A study of interpretability mechanisms for deep networks

Apurva Dilip Kokate
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/etd

Part of the Artificial Intelligence and Robotics Commons, and the Mechanical Engineering Commons

Recommended Citation

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
A study of interpretability mechanisms for deep networks

by

Apurva Kokate

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Soumik Sarkar, Co-major Professor
Jin Tian, Co-major Professor
Chinmay Hegde
Forrest Bao

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University
Ames, Iowa
2018

Copyright © Apurva Kokate, 2018. All rights reserved.
DEDICATION

I dedicate this thesis to my mother, Smt. Prema Kokate and my father, Shri Dilip Kokate whose support, motivation, and eternal love, since the beginning of my journey to the United States until now, has enabled me to accomplish this goal. I would like to thank my major professor for his support and patience in successfully achieving the research objective. Lastly, I thank my family for their loving guidance and financial assistance throughout the time I spent for my master’s degree.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>x</td>
</tr>
<tr>
<td>NOMENCLATURE</td>
<td>xi</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>xii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>xiii</td>
</tr>
<tr>
<td>CHAPTER 1. INTRODUCTION</td>
<td>14</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>15</td>
</tr>
<tr>
<td>1.2 Problem Definition</td>
<td>17</td>
</tr>
<tr>
<td>1.2.1 Axioms For Interpretability Mechanisms</td>
<td>18</td>
</tr>
<tr>
<td>1.3 Applications</td>
<td>20</td>
</tr>
<tr>
<td>1.4 Summary</td>
<td>21</td>
</tr>
<tr>
<td>CHAPTER 2. INTERPRETABILITY MECHANISMS</td>
<td>23</td>
</tr>
<tr>
<td>2.1 Literature Review</td>
<td>23</td>
</tr>
<tr>
<td>2.1.1 Deconvolution Networks</td>
<td>23</td>
</tr>
<tr>
<td>2.1.2 Guided Back-propagation (GBP)</td>
<td>25</td>
</tr>
<tr>
<td>2.1.3 Grad-CAM And Guided Grad-CAM</td>
<td>25</td>
</tr>
<tr>
<td>2.1.4 Saliency Maps</td>
<td>28</td>
</tr>
<tr>
<td>2.1.5 LIME (Local Interpretable Model-agnostic Explanations)</td>
<td>31</td>
</tr>
<tr>
<td>2.1.6 Activation Maximization</td>
<td>31</td>
</tr>
<tr>
<td>2.1.7 Integrated Gradients</td>
<td>31</td>
</tr>
<tr>
<td>2.1.8 Deep LIFT (Learning Important Features Through propagation):</td>
<td>32</td>
</tr>
<tr>
<td>2.2 Algorithm Comparison</td>
<td>32</td>
</tr>
<tr>
<td>2.3 Forward Backward Interpretability (FBI) Algorithm</td>
<td>35</td>
</tr>
<tr>
<td>2.3.1 Algorithm</td>
<td>37</td>
</tr>
<tr>
<td>2.3.2 Preliminary Results</td>
<td>40</td>
</tr>
<tr>
<td>CHAPTER 3. FRAMEWORK FOR TESTING INTERPRETABILITY MECHANISMS</td>
<td>41</td>
</tr>
<tr>
<td>3.1 Overview</td>
<td>42</td>
</tr>
<tr>
<td>3.1.1 Model Perturbation</td>
<td>43</td>
</tr>
<tr>
<td>3.1.2 Data Perturbation</td>
<td>44</td>
</tr>
<tr>
<td>3.2 Model Perturbation Metric</td>
<td>45</td>
</tr>
<tr>
<td>3.2.1 Calculating Elementwise Difference:</td>
<td>45</td>
</tr>
<tr>
<td>3.2.2 Calculating KL-Divergence</td>
<td>46</td>
</tr>
<tr>
<td>3.2.3 Calculating Architecture Difference</td>
<td>48</td>
</tr>
</tbody>
</table>
3.3 Framework ........................................................................................................... 49
   3.3.1 Experiment Setup ......................................................................................... 51
   3.3.2 Histogram Of Weights In Convolutional Layer 1 ........................................ 52
   3.3.3 Histogram Of Weights In Convolutional Layer 2 .......................................... 56
   3.3.4 Histogram Of Weights In Dense Layers....................................................... 60
3.4 Results .................................................................................................................. 64
   3.4.1 Saliency Map Results ................................................................................... 64
   3.4.2 Grad-CAM Results ....................................................................................... 74
   3.4.3 Graphical Results ......................................................................................... 84
   3.4.4 Data Perturbation Results ............................................................................ 87
3.4 Deduction Of Model Behavior ............................................................................. 89

CHAPTER 4. RESULTS ON MATERIAL SCIENCE DATASET ............................................. 90
  4.1 Material Science Dataset Results ...................................................................... 91
     4.1.1 Out Of Sample Generated Results ............................................................ 94
     4.1.2 Architecture Perturbation Results ............................................................ 96

CHAPTER 5. CONCLUSION .......................................................................................... 100
  5.1 Contribution ........................................................................................................ 100
  5.2 Discussion ........................................................................................................... 100

REFERENCES ............................................................................................................ 102
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Working of a basic Deep Neural Network</td>
<td>16</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Examples of Sensitivity, Implementation invariance and Performance at saturation</td>
<td>19</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Framework for Deconvolutional network</td>
<td>24</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Effect of CAM on model layers. (source: <a href="https://arxiv.org/abs/1610.02391">https://arxiv.org/abs/1610.02391</a>)</td>
<td>26</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Flow visualization for Grad-CAM and Guided Grad-CAM. (source: <a href="https://arxiv.org/abs/1610.02391">https://arxiv.org/abs/1610.02391</a>)</td>
<td>28</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Flow visualization for Saliency maps.</td>
<td>30</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Forward Backward Interpretability Diagram</td>
<td>37</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Algorithm for Forward Backward Interpretation</td>
<td>38</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Preliminary results on imagenet dataset.</td>
<td>40</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Left: Guided back propagation result Right: FBI result</td>
<td>40</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Showing results of Grad-CAM on Oxford flower dataset. Left: Training iteration 2799, Middle: Training iteration 3199, Right: training iteration 3599</td>
<td>42</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Algorithm for Element-wise difference.</td>
<td>46</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Algorithm for KL Divergence</td>
<td>47</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Calculating convolution layer output. (source: <a href="http://cs231n.github.io">http://cs231n.github.io</a>)</td>
<td>48</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Calculating pooling layer output. (source: <a href="http://cs231n.github.io">http://cs231n.github.io</a>)</td>
<td>49</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Backpropagation in neural networks (source: <a href="https://sebastianraschka.com">https://sebastianraschka.com</a>)</td>
<td>50</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Comparison of weights between model 1 and model 2 weights found in the first Convolution layer</td>
<td>52</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Comparison of weights between model 1 and model 3 weights found in the first Convolution layer</td>
<td>53</td>
</tr>
</tbody>
</table>
Figure 19: Comparison of weights between model 1 and model 4 weights found in the first Convolution layer

Figure 20: Comparison of weights between model 1 and model 5 weights found in the first Convolution layer

Figure 21: Comparison of weights between model 1 and model 6 weights found in the first Convolution layer

Figure 22: Comparison of weights between model 1 and model 7 weights found in the first Convolution layer

Figure 23: Comparison of weights between model 1 and model 8 weights found in the first Convolution layer

Figure 24: Comparison of weights between model 1 and model 9 weights found in the first Convolution layer

Figure 25: Comparison of weights between model 1 and model 2 found in the second Convolution layer

Figure 26: Comparison of weights between model 1 and model 3 found in the second Convolution layer

Figure 27: Comparison of weights between model 1 and model 4 found in the second Convolution layer

Figure 28: Comparison of weights between model 1 and model 5 found in the second Convolution layer

Figure 29: Comparison of weights between model 1 and model 6 found in the second Convolution layer

Figure 30: Comparison of weights between model 1 and model 7 found in the second Convolution layer

Figure 31: Comparison of weights between model 1 and model 8 found in the second Convolution layer

Figure 32: Comparison of weights between model 1 and model 9 found in the second Convolution layer

Figure 33: Comparison of weights between model 1 and model 2 found in the Dense layers
Figure 34: Comparison of weights between model 1 and model 3,4,5 found in the Dense layers................................................................. 61

Figure 35: Comparison of weights between model 1 and model 6,7,8 found in the Dense layers................................................................. 62

Figure 36: Comparison of weights between model 1 and model 9 found in the Dense layers................................................................. 63

Figure 37: Saliency maps results obtained over model 1 (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 65

Figure 38: Saliency maps results obtained over model 2. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 66

Figure 39: Saliency maps results obtained over model 3 (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 67

Figure 40: Saliency maps results obtained over model 4. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 68

Figure 41: Saliency maps results obtained over model 5. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 69

Figure 42: Saliency maps results obtained over model 6. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 70

Figure 43: Saliency maps results obtained over model 7. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 71

Figure 44: Saliency maps results obtained over model 8. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 72

Figure 45: Saliency maps results obtained over model 9. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 73

Figure 46: Grad-CAM results obtained over model 1. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 75

Figure 47: Grad-CAM results obtained over model 2. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 76

Figure 48: Grad-CAM results obtained over model 3. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 77
Figure 49: Grad-CAM results obtained over model 4. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 78

Figure 50: Grad-CAM results obtained over model 5. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 79

Figure 51: Grad-CAM results obtained over model 6. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 80

Figure 52: Grad-CAM results obtained over model 7. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 81

Figure 53: Grad-CAM results obtained over model 8. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 82

Figure 54: Grad-CAM results obtained over model 9. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 83

Figure 55: MSE comparisons against perturbation metric using Saliency maps and Grad-CAM...................................................................................................................... 85

Figure 56: SSIM comparisons against perturbation metric using Saliency maps and Grad-CAM...................................................................................................................... 86

Figure 57: Saliency map output against perturbed data. Original vs repeated pattern vs feature perturbation and additional features. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)...... 88

Figure 58: Deep Convolutional Network architecture for classifying microstructures........ 90

Figure 59: Confusion matrix created using prediction results on microstructure dataset....... 90

Figure 60: Saliency map results on microstructure dataset samples................................. 92

Figure 61: Grad-CAM results on microstructure dataset samples................................. 93

Figure 62: Symmetrically generated data results obtained on Saliency maps. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)................................................................. 95

Figure 63: Asymmetrically generated data results obtained on Saliency maps. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)................................................................. 95

Figure 64: Test models generated by architecture perturbation........................................ 96
Figure 65: Saliency map outputs against model 1 in pixel space over microstructure test sample

Figure 66: Saliency map outputs against model 2 in pixel space over microstructure test sample

Figure 67: Guided Grad-CAM outputs against model 1 in pixel space over microstructure test sample

Figure 68: Guided Grad-CAM outputs against model 2 in pixel space over microstructure test sample
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Comparison of algorithms</td>
<td>33</td>
</tr>
<tr>
<td>2</td>
<td>Test model configuration</td>
<td>51</td>
</tr>
<tr>
<td>3</td>
<td>MSE and SSIM over Saliency maps outputs</td>
<td>74</td>
</tr>
<tr>
<td>4</td>
<td>MSE and SSIM over Grad-CAM outputs</td>
<td>84</td>
</tr>
<tr>
<td>5</td>
<td>MSE and SSIM over Saliency maps vs Guided Grad-CAM outputs</td>
<td>99</td>
</tr>
</tbody>
</table>
NOMENCLATURE

DNN- Deep Neural network
CNN- Convolutional neural network
Grad-CAM- Gradient based class activation mapping
LIME- Local Interpretable Model Agnostic explanations
Deconv- Deconvolution
GBP- Guided backpropagation
MSE- Mean squared error
SSIM- Structural similarity index measure
Frwrd Bckwr- Forward Backward
FBI- Forward Backward Interpretation
I would like to thank my committee chair, Dr. Soumik Sarkar, and my committee members, Dr. Jin Tian, Dr. Chinmay Hegde, and Dr. Forrest Bao, for their guidance and support throughout the course of this research. I would like to thank Dr. Sarkar, for giving me the opportunities to work on the various projects and my lab group, Self-aware systems lab, for their backing during my research.

In addition, I would also like to thank my friends, colleagues, the department faculty, and staff for making my time at Iowa State University a wonderful experience. I would like to thank Veera and Charu for giving me a much-needed emotional support. Last but not the least, I would like to express my gratitude towards my parents for their constant motivation throughout the progress of my research.
ABSTRACT

Deep neural networks are traditionally considered to be “black-box” models where it is generally difficult to interpret a certain decision made by such models given a test instance. However, as deep learning is increasingly becoming the tool of choice in making many safety-critical and time-critical decisions such as perception for self-driving cars, the machine learning community has been extremely interested recently to build interpretation mechanisms for these so called black box deep learning models primarily to build users’ trust with the models. Many such mechanisms have been developed to explain behavior of deep models such as convolutional neural networks (CNNs) and provide visual interpretations of their classification decisions. However, there is still no consensus in the community about the specific goals and performance metrics for the interpretability mechanisms. In this thesis, we review the recent literature to arrive at a formal definition for the “Interpretability-problem” for CNNs with the help of different axioms. We observe that many recently proposed mechanisms do not adhere to the axioms of interpretability and hence not quite robust in performance. In this context, we propose a framework to test the interpretation algorithms under model perturbation and data perturbation. This framework tests the “sensitivity” of the algorithms and helps in evaluating “implementation invariance”, which are desired characteristics for any interpretability mechanism. We demonstrate our framework using two well-known algorithms namely “Saliency Maps” and “Grad-CAM” and introduce a new interpretability technique called “Forward-Backward Interpretability algorithm” that provides a systematic framework for visualizing information flow in deep networks. Finally, we also present visualization and interpretability results for an impactful scientific application involving microstructure-property mapping in material science.
CHAPTER 1. INTRODUCTION

Deep networks (DNN) are set of algorithms that are loosely modeled over the human brain to replicate how it maps input to output. Deep Convolutional networks are widely used in image classification due to their ability of capturing spatial information of a given data sample [21] [22] [24]. A convolution operation is a repetitive pointwise multiplication of a filter/stencil/feature over a portion of the input image while shifting the filter across the image with a given stride. The result of these multiplications is a vector/activation that denotes the degree of the presence of the filter in the image with a higher value indicating a higher probability of the filter occurring at that position. A non-linear activation then follows the convolution along with a pooling operation to learn the most relevant features of the image. The information gained from the pooled activations is processed by densely connected layers which learns their weights as too find the best combination that maximizes the correct output node in the network. A loss is generated if the wrong node is maximized and this error/loss is backpropagated to adjust the weights accordingly.

As an example of deep CNN’s effectiveness, state-of-the-art approaches show that such a model can efficiently make correct predictions of over 1.2 million images present in the ImageNet dataset with top 5 test error rates of 17 % [3]. This performance is considered as a major development as compared to traditional image classification/computer vision methods. However, merely relying on the accuracy of the model does not built enough trust in its results when used in medical diagnoses, disease classification and safety-critical human engineered systems. Further, there is a need to quantify such a trust and measure not only the correctness of predictions but also the reasoning behind them. This problem has been mentioned
repetitively as understanding the black box nature, explain-ability, or interpretability of Deep Networks.

In this Chapter, we will discuss the need for interpretability in Deep Networks when the models are showing high accuracy as well as when they are generating incorrect results. We will also extend the discussion to the need for a check on the correctness of the mechanisms that provide a solution to the “interpretability problem”. We will explain what we consider interpretability to be for Convolutional Networks and give a veritable definition to the solution. We will use axioms to provide clarity to the definition and further set the stage for an ideal interpretability mechanism. We will discuss various scenarios where explanations can be used to inference knowledge about the data as well as the model. We will end this Chapter with a summary of this thesis and what questions the reader can expect to find an answer to.

1.1 Motivation

The motivation for this thesis, comes from the need to have correct explanations for model predictions. This section probes the question of why we need to understand model behavior and why we need to analyze methods which prove an explanation before using them. We consider two cases to explain the need for Deep models to be interpretable. Let us consider a fundamental case where the model is making incorrect decisions and training it for a longer time has no effect on the accuracy of its predictions.

We have two options that may improve the model accuracy. One way is to change the training dataset fed during the training stage of the model. If we can visualize what the model has learnt till the current stage, we can feed in training data that includes more samples including features that the model is unable to pick up and further we can eliminate samples in the training data that have confused the model. A second option is to change the model itself.
If we were to explain parts of a model and visualize its learning, we can effectively change the model to perform in a better way.

A more difficult scenario to understand is that in which the model is showing high accuracy. In this case, we might not even ask for an explanation as the goal of a predictive model is being satisfied. To get an intuition of this we mention a well-known example in the interpretability community of classifying a husky dog. The model would classify the dog as a husky due to the snow around it. In this case, it would help the training if we were to include more samples with a husky in an indoor location. Then the model would pick up intricacies in how huskies look rather than basing it on snow which although may be correct in most cases but is fundamentally wrong. Hence, we can use visualizations of model behavior to make sure that the reasoning behind decisions made are in fact correct.

We as developers of machine learning systems are also could be liable to provide an explanation of the decisions made by our models to the users. For example, the right to

![Diagram of basic Deep Neural Network](image)

Figure 1: Working of a basic Deep Neural Network
explanation act in certain sectors provides rights to ‘meaningful information about the logic involved’ in applications which require a human to depend on a machine [19]. To adhere to this requirement, there is a critical need for methods to seek explanations for models such as deep CNNs.

Here we arrive at the need for the later part of the probed question; are interpretability mechanisms able to correctly portray the complete model behavior? It is easy to see that finding a solution to the “interpretability problem” is unusable if we cannot justify it or if it is not fault tolerant. It is observed that Interpretability algorithms do not perform well with adversarial examples. Hence this thesis is an effort to study the methods developed and form a collection of various adversarial examples that test the correctness of such algorithms.

1.2 Problem Definition

We will now restrict our attention to the “interpretability problem” in the domain of CNNs. As mentioned earlier, a 2D CNN works in the image domain and hence its explanations can also be mapped onto a pixel level grid. Although this problem is non-trivial it makes it easier to have a visual input and interpretation to give an intuitive sense of whether it is correct. Here it is worth mentioning that what may look correct may not be an accurate description of the model performance and vice versa. Hence whenever necessary we assume that our models are learning the correct features, and the generated explanations also are in accordance with how a human would be expected to make an elucidation.

As far as the scope of this thesis, we define interpretability of a CNN to be the exact pixels of the input image that have affected the prediction of the network. Here, we would like the most contributing pixels to be highlighted and subsequently reduce their intensity as we tread down the contribution. Given a threshold value of contribution, we aim to find a subset
of pixels in the input image who have a contribution higher than the subset. Consequently, we aim to generate the smallest subset of pixels in the input image that are most necessary for classifying the image in the target class.

### 1.2.1 Axioms For Interpretability Mechanisms

As it is a new-fangled problem we shall first define it to the best of our knowledge with axioms that have been defined in previous works. Intuitively, interpretability can be described as visualizing the information flow in the network and understanding what the model has learnt. We restrict ourselves to Convolutional networks and visualize the model through images. The subset that defines the explanation criteria define above, must follow the following axioms:

#### 1.2.1.1 Sensitivity

This axiom mentioned in [7] focuses on Class discriminative explanations. The axiom was originally proposed to compare the explanations of an image with a baseline, however we generalize it over any two images. It states that a non-zero score must be given to differing features of any two images with different class predictions. Intuitively, if two images have differing features and different class predictions then there is a high probability that the model has based its classification decision on these features rather than the ones in common between the images and has been magnified through the forward pass.

#### 1.2.1.2 Implementation invariance

This axiom mentioned in [7] states that interpretations provided by two functionally equivalent networks must be the identical. By functionally equivalent, we mean that the networks have the same output of every input that is fed to them. We extend this axiom further as, the difference in explanations given by interpretability mechanisms must vary with a
constant factor of change in models. Hence, models that show high variations in performance must have different explanations and models that have a small variation must show more identical explanations. It is also worth noting, that the same model should the exact same explanation for a given input. In other words, mechanism must not be probabilistic.

### 1.2.1.3 Performance at saturation

This axiom focuses on a specific scenario where the model is saturated. In other words, due to the perfect fit of data points the gradient signal through the model is zero. This is a common observation in huge networks with multiple fully connected layers. Interpretability mechanisms should be independent of this scenario and the explanations of the model must not be affected by the saturation of gradients. Intuitively, to avoid the scenario completely, the community has attempted to develop algorithms independent of gradient back propagation through the entire network. Although gradients give important information about the dependencies of the model towards change of input, backpropagating through several layers at once is prone to saturation.

![Figure 2: Examples of Sensitivity, Implementation invariance and Performance at saturation](image-url)
1.2.1.4 Local fidelity:

Local as well as Global Fidelity is described in [8]. This axiom asks interpretable mechanisms to strive for accurate description of model behavior at least in the vicinity of a test instance. We mention instance because it is difficult to interpret Deep Networks directly. We may want to approximate the model around a test example in a linear fashion to increase its interpretability. In other words, visualizations of model behavior should be faithful to the actual model components. A CNN model can best be described by its weights, biases, and the architecture in which they are combined. Hence to test against local fidelity we will ask the question of whether a method describes model instances truthfully. We can also interpret this axiom in terms of features present in the input image. The model must be able to recognize all important features that are local to a class. These features must be given positive scores. This measure ensures that important features affecting the classification are available in the visualizations and hence gives truthful visualizations.

1.2.1.5 Global fidelity:

Global Fidelity [8] is an extension of local fidelity over all the classes and instance of the model. This also describes a perfect interpretability mechanism because it essentially strives towards a complete inference of model performance using all the components that have contributed to classification.

1.3 Applications

This section we describe various circumstances where interpretability mechanisms play a crucial role in the development of AI based systems. Further, it is also critical that the interpretations follow all correctness guidelines and accurately describe the model behavior.
When Deep Networks are used in health-care domain there is absolute necessity for providing explanations as shown in [5]. They correctly identify the necessity to explain the predictions of their model and use a more interpretable gradient boosting tree as compared to Deep Networks.

Interpretability algorithms find increasing demands in case of human safety critical systems like self-driving cars, as shown in [4]. They use heat maps over input images to justify correctness of their system.

When used in disease classification, trust in model predictions can be strengthened through generating a visual explanation for parts of the image that highlight the affected area. In the attempt by Sambuddha Ghosal et al. [6] [25] they can replace expert opinions of diseased plant leaves with deep networks and add trust to the model by giving an explanation.

1.4 Summary

This thesis provides a study of various mechanisms that attempt to solve that “interpretability problem” for a given CNN model. In Chapter 2, we provide the reader with a helpful explanation of how some of the popular interpretability methods work and salient features responsible for their success. We also provide a comparison and contrast of the different methods and try to classify them into categories based on their nature. We also propose a new method called Forward-Backward Interpretation algorithm that serves as an intuitive method to realize interpretation of model. In Chapter 3, we discuss model perturbation and data perturbation experiments that are designed to test the correctness of a given interpretability mechanism. We propose a new model perturbation metric which quantifies the change between two given models. We also test our proposed framework on Saliency maps as well as Grad-CAM and discuss the results obtained. In Chapter 4, we apply the proposed
framework (whilst testing Saliency maps) on a dataset created with material science microstructure samples created at Iowa State University. The results of these experiments are used by Balaji Sesha Sarath, Pokuri et al. to produce new micro-structures that follow the morphological traits highlighted by Saliency maps and to justify correctness of results.
CHAPTER 2. INTERPRETABILITY MECHANISMS

This Section discusses various methodologies that have been developed, which attempt to visualize the information flow in a network. Each method makes a notable contribution to understanding the problem of “Interpretability” and have been studied with the intention of forming a new algorithm that combines the best aspects of all. We will also talk about the intuitions behind each algorithm and argue over their correctness.

2.1 Literature Review

2.1.1 Deconvolutional Networks

This technique [2] attempts to visualize what each Convolutional layer has learnt. The method creates an unsupervised learning model. The new network is basically another CNN in a reverse manner such that the filter maps can be reconstructed into an image and is composed of Convolutional layers, followed by a ReLU operation followed by unpooling. Such an architecture inverts data flow of a traditional CNN.

The Convolutional layer is constructed as a simple inverse of the original filter maps. This process is termed as an inverse convolution operation. This reconstructs top level feature maps by making them sparse and then visualizing them by a simple convolution using the newly created inverse filters. The sparsity depends on the size of the convolutional filters and the stride used in the original model.

The ReLU operation is performed just as a traditional ReLU operation in the original CNN. If we consider the new network as a backward pass, the ReLU layer can be understood as allowing only the positive gradients to propagate and hence reducing noise in the result.
Unpooling, reverses the effect of Max pooling in a CNN. The top activations of the original model are used as switches to form the unpooling layer. These switches record the signed maximum values (that would have been used in Max pooling to reduce the dimension of the input) and the locations of these values in the original input. It is worth to note here that the use of these switches has been argued in works of Springenberg, Dosovitskiy et al. [9] to be conditioned on the input image rather than visualizing model learning. Using these switches, the values are placed back into their correct switch location during unpooling, while the neighborhood values are set to zero. The size of this neighborhood depends on the stride used in the original model.

It is worth noting that this method generates the visualizations through forward propagation in a new network rather than back propagation as other methods that are discussed below. However, works of [9] discuss that deconvolution is equivalent to a backward pass of the top gradient through the original model which adds a support to the correctness of this approach.

![Figure 3: Framework for Deconvolutional network.](image-url)
2.1.2 Guided Back-propagation (GBP)

This technique [9] attempts to visualize the information learnt by a single neuron by using a backward pass through the model starting from the activations of the target neuron. While back propagating through each layer, the algorithm uses layer activation in case of convolutional layers and switches (mentioned in Deconvolutional Networks) in case of Max pooling layers. In the case of ReLU layers, the method converts the negative gradients to zero and zeroes down the backward values for negative activations. Negative gradients at a particular ReLU neuron, state that this neuron has a negative influence on the class that we are trying to visualize. The result is a cleaner and more accurate visualization where maximum noise has been eliminated. Computes gradients of the activation of a single neuron through the network to visualize the part of the image that. Guided back prop combines the deconvolutional approach [2] with backprop with the slight modification to ReLU pass. The method is guided in the sense of using negative activations to realize contribution is negative. Finally, to visualize entire model we backpropagate through output node. This method can be used to visualize parts of model separately as well.

2.1.3 Grad-CAM And Guided Grad-CAM

This technique [10] focuses on class-discriminative explanations, by this we mean that the reconstructions differ with different class inputs. Grad-CAM exploits the spatial information encoded in Convolutional layers. The method also substitutes fully-connected layers by a class activation map to further retain the spatial information that is lost in these layers. Gradients are backpropagated by chain rule derivation through this modified network to get a gradient of class output with respect to feature maps of the top convolutional layer. These gradients are passed through a global average pooling to obtain important weights of
each filter which are used to calculate a heat map. These heat maps provide a localization of the top features present in the image but are coarse and produce a low-resolution visualization which cannot be directly applied to the input image. To obtain a pixel space visualization, the heatmap is fused with guided backpropagation to form Guided Grad cam outputs which produce a support in the original image domain that approximately corresponds to a given object of the class detected. Hence the algorithm is based on (1) modifying the base network with class activation maps (2) backpropagation till the final convolutional layer filter maps (3) creating a heat map, and (4) using the heat map to get a pixel space visualization.

Class activation maps compromise model complexity and performance for highly class discriminative model architectures. Convolutional maps are global average pooled followed by a SoftMax layer that provides class scores. Grad-cam modifies this approach by combining the feature maps using a gradient of Output class with respect to feature maps in such a way that no change is required to be made to the model and hence can be applied to a variety of CNN architectures. Further no re training is required. The result is a localization map of weights given to each neuron.

Figure 4: Effect of CAM on model layers. (source: https://arxiv.org/abs/1610.02391)
Backpropagation of gradients is done because the model has essentially been modified and hence the weights of the feature maps do not have a direct correlation to the classification. These weights cannot be directly used as they do not have meaning. Specifically, a higher weight in the convolutional layer could have been modified further in the network to have a low overall effect on classification and vice versa. This is a common observation in Deep Networks as they are highly nonlinear. Hence new weights need to be calculated.

Intuitively, the weights of neurons in the localization map so obtained is equivalent to the importance of the feature maps. Hence, to obtain the most important features in the image, a weighted combination of all feature maps is performed. The heatmap is calculated as a RELU of the resultant vector to obtain the feature maps that have had a positive effect on the classification. Note that these feature maps are weighted, so we know which features have contributed the most positively. The heatmap is then expanded to the dimensions of the input image by bi-linear interpolation.

As mentioned earlier, heatmaps provide coarse visualizations. It is combined with guided back prop output by a pointwise multiplication. This method identifies that the last convolutional layer has the best compromise between high-level semantics and detailed spatial information. It also shows state of the art results with class discriminative visualizations. The results make intuitive sense to a novice observer and is used for the problem of visual question answering. However, since this method substitutes the second half of the network (the fully connected layers), we lose the information encoded in these layers. The result is that the visualizations obtained by Grad-CAM are highly volatile. In the sense that, the convolutional layers change at a greater rate than the model with each training data sample. While
classification, the information in the fully connected layers are used to weigh out these changes and the same nature should have been exploited during the interpretations as well.

2.1.4 Saliency Maps

This technique [1] uses backpropagation of gradients of the output class scores with respect to the input image to highlight the salient parts that is intuitively parts of the image where the output class has occurred and hence that are most required for the classification decision. The method considers the non-linear nature of Convolutional networks and differentiates the input with an approximated linear function. The result is a class saliency map that can be used to create fine-grained visualizations. The method includes a forward pass through the network, a back propagation of gradients, forming a class saliency map and finally using the map for $n$ visualizations.

Figure 5: Flow visualization for Grad-CAM and Guided Grad-CAM. (source: https://arxiv.org/abs/1610.02391)
In the forward pass, the method excludes the output of the final SoftMax layer and uses unnormalized class scores. The intuition behind this is to exploit the effect of change of input only on the target class where-as the SoftMax layer output records a relative importance of class scores and hence considers the effect of the input on other classes as well. Further, experiments conducted by Kotikalapudi, Rangjavendra.et.al in the keras-viz toolkit [15] shows that visualizations cannot be comprehended when normalized class scores are used.

The backpropagation is done with a chain rule derivative starting from the output layer i.e. the unnormalized class scores toward the input. The algorithm by default considers the top class for the backpropagation but any class number can be passed as an argument. It is basically asking the question of where the given class occurs in the image and hence this method can also be used for object detection. However, in experiments conducted by us, we observe the same saliency maps generated for different classes and hence the class discrimination aspect failed in those cases. For multi-dimensional images, this class score is calculated with respect to each channel. The result is a class saliency number for every pixel in the image calculated as the absolute values of these scores.

The class saliency map is simply a 2-dimensional array of the class saliency numbers, it is weakly supervised and is calculated as the maximum of scores for each channel for the number of channels in the image (3 in the case of RGB images and 1 in the case of black and white images). This map tells us about the (a) if the class occurs in that part of the image and (2) till what extent does the class occur.

The class saliency map is then passed through a thresholder which differentiates whether the pixel lies in the foreground or in the background. The use of thresholding enables visualizations to be more precise, as class scores are affected not only by the presence of a
feature/class in the input image but also by the absence of a certain feature/class. For example, while differentiating between the occurrence of a dog or a cat in an image, we are more likely to be confident of the presence of a dog because of the absence of cat whiskers in the image.

A Graph color segmentation algorithm is used, which propagates the saliency number all over the object on whose pixel it occurs, hence splitting the total image into a foreground and a background. The visualization is then generated by displaying the objects that occurs in the foreground while masking those that occur in the background.

This method achieves a good localization of important features of the image while giving a foreground/ background importance to the objects. Hence this method can be used to make a guess of the presence of which parts of the object contributed towards classification, the absence of which parts of the objects contributed to the classification and which parts of the object had nearly no contribution at all.

---

**Figure 6:** Flow visualization for Saliency maps.
2.1.5 **LIME (Local Interpretable Model-agnostic Explanations)**

This method [8] is independent of the model entirely and it learns the behavior by sampling instances of the data around its neighborhood. Lime helps to explain individual predictions and use the knowledge of these multiple predictions to understand how the model works. It attempts to search for an equation that replicates or is near to equivalent to the highly complicated formulations that are performed in a traditional deep network. It takes advantage of local fidelity, in the sense that, interpretable models that are close in the neighborhood of a given test model will provide the same sort of results and hence the same kind of explanations. To generate this equation, the algorithm performs various changes to the input of the image and observes what change has been made to the output class-score.

2.1.6 **Activation Maximization**

This technique [1] works by passing multiple inputs through the network to study what excites a particular neuron in the model. The authors, study neurons in the final class output layer by passing multiple perturbations of an image to realize the optimal solution. They start by passing in, the images present in the training set and work towards generating a new image by using the derivative of the output loss with respect to the input. If the neuron is getting activated, then the loss in minimum. By performing this over multiple iterations, we finally generate an image that activates the neuron maximally. And hence, it can be considered as an optimization problem of maximizing activation with respect to input.

2.1.7 **Integrated Gradients**

This [7] method calculates Gradient * Input, that computes the partial derivatives of the output with respect to each input feature. However, while Gradient * Input computes a single derivative, evaluated at the provided input x, Integrated Gradients computes the average
gradient while the input varies along a linear path from a baseline $\bar{x}$ to $x$. The baseline is defined by the user and often chosen to be zero.

2.1.8 Deep LIFT (Learning Important Features Through propagation):

This approach [11] is used for computing importance scores based on explaining the difference of the output from some ‘reference’ output in terms of differences of the inputs from their ‘reference’ inputs. The ‘reference’ input represents some default or ‘neutral’ input that is chosen according to what is appropriate for the problem at hand. Deep LIFT avoids placing potentially misleading importance on bias terms (in contrast to gradient*input). By allowing separate treatment of positive and negative contributions, the DeepLIFT-RevealCancel rule can identify dependencies missed by other methods.

2.2 Algorithm Comparison

Broadly, algorithms for interpreting the action of deep networks for this task can be grouped as follows:

Class-discriminative approaches, such as Class Activation Mappings (CAM, or its gradient-based variant [10], produce a support in the original image domain that approximately corresponds to a given object class detected in that image. However, such methods are coarse and only produce low-resolution visualizations, and as such cannot be directly applied to very high-resolution images.

On the other hand, pixel-space gradient-based methods such as deconvolution networks [2] and guided back-propagation [9] produce fine-grained features in a given image. However, gradient based methods suffer from either significant computational efficiency concerns or are susceptible to saturation phenomena due to vanishing/exploding gradients, or both.
In reference-based methods like DeepLIFT [11], this issue is alleviated by suitably using a second reference input to stabilize the estimates. However, choosing this reference image is qualitative and can be challenging.

Finally, model-agnostic approaches such as LIME [9] are theoretically sound and can be applied for interpreting deep convolution networks but involve solving challenging optimization problems.

Here we present a comparison of various techniques with respect to the axioms of interpretability. This table has been created with a combination of examples present in the literature as well as the results of experiments conducted by us and hence is accurate to the best of our knowledge.

Table 1: Comparison of algorithms

<table>
<thead>
<tr>
<th>Method</th>
<th>Root</th>
<th>Base IMG</th>
<th>Sensitivity</th>
<th>Implementation invariance</th>
<th>Performance at Saturation</th>
<th>Local Fidelity</th>
<th>Global Fidelity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grad-CAM</td>
<td>Filter weights</td>
<td>Input</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Guided Grad-CAM</td>
<td>Gradient</td>
<td>Input</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>LIME</td>
<td>Back prop</td>
<td>Sampled input</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes [8]</td>
</tr>
<tr>
<td>FBI</td>
<td>Frwrd Backwr</td>
<td>Frwrd IMG</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Grad-CAM shows sensitivity as the weights of the filters in the final convolutional layer are calculated through back propagation through a CAM. This enables it to be highly
differentiating while weighing out different filters and captures the class discriminative features. Guided Grad-CAM show sensitivity due to Grad-CAMs contribution to its results. As mentioned in [7] gradient based techniques do not show sensitivity and focus on unwanted features. Hence, guided back propagation and deconvolutional networks does not show sensitivity. DeepLift displays sensitivity by using a baseline and compute discrete gradient values with respect to this. Integrated gradients also use a similar approach to gain sensitivity. LIME itself works on sampling instances and hence its sensitivity depends on the quality of the samples.

Grad-CAM and guided Grad-CAM do not show implementation invariance as they overlook implementation by the CAM approach. This causes them to be highly dependent on the filter weights which are volatile with respect to each image. Gradient based methods show implementation invariance due to their base nature of calculating gradient of output with respect to input as shown in [7]. Hence saliency maps, guided back propagation and deconvolutional networks show implementation invariance. DeepLIFT violates implementation invariance as chain rule does not apply on discrete gradients during backpropagation. Integrated gradient uses continuous gradients to satisfy implementation invariance. LIME being a model agnostic method satisfies implementation invariance.

Gradient based methods are susceptible to performance at saturation. DeepLIFT and integrated gradients do not use pure gradients and avoids this scenario. LIME is purely independent of gradients and saturation phenomenon.

Grad-CAM and Guided Grad-CAM fail to show both local and global fidelity because of the change in base model that is made. Further, replacing fully connected layers results in loss of explanation for classification decision of the model. Other methods provide a complete
description of the model and hence provide faithful visualizations in the neighborhood of the test instance.

### 2.3 Forward Backward Interpretability (FBI) Algorithm

In this section, we introduce a new, systematic framework for visualizing information flow in deep networks [23]. We will discuss the intuition behind the approach and show preliminary results obtained during the development. The method attempts to bring together the concepts that have assured or are responsible for the success of the previously developed methods and focuses on dealing with nonlinearities present in CNN’s that make interpreting them a challenge. The problem that we study can be framed as such: given any trained deep convolution network model and a given test image, can we find the minimal set of pixels in the input image that a) are most important for the classification decision, and b) considering that only these pixels are existing in the image and the rest of the image is black, the model will make the decision posterior/class core. We also strive towards developing a method whose results are consistent/persistent through training epochs and only varies when the model behavior varies. The proposed method is also developed by keeping in mind the axioms. In other words, our method produces a compact support in the image domain that corresponds to a (high-resolution) feature that contributes to a given explanation.

Let’s now discuss the various methods that have motivated the development of FBI. The most closely related method is Deconvolutional Networks which also attempt to reconstruct the input. We closely follow guided back propagation during back propagation. Like GBP, we will also be back propagating layer by layer beginning from the output node towards the input. We will use forward activations to guide this backward pass to ensure the method is sensitive towards change in model’s performance. Just as in the case of GBP, we
will zero-out elements at positions which have both negative activations as well as negative gradients. The intuition behind this is to adhere local fidelity and give a non-zero score to neurons that have contributed to the classification in a positive manner.

We will use Grad-CAM’s approach of weighing out filter maps and use the top filter maps during back propagation. In addition to this, we will be masking out neurons/filter maps that do not lie in this range of top “k” filters. Just as all the previous approaches have created a pixel space visualization, we will be also using the same approach.

Saliency maps have motivated the method to back propagate through the entire network as we want to adhere to implementation invariance. If two models are equivalent, then our method will show equivalent outputs as all the information encoded in the deep network will be used.

This method does not depend on gradients. The decision is made to adhere to having a good performance at saturation also instead the method attempts to invert each other while coming towards your input. This has been motivated by the approach of deconvolutional networks that attempt to do the same thing. However, we will do this for fully connected layers and nonlinearities such as ReLU and SoftMax. Hence, at each point there is a transparent mathematical inversion performed. Our method is both computationally efficient as well as numerically robust. We will also be using the un-pooling operation described in deconvolutional networks while inverting the max-pooling layer.

Our novel masking scheme attempts to invert only the desired part of the model. This makes the method computationally efficient. To further aid, computational performance, we
will use direct mathematical calculations such as matrix-adjoint. The result is a visualization that contains less noise factor when compared to those of the guided backprop.

### 2.3.1 Algorithm

In this method, we aim to reverse the effect that each layer has had on the input during forward propagation. Due to the non-linear activations and irreversible pooling layers present in the model, a perfect reversal is not possible. However, we attempt to reverse to the maximum possibility by using forward activations to guide this process. During each backward pass we attempt to mask the lowest weights to make the result both cleaner with respect to noise and to reach the minimal subset of pixel important to explain the classifiers performance.
This algorithm is designed for networks that have been trained to an optimal state. This assumption is necessary as our algorithm highly depends on the forward pass through the network during classification and any error with the classification scores will result in a wrong guidance during the backward pass which in turn will provide incorrect visualizations. Further we assume that the classes are differentiated with a set of features which are learnt by the model and only when the network learns the features will the visualizations show their presence.

We start with performing a forward pass of the test image to get the activations at each neuron present in the model whose interpretation is desired. This gives us a “forward value” that we will use as this contains important information about which neurons have been excited by the image. Intuitively, this means that the features learnt by the neuron is present in the given test image.

---

**Algorithm 1: Forward Backward Interpretation**

**Input:** Neural Network $M$ with weights $W = (W_1, W_2, \ldots, W_L)$, 
$InputImage = X$, $Classnumber = c$, $F_0 = X$

**Output:** Interpretation=$I_0$

1. /* Class-discriminative feature map */
2. $I_0^c = ReLU \left( \sum_{k=1}^{K} \alpha_k^c F_k \right)$ /* Forward pass */
3. for $l = 1 : L-1$ do
4. $F_l = W_l F_{l-1}$
5. end

1. /* Backward pass for interpretation */
2. for $i = L-1 : 0$ do
3. $I_i = Mask(Adjoint(W^T_{i+1} I_{i+1}, F_l))$
4. end

---

Figure 8: Algorithm for Forward Backward Interpretation.
We start the backward pass after this stage. The goal is to identify important regions that explain the prediction of the learned network. The input to our backward algorithm is the class indicator vector where the element at the index of the selected class is 1 and the rest of the indices are zero. We observe the activation at all indices and propagate it to the previous layer. The subsequent layers attempt to invert their impact by processing the activation by the adjoint of the layer. Here adjoint is loosely defined as the inverted function due to the nonlinearities involved. Depending on what kind of layer is present we define the adjoint accordingly:

- **SoftMax Layer** - The SoftMax layer converts adjoint of the SoftMax layer performed by Maximum of activations for selected class and Minimum of activations otherwise

\[
\hat{z}_i = \begin{cases} 
\max(z^{(L)}) & i = c \\
\min(z^{(L)}) & i \neq c.
\end{cases}
\]

- **Fully Connected Layers** – Inverting the equation in forwards pass with a ReLU filter

\[
\hat{a}^{(l-1)} = (W^{(l)})^T (\hat{z}^{(l)} - b^{(l)}).
\]

- **Convolutional layers** – Top K convolution filter of the total activations for entire map followed by a deconvolution operation. The deconvolution computes adjoint by convolving the backward activation with the flipped filter weights of a corresponding filter.

- **Un-pooling** - Elementwise product of Forward activation with the minimum of Forward and Backward activations.

\[
\hat{a}_{up}^{(l)} = \min(a_p^{(l)}, \hat{a}_p^{(l)})
\]
2.3.2 Preliminary Results

Here we show results obtained by FBI algorithm on VGG 16 model pretrained with ImageNet dataset with top 1% predictions. The results are compared to that of guided backprop and are seen to have less noise factor. One example of a test image is shown below for detailed comparison.

Figure 9: Preliminary results on imagenet dataset.

<table>
<thead>
<tr>
<th>Input</th>
<th>FBI Interpretations [our method]</th>
<th>Guided Backprop Visualizations</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Input" /></td>
<td><img src="image" alt="FBI Interpretations" /></td>
<td><img src="image" alt="Guided Backprop Visualizations" /></td>
</tr>
</tbody>
</table>

Figure 10: Left: Guided back propagation result Right: FBI result
CHAPTER 3. FRAMEWORK FOR TESTING INTERPRETABILITY MECHANISMS

In this chapter, we will describe various tests that let us know about the correctness of interpretability mechanisms by using adversarial examples. We shall also describe our own model perturbation metric that allows us to quantify the change between two models. This metric is based on the model weights and biases and works on static trained models. We shall then show results over commonly used mechanisms and infer from their performance.

The motivation for this chapter comes from a lab experiment conducted before. The experiment used deep networks to classify the Oxford flower dataset [20] and model interpretations were obtained using Grad-CAM. We noticed that Grad-CAM visualizations would vary greatly over very well-trained models and it was hard to realize what change should be made in the model to improve its performance. At the same time, other visualization techniques such as saliency maps would not change at all over model perturbations which did not insight into the model behavior. We were intrigued to know more about what visualizations to trust. Having conducted a literature survey, we found what could only be expressed as chaos with respect to the view of the machine learning community over interpretability of deep networks. Hence, we wanted to try to consolidate various test strategies into a framework that would let other research groups like ours know which visualizations to trust and hence make an improvement with modifying their base network. The discussion on interpretability mechanisms is ever-growing and not all of the knowledge is compiled, but it is the start nonetheless towards consolidation of opinions.

We also realized that a lot of focus has already been made towards data perturbations and hence we tried to focus more on model perturbations while providing base data perturbations techniques in the hope that further consolidation will be made in the research
community. Introducing model perturbations focuses on the sensitivity of methods as well as implementation invariance, axioms described in Chapter. 1. The focus of our experiment is also to provide a qualitative approach towards testing interpretability in such a way that can be replicated over mechanisms following different basis.

Figure 11: Showing results of Grad-CAM on Oxford flower dataset. Left: Training iteration 2799, Middle: Training iteration 3199, Right: training iteration 3599

3.1 Overview

In this experimental framework, our aim is to create several test scenarios, and produce human understandable measures that approximately correspond to the correctness of an algorithm’s behavior with respect to sensitivity and implementation invariance. We take input as a base model and its weights along with certain test images. The model weights are then changed to get results of model perturbations and we make changes to the test data to get results of data perturbations. Interpretability is best understood under certain specifications, we list as the prerequisites to our framework as follows:

1. The Classification Decision of the model must be same for the given input.
2. The model should display high accuracy. In other words, it should be well trained on a variety of training samples.
3. Model should have equal sequence of layers
3.1.1 Model Perturbation

The aim of this experiment is to measure the effect of change in model weights on the explanations provided by interpretability algorithms. We expect that if two models are very similar the difference in the interpretations should be small, and vice versa. Hence, this experiment has a close correlation with testing the axiom of model invariance. The framework for this experiment is as follows:

We create a set of models that we have perturbed with respect to a single model. We will measure the extent of the perturbation done and assign a value to the difference in models with respect to the base model. We then run the interpretation mechanism whose correctness we want to test and visualize the correlation between model perturbation vs the difference in the outputs. An ideal interpretation mechanism will show a steady increase in the difference between the visualizations.

Now, we arrive at the question of how to perturb models. We realize that the model is composed of its weights and biases and we perturb these in order to induce slight changes to models. It is clear that, as we change the model, its performance would change and hence we notice differences in model accuracy. Change of model weights can be done via two methods:

1. Training the model:

   Model training is a natural way of causing modifications to the layers of the model. It is also the easiest way to observe changes in the top most layers while the lower layers remain fixed.

   During the training phase of the model, the explanations must remain constant over epochs. If the explanations vary greatly then the model might need more training, or the algorithm used for interpretation is not giving a stable result. In the latter case, the error is in the nature of the algorithm.
2. Changing weights:

Manual changes to the weights can be done. If the weights are changed at the lower convolutional layers there should be a drastic change in the explanations given and if the change is made at higher convolution layers the explanations must not change.

3.1.2 Data Perturbation

The aim of this experiment is to measure effect of change in Input on explanation of model decision. There have been various attempts to find images which completely collapse a given interpretation mechanism. Keeping these in mind we design a set of data perturbation techniques that will test if a mechanism if correct. We have listed down ideal behavior with respect to each perturbation. Note that the visualizations themselves are a way to let the observer know whether the interpretability mechanism has passed the test or not and we will also be basing our results on the visually available result and hence we have not currently developed a metric for it. One can easily develop a metric that tests the sanity of these images for probing for the desired pixel values. The data can be perturbed as follows:

1. Changing features:

   The explanation must change if any feature that is highlighted by the algorithm is changed. The explanation may or may not change when a feature that is not highlighted by the algorithm is changed.

2. Keeping only selected features: The classification decision as well as the explanation should not change if only the features highlighted by the algorithm are given as input to the network. This confirms that the features are correct and minimal.

3. Cyclic patterns/Multiple occurrences: Explanations must be consistent across repeating patterns
4. Adding another class to the input sample: The performance of an algorithm can be tested by adding features of another class to the input samples. The explanations for the highest occurring class should not differ vastly.

3.2 Model Perturbation Metric

The proposed experiment evaluates the sensitivity of “interpretability mechanisms” over slight changes made to the model. Hence it is important to first answer the question of how to quantify the difference between two given models. We will focus on very similar models that differ only slightly in (1) Model weights and (2) Model architecture. This subsection describes the methodology used to calculate the difference between models that show very similar behaviors with respect to test accuracies and test loss. We know that this performance is dependent on how the weights and biases of each layer modifies the input to classify it in a category. Hence, to quantify change in a model we will calculate the change in the model weights and further the Kullback-Leibler divergence (KL divergence) between the distribution of the weights per layer. Models that differ slightly in architecture can also show similar performance. We will calculate the difference in the architecture layer wise as well.

3.2.1 Calculating Elementwise Difference:

We calculate element-wise difference layer wise with a simple subtraction of the values of weights. We calculate the total of absolute values of each corresponding weights between model 1 and model 2. We then divide this total by the number of weights present in the layer. The total element-wise difference is calculated by summing up all the layers. The algorithm is shown below:
However, this metric is very stringent toward the models and requires the architecture to exactly be the same.

### 3.2.2 Calculating KL-Divergence

We calculate the KL-divergence of each layer. Then we calculate the total divergence as a summation of the divergence difference of all layers. For two given models, we’ll recover the weights as a numpy matrix and then flatten these weights. The result is a 1-D array containing all the weights of that layer. We will then calculate the probabilistic distribution using a Gaussian kernel density estimate for both models. A kernel density estimate is used to calculate the probability density function of a random variable. It smoothens the data and estimates the shape of a function $f$, that best fits over the given data samples. It is calculated according to the equation given below.

$$
\hat{f}_h(x) = \frac{1}{n} \sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)
$$

We use a Gaussian kernel denoted by the given formula.
The KL divergence is a measure of how one probability distribution diverges from a second, expected probability distributions. It is the expectation of the logarithmic difference between the probabilities at each discrete position of the input.

\[ D_{KL}(P\|Q) = -\sum_i P(i) \log \frac{Q(i)}{P(i)} \]

A KL divergence of one indicates that 2 distributions behave in differing manner and a KL divergence of zero indicates that we can expect similar behavior.

The KL divergence of this probabilistic distribution gives us a measure of other distribution of weights of model 1 diverges from model 2. The algorithm is shown below.

---

**Algorithm 1: K-L Divergence**

<table>
<thead>
<tr>
<th>( (M1, M2) )</th>
</tr>
</thead>
</table>

**Input:** Neural Network M1 with weights \( W1 = (W_1^1, W_2^1, ..., W_L^1) \), M2 with weights \( W2 = (W_1^2, W_2^2, ..., W_L^2) \)

**Output:** Total divergence

1. Total divergence = 0
2. for \( l = 1 : L-1 \) do
3. \( f_1 = Flatten(W_1^l) \) \( f_2 = Flatten(W_2^l) \)
4. \( k_1, k_2 = \text{gaussianKDE}(f_1), \text{gaussianKDE}(f_2) \)
5. \( z_1 = k_1(\text{meshgrid}(W_1^\text{min} : W_1^\text{max})) \)
6. \( z_2 = k_2(\text{meshgrid}(W_2^\text{min} : W_2^\text{max})) \)
7. \( e = KLD(z_1, z_2) \)
8. Totaldivergence += e
9. end

---

Figure 13: Algorithm for KL Divergence.

We may also note that, the KL-Divergence metric is less stringent on individual weights as compared to element-wise difference.
3.2.3 Calculating Architecture Difference

We calculate the difference in the architecture of model 1 against model 2 by looking at the output shapes of each layer.

The output shape is generated as a result of the parameters passed during instantiating layers.

Figure 14: Calculating convolution layer output. (source: http://cs231n.github.io)

When the input is of size \((W_1 \times H_1 \times D_1)\) where \(W_1\), \(H_1\) is the dimension of the input image and \(D_1\) is the number of channels. The output of the convolutional layer is calculated as \((W_2 \times H_2 \times D_2)\) where \(D_2\) is the number of filters. The output shape depends on the spatial extent, padding and stride of the convolution filter as shown below:

The output shape of a ReLU layer is also \((W_2 \times H_2 \times D_2)\) where the element \(x\) is simply a maximum between 0 and \(x\).

The output of the pooling layer is calculated as \((W_3 \times H_3 \times D_2)\) where \(W_3\) and \(H_3\) are reduced dimensions after down-sampling which depends on the spatial extent and stride. The calculating is shown below:
The output dimensions of fully connected layer are \((1 \times 1 \times D)\) where \(D\) is the number of nodes in the next layer. The output dimension of the final fully connected layer is \((1 \times 1 \times O)\) where \(O\) is the number of output classes.

We calculate the difference between each corresponding element in the output shape of the layer and take the absolute value of the total difference. We then divide the summation made on each layer by the total number of layers.

### 3.3 Framework

The experiment is run on the models having the following architecture:

- Conv2D (64,3), Conv2D (64,3), Maxpool (2), Dropout (0.25), Flatten, Dense (128), Dropout (0.5), Dense (10), Softmax.

Where: Conv2D (x,y) stand for an instantiation of a 2-dimensional convolution layer with x filters and y stride. Maxpool(z) stands for an instantiation of max pooling layer with z stride. Dropout(r) stands for an instantiation of Dropout layer with r dropout rate. Flatten stand for a
flatten layer. Dense(d) stand for a fully connected layer with d output nodes and SoftMax stand for a SoftMax layer.

We train the models on the “MNIST” dataset [18] with varying epoch and batch sizes to get slight variations in model. The MNIST dataset consists of handwritten digits as 28x28 grayscale images ranging from 0 to 9. It contains 60,000 training images along with a test set of 10,000 images.

We use modifications in backward passes of the gradients as guided, ReLU and vanilla backprop. These modifications control what the values of gradients are allowed through the network during the backward pass. This can be explained via the image shown below:

![Backpropagation in neural networks](https://sebastianraschka.com)

Figure 16: Backpropagation in neural networks (source: https://sebastianraschka.com)

Traditional or vanilla backpropagations allows all gradient values to be transfer to the previous layer. Guided backpropagation allows only those values to propagate that had a positive forward activation. ReLU back propagation allows only the positive gradient values
to propagate.

### 3.3.1 Experiment Setup

The table below shows the different models used for the experiment.

<table>
<thead>
<tr>
<th>Model number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Epoch</td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>1</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Batch Size</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Test Loss</td>
<td>0.052</td>
<td>0.035</td>
<td>0.036</td>
<td>0.046</td>
<td>0.029</td>
<td>0.028</td>
<td>0.050</td>
<td>0.032</td>
<td>0.029</td>
</tr>
<tr>
<td>Test Accuracy</td>
<td>0.983</td>
<td>0.988</td>
<td>0.991</td>
<td>0.984</td>
<td>0.991</td>
<td>0.991</td>
<td>0.983</td>
<td>0.989</td>
<td>0.992</td>
</tr>
<tr>
<td>KL-Divergence</td>
<td>Base Model</td>
<td>9.40</td>
<td>11.25</td>
<td>0.10</td>
<td>3.55</td>
<td>4.25</td>
<td>0.72</td>
<td>0.84</td>
<td>3.39</td>
</tr>
<tr>
<td>Element-wise difference</td>
<td>Base Model</td>
<td>0.59</td>
<td>0.75</td>
<td>0.40</td>
<td>0.47</td>
<td>0.51</td>
<td>0.38</td>
<td>0.44</td>
<td>0.48</td>
</tr>
</tbody>
</table>

We will use Mean squared error (MSE) and structural similarity index (SSIM) to calculate the difference between the output visualizations of the two methods. MSE is a risk function, corresponding to the expected value of the squared error loss or quadratic loss [13].

\[
\text{MSE}(\overline{X}) = \mathbb{E}\left[\left(\overline{X} - \mu\right)^2\right] = \left(\frac{\sigma}{\sqrt{n}}\right)^2 = \frac{\sigma^2}{n}
\]

SSIM is a perception-based model that considers image degradation as perceived change in structural information, while also incorporating important perceptual phenomena, including both luminance masking and contrast masking terms. [14]

\[
\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}
\]
In the equations shown above $\mu$ represents the average of data values, $\sigma$ represents the variance of the data while $c$ is a variable number.

The given models are tested against Saliency maps and Grad-CAM. The result for model 1 to model 9 are shown in the next section.

Here we show histograms plotted using the weight matrix of each model. We have plotted weights of each model in comparison with Model1. Each histogram represents the weights of Model1 vs Model N for layers Conv2D (64,3) called Conv layer 1, Conv2D (64,3) called Conv layer 2, Maxpool Dropout and Flatten do not have weights, Dense (128) called Dense layer 1, and Dense (10) called Dense layer 2.

### 3.3.2 Histogram Of Weights In Convolutional Layer 1

![Comparison of weights between model 1 and model 2 weights found in the first Convolution layer](image)

Figure 17: Comparison of weights between model 1 and model 2 weights found in the first Convolution layer
Model 1 vs Model 3 Conv layer 1

Figure 18: Comparison of weights between model 1 and model 3 weights found in the first Convolution layer

Model 1 vs Model 4 Conv layer 1

Figure 19: Comparison of weights between model 1 and model 4 weights found in the first Convolution layer
Model 1 vs Model 5 Conv layer1

Figure 20: Comparison of weights between model 1 and model 5 weights found in the first Convolution layer

Model 1 vs Model 6 Conv layer1

Figure 21: Comparison of weights between model 1 and model 6 weights found in the first Convolution layer
Figure 22: Comparison of weights between model 1 and model 7 weights found in the first Convolution layer

Figure 23: Comparison of weights between model 1 and model 8 weights found in the first Convolution layer
Model 1 vs Model 9 Conv layer1

Figure 24: Comparison of weights between model 1 and model 9 weights found in the first Convolution layer

3.3.3 Histogram Of Weights In Convolutional Layer 2

Model 1 vs Model 2 Conv layer2

Figure 25: Comparison of weights between model 1 and model 2 found in the second Convolution layer
Model 1 vs Model 3 Conv layer2

Figure 26: Comparison of weights between model 1 and model 3 found in the second Convolution layer

Model 1 vs Model 4 Conv layer2

Figure 27: Comparison of weights between model 1 and model 4 found in the second Convolution layer
Model 1 vs Model 5 Conv layer2

Figure 28: Comparison of weights between model 1 and model 5 found in the second Convolution layer

Model 1 vs Model 6 Conv layer2

Figure 29: Comparison of weights between model 1 and model 6 found in the second Convolution layer
Model 1 vs Model 7 Conv layer2

Figure 30: Comparison of weights between model 1 and model 7 found in the second Convolution layer

Model 1 vs Model 8 Conv layer2

Figure 31: Comparison of weights between model 1 and model 8 found in the second Convolution layer
Model 1 vs Model 9 Conv layer 2

Figure 32: Comparison of weights between model 1 and model 9 found in the second Convolution layer

3.3.4 Histogram Of Weights In Dense Layers

Model 1 vs Model 2 Dense 1 and 2

Figure 33: Comparison of weights between model 1 and model 2 found in the Dense layers
Model 1 vs Model 3 Dense 1 and 2

Figure 34: Comparison of weights between model 1 and model 3,4,5 found in the Dense layers

Model 1 vs Model 4 Dense 1 and 2

Model 1 vs Model 5 Dense 1 and 2

Figure 34: Comparison of weights between model 1 and model 3,4,5 found in the Dense layers
Model 1 vs Model 6 Dense 1 and 2

Model 1 vs Model 7 Dense 1 and 2

Model 1 vs Model 8 Dense 1 and 2

Figure 35: Comparison of weights between model 1 and model 6,7,8 found in the Dense layers
Model 1 vs Model 9 Dense 1 and 2

We observe the perturbation scores being reflected in the histograms as well. This is observed more in the Dense layer visualizations as compared to the Convolutional layer filter weights.

For example, model 3 which has a high perturbation value shows observable deviation from model 1 in terms of weights. Further model 7 with low perturbation does not show much difference. We will consider model 3 to be an outlier scenario due its high perturbation and later in the results we use this model to test the interpretability mechanisms at extremes. We also note that Model 5 has the next to least perturbation with respect to the base model.
3.4 Results

In this section we display the results of the proposed framework by applying it over Saliency Maps and Grad-CAM. We first show visualizations over samples generated by both methods and compare the difference in these visualizations. We then show a graphical plot to visualise the model behavior and make deductions after observing the plot. We will then show data perturbation results applied over a test image and use Saliency maps over the images.

3.4.1 Saliency Map Results

Here we show the output images obtained after applying Saliency Maps. We show the result for one test image per class. The classes range from 0 to 9. Each test image is a handwritten digit ranging between 0 to 9 which corresponds to its target class.

In the figures shown below there are 4 columns for each test case and 10 test cases for each model.

- Column 1 represents the original input fed to the model being tested.
- Column 2 represents the interpretations given by the model using vanilla backpropagation
- Column 3 represents the interpretations given by the model using guided backpropagation
- Column 4 represents the interpretations given by the model using ReLU backpropagation
Saliency map applied over test images for Model 1:

Figure 37: Saliency maps results obtained over model 1 (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)

These are the visualizations obtained after running Saliency maps over the base model configurations of this experiment. We will use these images as the base to calculate the relative change in the outputs.
Saliency map applied over test images for Model 2:

Figure 38: Saliency maps results obtained over model 2. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)

These are the visualizations obtained after running Saliency maps over the Model 2 configurations of this experiment. It had a KL-Divergence value of 9.4 and element-wise difference of 0.59.
Saliency map applied over test images for Model 3:

![Saliency Maps](image)

Figure 39: Saliency maps results obtained over model 3 (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)

These are the visualizations obtained after running Saliency maps over the Model 3 configurations of this experiment. It had a KL-Divergence value of 11.25 and element-wise difference of 0.75.
Saliency map applied over test images for Model 4:

These are the visualizations obtained after running Saliency maps over the Model 4 configurations of this experiment. It had a KL-Divergence value of 0.1 and element-wise difference of 0.40.
Saliency map applied over test images for Model 5:

Figure 41: Saliency maps results obtained over model 5. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)

These are the visualizations obtained after running Saliency maps over the Model 5 configurations of this experiment. It had a KL-Divergence value of 3.55 and element-wise difference of 0.47.
Saliency map applied over test images for Model 6:

Figure 42: Saliency maps results obtained over model 6. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)

These are the visualizations obtained after running Saliency maps over the Model 6 configurations of this experiment. It had a KL-Divergence value of 4.25 and element-wise difference of 0.51.
Saliency map applied over test images for Model 7:

Figure 43: Saliency maps results obtained over model 7. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)

These are the visualizations are obtained after running Saliency maps over the Model 7 configurations of this experiment. It had a KL-Divergence value of 0.72 and element-wise difference of 0.38.
Saliency map applied over test images for Model 8:

Figure 44: Saliency maps results obtained over model 8. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)

These are the visualizations obtained after running Saliency maps over the Model 8 configurations of this experiment. It had a KL-Divergence value of 0.84 and element-wise difference of 0.44.
Saliency map applied over test images for Model 9:

Figure 45: Saliency maps results obtained over model 9. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)

These are the visualizations obtained after running Saliency maps over the Model 9 configurations of this experiment. It had a KL-Divergence value of 3.39 and element-wise difference of 0.48.

We now calculate and compare the change in the interpretation over all the models. We use MSE and SSIM to quantify the change in images with respect to interpretations obtained from Model 1.

The results for saliency maps are compiled into a table and shown below. Graphical representations are shown at the end of the results for Grad-CAM.
The result is calculated as an average over all test images. M.S.E is calculated as an average M.S.E over vanilla, guided and ReLU backpropagation results. SSIM is calculated as an average SSIM over the 3 backpropagation techniques.

Table 3: MSE and SSIM over Saliency maps outputs

<table>
<thead>
<tr>
<th></th>
<th>Model 2</th>
<th>Model 3</th>
<th>Model 4</th>
<th>Model 5</th>
<th>Model 6</th>
<th>Model 7</th>
<th>Model 8</th>
<th>Model 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean MSE of image</td>
<td>986.06</td>
<td>1452.7</td>
<td>1030.8</td>
<td>861.56</td>
<td>1056.7</td>
<td>1070.7</td>
<td>1039.8</td>
<td>959.86</td>
</tr>
<tr>
<td>Mean SSIM of image</td>
<td>0.3767</td>
<td>0.28</td>
<td>0.4067</td>
<td>0.4633</td>
<td>0.3667</td>
<td>0.39</td>
<td>0.383</td>
<td>0.4067</td>
</tr>
</tbody>
</table>

3.4.2 Grad-CAM Results

Here we show the output images obtained after applying Grad-CAM. We show the result for one test image per class. The classes range from 0 to 9. Each test image is a handwritten digit ranging between 0 to 9 which corresponds to its target class.

In the figures shown below there are 4 columns for each test case and 10 test cases for each model.

- Column 1 represents the original input fed to the model being tested.
- Column 2 represents the interpretations given by the model using vanilla backpropagation
- Column 3 represents the interpretations given by the model using guided backpropagation
- Column 4 represents the interpretations given by the model using ReLU backpropagation
Grad-CAM applied over test images for Model 1:

![Image](image-url)

Figure 46: Grad-CAM results obtained over model 1. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)

These are the visualizations obtained after running Grad-CAM over the base model configurations of this experiment. We will use these images as the base to calculate the relative change in the outputs.
Grad-CAM applied over test images for Model 2:

Figure 47: Grad-CAM results obtained over model 2. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)

These are the visualizations are obtained after running Grad-CAM over the Model 2 configurations of this experiment. It had a KL-Divergence value of 9.4 and element-wise difference of 0.59. The features remain constant over inputs.
Grad-CAM applied over test images for Model 3:

![Grad-CAM results](image)

Figure 48: Grad-CAM results obtained over model 3. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)

These are the visualizations are obtained after running Grad-CAM over the Model 3 configurations of this experiment. It had a KL-Divergence value of 11.25 and element-wise difference of 0.75.
Grad-CAM applied over test images for Model 4:

![Grad-CAM Results](image)

Figure 49: Grad-CAM results obtained over model 4. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)

These are the visualizations are obtained after running Grad-CAM over the Model 4 configurations of this experiment. It had a KL-Divergence value of 0.1 and element-wise difference of 0.40.
Grad-CAM applied over test images for Model 5:

Figure 50: Grad-CAM results obtained over model 5. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)

These are the visualizations obtained after running Grad-CAM over the Model 5 configurations of this experiment. It had a KL-Divergence value of 3.55 and element-wise difference of 0.47. The features remain constant over inputs.
Grad-CAM applied over test images for Model 6:

Figure 51: Grad-CAM results obtained over model 6. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)

These are the visualizations are obtained after running Grad-CAM over the Model 6 configurations of this experiment. It had a KL-Divergence value of 4.25 and element-wise difference of 0.51.
Grad-CAM applied over test images for Model 7:

Figure 52: Grad-CAM results obtained over model 7. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)

These are the visualizations obtained after running Grad-CAM over the Model 7 configurations of this experiment. It had a KL-Divergence value of 0.72 and element-wise difference of 0.38.
Figure 53: Grad-CAM results obtained over model 8. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)

These are the visualizations obtained after running Grad-CAM over the Model 8 configurations of this experiment. It had a KL-Divergence value of 0.84 and element-wise difference of 0.44.
Grad-CAM applied over test images for Model 9:

Figure 54: Grad-CAM results obtained over model 9. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)

These are the visualizations obtained after running Grad-CAM over the Model 9 configurations of this experiment. It had a KL-Divergence value of 3.39 and element-wise difference of 0.48.
We now calculate and compare the change in the interpretation over all the models.

The result is calculated as an average over all test images. M.S.E is calculated as an average M.S.E over vanilla, guided and ReLU backpropagation results. SSIM is calculated as an average SSIM over the 3 backpropagation techniques.

The results for Grad-CAM are compiled into a table are shown below.

Table 4: MSE and SSIM over Grad-CAM outputs

<table>
<thead>
<tr>
<th></th>
<th>Model 2</th>
<th>Model 3</th>
<th>Model 4</th>
<th>Model 5</th>
<th>Model 6</th>
<th>Model 7</th>
<th>Model 8</th>
<th>Model 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean MSE of image</td>
<td>1713.5</td>
<td>2131.4</td>
<td>2003.5</td>
<td>1536.1</td>
<td>2114</td>
<td>1982.9</td>
<td>1558.8</td>
<td>2208.5</td>
</tr>
<tr>
<td>Mean SSIM of image</td>
<td>0.4767</td>
<td>0.09</td>
<td>0.45</td>
<td>0.5767</td>
<td>0.4167</td>
<td>0.4567</td>
<td>0.5133</td>
<td>0.4433</td>
</tr>
</tbody>
</table>

3.4.3 Graphical Results

To visualize the relative performance of both methods we plot a graphical representation of the results obtained above:

The green dots represent the results of Saliency maps, and the red dots represent the results of Grad-CAM.

The graphs are plotted as the Model perturbation metric on the x-axis and the change in visualizations against the y-axis. We expect that for low perturbation, the change in visualization should also be low.
Here we plot The Model perturbation metrics with respect to the MSE of corresponding results obtained by the two algorithms. We observe that for some cases saliency map show less change in results for low perturbation.
We observe that both methods do not show any steady change in interpretation. Excluding the outlier cases, Saliency maps show less volatile nature toward the way that they interpret model behavior. The maximum range of MSE for Saliency map is 591.1 and that for Grad-CAM is 672.4. In the case of SSIM maximum difference for Saliency Maps is 0.183 while that for Grad-CAM is 0.486. In this experiment one of the major things we measure is the reaction to slight perturbation of model. And hence we test implementation invariance. The ideal behavior of a interpretation algorithm on two closely related model would not change. We observe this slightly in Saliency maps but not at all in Grad-CAM

Hence, we deduce that Grad-CAM shows a more sensitive behavior towards model perturbation and Saliency maps show some degree of implementation invariance.
We also note that the behavior of methods remains the same over both KL-Divergence and element-wise difference. Hence, we argue that KL-Divergence can be used as an effective metric to record model behavior.

3.4.4 Data Perturbation Results
In this section we show results obtained of conducting data perturbation experiments over Saliency maps. We observe that salient features are highlighted as edges of digits and this interpretation remains constant over perturbation.

We have tested Saliency Maps against slight perturbation of features and adding additional noisy features to the image. We also test it against repeating features present in the image however due to the form of training of using only a single digit in the sample we do not observe any meaningful result.

In the figures shown below there are 4 columns for each test case and 10 test cases for each model.

- Column 1 represents the original input fed to the model being tested.
- Column 2 represents the interpretations given by the model using vanilla backpropagation
- Column 3 represents the interpretations given by the model using guided backpropagation
- Column 4 represents the interpretations given by the model using ReLU backpropagation
We use Model 2 in our test model configurations to carry out data perturbation test. Here we also note clearly the reason guided backpropagation initially displays no interpretation when the wrong class is probed. This is because the backpropagation depends on the positive forward activations.

We also note that the end of the digits is always shown in the visualizations and are preserved over data perturbation. This matches with the expected behavior of preserving the nature of interpretations over slight perturbations.

Figure 57: Saliency map output against perturbed data. Original vs repeated pattern vs feature perturbation and additional features. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)
3.4 Deduction Of Model Behavior

We focus our experiment to test implementation invariance and sensitivity axioms mentioned before.

We see in the results that both methods do not show a linear change of interpretation. Saliency Maps show less volatile behavior and therefore shows some degree of Implementation invariance. Grad CAM shows highly volatile behavior towards model perturbation. In the case of data perturbation, we see sensitivity displayed by saliency maps towards important features present in the input image. Both methods perform bad in case of outliers (high perturbation). Further we note that even when there is low perturbation between models there is still a noticeable difference in the output interpretations. We are intrigued by this and would like to study the behavior further.
CHAPTER 4. RESULTS ON MATERIAL SCIENCE DATASET

In this chapter, we present saliency maps and Grad-CAM applied over a microstructure dataset from the material science domain. We also show data perturbation results that test Saliency maps over repeating and cyclic patterns. The results are used to map micro structures to macro-scale property of materials. The data is generated using a 2-component Cahn Hilliard equation and represents the microstructure of an organic solar cell. There are 10 target classes based on electric charge between the cells. Class 0 represents lowest charge whereas Class 9 has the highest electric charge. We use the following network architecture to obtain the results.

![Deep Convolutional Network architecture for classifying microstructures](image)

Figure 58: Deep Convolutional Network architecture for classifying microstructures

The network achieves high prediction accuracy of 95%. The confusion matrix for test dataset is shown below.

![Confusion matrix created using prediction results on microstructure dataset](image)

Figure 59: Confusion matrix created using prediction results on microstructure dataset
4.1 Material Science Dataset Results

The relationship between active layer microstructure is used to study the nature of an organic solar cell. The microstructures are categorized according to degree to electric charge. Higher class level suggests higher electric charge.

We use both Saliency Maps and Grad-CAM over the microstructure samples to highlight important regions in the image that are used for classification. We test the samples extensively as the classification of microstructures has only ever previously been dealt with tedious manual methods that take a lot of time. The aim was to built an automated system that makes correct predictions while making it easy for the user to trust these decisions with the visualizations provided to them.

In the figures shown below there are 4 columns for each test case and 10 test cases for each model.

- **Column 1** represents the original input fed to the model being tested.
- **Column 2** represents the interpretations given by the model using vanilla backpropagation
- **Column 3** represents the interpretations given by the model using guided backpropagation
- **Column 4** represents the interpretations given by the model using ReLU backpropagation
Saliency maps applied to microstructure images. There is 1 test image per class:

Class 0 sample
Class 1 sample
Class 2 sample
Class 3 sample
Class 4 sample
Class 5 sample
Class 6 sample
Class 7 sample
Class 8 sample
Class 9 sample

Figure 60: Saliency map results on microstructure dataset samples
Grad-CAM applied to microstructure images. There is 1 test image per class:

Class 0 sample

Class 1 sample

Class 2 sample

Class 3 sample

Class 4 sample

Class 5 sample

Class 6 sample

Class 7 sample

Class 8 sample

Class 9 sample

Figure 61: Grad-CAM results on microstructure dataset samples
We observe that the boundaries of surfaces are highlighted which are focused towards the interior rather than the corners of these cells. Expert opinion suggests that this is in fact a correct observation seen in solar cells.

4.1.1 Out Of Sample Generated Results

We test whether saliency maps can highlight repeating patterns. The patterns are created manually to study the effect of newly generated structures. As seen below the patterns are highlighted when guided back propagation and ReLU backpropagation is used. Further, the patterns found towards the center are highlighted more strongly than those found towards the edges of the image.

In the figures shown below there are 4 columns for each test case.

- **Column 1** represents the original input fed to the model being tested.
- **Column 2** represents the interpretations given by the model using vanilla backpropagation
- **Column 3** represents the interpretations given by the model using guided backpropagation
- **Column 4** represents the interpretations given by the model using ReLU backpropagation
Figure 62: Symmetrically generated data results obtained on Saliency maps. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)

Figure 63: Asymmetrically generated data results obtained on Saliency maps. (Column 1: input Column 2: vanilla backprop Column 3: guided backprop Column 4: ReLU backprop)
4.1.2 Architecture Perturbation Results

We use a variation of the base model which differs slightly in architecture. We then use the architecture perturbation metric described before to calculate the perturbation and record MSE and SSIM of explanations generated by Saliency Maps vs Guided Grad-CAM.

Model 1

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv2d_9 (Conv2D)</td>
<td>(None, 97, 97, 4)</td>
<td>104</td>
</tr>
<tr>
<td>max_pooling2d_9 (MaxPooling)</td>
<td>(None, 48, 48, 4)</td>
<td>0</td>
</tr>
<tr>
<td>batch_normalization_9</td>
<td>(Batch, None, 48, 48, 4)</td>
<td>16</td>
</tr>
<tr>
<td>conv2d_10</td>
<td>(None, 44, 44, 16)</td>
<td>1616</td>
</tr>
<tr>
<td>max_pooling2d_10 (MaxPooling)</td>
<td>(None, 22, 22, 16)</td>
<td>0</td>
</tr>
<tr>
<td>batch_normalization_10</td>
<td>(Batch, None, 22, 22, 16)</td>
<td>64</td>
</tr>
<tr>
<td>conv2d_11 (Conv2D)</td>
<td>(None, 20, 20, 64)</td>
<td>9200</td>
</tr>
<tr>
<td>max_pooling2d_11 (MaxPooling)</td>
<td>(None, 10, 10, 64)</td>
<td>0</td>
</tr>
<tr>
<td>batch_normalization_11</td>
<td>(Batch, None, 10, 10, 64)</td>
<td>256</td>
</tr>
<tr>
<td>conv2d_12 (Conv2D)</td>
<td>(None, 8, 8, 128)</td>
<td>73854</td>
</tr>
<tr>
<td>max_pooling2d_12 (MaxPooling)</td>
<td>(None, 4, 4, 128)</td>
<td>0</td>
</tr>
<tr>
<td>batch_normalization_12</td>
<td>(Batch, None, 4, 4, 128)</td>
<td>512</td>
</tr>
<tr>
<td>flatten_3 (Flatten)</td>
<td>(None, 2048)</td>
<td>0</td>
</tr>
<tr>
<td>dense_9 (Dense)</td>
<td>(None, 512)</td>
<td>1049088</td>
</tr>
<tr>
<td>dropout_7 (Dropout)</td>
<td>(None, 512)</td>
<td>0</td>
</tr>
<tr>
<td>dense_10 (Dense)</td>
<td>(None, 128)</td>
<td>65664</td>
</tr>
<tr>
<td>dropout_8 (Dropout)</td>
<td>(None, 128)</td>
<td>0</td>
</tr>
<tr>
<td>dense_11 (Dense)</td>
<td>(None, 32)</td>
<td>4128</td>
</tr>
<tr>
<td>dropout_9 (Dropout)</td>
<td>(None, 32)</td>
<td>0</td>
</tr>
<tr>
<td>dense_12 (Dense)</td>
<td>(None, 10)</td>
<td>330</td>
</tr>
</tbody>
</table>

Total params: 1,204,914
Trainable params: 1,204,910
Non-trainable params: 4,04

Model 2

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv2d_13 (Conv2D)</td>
<td>(None, 97, 97, 3)</td>
<td>209</td>
</tr>
<tr>
<td>max_pooling2d_13 (MaxPooling)</td>
<td>(None, 48, 48, 3)</td>
<td>0</td>
</tr>
<tr>
<td>batch_normalization_13</td>
<td>(Batch, None, 48, 48, 3)</td>
<td>32</td>
</tr>
<tr>
<td>conv2d_14 (Conv2D)</td>
<td>(None, 44, 44, 32)</td>
<td>616</td>
</tr>
<tr>
<td>max_pooling2d_14 (MaxPooling)</td>
<td>(None, 22, 22, 32)</td>
<td>0</td>
</tr>
<tr>
<td>batch_normalization_14</td>
<td>(Batch, None, 22, 22, 32)</td>
<td>128</td>
</tr>
<tr>
<td>conv2d_15 (Conv2D)</td>
<td>(None, 20, 20, 128)</td>
<td>56992</td>
</tr>
<tr>
<td>max_pooling2d_15 (MaxPooling)</td>
<td>(None, 10, 10, 128)</td>
<td>0</td>
</tr>
<tr>
<td>batch_normalization_15</td>
<td>(Batch, None, 10, 10, 128)</td>
<td>512</td>
</tr>
<tr>
<td>conv2d_16 (Conv2D)</td>
<td>(None, 8, 8, 512)</td>
<td>593036</td>
</tr>
<tr>
<td>max_pooling2d_16 (MaxPooling)</td>
<td>(None, 4, 4, 512)</td>
<td>0</td>
</tr>
<tr>
<td>batch_normalization_16</td>
<td>(Batch, None, 4, 4, 512)</td>
<td>2048</td>
</tr>
<tr>
<td>flatten_6 (Flatten)</td>
<td>(None, 2048)</td>
<td>0</td>
</tr>
<tr>
<td>dense_13 (Dense)</td>
<td>(None, 512)</td>
<td>4134816</td>
</tr>
<tr>
<td>dropout_10 (Dropout)</td>
<td>(None, 512)</td>
<td>0</td>
</tr>
<tr>
<td>dense_14 (Dense)</td>
<td>(None, 128)</td>
<td>49644</td>
</tr>
<tr>
<td>dropout_11 (Dropout)</td>
<td>(None, 128)</td>
<td>0</td>
</tr>
<tr>
<td>dense_15 (Dense)</td>
<td>(None, 32)</td>
<td>4128</td>
</tr>
<tr>
<td>dropout_12 (Dropout)</td>
<td>(None, 32)</td>
<td>0</td>
</tr>
<tr>
<td>dense_16 (Dense)</td>
<td>(None, 10)</td>
<td>330</td>
</tr>
</tbody>
</table>

Total params: 4,901,424
Trainable params: 4,900,266
Non-trainable params: 1,160

Figure 64: Test models generated by architecture perturbation

We show below the visualizations obtained after running Saliency maps over the Model 1 and Model 2 configurations of architecture perturbation experiment. It has an architecture perturbation of 3.997 which is calculated using the metric described in section 3.2.3.
Figure 65: Saliency map outputs against model 1 in pixel space over microstructure test sample

Figure 66: Saliency map outputs against model 2 in pixel space over microstructure test sample
Figure 67: Guided Grad-CAM outputs against model 1 in pixel space over microstructure test sample

Figure 68: Guided Grad-CAM outputs against model 2 in pixel space over microstructure test sample
These are the visualizations are obtained after running Guided Grad-CAM over the Model 1 and Model 2 configurations of architecture perturbation experiment.

We now calculate and compare the change in the interpretation over the 2 models using MSE and SSIM of corresponding images. We tabulate the results and present the table below. This research can be further extended to study architecture perturbation in more detail and this has been listed as future work.

Table 5: MSE and SSIM over Saliency maps vs Guided Grad-CAM outputs

<table>
<thead>
<tr>
<th></th>
<th>Model1 vs Model2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture Difference</td>
<td>3.97</td>
</tr>
<tr>
<td>Mean MSE (Saliency)</td>
<td>605.9</td>
</tr>
<tr>
<td>Mean MSE (Guided Grad-CAM)</td>
<td>2499.09</td>
</tr>
<tr>
<td>Mean SSIM (Saliency)</td>
<td>0.39</td>
</tr>
<tr>
<td>Mean (Guided Grad-CAM)</td>
<td>0.5687</td>
</tr>
</tbody>
</table>

The aim of the experiment was to test architecture perturbation and also test two pixel space visualizations such as Guided Grad-CAM and Saliency maps.

We observe that Guided Grad-CAM shows higher MSE and SSIM over the images. The perturbation is not high neither low as compared to the KL-Divergence values. This gives us further confidence in suggesting Saliency map is implementation invariant as we have now compared two-pixel space methods of interpretation.
CHAPTER 5. CONCLUSION

This chapter discusses the contributions of this thesis towards understanding interpretability algorithms. It discusses the novel work towards devising a framework for testing these algorithms and the results obtained.

5.1 Contribution

In this thesis, we present a contrast of the algorithms with respect to the different axioms of interpretability. We propose a new algorithm and show its results in comparison with guided back propagation. Our main work is to devise a framework that tests algorithms with respect to model perturbation and data perturbation. We propose a novel metric that measures difference between two models. We provide a histogram representation of models to argue the correctness of the metric. We present results of the framework over saliency maps and Grad-CAM and compare the two. We also apply these explanation mechanisms on a material science dataset to develop scientific understandings.

5.2 Discussion

We note that there is a trade-off between sensitivity and implementation invariance. We also probe for an in-depth study of interpretability mechanisms as there is limited understanding of the ways they perform. In fact, recent study shows that the result of local explanations can be replicated using completely randomized weights in the network [17].

There is a need to formally define interpretability as well. The discussion in [16] mentions that implementation invariance is a poorly constructed axiom. In this thesis we have realized the need to define implementation invariance more strongly and hence we not only try to call two models similar due to their classification decisions but also introduce quantitative metrics that record the deviation of internal elements of the models.
Finally, the proposed framework is not a way of pointing flaws in the methods tested but simply a solution to test intricacies of various interpretability methodologies that have been developed or are being developed by the machine learning community. The framework is not a complete study of algorithms and can easily be improved by adding more test scenarios. Further the model perturbation quantification can also be further improved by using different metrics to quantify the difference in weight distributions and architecture.
REFERENCES


[16] Discussion of Saliency maps: https://openreview.net/forum?id=r1Oen--RW


[18] MNIST dataset: https://keras.io/datasets/


