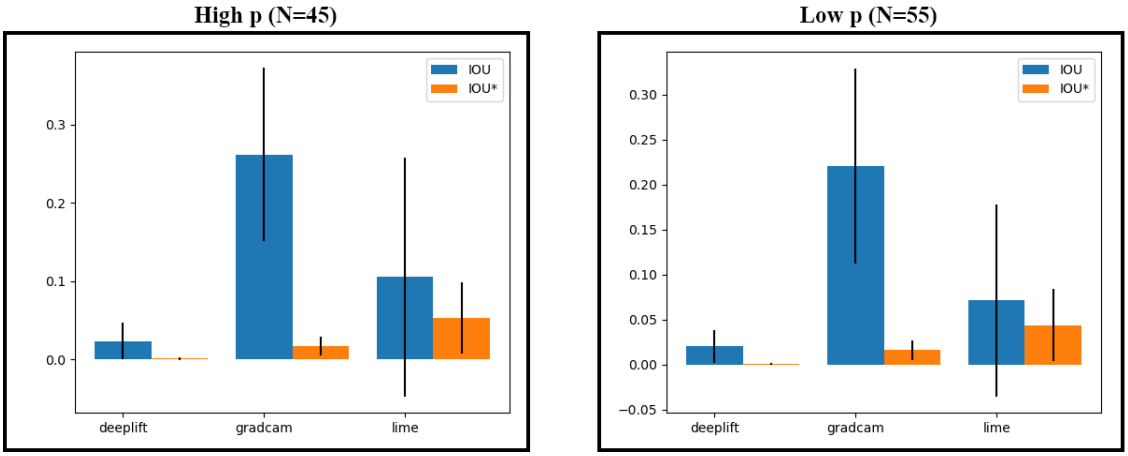

Figure 4.3: Results for a 100-instance attribution sample of ResNet50.

These results are more like the VGG16 results in Figure 4.1. Here, no noticeable difference between high and low confidence subsamples is observed, and GradCAM out-performs LIME on IOU but not IOU*, as in the VGG16 results.

4.2.4 Insights from Saliency Metrics

What did pixel-wise attribution (IOU) highlight?

SHAP and DeepLIFT were the best performers on VGG16 with variance in LIME noticeably high and GradCAM performing not quite as well in terms of mean IOU. However, both SHAP and DeepLIFT failed on other models, which is a problem explored further in Section 4.3.3.

The DeepLIFT saliency results contradict empirical results on its visual saliency. Its attributions were visually much noisier than others and this may have inflated its IOU comparatively if enough of the noise was in the ground truth region.

On balance, when considering across-model result generalisation (where GradCAM excels) and visual saliency (where SHAP and GradCAM excel), both SHAP and GradCAM should be considered superior for saliency in terms of pixel-level detail or 'discriminatory power'.

The impact of thresholding also appeared to make no difference to the ordinal differences between methods (Appendix C). It seems that a higher threshold led to only smaller IOU magnitude rather than a change in one method's performance relative to another.

What did strength of attribution (IOU*) highlight?

The IOU* metric results unfortunately did not derive much further insight. However, a couple of observations were still made:

- LIME outperformed all other methods on IOU*, but since superpixel regions are broadly brushed with the same attribution score, this can be attributed to some bias introduced from the methodology.
- DeepLIFT out-performed SHAP and GradCAM on the VGG16 results though this is possibly due to higher noise across the input space.

How strongly individual pixels are weighted may need to be rewarded more generously: attributions are only bounded between 0 and 1 (with absolute value taken), and so small regions of intensity are probably averaged out by the denominator union array. A suggestion for future work on weighted saliency metrics is to use larger weighting and/or an alternative to IOU.

4.3 Other Criteria

4.3.1 Performance

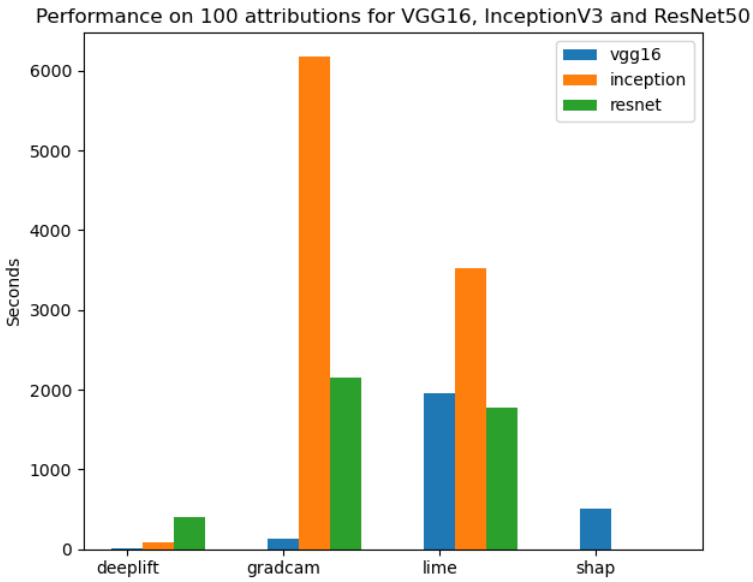

Figure 4.4: Performance/speed in seconds for VGG16, InceptionV3, and ResNet50.

Some insights from performance measurements in Figure 4.4 are described here.

These results are correlated with the time the underlying model takes to calculate a prediction, which is itself a function of the number of model layers and parameters. This is controlled across the panel at least, so the relative performance of each method can still be highlighted (and to some extent performance correlation with model complexity). For example, DeepLIFT's claims of fast performance due to requiring only a single backwards pass are supported by these results. It was also extremely fast for InceptionV3 and much slower on ResNet50, which cannot be explained easily, though two possible reasons are the denser convolutional layers in ResNet50 or a method internal bug.

A surprising insight is that GradCAM performed much worse than LIME for a 100-instance sample. A standard assumption in the literature is that perturbation techniques are necessarily much slower than other approaches. Further analysis however with Guided Backpropagation removed from the Guided GradCAM method used here saw its performance on InceptionV3 reduce from over 6000 seconds (Figure 4.4) to roughly 200 for the same sample size. This shows that the discriminative version of GradCAM comes at a significant performance cost for architectures with many layers, and similar or worse results would have been seen for SHAP for this project's methodology to apply Guided Backpropagation to its outputs as well.

4.3.2 Consistency

Analysis is provided below on consistency across attributions. Table 4.1 lists standard deviations for the method panel from the 1000-instance experiment run on VGG16 (error bars in Figure 4.1):

	IOU		**IOU***	
	High p	**Low p**	**High p**	**Low p**
LIME	0.145	0.122	0.052	0.055
SHAP	0.089	0.085	0.010	0.008
DeepLIFT	0.112	0.118	0.012	0.011
GradCAM	0.109	0.105	0.012	0.012

Table 4.1: Standard deviations on 1000-instance saliency metrics.

LIME suffers the highest variance over the sample, which corroborates findings by Alvarez-Melis & Jaakkola (2018) (Section 2.5.2) that LIME and model-agnostic, perturbation-based techniques generally are more prone to instability than gradient-based techniques.

For other methods, results were consistent across high and low confidence predictions, which was an unexpected finding. This may have been because of a certain baseline level of noise that each method outputs, meaning underlying model confusion did not translate into lower IOU or IOU* on average.

For consistency across models, it was noted in Section 4.2.4 that DeepLIFT performed strongly on VGG16 and very poorly on more complex architectures. This was therefore a strength of LIME and GradCAM: they performed slightly worse on IOU in the VGG16 experiment but better on more complicated architectures. In terms of consistency though, LIME's attributions increased in variance on the saliency metrics for the more complicated architectures (Figures 4.2 and 4.3). This is another blight on its reputation for instability.

4.3.3 Ease of Adaption

Observations on each method's implementation considerations are made in this section. Reference is made to their hyperparameter complexity and model compatibility, which together can be summarised as ease of adaption. In the following chapter, a higher level comparison of each method's approach (and related implementation considerations) is also provided.

GradCAM

A note on GradCAM made was that its performance significantly deteriorates when replacing its localisation 'heatmap' formulation with the discriminative upgrade proposed by the authors (i.e. multiplied with Guided Backpropagation). This is a compromise that practitioners can be aware of: for individual explanations, it may be desirable to use Guided GradCAM to see a higher resolution explanation, since at-scale evaluation is not a concern. For explaining many model predictions, or where localisation is more important to the use case than individual patterns or edges, the variant without Guided Backpropagation should be preferred.
Other insights about GradCAM's use cases:

- GradCAM's explanations were consistently visually salient, and also consistent in the saliency metric results. This could be because it is inherently designed for CNN architectures: the target of the final convolutional layer where the features are intuitively represented in the most abstract form seems to lead to more visually appealing explanations for model behaviour.

- GradCAM is suited as an 'off the shelf' option for CNN architectures. It requires no modification to internal gradient or backpropagation operations: a strength over backpropagation methods like DeepLIFT.

DeepLIFT

DeepLIFT performed strongly on VGG16 but appeared to fail completely on more complex architectures. The reason may be due to the implementation: DeepLIFT requires an architectural modification that has to be done for each layer type, which is to implement the custom backpropagation function that incorporates reference activations (i.e. transform gradients into attributions that represent differences from those reference activations). For this project, the implementation relied upon may not have been able to handle ResNet's residual layers or Inception's layer modules. Full neural network compatibility may be a theoretically true claim by the authors but these caveats (and determining reference activations) are a major practical drawback.

SHAP

The difficulty also observed in applying SHAP to more complicated model architectures suggests that other practitioners may have similar difficulties with their own models. SHAP is a higher level *specification* of an explanation method rather than a particular formulation (Section 2.4.2) which makes it difficult to evaluate independently of its model-specific approximations.

One of those model-specific approximations is based on Integrated Gradients and was the one relied upon in this project. This SHAP implementation saw almost identical results to DeepLIFT, in terms of saliency in Figure 4.1, and consistency in Table 4.1). This supports the suggestion that DeepLIFT is a more practical approximation of Integrated Gradients with a similar level of accuracy (Ancona et al. (2017), Section 2.5.1).

SHAP's key implementation requirements are a background sample of predictions needed to estimate the expectation of model output, and the choice of a layer target to explain. Neither of these were difficult to specify, though the choice of layer is somewhat arbitrary compared to GradCAM's specification of the final convolutional layer.

Though it couldn't be practically evaluated on other architectures, its visual saliency on VGG16 was appealing, and it is notable that SHAP highlighted different features to other methods for the same prediction (e.g. Figure 3.4). SHAP's software package also provides approximations for a number of model families [35], which was part of the motivation for its initial description as a model-agnostic method.

LIME

Although it is fully model-agnostic and therefore had fewer implementation difficulties, LIME has two important hyperparameters that have not been mentioned so far in discussion. This is the number of training samples of locally perturbed input features taken, as well as the number of features (superpixels) for each sample. Both were taken near the authors' defaults, as a suggested compromise between performance and accuracy [52].

The superpixel-based output of LIME is better suited for *localisation* saliency measures (not targeted by this project). In a sense LIME was slightly misrepresented by how it was adapted for pixel-wise attribution, with superpixel weights brushed over all pixels in each bounded region. However, this adaption step was necessary for evaluation with other methods, and superpixel weights still represent input space attributions.

4.4 Summary of Insights

4.4.1 Quantitative

	Saliency (IOU, IOU*) (Section 4.2.4)	**Consistency (Section 4.3.2)**	**Performance (Section 4.3.1)**
LIME	Low IOU; High IOU* explained by superpixel weighting	Poor over the sample; Unstable	Very slow for all three models
SHAP	High IOU; failed on other models; visually salient	Consistent for model confidence	Moderate speed; incomplete results
DeepLIFT	High IOU and IOU*; failed on other models; noisy explanations	Consistent for model confidence	Fast; results also incomplete
GradCAM	Average IOU and IOU*; visually salient	Consistent for model confidence and across models	Guided-GradCAM = very slow; Grad-CAM = fast

Table 4.2: Summary of quantitative evaluation insights.

4.4.2 Qualitative

	Relative Strengths	**Relative Weaknesses**
LIME	Full model compatibility; ease of adaption	Explanation instability; accuracy-linked hyperparameters
SHAP	Model and task compatibility; theoretic basis	Uncertain hyperparameter choice; competing approximations (+ bugs)
DeepLIFT	Neural network compatibility; task compatibility	Implementation complexity (increasing in model complexity)
GradCAM	Low hyperparameter complexity; flexibility in heatmap localisation vs pattern discrimination	Model inflexibility: designed for CNNs; task inflexibility: suited for image data

Table 4.3: Summary of qualitative insights.

Chapter 5

Discussion

The main insights from method evaluation were summarised at the end of the previous chapter. To conclude, some discussion is provided on project methodology, method approaches, obstacles encountered, the developed software framework, and other recommendations for future work.

5.1 Evaluation via Saliency Metrics

There are a number of considerations around proxy metric design for explanation quality, and strengths and limitations of the saliency metric approach itself are worth noting. Some have been mentioned in prior art but were empirically confirmed in this project:

1. Edges do form an important component of visual saliency [12], though localising an explanation coarsely is sometimes more useful than knowing how discriminative the model is in terms of edges and weights.

2. Human perception of explanation quality cannot feasibly be distilled into a single metric, though methods can still be *compared* on proxies of visual quality.

3. A method's errors are not easy to distinguish from underlying model confusion (i.e. model errors), at least in individual method evaluation contexts.

For each of these points, some associated novel insights were derived:

1. Method evaluation should account for both localisation and discriminative abilities of a method - a single localisation metric and a single pixel-wise metric are not mutually exclusive. A more sophisticated saliency metric may be able to optionally account for connected components, edges and pixel-wise attribution intensity simultaneously. At-scale metric evaluation also provides

different insights to simple visual inspection, and the former is not a replacement for the latter.

2. A *set* of higher level criteria, including Adebayo's randomisation-based sanity checks, the invariance checks in Table 2.1, as well as lower level, functionally-grounded saliency metrics, is the fairest way to compensate for the subjectivity of a single measure of explanation quality. However, in this project, a practical set of criteria of method performance and model compatibility were used for higher level evaluation, though finding exact conclusions on the basis of all criteria simultaneously was difficult.

3. In a context where the underlying model and model predictions are consistent across a panel of methods, as in this project, this can somewhat be controlled for.

5.2 Comparing Approaches

Backpropagation-based methods like DeepLIFT require custom backward functions that override activation functions and gradient operators. This gives them implementation hurdles for models with many layer types (i.e. hurdles increasing in model complexity): this was previously mentioned as an explanation for DeepLIFT's poor performance on InceptionV3 and ResNet50 in Section 4.3.3.

For GradCAM on the other hand, targeting only the final convolutional layer gives it a strength over others: it does not depend on complexity in earlier layers. Its limitation is that it can only be implemented for convolutional neural networks, therefore is dominant only in image-based tasks or other applications of that architecture. Less architecture-dependent methods in the gradient class include Gradient * Input and SmoothGrad.

Perturbation techniques, including LIME but also those proposed by Fong & Vedaldi [30] ('mask generation') and Zeiler & Fergus (2014) [13] ('Occlusion') have a clear interpretation in what they iteratively reveal. However, the requirement to test a large number of small, progressively increasing occlusion patches was shown to suffer bad performance even on simpler architectures like VGG16. For modern, more complex architectures, perhaps those trained on higher resolution images or taking longer to compute a single prediction, the performance may make the method unviable, unless a small number of explanations are required. The strength of LIME and the perturbation approach generally is its 'off the shelf' ease of adaption, which is true for practitioners and researchers working on *any* classification task beyond image data.

5.3 Obstacles & Methodology Enhancements

There was some disappointment around the bug in the SHAP implementation that meant it could not be applied to more complicated model architectures than VGG16. Other practitioners and researchers have noted similar difficulties in its GitHub issue list [35]. However, some results were still able to be generalised on InceptionV3 and ResNet50.

There was also some regret around saliency metrics being limited to the formulation based on pixel-wise attribution. The implicit procedure in the project's methodology was to transform attribution method outputs into a discriminative common form. This may have misrepresented the methods where a divergence from the authors' intent occurred (i.e. superpixels for LIME, heatmaps for GradCAM). It may have been fairer analysis to critique them on their own criteria (i.e. localisation of bounding boxes for GradCAM) *as well as* the saliency metrics and qualitative criteria used in this project. This would also have been a positive contribution towards reproducibility.

Finally, ImageNet is a widely used dataset for research in image classification, though the practical insights from the project may have been better highlighted on a domain-specific dataset. Real-world image data with bounding box annotations like ImageNet is difficult to come by however. Model training would also be necessary with such a dataset. For datasets without ground truth bounding box annotations, one possible evaluation metric could be based on noisy data: generated 'junk' examples could be fed into the model, and then checked if a drop in method attribution weight across the input space was correlated with the drop in model prediction confidence. This 'confusion invariance' idea is similar to (and can be combined with) other invariance criteria explored in Table 2.1.

5.4 Software Framework

Existing software packages combine only gradient-based methods or closely related variants. These toolsets can be redundant when related method outputs are so consistently similar (e.g. Figure 2.9 in Related Work). A key contribution of this project's toolset was to combine different explanation approaches for image data to be able to offer different explanations for any one instance. For example, Figures **??** and 3.4 in the Methodology highlight different, informative input space features among the method panel even for a single model's prediction.

Modular support for attribution methods and evaluation metrics was achieved through the object-oriented approach taken for the testing framework (Figure 3.8 in Methodology). However, an extra function to present attribution representations

differently (i.e. localisation 'heatmap' vs discriminatory pixel-map) or denormalise attributions could create the fairer evaluation testbed that was discussed in Section 5.1.

5.5 Other Recommendations for Future Work

5.6 Future work

Some directions for future work with regards to the evaluation methodology were identified already in this chapter. These aspects and other suggestions are summarised in Table X to conclude.

lthe breadth of supported methods, adding higher level support for adding new saliency metrics,

implementing methods directly, rather than relying on GitHub implementations by researchers and third party attempts to implement formulations described in a paper. The motivation for more reproducibility in machine learning is another reason to hand-implement these methods, to independently corroborate method author claims and find low level bugs (like SHAP's) more easily.

models at various stages of training

New datasets, biased dataset

Chapter 6

Conclusion

There are broad motivations for explainability in machine learning, and overlapping views in the literature for not only what a 'quality' explanation should look like but also what an explanation should target (i.e. activations, gradients or occluded inputs). As well as the various motivations, even for a single model's prediction, the range of possibly salient explanations for that prediction makes finding practical criteria for interpretability that much harder.

This thesis did not aim to address the issue of what defines interpretability. As well as this, the intention was not to provide a verdict on a single superior method in the panel. Instead, the project's aims were to independently provide an evaluation of *diverse* feature attribution methods in the image classification context though using a *common* set of criteria. This was to measure their strengths on a level playing field, using the developed testing framework, and thereby improve task-suited adoption of interpretability techniques via better understanding of their practical merits and considerations.

These aims were successfully achieved via the evaluation study carried out, and insights on the panel of methods picked for representativeness of approach were derived from quantitative and qualitative data. The developed testing framework includes modular support for underlying models, methods and evaluation metrics, which can help future researchers evaluate and apply methods to build more adoptable models.

Researchers and practitioners who wish to understand pieces of their model's complexity can also hopefully gain a more *thorough* understanding of predictions using methods from different approaches. This contribution of a multi-approach evaluation of feature attribution methods will hopefully achieve higher confidence in explanation techniques generally, and support their adoption in important applications like prediction debugging in medical imaging.

Appendix A

Attributer Class

```
import logging

# keras models
from keras.applications import InceptionV3, VGG16, ResNet50
from keras.applications.imagenet_utils import decode_predictions

# util
from eval.util.constants import *
from eval.util.image_util import ImageHandler, show_figure, apply_threshold,
        get_classification_mappings

# methods
from eval.methods.LIME import Lime
from eval.methods.deep_lift import DeepLift
from eval.methods.SHAP import Shap
from eval.methods.grad_cam import GradCam


def check_invalid_attribution(attribution, ih):
    # attribution should be a 2D array returned by each method
    # i.e grayscale not RGB
    if attribution is None:
        return 1
    # check shape of attribution is what is expected
    if attribution.ndim != 2:
        print('Attribution returned has {} dimensions'.format(attribution.ndim))
        return 1
    if attribution.shape != ih.get_size():
        print('Attribution returned with shape {} is not the expected shape of {}'.format(
            attribution.shape, ih.get_size()))
        return 1
    return 0


class Attributer:
    def __init__(self, model_name: str):
        self.models = {
            VGG: VGG16,
            INCEPT: InceptionV3,
            RESNET: ResNet50
        }
```

```
        self.curr_model_name = model_name
        self.curr_model = self.load_model(model_name)
        # set up methods
        self.lime_method = None
        self.deep_lift_method = None
        self.shap_method = None
        self.gradcam_method = None
        # Classes for imagenet
        self.class_map = get_classification_mappings()

    def load_model(self, model_name: str):
        self.curr_model_name = model_name
        print('Loading {} model architecture and weights...'.format(model_name))
        return self.models[self.curr_model_name](weights='imagenet')

    def build_model(self):
        """Function returning a new keras model instance.
        """
        if self.curr_model_name == VGG:
            return VGG16(include_top=True, weights='imagenet')
        elif self.curr_model_name == INCEPT:
            return InceptionV3(include_top=True, weights='imagenet')
        elif self.curr_model_name == RESNET:
            return ResNet50(include_top=True, weights='imagenet')

    def initialise_for_method(self, method_name: str, layer_no: int = None):
        if method_name == LIFT and self.deep_lift_method is None:
            self.deep_lift_method = DeepLift(self.curr_model, self.build_model)
        elif method_name == LIME and self.lime_method is None:
            self.lime_method = Lime(self.curr_model, self.curr_model_name)
        elif method_name == SHAP and self.shap_method is None:
            print(layer_no)
            self.shap_method = Shap(self.curr_model, self.curr_model_name,
                                    layer_no)
        elif method_name == GRAD and self.gradcam_method is None:
            self.gradcam_method = GradCam(self.curr_model, self.build_model,
                                          layer_no)

    def predict_for_model(self, ih: ImageHandler, top_n: int = 5,
                          print_to_stdout: bool = True) -> (str, float):
        # returns a tuple with the top prediction, and the probability of the
        # top prediction (i.e confidence)
        logging.info('Classifying...')
        predictions = self.curr_model.predict(ih.get_processed_img())
        decoded_predictions = decode_predictions(predictions, top=top_n)

        # print the top 5 predictions, labels and probabilities
        if print_to_stdout:
            print('Model predictions:')
        max_p = 0.00
        max_pred = ''
        for (i, (img_net_ID, label, p)) in enumerate(decoded_predictions[0]):
            if print_to_stdout:
                print('{}: {}, Probability={:.2f}, ImageNet ID={}'.format(
                    i + 1, label, p, img_net_ID))
            if p > max_p:
                max_p = p
                max_pred = label
        if print_to_stdout:
```

```
            print('')
        return max_pred, max_p

    def get_good_examples(self, cap: int = 1001):
        good_examples = []
        for i in range(1, cap):
            ih = ImageHandler(i, self.curr_model_name)
            max_pred, p = self.predict_for_model(ih, top_n=1,
                                                 print_to_stdout=False)
            if p > 0.9:
                good_examples.append(i)
        return good_examples
```

```
def attribute(self, ih: ImageHandler, method: str, layer_no: int = None,
              take_absolute: bool = False, take_threshold: bool = False,
              sigma_multiple: int = 0,
              visualise: bool = False, save: bool = True):
    if layer_no is None:
        layer_no = LAYER_TARGETS[method][self.curr_model_name]
    self.initialise_for_method(method_name=method, layer_no=layer_no)
    # get the 2D numpy array which represents the attribution
    attribution = self.collect_attribution(ih, method=method, layer_no=layer_no)
    # check if applying any thresholds / adjustments based on +ve / -ve evidence
    if take_threshold or take_absolute:
        attribution = apply_threshold(attribution, sigma_multiple,
                                      take_absolute)
    if check_invalid_attribution(attribution, ih):
        return
    if save:
        ih.save_figure(attribution, method)
    if visualise:
        show_figure(attribution)
    return attribution

def attribute_panel(self, ih: ImageHandler, methods: list = METHODS,
                    take_threshold: bool = False, sigma_multiple: int = 0,
                    take_absolute: bool = False,
                    visualise: bool = False, save: bool = True):
    output_attributions = {}
    for method in methods:
        layer_no = LAYER_TARGETS[method][self.curr_model_name]
        output_attributions[method] = self.attribute(ih=ih, method=method,
                                                     layer_no=layer_no,
                                                     take_absolute=take_absolute,
                                                     take_threshold=take_threshold,
                                                     sigma_multiple=sigma_multiple,
                                                     visualise=visualise, save=save)
    return output_attributions

def collect_attribution(self, ih: ImageHandler, method: str,
                        layer_no: int = None, print_debug: bool = True):
    """Top level wrapper for collecting attributions from each method. """
    if print_debug:
        print('Collecting attribution for `{}``'.format(method))
    if method == LIFT:
        return self.deep_lift_method.attribute(ih)
    elif method == LIME:
        return self.lime_method.attribute(ih)
    elif method == SHAP:
        self.shap_method.reset_explainer(layer_no=layer_no)
        return self.shap_method.attribute(ih)
    elif method == GRAD:
        self.gradcam_method.reset_layer_no(layer_no=layer_no)
        return self.gradcam_method.attribute(ih)
    else:
        print('Error: Invalid attribution method chosen')
        return None
```

Appendix B

Evaluator Class

```
import os
import pandas as pd
import numpy as np

# Attributer class
from eval.Attributer import Attributer

# util
from eval.util.constants import *
from eval.util.image_util import ImageHandler, show_figure,
        show_intersect_union_subfigures
from eval.util.imagenet_annotator import get_mask_for_eval


class Evaluator:
    def __init__(self, metric: str, model_name: str, att: Attributer = None):
        if att is None:
            self.att = Attributer(model_name=model_name)
        else:
            self.att = att
        self.metric = metric
        self.model_name = model_name
        self.file_headers = METHODS  # [m + "+_" + self.model_name for m in METHODS]
        self.result_file = "{}/{}/{}_results.csv".format(
            RESULTS_EVAL_PATH, model_name, metric)
        self.results_df = self.read_file(self.result_file, wipe=False)

    def read_file(self, file_path: str, wipe=False):
        # gets a dataframe from the results file
        if wipe:
            f = open(file_path, "w+")
            f.close()
            df = pd.DataFrame(columns=['img_no'] + self.file_headers).set_index(
                'img_no')
            return df
        if not os.path.exists(file_path):
            with open(file_path, "w") as f:
                f.write(','.join(['img_no'] + self.file_headers))
        df = pd.read_csv(file_path).set_index('img_no')
        return df
```

```
def write_results_to_file(self):
    self.results_df.to_csv(self.result_file, index=True, index_label='img_no')

def get_image_handler_and_mask(self, img_no):
    # this gets the image wrapped in the ImageHandler object, and the
    # bounding box annotation mask for the image,
    # ImageHandler is used to calculate attributions by each method, and the
    # mask is used for evaluation
    ih = ImageHandler(img_no=img_no, model_name=self.model_name)
    # bounding box in the format of the model's input shape / attribution shape
    annotation_mask = get_mask_for_eval(img_no=img_no, target_size=ih.get_size(),
                                        save=False, visualise=False)
    return ih, annotation_mask

def collect_panel_result_batch(self, experiment_range: list):
    new_rows = {}
    for img_no in experiment_range:
        new_row = {}
        ih, annotation_mask = self.get_image_handler_and_mask(img_no)
        for method in METHODS:
            result = self.collect_result(ih, annotation_mask, method)
            if img_no <= len(self.results_df.index):
                self.results_df.at[img_no, method] = result
            else:
                new_row[method] = result
        new_rows[img_no] = new_row
        if (img_no % 10) == 0:
            self.append_to_results_df(new_rows)
            new_rows = {}
    self.append_to_results_df(new_rows)

def collect_result_batch(self, method: str, experiment_range: range):
    new_rows = {}
    for img_no in experiment_range:
        ih, annotation_mask = self.get_image_handler_and_mask(img_no)
        result = self.collect_result(ih, annotation_mask, method)
        if img_no <= len(self.results_df.index):
            self.results_df.at[img_no, method] = result
        else:
            new_rows[img_no] = {method: result}
        if img_no % 10 == 0:
            self.append_to_results_df(new_rows)
            new_rows = {}

    self.append_to_results_df(new_rows)

def append_to_results_df(self, new_rows_dict, write=True):
    new_data = pd.DataFrame.from_dict(new_rows_dict,
                                      columns=self.file_headers,
                                      orient='index')
    self.results_df = self.results_df.append(new_data, sort=True)
    if write:
        self.write_results_to_file()

def collect_result(self, ih: ImageHandler, mask, method: str):
    # threshold for each attribution's "explainability" is the number of std
    # deviations above the mean contribution score for a pixel
    if self.metric == INTERSECT:
        return self.evaluate_intersection(ih, mask, method,
```

```
                                                sigma=INTERSECT_THRESHOLD)
        elif self.metric == INTENSITY:
            return self.evaluate_intensity(ih, mask, method,
                                           sigma=INTENSITY_THRESHOLD)

    def evaluate_intersection(self, ih: ImageHandler, mask, method: str,
                              sigma: int, print_debug: bool = False) -> float:
        # calculate an attribution and use a provided bounding box mask to
        # calculate the IOU metric. attribution has threshold applied, and abs
        # value set (positive and negative evidence treated the same)
        attribution = self.att.attribute(ih=ih,
                                         method=method,
                                         layer_no=LAYER_TARGETS[method][self.model_name],
                                         take_threshold=True, sigma_multiple=sigma,
                                         take_absolute=True,
                                         visualise=False, save=False)
        # calculate the intersection of the attribution and the bounding box mask
        intersect_array = np.zeros(attribution.shape)
        intersect_array[(attribution > 0.0) * (mask > 0.0)] = 1
        # get the union array for the IOU calculation
        union_array = np.zeros(attribution.shape)
        union_array[(attribution > 0.0) + (mask > 0.0)] = 1
        # calculate intersection and union areas for numerator and
        # denominator respectively
        intersect_area = intersect_array.sum()
        union_area = union_array.sum()
        intersection_over_union = intersect_area / union_area
        print('Evaluating ‘{}‘ on example ‘{}‘ ({})'.format(
            method, ih.img_no, 'intersection'))
        if print_debug:
            #print('--Mask Area =\t {}'.format(mask_area))
            print('--Intersect Area =\t {}'.format(intersect_area))
            print('--Union Area =\t {}'.format(union_area))
            print('--Intersection / Union =\t {:.2f}%'.format
                  (intersection_over_union * 100))
            print('')

        return intersection_over_union

    def evaluate_intensity(self, ih: ImageHandler, mask, method: str, sigma: int,
                           print_debug: bool = False) -> float:
        # # calculate an attribution and use a provided bounding box mask to
        # calculate the IOU metric attribution has threshold applied, and abs
        # value set (positive and negative evidence treated the same)
        attribution = self.att.attribute(ih=ih,
                                         method=method,
                                         layer_no=LAYER_TARGETS[method][self.model_name],
                                         take_threshold=True, sigma_multiple=sigma,
                                         take_absolute=True,
                                         visualise=False, save=True)
        # calculate the weight/confidence of the attribution intersected with
        # the bounding box mask
        intensity_array = np.copy(attribution)
        intensity_array[(attribution > 0.0) * (mask < 0.1)] = 0
        # get the union array for the IOU* calculation
        union_array = np.zeros(attribution.shape)
        union_array[(attribution > 0.0) + (mask > 0.0)] = 1
        intensity_area = intensity_array.sum()
        union_area = union_array.sum()
```

```
intensity_over_union = intensity_area / union_area
print('Evaluating ‘{}‘ on example ‘{}‘ ({})'.format(
    method, ih.img_no, 'intensity'))
if print_debug:
    print('— Intersect Area = \t {}'.format(intensity_area))
    print('— Union Area = \t {}'.format(union_area))
    print('— Intensity / Union = \t {:.2f}%'.format(intensity_over_union * 100))
    print('')

return intensity_over_union
```

Appendix C

VGG16 Supportive Results

Supportive results for Section 4.2.1. These results are from a smaller experiment run on VGG16 with a different threshold set: 1 std dev. for IOU and 2 for IOU*.

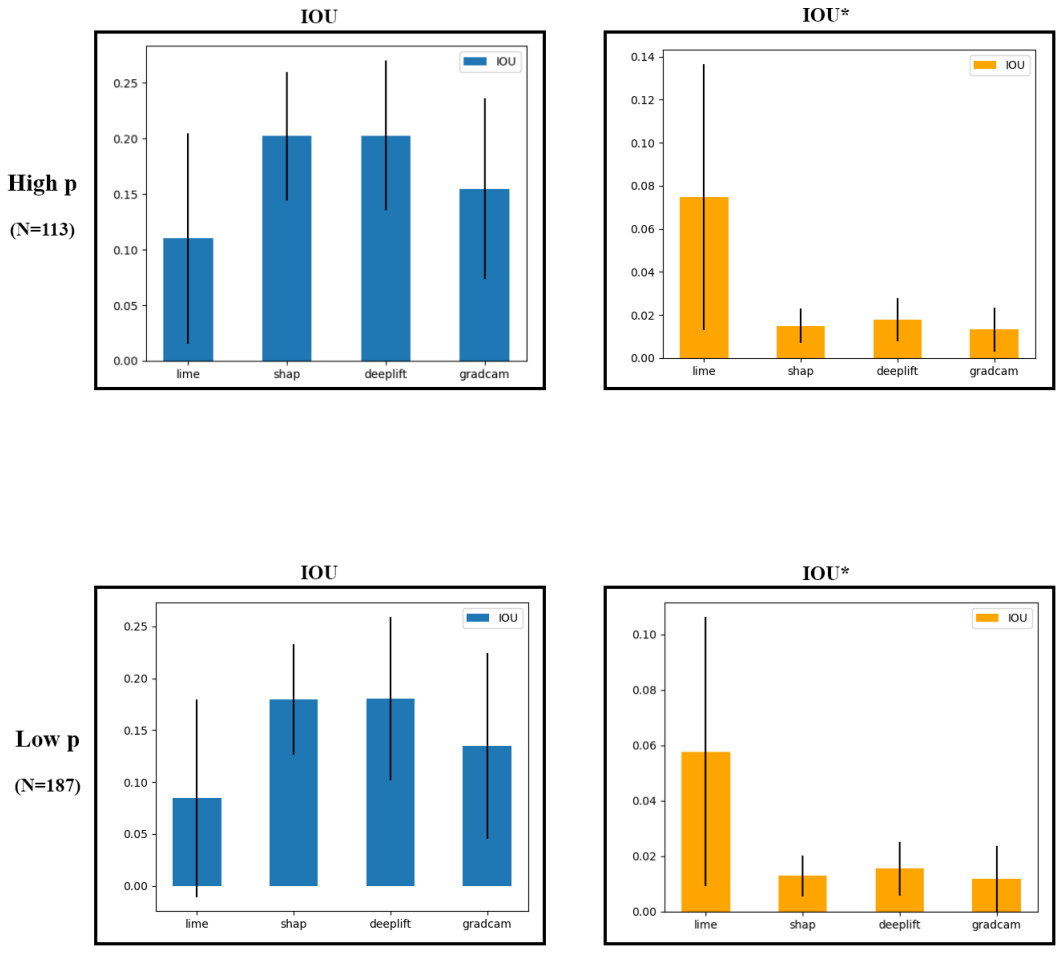

Figure C.1: Results for a 300-instance attribution sample of VGG16.

Bibliography

[1] J. R. Zech, M. A. Badgeley, M. Liu, A. B. Costa, J. J. Titano, and E. K. Oermann, "Confounding variables can degrade generalization performance of radiological deep learning models," *CoRR*, vol. abs/1807.00431, 2018. arXiv: 1807.00431. [Online]. Available: http://arxiv.org/abs/1807.00431.

[2] S. Kiritchenko and S. M. Mohammad, *Examining gender and race bias in two hundred sentiment analysis systems*, 2018. arXiv: 1805.04508 [cs.CL].

[3] D. Diviny, *How to effectively use machine learning to implement public policy*, 2019. [Online]. Available: https://www.nousgroup.com/insights/effectively-use-machine-learning-implement-public-policy/.

[4] C. Rudin, *Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead*, 2018. arXiv: 1811.10154 [stat.ML].

[5] T. Peltola, "Local interpretable model-agnostic explanations of bayesian predictive models via kullback-leibler projections," *CoRR*, vol. abs/1810.02678, 2018. arXiv: 1810.02678. [Online]. Available: http://arxiv.org/abs/1810.02678.

[6] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, *Object detectors emerge in deep scene cnns*, 2014. arXiv: 1412.6856 [cs.CV].

[7] R. Fong and A. Vedaldi, "Net2vec: Quantifying and explaining how concepts are encoded by filters in deep neural networks," *CoRR*, vol. abs/1801.03454, 2018. arXiv: 1801.03454. [Online]. Available: http://arxiv.org/abs/1801.03454.

[8] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, and R. Sayres, *Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav)*, 2017. arXiv: 1711.11279 [stat.ML].

[9] I. J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples*, 2014. arXiv: 1412.6572 [stat.ML].

[10] A. Shrikumar, P. Greenside, and A. Kundaje, *Learning important features through propagating activation differences*, 2017. arXiv: 1704.02685 [cs.CV].

[11] P.-J. Kindermans, K. T. Schütt, M. Alber, K.-R. Müller, D. Erhan, B. Kim, and S. Dähne, *Learning how to explain neural networks: Patternnet and patternattribution*, 2017. arXiv: 1705.05598 [stat.ML].

[12] J. Adebayo, J. Gilmer, M. Muelly, I. J. Goodfellow, M. Hardt, and B. Kim, "Sanity checks for saliency maps," *CoRR*, vol. abs/1810.03292, 2018. arXiv: 1810.03292. [Online]. Available: http://arxiv.org/abs/1810.03292.

[13] M. D. Zeiler and R. Fergus, *Visualizing and understanding convolutional networks*, 2013. arXiv: 1311.2901 [cs.CV].

[14] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008. [Online]. Available: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[15] A. Fabisch. (2014). "T-sne in scikit learn," [Online]. Available: https://alexanderfabisch.github.io/t-sne-in-scikit-learn.html.

[16] J. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, Nov. 2000. DOI: 10.1214/aos/1013203451.

[17] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, *Striving for simplicity: The all convolutional net*, 2014. arXiv: 1412.6806 [cs.LG].

[18] A. Shrikumar. (2019). "Deeplift github repository," [Online]. Available: https://github.com/kundajelab/deeplift.

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

[20] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller, "Explaining nonlinear classification decisions with deep taylor decomposition," *Pattern Recognition*, vol. 65, pp. 211–222, May 2017, ISSN: 0031-3203. DOI: 10.1016/j.patcog.2016.11.008. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2016.11.008.

[21] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K.-R. Mueller, *How to explain individual classification decisions*, 2009. arXiv: 0912.1128 [stat.ML].

[22] K. Simonyan, A. Vedaldi, and A. Zisserman, *Deep inside convolutional networks: Visualising image classification models and saliency maps*, 2013. arXiv: 1312.6034 [cs.CV].

[23] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, *Learning deep features for discriminative localization*, 2015. arXiv: 1512.04150 [cs.CV].

[24] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," *International Journal of Computer Vision*, vol. 128, no. 2, pp. 336–359, Oct. 2019, ISSN: 1573-1405. DOI: 10.1007/s11263-019-01228-7. [Online]. Available: http://dx.doi.org/10.1007/s11263-019-01228-7.

[25] M. Lin, Q. Chen, and S. Yan, *Network in network*, 2013. arXiv: 1312.4400 [cs.NE].

[26] A. Chattopadhay, A. Sarkar, P. Howlader, and V. N. Balasubramanian, "Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks," *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Mar. 2018. DOI: 10.1109/wacv.2018.00097. [Online]. Available: http://dx.doi.org/10.1109/WACV.2018.00097.

[27] M. Sundararajan, A. Taly, and Q. Yan, *Axiomatic attribution for deep networks*, 2017. arXiv: 1703.01365 [cs.LG].

[28] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje, *Not just a black box: Learning important features through propagating activation differences*, 2016. arXiv: 1605.01713 [cs.LG].

[29] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg, *Smoothgrad: Removing noise by adding noise*, 2017. arXiv: 1706.03825 [cs.LG].

[30] R. C. Fong and A. Vedaldi, "Interpretable explanations of black boxes by meaningful perturbation," *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017. DOI: 10.1109/iccv.2017.371. [Online]. Available: http://dx.doi.org/10.1109/ICCV.2017.371.

[31] M. T. Ribeiro, S. Singh, and C. Guestrin, *"why should i trust you?": Explaining the predictions of any classifier*, 2016. arXiv: 1602.04938 [cs.LG].

[32] ——, "Anchors: High-precision model-agnostic explanations," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[33] M. T. Ribeiro. (2020). "Anchors github repository," [Online]. Available: https://github.com/marcotcr/anchor.

[34] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017, pp. 4765–4774. [Online]. Available: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[35] S. Lundberg. (2020). "Shap github repository," [Online]. Available: https://github.com/slundberg/shap.

[36] K. Aas, M. Jullum, and A. Løland, *Explaining individual predictions when features are dependent: More accurate approximations to shapley values*, 2019. arXiv: 1903.10464 [stat.ML].

[37] Z. C. Lipton, *The mythos of model interpretability*, 2016. arXiv: 1606.03490 [cs.LG].

[38] F. Doshi-Velez and B. Kim, *Towards a rigorous science of interpretable machine learning*, 2017. arXiv: 1702.08608 [stat.ML].

[39] M. Ancona, E. Ceolini, C. Öztireli, and M. Gross, *Towards better understanding of gradient-based attribution methods for deep neural networks*, 2017. arXiv: 1711.06104 [cs.LG].

[40] J. Choe, S. J. Oh, S. Lee, S. Chun, Z. Akata, and H. Shim, *Evaluating weakly supervised object localization methods right*, 2020. arXiv: 2001.07437 [cs.CV].

[41] P. Dabkowski and Y. Gal, *Real time image saliency for black box classifiers*, 2017. arXiv: 1705.07857 [stat.ML].

[42] D. Alvarez-Melis and T. S. Jaakkola, *On the robustness of interpretability methods*, 2018. arXiv: 1806.08049 [cs.LG].

[43] P.-J. Kindermans, S. Hooker, J. Adebayo, M. Alber, K. T. Schütt, S. Dähne, D. Erhan, and B. Kim, *The (un)reliability of saliency methods*, 2017. arXiv: 1711.00867 [stat.ML].

[44] M. T. Ribeiro, S. Singh, and C. Guestrin, *Model-agnostic interpretability of machine learning*, 2016. arXiv: 1606.05386 [stat.ML].

[45] M. Ancona. (2019). "Deep explainer github repository," [Online]. Available: https://github.com/marcoancona/DeepExplain.

[46] D. Race. (2018). "Skater github repository," [Online]. Available: https://github.com/oracle/Skater.

[47] R. Fong and A. Vedaldi. (2019). "Torchray github repository," [Online]. Available: https://github.com/facebookresearch/TorchRay.

[48] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[49] K. API. (2019). "Keras applications," [Online]. Available: https://keras.io/api/applications/.

[50] V. Petsiuk. (2017). "Gradcam github repository," [Online]. Available: Keras%20implementation%20of%20GradCAM.

[51] M. Alber. (2019). "Deeplift wrapper github repository," [Online]. Available: `https://github.com/albermax/innvestigate`.

[52] M. Ribeiro. (2020). "Lime github repository," [Online]. Available: `https://github.com/marcotcr/lime`.