Faithful Saliency Maps: Explaining Neural Networks by Augmenting ”Competition for Pixels”

Jorma Peer Görns
Applied Mathematics

supervised by
Professor Himabindu LAKKARAJU

Cambridge, MA
April 2020
Abstract

For certain machine-learning models such as image classifiers, saliency methods promise to answer a crucial question: At the pixel level, where does the model look to classify a given image? If existing methods truthfully answer this question, they can bring some level of interpretability to an area of machine learning where it has been inexcusably absent: namely, to image-classifying neural networks, usually considered some of the most "black-box" classifiers. A multitude of different saliency methods has been developed over the last few years—recently, however, Adebayo et al. [1] revealed that many of them fail so-called "sanity checks": That is, these methods act as mere edge detectors of the input image, outputting the same convincing-looking saliency map completely independently of the model under investigation! Not only do they not illuminate the inner workings of the model at hand, but they may actually deceive the model investigator into believing that the model is working as it should. To fix these deceptive methods and save them from the trash pile of discarded research, Gupta and Arora [11] proposed an algorithm called competition for pixels. Yet as we uncovered, competition can be deceiving itself! This thesis makes three main contributions: (1) It examines competition for pixels, showing that the algorithm has serious issues in the few-class setting. (2) It proposes an augmentation of the competition algorithm designed to address these issues. (3) It experimentally verifies the effectiveness of said augmentation.
Acknowledgements

I first want to thank my thesis advisor, Professor Himabindu Lakkaraju of both SEAS and the Harvard Business School, for her phenomenal support. Hima encouraged me to not be satisfied with having discovered certain shortcomings of the competition algorithm, but to want to go further and find ways of augmenting it. Furthermore, she has been a constant source of insightful comments and advice—this thesis would not have been possible without her. I also want to extend my gratitude to the Associate Directors of Undergraduate Studies in Applied Mathematics, Dr. Margo Levine and Dr. Sarah Iams. Margo and Sarah have been very accommodating and have done an outstanding job of looking after their thesis students considering this year’s extraordinary circumstances caused by the current pandemic. Furthermore, I wish to express my gratitude to my high-school CS teacher, Dr. Detlef Kähler, who first introduced me to the exciting field of computer science. It should go almost without saying that I am also exceptionally grateful for Professor Finale Doshi-Velez of SEAS, who agreed to serve as my second reader—despite already having 14 (!) other theses to read. Finally, I am severely indebted to my family (especially to my mother, Martina Lange-Görns), to my friends (particularly to Philippe Noël, who convinced me to take CS 181), and to my girlfriend, Nicky Wojtania. Their unwavering love is a constant force in everything that I do; and their dedication to this thesis has at times rivaled my own. Thank you all!

- Jorma Görns
## Contents

1 Introduction ........................................... 7
   1.1 The Need for Interpretability ......................... 7
   1.2 Scope of This Thesis ................................ 8
   1.3 Our Contributions .................................... 8

2 Related Work ........................................... 9
   2.1 Some Intrinsically Interpretable Models ................. 10
      2.1.1 Sparse Linear Models .............................. 11
      2.1.2 Rule Lists and Rule Sets .......................... 12
   2.2 Some Post-Hoc Interpretability Methods .................. 12
      2.2.1 LIME ............................................. 13
      2.2.2 Saliency Methods .................................. 14
      2.2.3 The Power of Saliency Methods: An Example ........... 16
   2.3 Sanity Checks for Saliency Maps ........................ 17
      2.3.1 The Data Randomization Test ......................... 19
      2.3.2 The Model Randomization Test ....................... 19
   2.4 Competition for Pixels .................................. 20

3 Our Method ............................................ 22
   3.1 Our Motivation: The Completeness Property ............. 22
   3.2 Our Augmentation: Class-Internal Normalization ......... 23
## 4 Experimental Results

4.1 Where Competition Struggles: Many Versus *Few* Classes .......................... 25
   4.1.1 Motivation ...................................................................................... 25
   4.1.2 Experimental Setup .......................................................................... 26
   4.1.3 Results ............................................................................................. 26

4.2 Identifying Saliency Methods Left Behind by Vanilla Competition .............. 27
   4.2.1 Motivation ...................................................................................... 27
   4.2.2 Experimental Setup .......................................................................... 28
   4.2.3 Results ............................................................................................. 29

4.3 Evaluating the Effect of Different Types of Normalization ......................... 30
   4.3.1 Motivation ...................................................................................... 30
   4.3.2 Experimental Setup .......................................................................... 30
   4.3.3 Results ............................................................................................. 30

## 5 Conclusion & Future Work

32
List of Figures

2.1 The accuracy-interpretability tradeoff ........................................ 10
2.2 Rule lists and rule sets ............................................................. 12
2.3 LIME .......................................................... 13
2.4 Vanilla-gradient saliency maps are noisy ................................. 15
2.5 An example of the power of saliency methods ......................... 16
2.6 An example of the model randomization test ............................ 18
2.7 The competitive LRP algorithm ................................................ 20

4.1 Competition struggles in the few-class setting ......................... 27
4.2 Competition works as desired when there are many classes to be distinguished 27
4.3 CNN architecture used to classify MNIST data .......................... 28
4.4 Competition does not help an LRP variant pass the sanity checks in the
    few-class setting .................................................................... 29
4.5 Effects of different types of normalization on the competition algorithm in
    the sanity-check setting ......................................................... 30
4.6 Effects of different types of normalization on the competition algorithm in
    the non-sanity-check setting .................................................. 31
Chapter 1

Introduction

1.1 The Need for Interpretability

As machine-learning models are rapidly being deployed in high-stakes arenas of decision-making such as the justice system [4, 9], autonomous driving [10], and healthcare [20], the development of new algorithms has vastly outpaced the development of methods to inspect them. "Black-box" models—most notably, deep neural networks—may seem appealing due to their human-comparable (and sometimes even superhuman) performance on tasks such as image classification. However, accuracy alone is an insufficient metric for the quality of a model: In fact, understanding how a model arrives at a prediction is often just as important as the prediction itself. Without principled ways of examining their behavior, deploying these models in critical areas can be, at best, negligent and, at worst, deadly. More specifically, interpretable models have the following advantages over opaque ones: They can be audited for fairness; they can be more easily debugged; potential problems can be identified before deployment; new insights can be gathered from observing their way of making decisions; and they engender trust in the end user.
1.2 Scope of This Thesis

We have pointed out that to trust and to safely deploy complex machine-learning models in high-stakes arenas of decision-making, we must find ways of lifting the dense, black curtain that often obscures them: One such way of conferring a level of interpretability on deep neural networks are saliency maps. Saliency maps promise to answer the following important question:

At the pixel level, where does a model look to classify a given image?

Of course, knowing where a black-box classifier is looking to make a prediction does not by itself open the black box completely—crucially, it does not answer the question what the model sees in the pixels that it is paying special attention to—but it is extremely valuable information, regardless (more on this in section 2.2.3, "The Power of Saliency Methods: An Example"). This thesis focuses on saliency maps as a tool to explain the behavior of image-classifying neural networks. More specifically, it is concerned with the problem of generating faithful saliency maps, that is, saliency maps that really highlight the pixels in the input image that caused the model’s prediction—and do not act as mere edge detectors of the input image (as we will see in the "Related Work" chapter, not all saliency methods are faithful). Ultimately, this thesis deals with the Competition for Pixels algorithm, an algorithm designed to make faithful unfaithful saliency methods [11].

1.3 Our Contributions

This thesis makes three main contributions:

1. It examines competition for pixels, showing that the algorithm has serious issues with creating faithful saliency maps in what we term the few-class setting (section 4.2).

2. It proposes an augmentation of the competition algorithm designed to address these issues (chapter 3).

3. It experimentally verifies the effectiveness of said augmentation (section 4.3).
Chapter 2

Related Work

This chapter provides a short overview of the field of interpretability research, detailing some intrinsically interpretable models and post-hoc interpretability methods before turning towards saliency methods, sanity checks for saliency maps, and the competition algorithm.

To confer interpretability on machine-learning models and understand how they arrive at their predictions, two different approaches naturally suggest themselves: We can either (1) build models that are INTRINSICALLY interpretable or (2) explain otherwise opaque models in a POST-HOC fashion (for instance, through saliency maps). The intrinsic approach has the benefit of faithful explanations, that is, explanations that always and without a doubt reflect the model’s true decision process—the explanation is the model! However, this approach also has a crucial limitation: It appears\(^1\) that there exists some sort of tradeoff between model complexity and model performance (see figure 2.1 for a toy visualization of this accuracy-interpretability tradeoff). More complex, black-boxy models, such as deep neural networks, tend to outperform transparent models on many important tasks, including image classification. That is why both intrinsic and post-hoc explanations have generated a lot of research interest in recent years.

\(^1\)While there seems to be somewhat of a consensus on its existence in the machine-learning community, some interpretability researchers, most prominently perhaps Cynthia Rudin, do not believe in the existence of an accuracy-interpretability tradeoff [15]. This lack of an overall consensus may point towards the lack of a rigorous investigation of the supposed tradeoff.
Figure 2.1: An illustration of the accuracy-interpretability tradeoff, reproduced from Rudin and, in turn, adapted from DARPA [15, 6]. Every dot represents a type of model. Notice the general trend: The better a model performs, the worse it can be explained. Notice also that the human brain fits nicely into this tradeoff: While we exhibit outstanding learning performance (state-of-the-art image-classifying neural networks, for example, only beat us by a small margin), we often struggle to explain why we made a certain decision—and struggle even more to do so faithfully: Imagine seeing a picture of a dog. Classifying it as a dog is usually easy—but why exactly did you make that decision? Was it because of the ears, the tail, the snout, the stuck-out tongue? Was it because it looked just like the dog that lived next door to your childhood home? Or was it due to the distinct German-Shepherd-like pattern of its fur? All of these explanations make sense—but which one (or which combination of them) did you actually employ in your decision-making process? Rarely can we humans ever answer the why question with certainty.

2.1 Some Intrinsically Interpretable Models

As a result of the accuracy-interpretability tradeoff, intrinsically interpretable models are unfit for modeling complex data formats such as images or spoken language. However, they are especially attractive for modeling tabular data (such as patient health records): For this kind of data, their performance is often on a par with that of more complex models. Since they also provide faithful explanations (that is, explanations that truly reflect the model's decision-making process), they should be preferred over post-hoc explanations when dealing with tabular data. We now take a closer look at some intrinsically interpretable models.
2.1.1 Sparse Linear Models

Linear models, such as the omnipresent linear regression, roughly speaking take the fol-
lowing form, where \( y \) is the target scalar, \( x \) is the \( d \)-dimensional feature vector, \( \theta \) is the
\( d \)-dimensional feature-weight vector, \( \theta_0 \) is the bias, \( \epsilon \) is the noise term, and \( i = 1, \ldots, n \):

\[
y_i = \theta_0 + \theta_1 x_{i1} + \cdots + \theta_d x_{id} + \epsilon_i
\]

Thus, to predict the value associated with an unseen data point, we simply take the
inner product between its features and the feature-weight vector. That is why linear models
naturally lend themselves to an easy interpretation:

\textbf{Feature weight is feature importance.}

In other words: A one-unit increase in a feature \( j \) is associated with a \( \theta_j \) change in the
model’s output. Small feature weights correspond to features that have little influence on
the model’s output, while large positive and large negative feature weights correspond to
features that are important for the model’s decision-making process.

However: If the data is very high-dimensional—that is, if the data points have many
features—interpretability is seriously diminished. For concreteness, consider data with 1,000
dimensions. Given a data point like that, it is hard (if not impossible in an acceptable
amount of time) for a human to reason about what the model will predict. This is where
\textbf{sparse linear models} come to the rescue: Sparse linear models aim to set as many feature
weights as possible to 0, effectively decreasing the number of features that a human observer
has to reason about, while maintaining a small residual sum of squares. They achieve this
by adding an L1-norm regularization term (also called the ”lasso”, least absolute shrinkage
and selection operator) to the standard linear-regression minimization objective [19]. \( \lambda \) is a
tunable hyperparameter for this penalty term:

\[
\arg\min_{\theta \in \mathbb{R}^d} \sum_{i=1}^{n} \left( y_i - \sum_{j=1}^{d} x_{ij} \theta_j \right)^2 + \lambda \sum_{j=1}^{d} |\theta_j| 
\]
2.1.2 Rule Lists and Rule Sets

Decision rules are probably the most interpretable building blocks to construct a machine-learning model from. They take the form:

\[
\text{if CONDITION then DECISION}
\]

Decision rules can further be assembled\(^2\) into rule lists (see figure 2.2 on the right) by nesting multiple decision rules. If certain criteria are met—for instance, the list must be short and the conditions must be sufficiently simple—then rule lists present a textbook example of an intrinsically interpretable model. That is not to say, however, that they cannot be improved upon: Rule sets were designed to address the fundamental problem of rule lists, namely that for a rule to apply, it must first be verified that all of the conditions above that rule are false [12]. In a rule set, decision rules apply independently of other rules in the set.

![Figure 2.2: The left box contains a rule set while the right box contains a rule list learned from the same dataset. A rule in a rule list only applies if all of the conditions above it are false. A rule in a rule set applies independently of other rules in the set. Reproduced from Lakkaraju et al. [12]](image)

2.2 Some Post-Hoc Interpretability Methods

We now turn to post-hoc interpretability methods. In particular, we discuss a model-agnostic method (LIME), before detailing some popular saliency methods—plus, we show an impressive example of their power to help make black-box models safe for deployment.

\(^2\)An in-depth description of how to mine decision rules and learn decision lists and, respectively, decision sets lies beyond the scope of this thesis.
2.2.1 LIME

LIME (Local Interpretable Model-agnostic Explanations) was designed to explain the predictions of any type of classifier [14]. The idea behind it is simple: While the global decision boundaries of a classifier may be far too complex to approximate well with an interpretable model, locally (that is, for a given data point), such an approximation is usually possible with high fidelity. LIME thus proceeds as follows (for a visualization, see figure 2.3): First, it generates a set of perturbations of the data point whose classification is to be explained. Then, it lets the model under investigation classify all of these perturbations. Finally, LIME fits an interpretable model (such as a sparse linear model) to the perturbations. This model is then used to explain the classification of the original data point.

Figure 2.3: An instructive visualization of LIME: The bold red cross is the data point to be explained. Blue dots and other red crosses represent perturbations of this data point. The complex decision boundary of the model under investigation is also shown. Finally, the gray, dashed line is an interpretable classifier that was fit on the perturbations (in a weighted fashion—the closer the perturbation is to the data point of interest, the more the interpretable classifier cares about classifying it in accordance with the original model). The interpretable model can now be used to explain the classification of the data point of interest by the original classifier with high local fidelity. Reproduced from Ribeiro et al. [14]

Another local, model-agnostic interpretability method is SHAP (SHapley Additive exPlanations) [13]. Explaining SHAP, however, lies beyond the scope of this thesis.
2.2.2 Saliency Methods

As discussed previously, saliency methods seek to answer the following question: At the pixel level, where does a model look to classify a given image? While a large number of different saliency methods has been developed, many of them are gradient-based, that is, they follow the same basic principle:

Many work by computing the gradient of the model output with respect to an input.

For concreteness, the input may be an image of a handwritten 7 and the output (the model’s prediction) may be any one of the digits 0 to 9. Intuitively, the gradient tells us how much the model’s output will change if we jiggle the input features a little bit (the pixels of the image). Taking the gradient of class 7 with respect to the specified input then shows us which pixels of the image provide the model with strong evidence for class 7 (large positive gradients) and which pixels provide the model with strong evidence against class 7 (large negative gradients). Such saliency maps are often visualized in a manner resembling that of a heat map.

Vanilla Gradient

Let the model input $x \in \mathbb{R}^d$; let a specific class $c = 1, \ldots, C$; let the model under investigation be $M : \mathbb{R}^d \rightarrow \mathbb{R}^C$; and let the saliency map $S : \mathbb{R}^d \rightarrow \mathbb{R}^d$. Then, to calculate the vanilla gradient map (as described above) for a specific class, simply [16]:

$$S_{\text{vanilla}, c}(x) = \frac{\partial M_c(x)}{\partial x}$$

Unfortunately, however, creating a heat map from the ”pure” gradient does not produce good results: It yields very noisy saliency maps, making it hard for a model investigator to ascertain which parts of the image the model is actually paying special attention to (see

---

4While most of saliency-map research has focused on image data, the methods generalize nicely to other kinds of data, including text and audio.
This shortcoming provided the initial motivation for the multitude of other saliency methods that have been developed since the introduction of vanilla gradient.

Figure 2.4: Vanilla-gradient saliency maps for three different inputs. Notice the noisy, hard-to-interpret results. Reproduced from Simonyan et al. [16]

**Gradient ⊙ Input**

Probably the simplest way of combating noise in a vanilla-gradient saliency map is to calculate the Hadamard (elementwise) product of the map and the input. **Gradient ⊙ Input** does just that:

\[
S_{\text{grad} \odot \text{in},c}(x) = x \odot \frac{\partial M_c(x)}{\partial x}
\]

Gradient ⊙ input is also closely related to **Layerwise Relevance Propagation (LRP)**, a saliency method that we will see again in the "Experimental Results" chapter [5].

**Integrated Gradients**

**Integrated gradients** depends on a featureless baseline input \( \bar{x} \) [18]. Usually, \( \bar{x} \) is set to zero. Then, to compute the integrated-gradients saliency map for some input \( x \):

\[
S_{\text{integrated},c}(x) = (x - \bar{x}) \int_0^{\alpha} \frac{\partial M_c(\bar{x} + \alpha(x - \bar{x}))}{\partial x} \, d\alpha
\]
Note: The three saliency methods presented thus far represent only a small fraction of the total number of existing saliency methods.

2.2.3 The Power of Saliency Methods: An Example

Saliency methods can be a very helpful tool to debug machine-learning models. An especially impressive example of this capability is provided by Zech et al. [20]: Wondering why a pneumonia-detecting convolutional neural network performed well in the hospital whose data it was trained on—but far worse outside of it—the authors turned to saliency methods. This led them to discover that the network under investigation had "learned to detect a metal token that radiology technicians place on the patient"—instead of the actual presence or absence of pneumonia (see figure 2.5)! Since (in the hospital that the training data had originated from) a patient having such a token placed on them was correlated with them having pneumonia, the model had learned to detect this proxy, rather than the actual disease—only by using saliency methods could this previously overlooked data flaw be uncovered. Conversely, this example also indicates how saliency maps can increase trust in a good model: Given a patient with pneumonia, had the saliency maps highlighted the parts of their lung that contain the disease, then we could have been more certain than before that the model was making the right decisions for the right reasons.

Figure 2.5: These saliency maps show where a supposedly pneumonia-detecting CNN is looking to make classifications of "pneumonia" or "no pneumonia." (A) shows the average saliency map of many input images—a "global saliency map", so to say. Notice that the model is paying most of its attention to the corners of the images, that is, to the places in which the metal tokens would be placed! (B) and (C) are instances of patients who had metal tokens placed on them. Overlaying their images with the saliency maps generated for them reveals that the CNN does not pay attention to their lungs for making its classification, but rather to the metal tokens. Reproduced from Zech et al. [20]
2.3 Sanity Checks for Saliency Maps

As discussed earlier, there exists a large multitude of saliency methods. Interestingly, given the right circumstances (for instance, if the model under investigation uses certain activation functions), some of these methods may actually reduce to each other [3]. While some work has been done to unify existing methods through placing them on a rigorous theoretical foundation—most notably by Ancona et al.—thus far, the field is lacking just such a common framework for thinking about its methods. This may explain why their development has "often been guided by visual appeal", as Adebayo et al. note [1]. In the wake of this development:

Saliency methods have emerged that are visually appealing, but actually illuminate neither the model, nor the data that it has been trained on!

More specifically, these methods serve as mere edge detectors of input images [1]. What makes them dangerous, then, is their appeal to confirmation biases: Since these methods always produce saliency maps that make intuitive sense to the model’s investigator (that is, saliency maps that trace the input image), they will inevitably cover up all model issues. Thereby, they generate trust in models that can be seriously flawed (think back to the pneumonia example).

To identify such dangerous methods, Adebayo et al. introduced two so-called sanity checks for saliency maps [1]: the data-label randomization test and the model-parameter randomization test. Both tests evaluate the sensitivity of saliency methods to certain manipulations of the data or the model. In particular, if a method is insensitive to these manipulations (that is, if it produces very similar saliency maps after the manipulation has been applied), we say that it fails the corresponding sanity check and reject the method. We now go on to discuss both tests in more detail.
Figure 2.6: This is an example of the model randomization test: Given (A) a radiograph of a hand, the model’s task is to predict the age of the subject. The saliency map generated by the method guided-backpropagation SmoothGrad (B) generates trust in the model: It seems to highlight the hand’s growth plates, the best indicators of the age of the subject. However, this trust is clearly undeserved: After randomizing the model’s weights (C), guided-backpropagation SmoothGrad produces virtually the same saliency map as before! We therefore say that the saliency method fails the sanity check. Reproduced from Adebayo et al. [1]
2.3.1 The Data Randomization Test

Zhang et al. famously showed that neural networks can easily memorize arbitrarily labeled data [21]. Inspired by this finding, the data-label randomization test compares a method’s saliency maps generated for some model trained on some dataset to that method’s saliency maps for the same architecture trained on the same dataset—but with all the labels randomly permuted! Randomly permuting completely breaks the relationship between images and associated labels. As a result, the model can only memorize the training data labeling; and cannot perform better on the test data than a randomly guessing model. Put differently, the model will have acquired no real, generalizable knowledge—it will not have learned. If a method is not sensitive to randomly labeled training data, then it is not sensitive to the difference between a useful and a useless model. Consequently, such a method can provide no insight into the model it is supposed to illuminate.

2.3.2 The Model Randomization Test

The model-parameter randomization test compares a method’s saliency maps generated for a well-trained model to that method’s saliency maps after the model parameters (for instance, the weights and biases of a neural network) have been randomly reset. This effectivelly makes the model “unlearn” the patterns that it has learned during training. As a result, the model will be incapable of classifying the training data or any test data well. If a method is not sensitive to randomly set model parameters, then, once again, it is not sensitive to the difference between a useful and a useless model. Therefore, such a method can provide no insight into the model it is supposed to illuminate. For an example of the model randomization test in action\(^5\), see figure 2.6.

Finally, Adebayo et al. not only introduce the two sanity checks, but also show that many saliency methods fail them.

---

\(^5\)This example is also interesting because it shows saliency maps being applied not to a classification, but to a continuous prediction task (that is, age in years).
2.4 Competition for Pixels

In response to the findings of Adebayo et al., Gupta and Arora proposed an algorithm called competition for pixels [11]. Attractively, competition for pixels can be applied to any saliency method—and should ideally make any previously failing method then pass the sanity checks. Below is a formal description of competition for pixels applied to layerwise relevance propagation (LRP):

Let $LRP[j, \text{Element}]$ be the LRP score of Element when decomposing output node $j$.
Let $y$ be the index of the chosen output node.

Input: An image $\in \mathbb{R}^d$ and a neural network $S : \mathbb{R}^d \rightarrow \mathbb{R}^C$

for $\text{Element in Image}$ do
  Calculate $LRP[i, \text{Element}]$ for all output nodes $i$
  if $LRP[y, \text{Element}] > 0$ then
    if $LRP[y, \text{Element}] \geq LRP[i, \text{Element}]$ then
      Make the corresponding element of $H$ the $LRP[y, \text{Element}]$
    end
  else
    if $LRP[y, \text{Element}] \leq LRP[i, \text{Element}]$ then
      Make the corresponding element of $H$ the $LRP$ score of Element
    end
  end
end

Figure 2.7: A formal description of the competitive LRP algorithm. Reproduced from Gupta and Arora [11]

Less formally, the competition algorithm consists of two steps:

1. Compute the saliency maps not just for the class of interest, but rather for all possible classes.

2. Then, let these maps compete with each other!

Consider, as a concrete example, an MNIST classifier [8]. We may be interested in finding the competitive saliency map for digit 7. To do so, we compute the saliency maps for all digits from 0 to 9. Then, we say that 7 wins a pixel if—for that pixel—it has either the largest positive or largest negative gradient among all classes. If a pixel is not won,
we set it to zero in 7’s competitive saliency map. As a result, competitive saliency maps
tend to be sparser than the non-competitive maps that they are built from—especially when
there are many classes to compete with (and it is therefore hard to win any one pixel).
Chapter 3

Our Method

In this chapter, we detail both our augmentation of the "competition for pixels" algorithm described above—specifically, an augmentation through a certain type of normalization—and our motivation behind this augmentation, a property of some saliency methods called "completeness."

3.1 Our Motivation: The Completeness Property

Recall that saliency methods assign a (positive or negative) importance score to each pixel in the input. Recall also that saliency maps are computed with respect to some class of interest. Associated with that class is a logit value—in other words, the non-normalized probability of that class. Completeness, defined as follows, is generally considered a desirable property for any saliency method [11]:

A saliency method is COMPLETE if the sum of pixel scores equals the logit value.

Importantly, not all saliency methods are complete. Gupta and Arora believe, however, that for competition to work correctly, it must be applied to a complete saliency method: For a non-complete method, they argue that it would be "unclear how to compare scores
across labels, since this could end up being an “apples vs oranges” comparison due to potentially different scaling” [11]. Put differently, the authors worry that—given a non-complete method—a class may undeservedly win or lose pixels simply because its pixel scores are operating on a larger or smaller scale than those of other classes.

This worry seems appropriate: Different scales between classes would indeed make a mockery out of competition. However:

Even when a method is complete, classes may undeservedly win or lose pixels!

Crucially, pixel scores are often both positive and negative. Thus, even for a complete method, two classes with similar logit values could be operating on vastly different scales. Consider, for concreteness, a binary classifier. For a given input image, half of one class’s pixel scores might be 100 (these pixels provide evidence for that class) while the other half might be -100 (evidence against that class). Assuming the saliency method is complete, the class’s logit value would be 0. For the other class, half of all pixel scores might be 1 with the other half being -1. This class also has a logit value of 0; thus, since the model deems both classes equally likely, the image must contain equally as many pixels in favor of the first as in favor of the second class. However, if we applied competition to this problem, the first class would be much more likely to win any one pixel (since the scales between the classes differ by two orders of magnitude)—even though it does not deserve to do so and even though the method is complete!

3.2 Our Augmentation: Class-Internal Normalization

Instead of relying on completeness, we propose to NORMALIZE all class-specific saliency maps before they start competing with each other. This augmentation of competition for pixels has two advantages:
1. Normalization ensures that competition between classes is fair by bringing all class-specific pixel scores into the same range.

2. It allows for the application of the competition algorithm to saliency methods that are not complete.

We considered three different normalization methods: Let $S$ be an $m \times n$ saliency map and let $i \in \{1, ..., m\}$, $j \in \{1, ..., n\}$. Then, we can normalize $S$ to have zero mean and unit standard deviation as follows:

$$S_{ij} \leftarrow \frac{S_{ij} - \bar{S}}{\sigma(S)}$$

To normalize all values of $S$ to lie in the interval $[-1, 1]$, do:

$$S_{ij} \leftarrow 2 \times \frac{S_{ij} - S_{\min}}{S_{\max} - S_{\min}} - 1$$

And finally, to normalize all values of $S$ to lie in the interval $[0, 1]$:

$$S_{ij} \leftarrow \frac{S_{ij} - S_{\min}}{S_{\max} - S_{\min}}$$

As it turns out, normalizing all values of competing saliency maps to lie in the interval $[-1, 1]$—boxed above—is the best augmentation of competition for pixels (see next chapter).
Chapter 4

Experimental Results

In this chapter, we present our experimental evaluation of the competition algorithm and of our augmentation. Specifically, we conduct three experiments: (1) We compare the performance of the competition algorithm for classifiers with many versus classifiers with few classes. (2) We identify saliency methods that are failed by competition in the few-class setting. (3) And we evaluate which type of normalization is best suited to augment the competition algorithm.

4.1 Where Competition Struggles: Many Versus Few Classes

4.1.1 Motivation

Let $C$ be the total number of classes in the classification problem at hand and let input images be of dimensions $i$ and $j$. Then, since a pixel can only be won by a single class, the average amount of pixels won by a class is: $\frac{ij}{C}$. As a result, the more classes are competing, the sparser the competitive saliency maps should become (recall that a pixel in a specific class’s competitive saliency map is set to 0 if it is not won). Thus, we hypothesized, competition might struggle in the few-class setting: If a saliency method fails the sanity
checks—that is, if it still displays the structure of the input image after the model or the data have been randomized (instead of being blank or randomly noisy)—zeroing out just a few pixels would likely not be enough to make this structure disappear.

4.1.2 Experimental Setup

We examined the class-number-dependent effectiveness of competition for all saliency methods in the Python library `innvestigate` and a pretrained VGG-16 model with ImageNet weights [2, 7]. Specifically, we conducted the model-parameter randomization test by randomizing the model’s penultimate (fully-connected, pre-softmax) layer according to independent draws from a Uniform distribution over [-1, 1] (as suggested by Adebayo et al.). The maximum number of competing classes was 1000. By randomly sampling a certain number of competitor classes (say, 15), we sought to approximate the behavior of a model with VGG-16 architecture trained to distinguish only these 15 classes. We chose this approach because—even with transfer learning—retraining models to distinguish a multitude of different class numbers would have been forbiddingly resource-intensive.

4.1.3 Results

We found that, while competition works as desired for classification problems with many classes (see figure 4.2), it does not work consistently across saliency methods when a smaller number of classes is to be distinguished (see figure 4.1). That is to say that in the few-class setting\(^1\), competition only helps some previously failing saliency methods pass the sanity checks—but it seems to fail to help others. However, the few-class setting is arguably the most common setting in image classification. For instance, in medical imaging, we may wish to distinguish between benign tumors, malign tumors, and no tumors. Rarely, in practice, are we dealing with ImageNet-like, many-class problems. This speaks to the importance of an augmentation to make competition work reliably.

\(^1\)While the exact boundaries of the few-class setting depend on many parameters (most importantly on the saliency method under investigation), as a rule of thumb, we determined the few-class setting to be something like \(\leq 20\) classes.
Figure 4.1: This figure shows the diminished effectiveness of competition when there are few classes to be distinguished. Saliency method: gradient ∘ input; number of classes: 15. Model randomization test on VGG-16. Input image on the left (Scott Martin/Macaulay Library at the Cornell Lab of Ornithology (ML47337561)), gradient ∘ input map in the middle, competitive gradient ∘ input map on the right. Notice that the competitive map is far from being blank: It still clearly highlights the bird’s eye, the most salient part of the non-competitive map. This effect (non-blankness) is even more pronounced when the number of competing classes is even smaller.

Figure 4.2: This figure shows that competition works as desired when there are many classes to be distinguished. Saliency method: guided backpropagation; number of classes: 1000. Model randomization test on VGG-16. Input image on the left (Scott Martin/Macaulay Library at the Cornell Lab of Ornithology (ML47337561)), guided backpropagation map in the middle, competitive guided backpropagation map on the right. Notice that the competitive map is completely blank (as desired)!

4.2 Identifying Saliency Methods Left Behind by Vanilla Competition

4.2.1 Motivation

To determine which types of normalization are best suited to augment the competition algorithm, we had to first find out which saliency methods were left behind by competition—
that is, we had to determine which saliency methods failed the sanity checks in the few-class setting even when competition is applied to them. A good augmented algorithm must then help all of these methods pass the sanity checks.

4.2.2 Experimental Setup

We performed both sanity checks—the data-label randomization test (by training a CNN (see figure 4.3) to high accuracy on MNIST data\(^2\) with randomly permuted labels) and the model-parameter randomization test (by training the same CNN architecture to high accuracy on correctly labeled data, then randomizing the weights of the layer right before the output layer)—again on all saliency methods in the Python library investigate. Then, we could determine by simple visual inspection which saliency methods are failing the sanity checks: If the maps that a method generates still display the structure of the input images despite the data labels or the model parameters having been randomized, that method fails the corresponding sanity check. Next, we applied competition to the failing methods to observe if it would help them pass the sanity checks or not.

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv2d_150 (Conv2D)</td>
<td>(None, 24, 24, 32)</td>
<td>832</td>
</tr>
<tr>
<td>max_pooling2d_149 (MaxPool)</td>
<td>(None, 12, 12, 32)</td>
<td>0</td>
</tr>
<tr>
<td>conv2d_151 (Conv2D)</td>
<td>(None, 8, 8, 64)</td>
<td>51264</td>
</tr>
<tr>
<td>max_pooling2d_149 (MaxPool)</td>
<td>(None, 4, 4, 64)</td>
<td>0</td>
</tr>
<tr>
<td>flatten_75 ( Flatten)</td>
<td>(None, 1024)</td>
<td>0</td>
</tr>
<tr>
<td>dense_149 ( Dense)</td>
<td>(None, 1024)</td>
<td>1049600</td>
</tr>
<tr>
<td>dense_150 ( Dense)</td>
<td>(None, 10)</td>
<td>10250</td>
</tr>
</tbody>
</table>

Total params: 1,111,946
Trainable params: 1,111,946
Non-trainable params: 0

Figure 4.3: CNN architecture used to classify MNIST data. This is the same architecture as employed by Adebayo et al.

\(^2\)Since MNIST data consists of 10 different classes, we are in the few-class setting \([8]\).
4.2.3 Results

Indeed, verifying our results from the previous experiment, many methods were not helped by competition to pass the sanity checks for MNIST data (see, as a visual example, figure 4.4). The complete results are depicted in table 4.1. Note that layerwise relevance propagation works using specific propagation rules [5]—depending on the propagation rule being used, it shows vastly different responses to sanity checks and competition and generates vastly different saliency maps. The name of the propagation rule used corresponds to its name in the investigate library.

Figure 4.4: This specific type of LRP ("lrp.z") fails the sanity checks: The 7 can still clearly be made out in the center image (the LRP map). Applying competition makes the map sparser—but the 7 remains visible.

Table 4.1: All saliency methods in this table fail the sanity checks. When there are many classes, all of them are helped to pass these checks through competition—but in the few-class setting, competition only helps the methods in the right column pass the checks while it fails to help the methods in the left column. The goal of normalization, therefore, is to make the left methods pass the checks in the few-class setting.
4.3 Evaluating the Effect of Different Types ofNormalization

4.3.1 Motivation

Previously, we proposed three possible normalization methods to augment the competition algorithm. In this experiment, we evaluate these three methods to see which ones present the best augmentation of competition, that is, help the methods in the left column of table 4.1 pass the sanity checks. We then evaluate whether or not these normalizations, in combination with competition, still generate good saliency maps in the non-sanity check case (that is, in the normal use case).

4.3.2 Experimental Setup

The experimental setup is the exact same as in the experiment above—with one deciding difference: We now normalize the saliency maps that a given method produces (according to one of the three normalization methods) before we let them compete. In addition to this, we conduct an experiment for the non-sanity check case by generating normalized, competitive saliency maps for a well-trained CNN (no parameters or data labels randomized, architecture as above).

4.3.3 Results

We show representative results in figure 4.5 and figure 4.6 below. They are analogous for all methods in the left column of table 4.1.

![Figure 4.5: Effects of different types of normalization on the competition algorithm in the SANITY-CHECK SETTING (see below for detailed description)](image)
As shown in figure 4.5, LRP with propagation rule "sequential_preset_b" fails the sanity checks: Even though the data labels have been randomized in this experiment, we can still clearly make out the digit 7 in the second image from the left. Adding competition does not change this fact. Consider now the three different normalization methods: While normalizing all saliency maps to have zero mean and unit standard deviation before letting them compete increases sparsity of the resulting competitive map, we can still make out the 7—therefore, this normalization method is not suited to augment the competition algorithm. Normalizing all values to lie in the interval \([-1, 1]\), however, makes it impossible to perceive the 7 anymore—therefore, this normalization method is well-suited to augment competition. Finally, normalizing all values to lie in the interval \([0, 1]\) creates a super-sparse competitive map—however, as shown in figure 4.6, it also turns saliency maps blank in the non-sanity-check setting. Therefore, it is not suited to augment competition.

![Figure 4.6: Effects of different types of normalization on the competition algorithm in the non-sanity-check setting](image)

Figure 4.6 shows saliency maps for a well-trained CNN (that is, it depicts the non-sanity-check setting). The second image from the left shows the output of LRP (LRP was chosen to be consistent with figure 4.5, yet the results presented here hold for all examined methods). Notice that normalizing to the interval \([-1, 1]\) and then applying competition produces an almost identical saliency map to "pure" LRP—therefore, normalizing to \([-1, 1]\) preserves the quality of the underlying saliency maps. Normalizing to \([0, 1]\) and applying competition, however, results in a virtually blank competitive saliency map—therefore, this type of normalization is not suited to augment competition.
Chapter 5

Conclusion & Future Work

We have seen that the competition algorithm can work inconsistently in the few-class setting. Normalizing all competing saliency maps to the interval \([-1, 1]\) augments competition to work more consistently. This effect could likely be strengthened even more if we also chose to ignore negative pixel scores, that is, evidence against a class (since this would create even sparser saliency maps). Importantly, \([-1, 1]\)-normalization preserves the quality of the underlying saliency maps in the non-sanity-check case. Regarding the experiments conducted, one word of caution is in order: Note that the methods in the left column of table 4.1 are not helped by few-class competition for the specific model architecture and data used in the corresponding experiment. It may be premature to conclude that these methods are not helped by competition in general.

In addition to this, two questions naturally suggest themselves: (1) Why may competition perform inconsistently in the few-class setting? It is easy to gain an intuitive answer: When there are many classes, there are more competitors for pixels. As a result, it becomes harder to win any one pixel. This leads to overall sparse or "blank" maps. In the few-class setting, however, it is much easier to win pixels. A saliency method that still displays the input even though the model or the data has been randomized may lose a few pixels here and there—but this may not be enough to turn its competitive saliency map blank. (2)
Why does normalizing help; and why do some types of normalization help more than others? This is a promising avenue for further research. One possible explanation may lie in completeness; another possible factor could be the positivity of saliency maps. Both may provide part of the answer, but they do not seem to give the full picture just yet.
Bibliography


