Developing Provably Robust Explanation Methods for Image Classifiers

André Shannon

Mentored by Dr. Douglas Szajda

Honors Thesis

Submitted to:

Computer Science Department
University of Richmond
Richmond, VA

April 29, 2022

1 Introduction

Machine learning systems are being developed to solve all sorts of problems. Some of the applications include determining whether to give someone a loan, the likelihood of recidivism, which areas in a community are more likely to have crime, whether computer code is malicious or benign, or whether an autonomous vehicle is approaching a stop sign. All these applications can have a significant impact on people's lives, yet we don't fully understand the "reasoning" behind these models. Users of such models need to know whether they contain biases and whether they are basing decisions on reasonable principles. These needs have given rise to *explanation methods*, which attempt to expose the reasoning behind specific machine learning model decisions. However, through our research, we have found that explanation methods are not as reliable as we would like them to be, hence our attempt to create provably robust explanation methods.

2 Background & Motivation

2.1 Machine Learning

At a high level, the basic idea of machine learning is to have a model train on a large dataset (the *training data*), in order to generalize from that data to new instances of data from the same (perhaps unknown) distribution. The hope is for the model to learn how to accomplish a task on future inputs, given several successfully completed examples of the task. Machine learning models can have billions of parameters, on which the specific behavior of the model depends. The process of "learning" involves determining the parameter values that optimize the performance of the model on the training data set. Once trained, the parameters are fixed and the hope is that the model has captured the essential "knowledge" that will enable it to do the same task on data it has not yet seen. This process of learning from a training data set is known as *supervised* learning[1].

Though there are several different types of machine learning models, we focus here on supervised machine learning *classifiers*. A classifier is a model that takes in input and tries to determine which class it belongs to. For example, an image classifier takes an image as input and outputs a *class* for the image (e.g., is this the image of a dog, or a cat, or a school bus). Of course, a computer does not process images in the same way it is believed that humans do. Instead, image classification models operate on the thousands of pixels – intensity values for red, green, and blue – that describe the appearance of the image at specific points. Computers store an image by saving pixel values, so when we give an image to a classifier as input, any processing is essentially mathematical operations performed on pixel values.

Like machine learning models in general, a classifier can be supervised or unsupervised. An unsupervised image classifier, might be given a collection of images with the hope that it will "cluster" similar images, effectively creating its own classes. Supervised machine learning models are trained on labeled data, so that all training examples have a corresponding label specifying the class to which it belongs. The process of training involves adjusting parameters

[1]Models that attempt to extract properties from data without any apriori knowledge of what is correct, fall under the umbrella of *unsupervised learning*.

so that the greatest possible proportion of training examples are classified correctly. (We note that it generally is not possible to correctly classify all training examples correctly.) Goodness of a specific set of parameters is measured using a task specific *loss function*, which quantifies the difference between actual and ideal model performance.

To a human, the (potentially billions of) final learned parameter values can appear somewhat arbitrary. Given a particular parameter selected from a large model, an expert on that model might not be able to describe its significance. Supervised machine learning classifiers, especially the large models used in real-world applications, can be extremely successful in classifying inputs, yet a user of that model may have no idea why the model classifies a specific input in the way that it does. This is problematic for many reasons, especially considering some of the applications of machine learning models. The need to understand how a model "reasons", and in particular how the model can be intentionally or unintentionally fooled, is highly important and gives rise to *explanation methods*.

2.2 Robustness of Explanation Methods

We will describe some specific explanation methods in more detail below, but essentially we want them to tell us what features of the input are significant factors for the classification of that input. If we are working with an image classifier, what pixels are most important in a classification?

During the summer of 2021, we experimented with applying an explanation method to automated Voice Processing Systems (we worked with DeepSpeech2, but other examples include Siri or Alexa) and wanted to know which frequencies were most important in a classification of a character. However, things turned out to be more complicated than we had anticipated, a development that we should have foreseen, for reasons discussed later. What our experiments showed was that for a given character in the same word, say "c" in "cat", spoken by different people, the explanations varied. We were hoping to find certain frequencies that would almost always be associated with certain characters, but what we found was that the explanations varied wildly even for just one character.

We decided to reconsider this issue in the context of image classification and the associated explanation methods, as examples in that space are more easily manipulated and there is a richer recent history. What our group and others found is that explanations can vary significantly. For example, given a picture of a dog and a corresponding explanation, we could slightly tweak the image by changing several pixel values and end up with a completely different explanation. The explanation methods lacked *robustness*, they could not produce similar explanations for images that were almost identical. Thus we shifted our focus to developing provably robust explanation methods for image classifiers.

3 Related Work

In this first subsection, we consider two explanation methods that serve as a basis for our study.

3.1 LIME (Local Interpretable Model-agnostic Explanations)

The LIME explanation method [6] seeks to explain why a classifier, f, classifies an input, x, as it does. It goes about this by establishing an "interpretable space" and a mapping from the input space of f (which we will call the raw feature space) to the interpretable space. This is not always necessary – there are contexts in which the features of input vectors are well understood, in which case the interpretable space and raw feature space may be identical. For other situations, however, knowledge of which features are considered "important" to the model provides little insight into the "reasoning" behind a classification decision. The interpretable space is thus context dependent. Nor is it unique, as there may be many appropriate interpretable spaces. Regardless, the interpretable space is chosen by whomever is implementing the method. The primary requirement is that it needs to be interpretable in the sense that each dimension represents a feature of the input data that a human can reasonably understand.

The fundamental insight driving LIME is the notion that a linear model can be used to approximate local behaviors of the model in question. The value in this lies in the transparency of linear models. A linear model can be written as $g(x) = \beta^T x + \epsilon$ where x is a vector with a variable for every dimension in the interpretable space, β is a vector with a coefficient corresponding to every variable in x, and ϵ is a constant. We say linear models are transparent because given a feature (dimension) from the input space, in this case the interpretable space, we can see how much of an impact it will have on the model by looking at its corresponding coefficient. The magnitude of the coefficients of the linear model can, in theory, tell us which features in the interpretable space are most important in classifying x; the larger the coefficient magnitude corresponding to a given feature, the more important the feature. Since features are interpretable, a human can understand what those features represent and thus have a good idea of what the model is actually basing its decision on in classifying x.

Now that we have the big picture of the LIME explanation method, let us explore it in more detail. Let us assume a classifier $f : \mathbb{R}^d \to \mathbb{R}$ that accepts d dimensional inputs and maps a given input x to a probability that it is a member of a specific class. Let us further assume that the dimension of the interpretable space is d', which may be different from the dimension, d, of the raw feature space. The linear model we want to create, $g : \{0,1\}^{d'} \to \mathbb{R}$, will map elements of the interpretable space to probabilities that they are members of the same specific class. Elements of the interpretable space will be binary vectors of length d' since each dimension represents a feature and an element will indicate whether the data exhibits that feature, with a 1, or does not, with a 0. For example, $x' = \{1,0,0,1,0,1,0,0\} \in \{0,1\}^8$ exhibits the first, fourth, and sixth feature in this eight dimensional interpretable space, but not the other features. For ease, we will denote the domain of f as D_R (raw feature space) and the domain of g as D_I (interpretable feature space). Let us also mathematically define the mapping, $\gamma : D_R \to D_I$, from the raw feature space to the interpretable space, so given $x \in D_R$, we can obtain $\gamma(x) = x' \in D_I$. We also need γ to be at least partially invertible so that given $x' \in D_I$, we can obtain $\gamma^{-1}(x') = x \in D_R$ where $\gamma^{-1} : D_I \to D_R$ can be a pseudo-inverse.

To train the linear model in the interpretable space as an approximatation of the behavior of f near x, we first find $\gamma(x) = x'$ and sample around $x' \in D_I$ to obtain a set, Z, of perturbed samples. We sample around or "perturb" x' by choosing a random number of "bits" (if we view x' as a binary vector) and a random collection of that number of bits to set to zero. This is essentially creating data points that do not have certain features that the original data point did. The number of perturbations is a hyper-parameter for the method and the LIME authors used $|Z| = 500$ or 1000. The set Z will be our inputs to the linear classifier that we want to train, so now we need to obtain the corresponding

probabilities that the elements of Z are members of a specific class. To accomplish this, for each $z' \in Z$, we find $\gamma^{-1}(z') = z$ and compute $f(z)$ to get the probability. Given the inputs and corresponding outputs, we can train our linear model using a loss function similar to this:

$$L(f, g) = \sum_{z, z' \in Z} (f(z) - g(z'))^2.$$

This is not quite satisfactory, however. Since we want to approximate the behavior of f around x, we want the training data to be relatively "local" to x. When we compute $\gamma^{-1}(z') = z$, z might be far from x even if we changed relatively few bits of x' to obtain z'. Thus we want to add weights to the data that we train g on so that data points with z closer to x are weighted or counted more heavily than points that are not as close to x. For this, we need to define a distance metric, D, in the raw feature space, D_R. Some examples include the L^2 distance for images or cosine similarity for text. We define our weight function as an exponential kernel, $\pi_x(z) = exp(-D(x, z)^2/(\sigma^2))$, where σ is another hyper-parameter. We can imagine the exponential kernel as a bell shaped curve centered on x that weighs values based on proximity to x. The hyper-parameter σ determines how "wide" the bell is. The addition of the weight function results in the final loss function:

$$L(f, g, \pi_x) = \sum_{z, z' \in Z} \pi_x(z)(f(z) - g(z'))^2.$$

3.2 LEMNA (Local Explanation Method using Nonlinear Approximation)

LEMNA [2] is another explanation method that builds upon LIME. It is designed to handle feature dependency using a technique called fused lasso. In addition, LEMNA incorporates a non-linear local approximation. LEMNA differs from LIME in that it assumes that the raw feature space is interpretable, so linear models are trained in the raw feature space and not a separate interpretable space. As a reminder, a linear model looks like $g(x) = \beta^T x + \epsilon$ where x is an input vector from the raw feature space, β is the vector of the coefficients of the model, and ϵ is some constant. We will use y_i to denote the true label of a given input sample, x_i. With this notation, the loss function for training $g(x)$ would look like

$$L(g(x), y) = \sum_{i=1}^{N} ||\beta x_i + \epsilon - y_i||,$$

where N is the number of samples the model is being trained on and $|| \dots ||$ is the L2-norm.

Fused lasso is a term that can be added to the loss function that will penalize adjacent features for being similar but not the same. Given the loss function, fused lasso will add another term to be minimized:

$$\sum_{j=2}^{M} ||\beta_j - \beta_{j-1}|| \leq S,$$

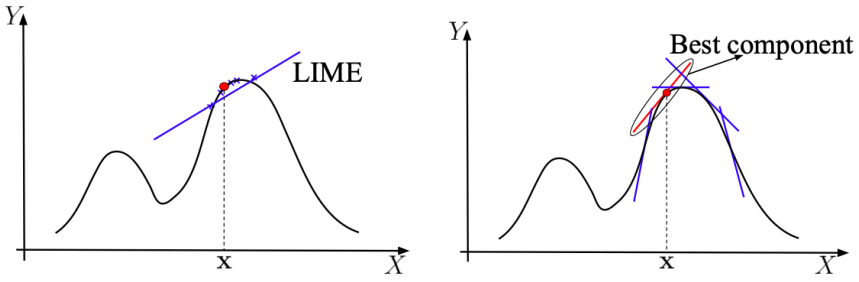
where M is the dimension of the raw feature space and the threshold, S, is a hyper-parameter for which differences bigger than S will be ignored in the loss function. This will have the effect of "grouping" clusters of adjacent coeffi-

cients so that they have very similar values. The intuition is to get our linear model to look something like

$$g(x) = \beta_1(x_1 + x_2 + x_3) + \beta_2 x_4 + \beta_3(x_5 + x_6) + \cdots + \beta_k x_M + \epsilon,$$

where adjacent features have the same coefficients so they can be treated as a group of features with one coefficient. This should catch dependencies between features and allow the explanation to indicate groups of features to be important.

Now we have seen how fused lasso is used to capture feature dependencies, lets look at using a mixture regression model for approximating nonlinear local boundaries. A mixture regression model is different from a linear model in that it clusters the data points and creates a linear model for each cluster of points. Each component has a weight assigned to it; the cluster containing x likely getting the biggest weight. This is illustrated in Figure 1.



(a) Linear regression model. **(b) Mixture regression model.**

Figure 1: Linear regression model vs mixture regression model [2]

The prediction of the mixture model will be

$$g(x) = \sum_{k=1}^{K} \pi_k(\beta_k x + \epsilon_k)$$

where K is a hyper-parameter for the number of clusters and linear models the mixture regression model will use and π_k is a weight function that will apply a weight to the prediction of each corresponding linear model. Integrating this into the loss function with the fused lasso we developed earlier, we get

$$L(g(x), y) = \sum_{i=1}^{N} ||g(x_i) - y_i||,$$
$$\text{subject to} \sum_{j=2}^{M} ||\beta_{kj} - \beta_{k(j-1)}|| \leq S, k = 1, \ldots, K,$$

where $g(x)$ is now the mixture regression model and β_{kj} indicates the j^{th} parameter for the k^{th} linear model.

3.3 Adversarial Samples

Adversarial samples are inputs specifically crafted by an adversary so that a classifier incorrectly classifies them. The trick for the adversary is to modify the input so slightly that a human does not notice any changes. For example, an adversary could take an image of a cat, make small changes to certain pixel values, and an image classifier might predict that it is a school bus while a human would have no trouble recognizing it as a cat, nor would they notice any change to the image. Figure 2 gives one such example of adding slight changes to pixels in an image of a stop sign and it being completely miss-classified. These attacks can be targeted or untargeted. An untargeted attack just attempts to fool the classifier into predicting that the adversarial sample is some other class than its original class while a targeted attack attempts to fool the classifier into predicting an adversary-chosen class.

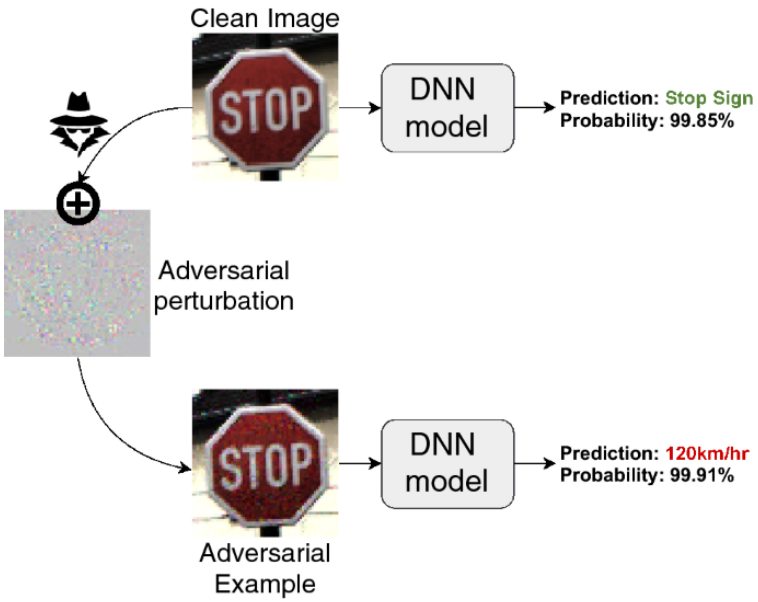


Figure 2: An example of almost imperceptible noise being added to an image and it being miss-classified [3]

Here, it is useful to explain the concept of a decision boundary. Consider dividing the input space into regions, with each region containing inputs that are classified the same by our classifier. This might look something like Figure 3 where there are three output classes the input space is two-dimensional. The decision boundaries for the model are the boundaries of these regions. Under the assumption that the color of a data point indicates the correct classification, we see in Figure 3 that not all the samples are correctly classified, which is typical. Note also that decision boundaries can be complicated and irregular.

To create an adversarial sample, we want it to be very close to the original sample within the input space so that a human would not notice the difference between the original and adversarial sample. We also want it to be miss-classified by the classifier, so we are going to perturb the input just enough so that we "move" it across a decision boundary. For a targeted attack, we want to not only move the input to another region, but to a region corresponding to the intended (incorrect) output class. One way of doing this, if the adversary has white box access to the classifier (has access to all of its parameters), is to consider the gradient of the output with respect to the inputs. If certain inputs

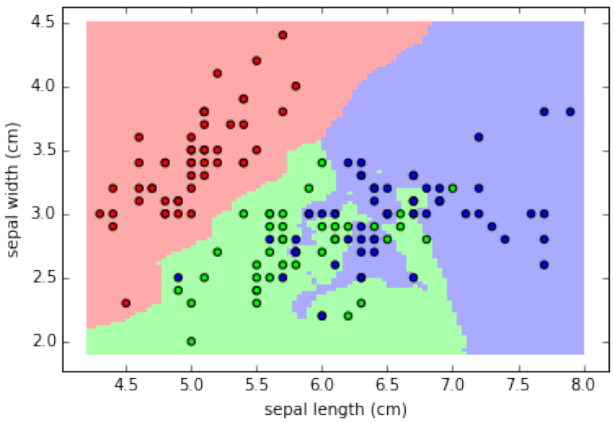


Figure 3: A simple decision boundary [8]

have big gradients, then slightly changing that input will have big effects on the output. If an adversary wants to create a targeted attack, they positively shift inputs that have big positive gradients and negatively shift inputs that have large negative gradients when considering their target class as the desired output of the model. This will yield a big change in the output of the model with minimal change to the input, making it almost imperceptible to humans.

The fact that there has been a lot of success with these attacks on top performing models tells us that adversaries are able to find the decision boundaries of their target classes very close to the original sample in almost all cases. This implies that the decision space of a classifier is not neatly separated into distinct classes where all the instances of one class are grouped together, but rather an extremely complex jumble of classes where you can reach almost any other class within a very small distance from any given point as in Figure 4. Note that Figure 4 has only two input dimensions, but a real world classifier is likely to have many more, leading to a even more complex decision boundary that spans more dimensions than we can visualize.

Figure 4: A more complex decision boundary [7]

If we think back to the explanation methods, they are trying to give us a local approximation of the model, and thus provide insight into the decision boundary near the input of interest. This might make sense with very distinct clear cut boundaries between one class and another, but from the success of adversarial samples, which gives us a far more complex picture of the decision boundaries, what are these models telling us? They are giving us coefficients of the features that the classification is most sensitive to for that instance. The higher magnitude coefficients are telling us which features, when slightly changed, are most likely to cause that specific sample to cross a decision boundary and cause a miss-classification. In other words, the explanations are telling us which dimensions contain the closest decision boundary(s). When considering the complexity of the decision boundary, features explaining one instance can be completely different from the features explaining a different instance of the same class. When we think back to what we wanted from the explanation model, an indication of which features were important in classifying a particular sample as a class, we get that, but it is not as telling as we anticipated it to be. Instead of a single "catalog" entry of the features that cause classification to be one specific class, there are a multitude of "catalog" entries at the least. This remains useful, but is certainly less satisfactory than a more general understanding.

Among the reasons that explanation methods are useful is that they can help detect adversarial samples. The idea is that adversarial samples yield poor or wrong explanations which humans can examine and tell apart from their benign counterpart. The first two rows in Figure 5 illustrate the different explanations or interpretations an explanation method gives for benign and adversarial samples. With an image, we expect the explanation method to emphasize the pixels that a human would be looking at most to determine its contents. If, instead, the pixels the explanation emphasizes are more random and unrelated to the main object in the image, this gives us a clue that the model is basing its prediction on something it should not be and that it might be an adversarial sample.

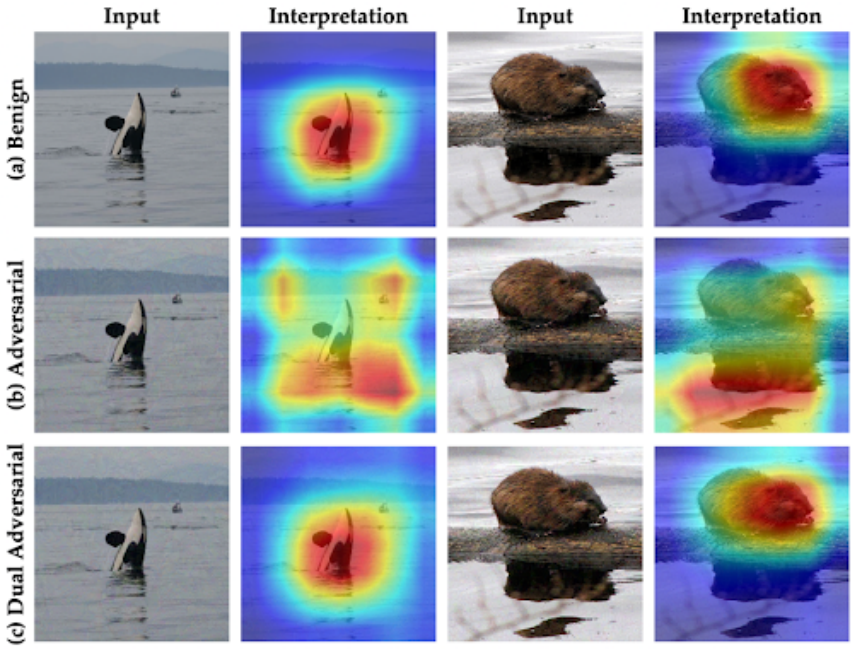


Figure 5: Sample (a) benign, (b) regular adversarial, and (c) dual adversarial inputs and interpretations on ResNet (classifier) and CAM (interpreter) [9].

Zhang et al. [9] show that there are attacks that can simultaneously fool both the classifier *and* the explanation model. An example is shown in the bottom row of Figure 5 where the adversarial sample is crafted so that it is miss-classified by the classifier and the explanation is very similar to the explanation of its benign counterpart. Zhang et al. achieve this using the gradient of the output with respect to the inputs as described earlier, but they add a term to the loss function used to calculate the gradients. In addition to quantifying the difference between what their adversarial sample classifies as and the target class they want it to classify as, they add a term to quantify the difference between the explanation of the benign sample and the explanation of their sample. Now large gradients correspond to inputs that can be shifted and significantly change the output of the classifier towards the target class and have a similar explanation as the benign sample. Perturbing those inputs, one can craft adversarial samples that fool both the classifier and explanation method.

The success of Zhang et al. in creating adversarial samples that fool the classifier and the explanation method is yet another reason for robust explanation methods. Knowing that one can change the explanation with very slight changes to the input shows the lack of robustness in explanations. The explanation of an input that has been imperceptibly changed should not significantly differ from the explanation of the original input.

3.4 Certified Adversarial Robustness via Randomized Smoothing

In their paper, Cohen et al. [1] build upon the work of Lecuyer et al. [4] and Li et al. [5] who first proved robustness guarantees using the "randomized smoothing" technique. Using this technique, they were able to define a formula to calculate a ball-like space around a point you want to certify for which any sample in that space will be classified the same as the certified point. Cohen et al. then prove a tight bound on the radius of the ball-like space.

The randomized smoothing technique consists of taking a classifier f which classifies well under Gaussian noise and creating a "smoothed" classifier g. It defines $g(x)$ to be the most probable class that f would classify the random variable $\mathcal{N}(x, \sigma^2 I)$ as. In other words, if we were to perturb x slightly to get $x + \epsilon$ where $\epsilon = \mathcal{N}(0, \sigma^2 I)$, $g(x)$ would return the most likely class returned by $f(x + \epsilon)$:

$$g(x) = \underset{c \in Y}{\operatorname{argmax}} \, \mathbb{P}(f(x + \epsilon) = c)$$

where Y is the set of possible classes.

However, as it is generally computationally infeasible to compute what $f(x+\epsilon)$ is most likely to be when ϵ varies over a given normal distribution, Cohen et al. describe Monte Carlo algorithms for evaluating $g(x)$ and certifying the robustness of g around x. A Monte Carlo algorithm is a way of approximating (and knowing the probability of being incorrect) by randomly sampling a space a given amount of times and then using those finite number of samples for your calculations. For evaluating $g(x)$ using this technique, get n samples of noise, $\epsilon_1, \epsilon_2, \ldots, \epsilon_n$ from $\mathcal{N}(0, \sigma^2 I)$, and for each, find $f(x + \epsilon_i)$. Next, find the class that is predicted the most and return that as the smoothed prediction if its count is significantly more than the other classes. In the case where its count is not significantly more than the other classes, Cohen et al. have the smoothed classifier abstaining from making a prediction.

To certify a point, Cohen et al. use a much larger number of samples to get a better estimate of the lower bound

of the percentage of the top predicted class compared to the total number of samples. They then use this lower bound to calculate the certified radius. This radius corresponds to a "ball" around the point for which any samples from that region would be classified the same as the certified point with almost 100 percent chance. Since the radius is dependent on the lower bound of the top predicted class being predicted, the higher that bound, or the more likely $f(x+\epsilon)$ is one particular class, the bigger the certified radius.

4 Current Work & Challenges

In attempts to develop a provably robust explanation method, we have applied the randomized smoothing technique to the explanations themselves. We want to create a "smoothed" explanation model that will take the most probable explanation for the random variable $\mathcal{N}(x, \sigma^2 I)$ as the explanation of why $f(x)$ classifies as it does. Given such a smoothed explanation model, we would then use a similar technique to that of Cohen et al. to certify radii around points for which all explanations within those hyperspheres are the same as their certified points.

However, this has turned out to be a much trickier problem than anticipated. For starters, distinguishing between classes is trivial, but distinguishing between explanations or determining when two explanations are similar is not so trivial. Recall that an explanation is a vector of coefficients corresponding to the linear model that approximates the behavior of the original model around a particular point. If the linear model has 60 inputs (not unreasonable), then we are dealing with an explanation that is in $\mathbb{R}^{60}$. While there are a finite number of image classes for a given classifier, there are infinitely many possible explanations and the chances of getting two identical explanations is effectively zero. When certifying samples, we need a majority class or, in this case, a majority explanation for which we find the lower bound on the probability that it is predicted, and use that to determine the certified radius. If all explanations are different, then we can never find a majority, much less a majority for which we would be able to certify a significant radius.

Thus, we need some sort of similarity metric that effectively creates equivalence classes of explanations. One idea is to focus on the relative values of coefficients. If we have 60 coefficients and we represent an explanation as a vector of indices from largest magnitude coefficient to smallest, then we now have only $60!$ different possible explanations. Not infinitely many, but still far from ideal. So, in addition, define a distance between two explanations such that explanations within a given threshold are counted as the same. Note that we do not necessarily need two "equivalent" explanations to have the same exact sequence of indices. Rather, explanations such that the group of highest magnitude coefficients occur at similar indices might well be sufficiently similar to be considered equal.

We are currently experimenting with various distance metrics. One idea is using *edit distance* which is usually applied to DNA gene sequencing. Edit distance counts the minimum number of additions, deletions, and substitutions it takes to get from one sequence to another, or in our case, one explanation to another. This metric is nice because two explanations with similar sequences of indices starting near the same positions, would be "close" to each other. However, we are more concerned with the highest magnitude coefficients, so we would like some way to weight differences for the first several indices more heavily. This is something we are still working on implementing.

Another more general challenge we are facing is working in such high dimensions. When using an explanation

method such as LIME, recall that the input space to the local explanation model is the interpretable space (or the raw feature space using LEMNA). This space can be in very high dimensions. For example, classifying images usually has a dimension for every pixel in an image in the raw feature space. The authors of LIME clustered pixels together into "super-pixels" and defined the interpretable space to consist of a dimension for every super-pixel. We are also using super-pixels and for a 299 by 299 image, we have 60 super-pixels and hence 60 coefficients in our explanations.

Why is high dimensionality a problem? The space gets exponentially bigger with every dimension. Recall that LIME and LEMNA rely on perturbation based sampling methods to get data to train the explanation models on, so higher dimensions means exponentially more samples to get the same coverage of the space. This is not good because we simply do not have the computational power and time required to handle the number of samples needed to reasonably cover high dimensional spaces. Thus we look towards some form of dimensionality reduction such as clustering pixels together into super-pixels.

With the smoothed explanation model we are developing, a singular explanation is defined to be the most likely explanation returned when considering why $f(x)$ classifies as it does where x is the random variable $\mathcal{N}(x, \sigma^2 I)$. We want to use a Monte Carlo algorithm to approximate this and that requires a lot of samples and their corresponding explanations. The better we want the approximation, the more samples we need and the more computationally expensive it becomes.

5 Future Work

First and foremost we would like to develop a method to equate explanations. This is probably going to involve more trial and error, where we have an idea, implement it, and then test to see if it works how we want it to or if we need to make more changes. Once we have a reasonable way of doing that, we could start seriously testing the robustness of the explanation method. Would could see how big a typical certified radius is and try to find adversarial samples that fool it inside that radius. Right now, we are working with the Inception_v3 image classifier that was trained on the ImageNet data set. If our robust explanation method is successful, we could try applying it to other classifiers. We could also go back to automated Voice Processing Systems and see if or how the explanations differ from what we were originally getting.

6 Conclusion

As machine learning models get more complex and more adversarial attacks are discovered, we find ourselves in need of explanation methods that can reveal what classifiers are basing their decisions on. However, the current explanation methods are not as reliable as we would like them to be and vulnerable to adversarial attacks. Hence our work attempts to develop a provably robust explanation method. We are currently working on modifying the randomized smoothing technique and proofs of robustness guarantees from the work of Cohen et al. to work with explanation methods. Hopefully, we can then move on to testing the robustness of our method and comparing it to others.

References

[1] J. M. Cohen, E. Rosenfeld, and J. Z. Kolter. Certified adversarial robustness via randomized smoothing. *CoRR*, abs/1902.02918, 2019.

[2] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing. Lemna: Explaining deep learning based security applications. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 364–379, New York, NY, USA, 2018. Association for Computing Machinery.

[3] A. Kherchouche, S. A. Fezza, and W. Hamidouche. Detect and defense against adversarial examples in deep learning using natural scene statistics and adaptive denoising. *CoRR*, abs/2107.05780, 2021.

[4] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana. Certified robustness to adversarial examples with differential privacy, 2018.

[5] B. Li, C. Chen, W. Wang, and L. Carin. Second-order adversarial attack and certifiable robustness. *CoRR*, abs/1809.03113, 2018.

[6] M. T. Ribeiro, S. Singh, and C. Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery.

[7] F. C. M. Rodrigues, M. Espadoto, R. Hirata, and A. C. Telea. Constructing and visualizing high-quality classifier decision boundary maps. *Information*, 10(9):280, Sep 2019.

[8] S. W. scikit-learn: Classification algorithms on iris dataset. http://stephanie-w.github.io/brainscribble/classification-algorithms-on-iris-dataset.html, 2016.

[9] X. Zhang, N. Wang, S. Ji, H. Shen, and T. Wang. Interpretable deep learning under fire. *CoRR*, abs/1812.00891, 2018.