Interpreting Deep Neural Networks Through Backpropagation

Miguel Coimbra Vera
miguel.coimbra.vera@tecnico.ulisboa.pt
IST/Universidade de Lisboa, Av. Rovisco Pais, 1, 1049-001 Lisboa, Portugal

***Abstract*—Deep Learning is one of the fastest growing technologies of the last five years, which has lead to its introduction to a wide array of industries. From translation to image recognition, Deep Learning is being used as a one-size-fits all solution for a broad spectrum of problems. This technology has one crucial issue: although its performance surpasses all previously existing techniques, the internal mechanisms of Neural Networks are a black-box. In other words, despite the fact that we can train the networks and observe their behaviour, we can't understand the reasoning behind their decisions or control how they will behave when faced with new data. While this is not a problem in some fields where these networks are being applied to domains where mistakes can be afforded, the consequences can be catastrophic when applied to high-risk domains such as healthcare, finance or others. In this thesis we explore a generic method for creating explanations for the decisions of any neural network. We utilize backpropagation, an internal algorithm common across all Neural Network architectures, to understand the correlation between the input to a network and the network's output. We experiment with our method in both the healthcare and industrial robotics domains. Using our methodology, allows us to analyse these previously opaque algorithms and compare their interpretations of the data with more interpretable models such as decision trees. Finally, we release DeepdIVE (Deriving Importance VEctors) an open-source framework for generating local explanations of any neural network programmed in Pytorch.**

***Keywords*—Deep Learning; Interpretability; Machine Learning; Artificial Inteligence; Neural Networks;**

I. Introduction

Machine learning is not a new field, the concept that we could teach machines how to "learn" the solution to a problem by themselves has been around for a very long time. Due to limitations in techniques and computational power, the algorithms used in the early days of the field were considerably simpler than the ones used today. As such, it was easy to interpret the algorithms used for these regressions and deduce *why* they had made a certain decision. In the last few years, partly due to enormous advances in computational power and partly due to new optimization techniques, Neural Networks have become the de facto standard for tasks such as Natural Language Processing, and many others. [1] With this increase in adoption, researchers have started looking at ways to expand the use these networks to areas such as finance, medicine or industrial processes.

This has brought into the forefront one of the major drawbacks of Neural Networks, they're extremely hard to interpret. They are composed of several interconnected layers of sometimes thousands of "neurons", each of which applies a mathematical transformation to received data. Their fantastic generalization capabilities are related to the fact that they use distributed representations in their hidden layers, that is, their "knowledge" is distributed unevenly by a set of neurons. [2] Because the minor effects of any particular "neuron" rely on the effects of the activations of every other unit in that layer, it is extremely hard to retrieve any meaningful information about their role in a decision. [3]

This "black box" nature of neural networks has become an enormous barrier of adoption in areas where decisions have irreversible consequences. In areas such as NLP, the consequences for failure are fairly minor, as such, the focus on interpretability as remained an afterthought to performance. On the other hand, in areas like medicine, where a wrong choice might prove deadly, it is of utmost importance to understand *why* the network has made a decision.

To illustrate the importance of interpretability, Caruana et al. [4] detail a model trained to predict probability of death from pneumonia. This model learned to assign less risk to pneumonia patients if they had asthma. Patients with asthma, when arriving at the hospital, received more aggressive treatment due to their condition. As a result, the model concluded that asthma was predictive of lower risk of death. If this model were to be deployed to aid in triage, these patients would receive less aggressive treatment. Acting in blind faith could have catastrophic consequences. So, how can doctors act upon predictions without knowing the reasoning behind the result?

In light of these facts, model interpretability has become of paramount importance for widespread adoption in high-risk areas. For the purpose of this work, we will consider interpretability as a source of trust. And trust, which we define as human confidence on the reasons behind the network's prediction. This confidence relies on an observable causality between inputs to the network and its outputs. As such, we can say that an interpretation of a decision would be an explanation of how the input values were at the genesis of that decision.

In this work, we propose a variation of backpropagation based techniques to interpret *any* deep neural network. By calculating the gradients at decision time we can create a causal relationship between input features and the network's output. With it, we can accurately attribute decisions to a specific cause or causes. This provides a generic method for Neural Network Intrepretability.

The rest of this work details previous work, our approach, our methodology, results and an explanation of DeepDIVE our open-source implementation of the methods proposed in the body of this work. But first, as an attempt to introduce unfamiliar readers to the core concepts behind this work we make

an endeavor to explain Neural Networks and backpropagation as plainly as possible.

II. Related Work

A. Interpretability

Due to the enormous growth in it usage, Machine Learning has naturally started seeping into critical areas. When tackling these kinds of issues, the fact that humans are incapable of understanding advanced models in machine learning takes a whole other dimension [5]. Machine Learning model *Interpretability* came into the spotlight as a way to solve this problem. The reasoning is that, if we can accurately pinpoint how the model is making a decision, we can take actionable steps to correct its behaviour. Despite this, not many people have tried defining what Interpretability *is*. Counter-intuitively papers often make allegations about the interpretability of numerous models.

In the research community, researchers have taken the stance of adopting a definition for interpretability that makes intuitive sense to them. This has the unwanted effect of the definition for interpretability changing considerably from paper to paper. The study by Lipton [6] compiled a list of the different descriptions of interpretability found throughout the spectrum of machine learning literature. Some researchers attempt to describe interpretability as ”simulatability”, whether a person can contemplate the entire model at once. Or in other words, whether one can examine the entirety of the model and understand what it is doing and how. This notion is also adopted by Ribeiro et al. which suggests that an interpretable model is one that ”can be readily presented to the user with visual or textual artifacts” [7]. Other authors suggest different traits to judge interpretability such as decomposability or algorithmic transparency.

By none of these metrics are neural networks considered interpretable. But it can also be argued that the different metrics have different goals or properties that they strive for. Not all models that are typically considered interpretable would fit even two of those metrics. We believe that this is because there can be no discussion about interpretability without considering, not only what makes a model interpretable, but what it is we are striving for with interpretability.

Luckily, Lipton [6], when compiling the different metrics used to classify an interpretable model, also studied and collected the different properties interpretability research strives for. These properties are: Trust, Casuality, Transferability, Informativeness and Fair Decision-Making. In our work we will mainly focus on interpretability as a source of trust and information.

B. Interpretable Models as Explanations

When considering ways to provide explanations for the behaviour of deep neural networks, one of the most common approaches is a delegation of responsibility. Instead of looking at the model's internal mechanisms for an explanation, the task of explaining the model's behaviour is outsourced to an external, simpler and interpretable model.

Ribeiro et al. propose an approach capable of explaining *any* model, including neural networks, by treating the model as a black-box. They suggest that two things are essential for an explanation of a machine learning model: The explanation must be readable by a human and it needs to be locally faithful [7]. They claim that it is often impossible for an explanation to be entirely faithful unless it is the complete description of the model itself. As such, they argue that explanations don't need to represent the entirety of the model's behaviour. They only need to be locally faithful, or in other words, accurately describe the behaviour for that particular decision.

Taking this into account, the authors developed a framework by the name Locally Interpretable Model-Agnostic Explanations (LIME). Its goal is to use an interpretable model to provide an interpretable representation locally faithful to a prediction of another classifier. An example is provided in figure 1

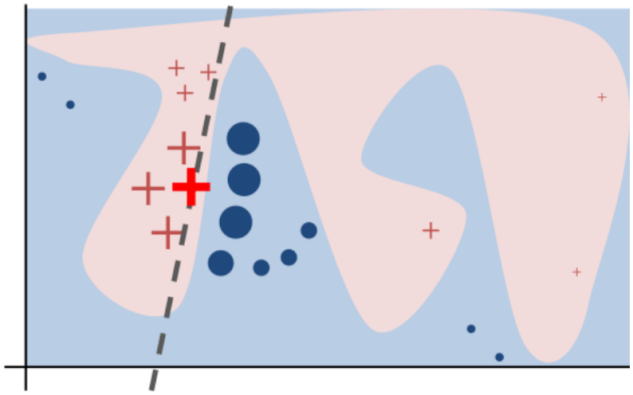

Fig. 1: An example for LIME. The black-box model's decision function f is represented by the pink/blue background. A linear model is used to create a local explanation. The bold red plus sign is the instance being explained. LIME samples instances by perturbation (red plus signs and blue circles), gets predictions using f and weights them by proximity to the original instance. The dashed line is the learned explanation, it is locally (but not globally) faithful. Image sourced from [7].

Another similar approach was developed by Frosst et al. Instead of trying to understand how a deep neural network makes its decisions, they use the network to train a decision tree that mimics the function discovered by the network but works in a completely different way [8]. They experimented using a technique labeled distillation [9] to extract the knowledge from the tree to a type of decision tree that makes soft decisions [10].

Both of the aforementioned approaches have glaring problems. LIME, while an extremely interesting approach that can be applied to any model, has some glaring problems. First, it tries to explain changes in a model's behaviour by first order changes in a discrete interpretation of the input space. In other words, by measuring the changes with permutations, there's no guarantee that the behaviour being modeled is actually from the space surrounding the original input in the target model's space. Secondly, for each decision of the model we intend to explain, we must train a new explainable model. This adds a significant overhead to providing explanations to more complex models with large runtimes, as the model needs to be run several times in order to generate an explanation. Distilling a neural network

to a soft decision tree, on the other hand, is also a far from ideal solution. While it is easy to generate an explanation, we only have to follow the path with the greatest probability from the root to the leaf. It does not explain neural networks directly, only by proxy through another model. There is no guarantee that this other model will learn the same distribution as the original network. Furthermore, these explanations can prove hard to interpret depending on the format of the data. It is easy to think of complex examples where these explanations would quickly become too bothersome and complicated to interpret. Moreover, since the decision trees can only attain a performance significantly lower than the networks, we must sacrifice performance of the model for an explanation.

C. Inner Model Visualizations

Another type of approach to deciphering neural networks, in contrast to the ones in the previous section, are based on using the inner mechanisms of the model to get information on how they are making the prediction. It should be noted however, that before the recent rise in popularity of Deep Learning, there had already been some attempts to interpret the networks' inner workings [11], [12].

In 2009, Erhan et al. discovered that, counter-intuitively, it is possible (and simple) to interpret deep architectures at the unit level [13]. Before his work, the only qualitative method to compare and explore features representations by deep networks was by examining the "filters" learned by the network [14], [15]. That is, the linear weights in the input-to-first layer weight matrix represented in input space. Erhan et al.'s work was the first to develop methodologies to visualize representations of arbitrary layers of a deep architecture, in this case, Deep Belief Networks [16]. They discovered a curious phenomenon: the responses of the internal units to images, as a function in image space, are unimodal. Knowing this, by finding the dominant mode, we can create a good characterization of how the unit is processing the information. With recent works such as Olah et al. [17] pushing this medium forward, trying to combine both visualization and attribution to create rich interface for humans, the gap between feature visualization and feature attribution is closing. It is however, still too wide to be considered an effective solution to our problem.

Meanwhile on the other side of this gap, more or less simultaneously to Erhan et al's work, Baehrens et al proposed a novel approach for interpreting Bayesian classifiers [18]. Their goal was similar to our own, to peek into the black-box of modern classifiers and provide an answer to *why* a certain decision was made by the classifier. Their approach uses the fact that the bayesian classifiers are optimized using numerical algorithms such as gradient descent. With that in mind, they use class probability gradients in order to create local explanations of the classifier. This creates a vector that "points" away from the chosen class, in essence, providing a weighted vector of the features which influenced the classifier's decision. This can easily convey the relevant features for a single prediction, both the ones that contributed positively and negatively to the final class.

This type of approach made its way into neural network research when Simonyan et al. [19] based on Erhan et al.'s and Baehren et al.'s works, proposed using the gradient of the output with regards to pixels of an input image to create a saliency map (Figure 2) as a way to interpret deep convolutional neural networks [20]. It mixed the ideas from previous works [13], [18], gradients as explanations and creating explanations using input space. The authors propose that the gradient of the output class with regards to the pixels of the input image can create a weighted vector of contributions from the input pixels, or how much each pixel of the input image contributed to the final classification. At the time when this was proposed, the most common approach to interpreting convolutional networks was a specific network architecture called DeconvNet [21]. This approach required training an additional network with a specific architecture on top of an existing convolutional neural network. It was unpractical as it required training a specific network on top of an existing network in order to interpret the latter.

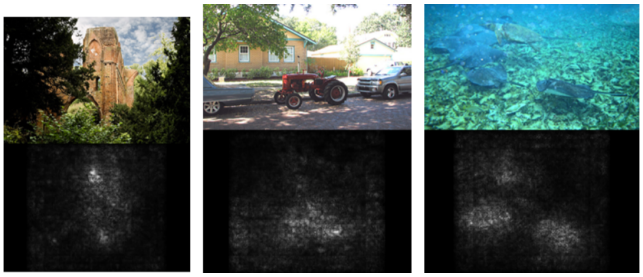

Fig. 2: The maps were extracted using a single backpropagation pass through a classification ConvNet. These maps represent the top predicted class in ILSVRC-2013 test images. Image sourced from Simonyan et al. [19]

Since the introduction of this method, several other authors have proposed different gradient based methodologies. Springenberg et al. unsatisfied with the different ways default gradients and DeconvNets treated ReLUs, suggested further homogenizing both approaches and proposed guided backpropagation [22].

Other approaches to solving this problem include Layerwise Relevance Propagation (LRP) [23], an approach for calculating importance scores in a layer-by-layer approximation of backpropagation. This method was later demonstrated to be equivalent within a scaling factor to an elementwise product between the saliency maps of Simonyan et al. and the input [24], [25]. Sundarman et al. suggested that instead of computing the gradients only with the current value of the input, we could integrate the gradients as inputs are scaled up from a certain starting value [26]. Nonetheless recent works have shown that this approach can still provide highly misleading results [27].

III. Backpropagating Feature Importance

When we began working on the issue of unwrapping the black-box that are modern deep neural networks, we had a clear goal: to be able to provide instance-based explanations. A feed-forward artificial Neural Network is organized in layers with each layer performing different transformations on its inputs. The goal of these networks is to approximate a certain function f^*. Each of its layers can be seen as a function and a network can be seen as a chain of functions. Backpropagation is used to

calculate the gradient of the loss function w.r.t weights of the connection between the layers of the network. This is employed by the gradient descent optimization algorithm to adjust the weight of neurons with the previously calculated gradient of the loss function. Assuming x as a vector of inputs and θ as the parameters of the network, the gradient at time t can be calculated as:

$$\nabla = \frac{\partial Loss(x, \theta^t)}{\partial \theta}$$

Although this is commonly used as a way to train a network such that it can learn appropriate internal representations that allow it to approximate f^*, we propose generalizing the method by Simonyan et al. [19] to be used in a wider spectrum of networks. His original proposal and most other subsequent works [22], [25], [26], [28], apply this methodology solely to image classification networks with convolutional layers. With this work we intend to explore a generalization of this approach that can be applied to *any* network architecture.

Specifically, consider a system that classifies some input into a class from a set C. Given an input x, many networks will compute a class activation function S_c for each class $c \in C$. The final result will be determined by which class has the highest score.

$$class(x) = argmax_{c \in C} S_c(x)$$

We define an explanation vector of a data point x_0 as the the gradient of the output w.r.t the input, x at $x = x_0$. With that, we estimate a weighted vector of feature importances. If S_c is piecewise differentiable w.r.t x, for all classes c and over the entire input space, one can construct a feature importance vector $I_c(x)$ by computing:

$$I_c(x_0) = \left.\frac{\partial S_c(x)}{\partial x}\right|_{x=x_0} \quad (1)$$

This explanation vector is a local n-dimensional representation of the behaviour surrounding the decision point on the surface of the function learned by the network. By having a representation of the behaviour around that point of the manifold we can measure the sensitivity to change of the function value with respect to changes in its arguments.

In a more intuitive manner, I_c represents how much difference a change in each feature of x would cause in the classification score of class c. Thus, entries in I_c with large absolute values highlight features that will influence the label of x_0. On the other hand, the sign of the individual entries expresses whether the prediction would increase or decrease when the corresponding feature is increased locally.

This can also be motivated with a simple example. Consider a linear model for the class c:

$$S_c(x) = w_c^T x + b_c$$

assuming x is represented in a vectorised form, and w_c and b_c are respectively the weight vector and the bias of the model. In this case, we can easily see that the magnitude and sign of the elements of w will define the importance of the corresponding features of x for the class c. In the case of deep Neural Networks, the class score $S_c(x)$ is a highly non-linear function of x. Because of this, it is extremely hard to apply this reasoning. However, given an input x_0, we can approximate $S_c(x)$ with a linear function in the neighbourhood of x_0 by computing the first order Taylor expansion.

$$S_c^{'}(x) \approx d^T + b$$

where d is the derivative of S_c w.r.t the input x at the point x_0, as defined in 1. By definition, the gradient of a function $f(x)$ is the linear approximation of that function around x. This linear approximation can also be thought of as a Taylor series approximation with a Taylor polynomial of degree 1. As such, the gradient can be seen as an approximation of the linearization of the function and, in essence, of feature attribution.

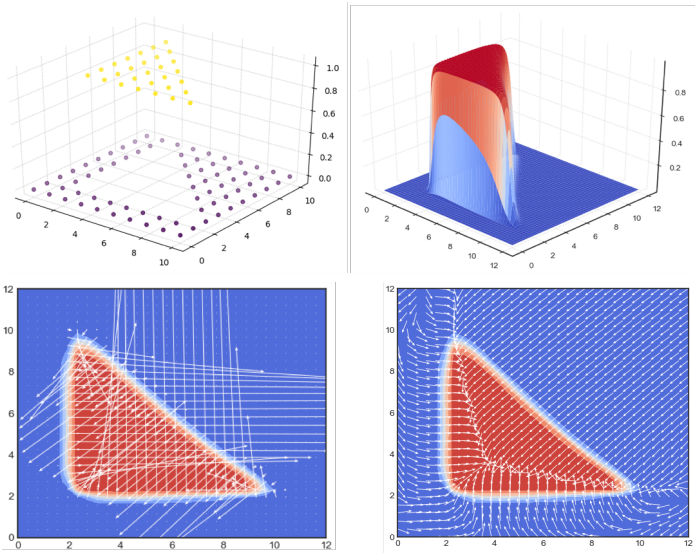

Fig. 3: Explanation of individual classifications of a simple Neural Network trained on a toy problem. On the top we have the dataset and the learned decision curve, left and right respectively. On the bottom, the visualization of the local explanation for points equally spaced throughout the dataset. On the left, the real values of the importance of each feature and on the right a normalized version.

On the top left corner of Figure 3 we can see the training data for the simple classification task. On the top right is the model learned using a simple network which outputs the probability of the point belonging to the yellow points. On the bottom left corner we can see the probability gradient (background) together with local explanation vectors generated by calculating the gradient of the output w.r.t the input. We can see that along the borders of the triangle the explanations point overwhelmingly towards the triangle class. Furthermore it is noticeable that along the hypotenuse and at the corners of the triangle both features are considered important for the explanation while along the edges the importance is dominated by one of the two features. In the bottom right we can see the normalized explanation vectors, this gives us a good representation of the direction of explanations without considering their magnitude.

Since we use local gradients of the trained model as explanatory vectors, they will always follow the model and its assumptions, providing a perfectly faithful explanation. On

the other hand, by capturing the gradient of the model we can experiment and measure the influence of changes in any direction, including weighted combinations of variables.

Overall, we believe that our approach is one of the most reliable and efficient ways of interpreting deep Neural Networks. By using backpropagation to explain the behaviour of the net we not only manage to do it in a single backwards pass, which is very efficient, but we also provide explanations that are completely faithful to the model's behaviour.

IV. Experiments and Results

In order to test the viability of our approach, we need to test it's performance in real networks. However, there are some problems with the current state of interpretability research. Contrary to other popular areas in machine learning, such as natural language or computer vision, there are no standardized tests for interpretability. Each new paper or approach that is suggested defines new evaluation metrics and experiments. This creates a fragmented research landscape as approaches are evaluated in different ways, allowing authors to choose the light than want to show them in. We try our best to follow best practices and experiments laid out by other recognized authors in the field [17], [27], [28].

A. Medical Domain

1) Context: Machine learning applied to healthcare is booming, every year several new approaches are suggested to solve pattern-detecting problems such as detecting pneumonia or tuberculosis in X-rays [29], [30] and new surveys analyse the impact this technology might have in the way healthcare works. But although some of these systems already achieve state-of-the-art results and have much better performance than human experts, adoption has been slow [31]. Interpretability, or lack thereof, has been one of the main reasons for the reluctant adoption of Deep learning in healthcare. As with anything related to Human health and security, there is an enormous amount of uneasiness in letting a system we do not fully comprehend make decisions that will directly influence our health. Furthermore, it is easy to imagine examples such as the one by Caruana et al. [4] where seemingly high performing models are making a mistake and invalidating their own results. In healthcare, this *cannot* happen.

With this in mind, we set out to find a dataset we could use that would be representative of the real world circumstances of a machine learning problem in the medical domain. Luckily, concurrently to our work, another student from our home institution (Instituto Superior Tcnico) was pursuing a thesis in this field [32]. Their work was based on a set of medical questionnaires from the biggest pediatric hospital in Portugal, Hospital D.Estefnia. These were filled by doctors to assess risk factors in children being evaluated for mental health issues.

2) Data Analysis: The data provided to us was retrieved from paper forms. The ones we had access to were archived cases from recent years, as such, our colleagues could only retrieve 322 datapoints. The main problem with such a low amount of data, is that because of the low amount of samples, if split randomly, the distributions in the train and test set end up being significantly different. This proved to be a really challenging problem to apply Deep Learning to, has it typically requires large amounts of data to learn the statistical relationships present in the data.

These forms were filled by doctors when a patient arrived at the hospital and used as a sort of triage/diagnosis mechanism to access a patient's risk level. On a patient's arrival, doctors ask the patient several questions regarding their behaviour. These questions are split into four different segments, each regarding a certain type of behaviour. Then, by solving the form, the doctors would rate the risk level of the child in four different areas: Suicide, Auto-Aggressiveness, Hetero-Aggressiveness and the risk of the child running away. It should be noted that although the forms had pre-determined rules on how to diagnose a patients according to their answers, doctors often followed their own heuristics. As such, what we are trying to do here is not so much learning the rules of the forms but heuristics developed by the doctors' years of experience.

Our goal with this dataset was to train a classifier that could, by receiving the answers to the questions in the forms, correctly attribute levels of risk to a new patient. What our colleagues did was train a different decision tree for each risk area. However, neural networks are able to learn and represent much more complex relationships present in the data. As such, we felt that it would be much more interesting to try to model the relationships between the different risk areas. This way, we can verify if questions in some areas areas influence the assessment of the doctors when classifying other, different, areas. Lastly, one of the other simplifications our colleagues did was based on the protocol of the hospital. Protocol stated that doctors should only intern children with very high risks in one of the areas. Taking that into account our colleagues reduced the dataset to two classes: very-high risk and not-very-high risk. Or in other words, whether a patient should be interned or not. Doing this, they managed to diminish the amount of possible outcomes of the classifier to 16 from 256 (2^4 vs 4^4).

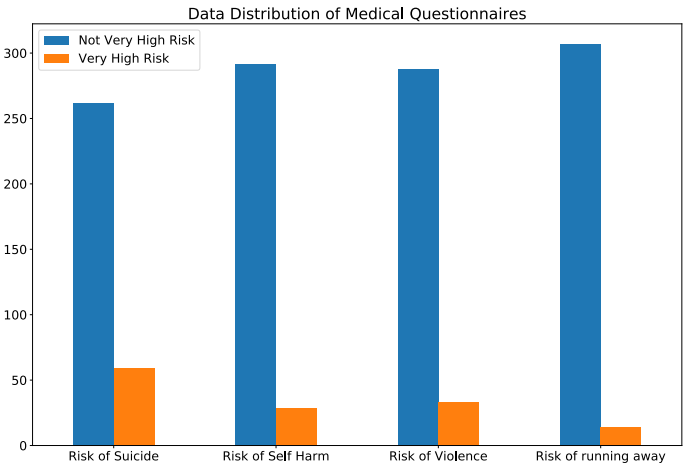

Fig. 4: Data for this problem was extremely skewed, with the least imbalanced class (Risk of Suicide) having close to 5:1 ratio and the most imbalanced class having around 10:1 ratio between Not Very High and Very High risk.

This change brought with it another problem, besides having a low amount of data, the data was also heavily skewed in favor

of non-high risk patients. The ratio was about 39:5 in high risk to non-high risk data points, this can be easily visualized in Figure 4.

A common solution to these kinds of situations is to re-balance the dataset and the previous work of our colleagues had done just that. They used the SMOTE (Synthetic Minority Over-Sampling) algorithm [33]. However, they did this individually for each of the different risk factors. This makes it so that they had a different dataset for training each of their trees. We, on the other hand, intended to train a single network for the entire dataset. As such, since we could not use their exact datasets and SMOTE does not possess a clear multiclass implementation we experimented with random over sampling but it did not yield the results we had hoped for. In the end, we had to settle for using the original dataset without any major modifications.

3) Network Architectures: This being a straightforward classification problem we argue that a simple feed-forward network is the most appropriate approach. Our suggested architecture was nicknamed ReLUSigNET. It is composed of four linear layers with ReLU activations on the first three and ending with a sigmoid in order to squash the output neurons into values between 0 and 1. This makes it so that each of the final four neurons will give us a binary result for each risk area (Suicide, Auto-Aggressiveness, Hetero-Aggressiveness and the risk of running away). Furthermore, in order to help with regularization and prevent overfitting we used Dropout on all layers except for the input and output [34]. This technique consists in randomly "dropping" (zeroing) some of the neurons of the layer and it is used to prevent complex co-adaptation of neurons.

4) Results: Overall, the results achieved by our solution were quite good. The results can easily be visualized in table I. It should be noted however, that directly comparing the results is a bit misleading. As the form is divided into several different blocks of questions, each block measuring one kind of risk, the feature space of these trees is quite small. By training one tree for each risk they only use the questions from the corresponding block and, since they generated synthetic points for each of the risks they reinforce certain patterns that might not be accurate when taking into account the complete data.

Although our results are slightly lower than the ones achieved by our colleagues, we believe this is an extremely good result. In training the decision trees, around 650 datapoints were used for *each* model. This means that they had twice our data for each of their 4 different problems. In the end this adds up to about 8 times more total data than the original dataset. We used *half* their data in order to learn one single model that could solve the 4 problems at once with a score within approximately 0.4% of theirs. We believe it is a testament to Neural Network's generalization capabilities that using regularization techniques like dropout our we managed to equalize the performance of the Decision Trees using much less data.

One of the reasons our colleagues chose decision trees as the classifier to use in this problem was their ability to provide instance-specific explanations to the Doctors. With it they could provide an easy explanation for a Human with domain specific knowledge to evaluate.

		Decision Trees	ReLUSigNET
Suicide	Accuracy	0.9	0.91
	Precision	0.9	0.69
	Recall	-	0.82
Heteroaggression	Accuracy	0.91	0.92
	Precision	0.91	0.66
	Recall	-	0.75
Autoaggression	Accuracy	0.97	0.95
	Precision	0.97	0.87
	Recall	-	0.77
Running Away	Accuracy	0.99	0.97
	Precision	0.99	0.80
	Recall	-	0.80
Average	Accuracy	**0.94**	**0.94**
	Precision	**0.94**	0.76
	Recall	-	0.79

TABLE I: Although both systems achieve high accuracies, the decision trees have a higher precision. It should be noted that we could not measure recall for the trees since we did not have access to the final trained trees.

As we mentioned before, it is extremely hard to measure interpretability. One thing we could have done is to experiment with the technique from Frosst et al. [8] and train our own decision tree based on the knowledge of our network.

In this case, we devised a completely different test for the explanations of our model. Since our explanations are supposed to give us a weighted vector of feature importance, we can judge the truth of an explanation by perturbing samples according to the importance of each feature. Our goal was to perturb the *second* most important feature in each sample, and, if the classification changed, check if changing the most important feature would *also* change the classification. As such, we moved all the second most important features in 100 randomly sampled datapoints. The results were astoundingly positive. In 100% of the cases where the second feature changed the predicted classification, the most important one also did. Thus, all proof indicates that, our explanations are indeed faithful to the local behaviour of the model.

Secondly, in order to test the impact the rest of the features provided by the local explanations produced by our approach we devised another simple test. Using the weighted vector of feature importances we perturb the x most important features for that classification decision. After altering these features we re-classify the data point and verify if the classification output is still the same. We took 100 random samples of the dataset (with a different random seed than the previous experiment) and calculated their explanation vectors. The results can be seen in table II. It is extremely interesting to notice that as we increase the number of features that we alter we start seeing diminishing returns. This makes sense as datapoints that are close to the decision boundary are easy to change but as we get further and further away, more and more features need to be changed. With a 25-dimensional input space it is very interesting that we can change almost half of the classifications of some risk areas with as little as two feature changes. On the other hand, risk areas like Running Away are clearly dependant on a lot more features, thus, they require perturbing a lot more features for their classification to change. It is also interesting that the two risk areas with the highest accuracy are the hardest ones to change. This suggests that the network is capturing

	Suicide	Heteroaggr.	Autoaggr.	Running Away
Top 1	29%	18%	11%	3%
Top 2	66%	43%	24%	8%
Top 3	71%	58%	33%	15%
Top 4	76%	62%	40%	20%

TABLE II: As the number of features that we perturb increases, the amount of datapoints whose classification is changed increases. As we reach 4 features we can clearly start seeing diminishing returns, implying that these datapoints that weren't changed are extremely far from the decision boundary of the network.

non-linear dependencies between several features, making a large number of features important and not just the top 4.

Overall, the results of our classifier were good. However, the goal of this experiment was to have a way to test our explanations. In this field, our first test proved that the feature importances we calculate with our method are meaningful and representative. In other words, they confirmed that a certain feature having a higher importance is actually meaningful and higher importances correlate with actual higher weight for the model's decisions. Furthermore, our second test shows the relevance of feature importance. Without an interpretability method like ours it would be impossible to ever know what features of the 25 were being relevant for the network's predictions. Now, not only do we know the important features but we have a method so accurate that we can change up to 30% of the network's classifications just by changing one feature (and maybe even more had we measured if changing that feature had any effect on other risk areas). This gives us extremely important information about the reasons behind the network's decisions. In contrast with having to blindly trust a high-performing model, Humans can now retrieve this information and use it to inform their decisions.

B. Industrial Domain

1) Context: With automation playing an ever greater role in factories throughout the world a lot of new research has appeared in the machine learning community with regards to autonomous robots. In a lot of cases, these "robots" are specialized arms that, due to their ability to move in a precise way in three dimensions, are used for a plethora of different jobs. However, with the black box nature of deep learning algorithms, this has raised many concerns within the scientific community. Currently, there are no mechanisms that can guarantee and certify the behaviour of a deep neural network. Interpretability, although not a guarantee of the network's behaviour, is a stepping stone on the path to creating solutions to understand the complete behaviour of the network. For now, it can be a valuable resource for machine learning practitioners, they can use it to verify if the machine is learning what it should, behaving correctly, amongst other things.

However, considering the difficulties on testing interpretability expressed in the previous chapter, it was extremely hard to find a relevant dataset that also provided information we could use to test our explanations. With this in mind we decided to design our own dataset. By doing this we know all desired behaviours and we can compare them experimentally to the network. Being able to experiment with our model and knowing the behaviours to expect, we can review our experiments and consequent explanations in a more empirical way. This is part of the reason why computer vision as been such a focused area in interpretability research. It is extremely easy to look at an image of a dog, the network's explanation and visually verify if it makes sense.

Taking this into consideration, we designed the problem of a two-limbed robot arm with two degrees of freedom capable of moving in two dimensions. The entire behaviour of the arm can be represented with the following equations:

$$\begin{aligned} x(\theta_1, \theta_2) &= \sin(\theta_1) + \sin(\theta_1 + \theta_2) \\ y(\theta_1, \theta_2) &= \cos(\theta_1) + \cos(\theta_1 + \theta_2) \end{aligned} \tag{2}$$

We trained networks as regressors to estimate the position of the robot's hand, given the angles of both joints of the arm. To successfully accomplish this, the network would have to learn expression 2. Another interesting characteristic of having an equation to define your dataset is that we can easily calculate it's gradients. Having these theoretical gradients, we can actually confirm the explanations given by our approach. Assuming that a network that has a very low error is learning a function extremely similar to 2, then, the networks gradients will have to be similar to the theoretical gradients. Else, the network would not be able to mimic the original function across the dataset. It should be noted however, that since our network receives two angles and outputs x,y coordinates, our interpretation of the network will be relative to only one of the axis, either x or y. So each explanation should tell us "How does altering each angle influence movement in this axis?". In the end, this should be the same as the derivative of 2 w.r.t the axis in question.

2) Data Analysis: One of the characteristics of our dataset is that 2 is periodic with a period of $2\pi rad$ or 360. Let f be 2 this means that $f(x + P) = f(x)$ where $P = 2\pi$. Let l and u be the lower and upper bound of the angles, respectively. Having the equation 2 to define the properties of our synthetically generated dataset, generating it was extremely easy. We sampled 50k random number pairs between 0 and 1, from a uniform distribution and applied them to 2. With d being a random datapoint, c the coordinates associated with it and $U(0, 1)$ a uniform distribution between 0 and 1, the dataset generation can be summarized in the following expressions:

$$d = (U(0, 1) * (u - l)) - |l|$$

$$c = f(d)$$

The dataset we used for the majority of our experiments was defined as $l = 0$ and $u = 2\pi$.

3) Results: Usage of a synthetic dataset had a lot of upsides for studying our problem. Although it is still hard to devise good tests for interpretability. The ability to compare our results to the "right answer" is invaluable.

The networks had an extremely high performance with RMSE ¡ 0.01 However, when increasing the range, the networks couldn't manage to keep up with the complexity of the data. They could not learn periodicity at all. In hindsight, this should have been expected since the activation functions we

picked for our networks are non-periodic. They can learn the behaviour of the function in a certain range of values, but won't be able to reproduce it outside the learned range.

4) Using Gradients for Visualization: As we mentioned in our literature review section, one of the most common intersections of interpretability in research is visualization. Most methods of interpreting a network can be used in a way or another to create interesting visualizations about the behaviour we are interpreting. Our toy problem, being 2 dimensional, makes for the perfect target to represent in plots. As stated before, an explanation for this toy problem means "How does altering each angle affect this axis?". Thinking of our own arm it is easy to notice positions where moving one of our joints almost doesn't move our hand. With this in mind we set out to represent if the networks could capture this behaviour and, more importantly, if our explanations would reflect it. We sampled random points throughout the dataset and created explanations for them.

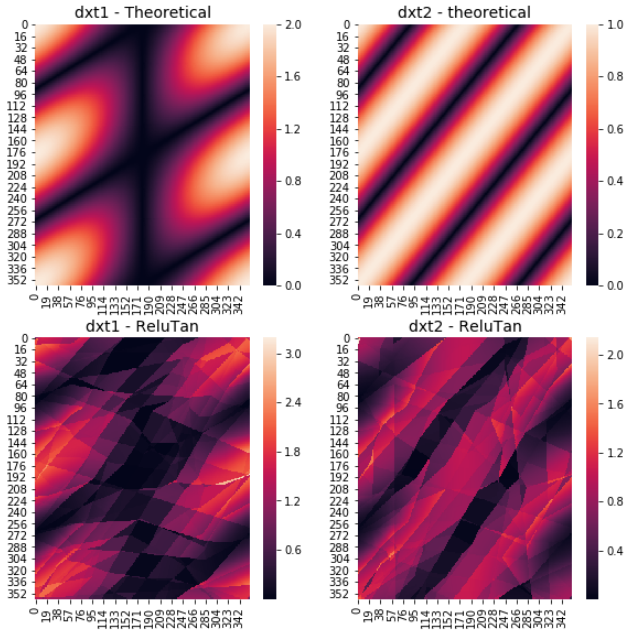

Fig. 5: Comparison of dx/dθ1 and dx/dθ2 between the theoretical values and ReLUTan. Although the accuracy of ReLUTan is extremely high we can see that the fact that it uses ReLUs makes the gradients much harsher than the original gradients of equation 2.

This revealed some interesting insights. Biased samples overlap with non-biased samples and sometimes it is even noticeable that samples biased to one angle are in the midst of samples biased to the other angle. This is a reflection of the fact that biases being plotted in output space. Sometimes there are several different positions that the arm can assume while having the hand in one place and for each of these our explanation will vary. Hence, different biases juxtaposed. Another interesting revelation was that the plots for the x and y axis are incredibly similar if one rotates one of them by 90deg. In hindsight this should have been obvious because if we are moving in the x axis and apply a 90deg rotation to the shoulder, our arm will now be moving on the y axis. Using this visualization, we could extrapolate that the network is indeed learning the behaviour of an arm (and we did spend more time experimenting and confirming behaviour with our own arm than we dare to admit). This allows us to have at least some certainty of what the network is in fact learning. As of right now, most practitioners training neural networks have no ways to test things like these and rely on raw performance numbers to check if their network is learning. As we have seen before this can lead to critical errors. [4], [7]

Another experiment in the realm of visualization that becomes possible with this dataset is comparing the gradients of the function throughout its entire domain. In other words, with this toy problem we can compare the function learned by the network directly with the real target function. This can be seen in figure 5.

It is extremely interesting to be able to peer into the inner workings of a network in this way. We can see actual proof that ReLUTan is indeed learning the exact objective function. More importantly however this is proof that our explanations are retrieving a very good first degree Taylor approximation of the networks decision function. Our method for deriving feature importance has created a mapping that very closely resembles the gradients of the theoretical function. As such, if we assume that, in any function, it's gradients are good indicators of local feature importance then it follows that our method is capable of retrieving those values of local features importance.

5) Gradients as Explanation: What we set out to do with our method was to provide a method to retrieve feature importance from a specific prediction of the network. Following the thought process that these gradients do in fact provide valuable information about the local behaviour of the function we can easily transform the gradient visualizations into more useful explanations. By combining both $\frac{dx}{d\theta 1}$ and $\frac{dx}{d\theta 2}$ we can create an explanation for movement in the x axis. This explanation is a vector that indicates how, in the local space, should the angles move in order to increase x. By doing this along the entire feature space (since it is 2-dimensional) we can create a graph that shows how these explanations vary throughout the $[0, 2\pi]$ range on both angles. This can be seen on figure 6.

This plot was produced by taking samples at every 12deg for the entirety of the input space and using our method to retrieve explanations for both the x and y axis. Here we can easily visualize the biases in the movement of the arm. An unbiased position is one where the explanatory arrow has a perfect 45deg angle, as otherwise, one of the angles has a bigger effect on the movement of the arm. This plot can also serve as a representation of the behaviour of the arm throughout the data set.

Finally, we wanted to confirm that our method of retrieving explanations was actually producing faithful explanations of the model throughout the entire domain. In order to test this we set out to retrieve explanations from the entirety of our dataset. Having these explanations we filtered them by the ratio of the importance of one angle w.r.t the other. Or in other words, retrieved all samples where the absolute importance of $\theta 1$ was $threshold$ times higher than the absolute importance of $\theta 2$.

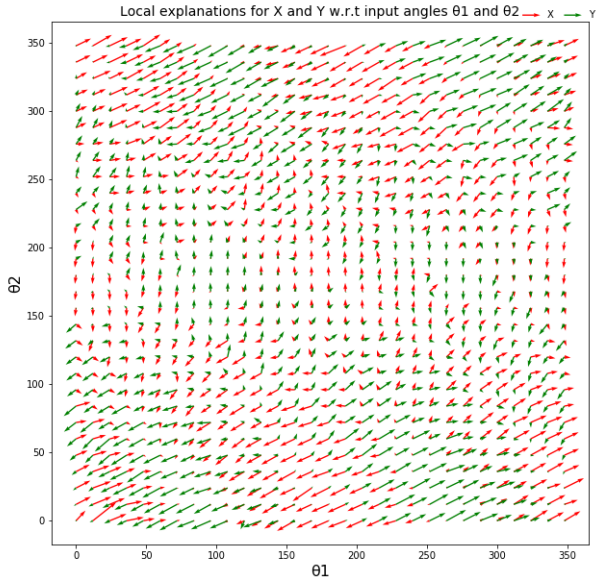

Fig. 6: Local explanations retrieved from ReLUTan. Each arrow color represents an explanation for one of the axis of movement. The magnitude of the arrow represents the amount of movement predicted. Arrows on the borders of the input space are generally bigger due to the network's inability to predict correctly outside this range.

Threshold\Axis	X	% of all data	Y	% of all data
1.5	99.50%	77.8%	99.78%	79.5%
2	99.80%	58.3%	99.82%	57.2%
3	99.82%	37%	99.79%	36.3%
4	99.86%	27.4%	99.86%	25.9%

TABLE III: From the biased data retrieved by filtering explanations where the importance ratio between angles was threshold we can see that movement on both axis is extremely faithful to the explanations. Most of the datapoints where the movement of the arm didn't correspond to the explanation are located in the edges of the dataset.

For each of these points we moved both angles of both joints in two directions and recorded their movements. The idea was to verify if the explanation would be faithful to the results of the network I.e. if the moving the "most important" angle actually caused the biggest change in the axis the explanation was generated for. The results can be seen on table III.

Here, we can clearly see from the results that the explanations are astoundingly faithful to the model's behaviour. Even with the smallest threshold we used (1.5) which captured around 80% of the dataset we had an average of **99.64%** of datapoints where the movement corresponded to the expected from the explanation. This are great results, not because they allow us to retrieve any special insight from the network or the problem at hand but because they show us that the explanations are truly reflecting the underlying behaviour of the model. As such, when using our method of creating explanations on a more complex, real-world dataset where we have no reference to compare our results to, we can be fairly confident that whatever our explanations are telling us about the network's internal decision function is faithful and precise.

V. DeepDIVE: Deriving Importance VEctors

DeepDIVE is an open-source framework developed in the context of this work. It was developed as a way to easily allow anyone to experiment with our interpretability method and it is available publicly at: https://github.com/captainvera/deepdive. It implements our method of backpropagating feature importance for networks designed using Pytorch [35]. By making an open-source version available we hope to foster experimentation and help fellow machine learning practitioners understand what is inside their black-box.

A. Architecture

Although our method can be applied to any neural network, the desired output of the explainer can vary wildly depending on the type of data being used by the model. We tried to take this into account while developing DeepDIVE and provide ways for users to easily retrieve explanations in several different formats. In order to accomplish this is as efficiently as possible we separated the logic of retrieving explanations from the delivery of those explanations. We constructed both an `_explainer` and an `_explanation` classes which are responsible for implementing generic behaviour. The behaviour required for creating explanations and for displaying those in a manner the user wishes for, respectively.

This root explainer class implements our interpretability methodology in a model agnostic way. At creation, the class receives a pytorch model and other configuration parameters. This class also implements `_generate_explanation` which is a model agnostic implementation of our method of deriving feature importance using backpropagation. We used Numpy [36] matrices as the root data type. This allows us to easily implement different methods that export explanations into different data types such as pandas [37] dataframes, pythons lists or dictionaries.

In order to facilitate the usage of our method on different domains we then created child classes of the `_explainer` and the `_explanation` classes. As of this writing, `numeric_explainer` and `image_explainer` have been implemented for numerical and image data respectively. After any of the explainer classes is instantiated using a pytorch model, the method `generate_explanation` can be called in order to generate an explanation for a prediction. These provide useful features for displaying the each respective type of data, this ranges from producing plots for feature importance in numeric data to creating saliency maps in images (reconnecting with the work by Simonyan et al. [19]).

This facilitates the integration of our module into existing work and allows easy experimentation.

VI. Conclusion

We have presented a method that can retrieve a linear representation of the causal relationship between the input features of a model and the model's decisions. Furthermore, our method can theoretically be applied to *any* neural network.

Although we only tested our method with feedforward networks, the backpropagation algorithm being used at the core of our approach is used in all popular types of networks used in a plethora of different areas. Furthermore, since our method relies solely on backpropagation, it is incredibly fast, easy to implement and scalable. We use one of the most robust features of deep learning frameworks to generate our explanations. This means anyone, anywhere can implement our method, meaning it is not a method to be used sparingly, but something that can easily be adopted by a wide range of machine learning practitioners.

On both problems where we tested our approach, the explanations generated by our methodology are incredibly faithful to the underlying models. The explanations we generated allow us to understand the local behaviour of the model and, in cases where visualization allows, they can also provide us with insights regarding the model's global behaviour.

Finally, we presented deepDIVE (Deriving Importance VEctors) an open-source implementation of our proposed method for generating explanations. We hope that by providing an open-source framework with support for numeric and image data we can foster experimentation and allow the largest possible number of people to reap the benefits of our research.

REFERENCES

[1] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," 12 2016.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[3] F. Dalvi, N. Durrani, H. Sajjad, Y. Belinkov, A. Bau, and J. Glass, "What is one grain of sand in the desert? analyzing individual neurons in deep NLP models," *CoRR*, vol. abs/1812.09355, 2018. [Online]. Available: http://arxiv.org/abs/1812.09355

[4] R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad, "Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1721–1730.

[5] B. Kim, "Interactive and interpretable machine learning models for human machine collaboration," Ph.D. dissertation, Massachusetts Institute of Technology, 2015.

[6] Z. C. Lipton, "The Mythos of Model Interpretability," *ArXiv e-prints*, Jun. 2016.

[7] M. Tulio Ribeiro, S. Singh, and C. Guestrin, ""Why Should I Trust You?": Explaining the Predictions of Any Classifier," *ArXiv e-prints*, Feb. 2016.

[8] N. Frosst and G. Hinton, "Distilling a Neural Network Into a Soft Decision Tree," *ArXiv e-prints*, Nov. 2017.

[9] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," *ArXiv e-prints*, Mar. 2015.

[10] O. Irsoy, O. T. Yildiz, and E. Alpaydin, "Soft decision trees," in *International Conference on Pattern Recognition*, 2012.

[11] D. G. Garson, "Interpreting neural network connection weights," 1991.

[12] J. D. Olden and D. A. Jackson, "Illuminating the "black box": a randomization approach for understanding variable contributions in artificial neural networks," *Ecological modelling*, vol. 154, no. 1, pp. 135–150, 2002.

[13] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing higher-layer features of a deep network," 01 2009.

[14] G. Hinton, S. Osindero, M. Welling, and Y.-W. Teh, "Unsupervised discovery of nonlinear structure using contrastive backpropagation," *Cognitive science*, vol. 30, no. 4, pp. 725–731, 2006.

[15] K. Kavukcuoglu, M. Ranzato, R. Fergus, and Y. LeCun, "Learning invariant features through topographic filter maps," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1605–1612, 2009.

[16] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[17] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev, "The building blocks of interpretability," *Distill*, 2018, https://distill.pub/2018/building-blocks.

[18] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K.-R. MÃžller, "How to explain individual classification decisions," *Journal of Machine Learning Research*, vol. 11, no. Jun, pp. 1803–1831, 2010.

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

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[21] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[22] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for Simplicity: The All Convolutional Net," *ArXiv e-prints*, Dec. 2014.

[23] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PloS one*, vol. 10, no. 7, p. e0130140, 2015.

[24] P.-J. Kindermans, K. Schütt, K.-R. Müller, and S. Dähne, "Investigating the influence of noise and distractors on the interpretation of neural networks," *arXiv preprint arXiv:1611.07270*, 2016.

[25] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje, "Not just a black box: Learning important features through propagating activation differences," *CoRR*, vol. abs/1605.01713, 2016. [Online]. Available: http://arxiv.org/abs/1605.01713

[26] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," *CoRR*, vol. abs/1703.01365, 2017. [Online]. Available: http://arxiv.org/abs/1703.01365

[27] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," *arXiv preprint arXiv:1704.02685*, 2017.

[28] D. Smilkov, N. Thorat, B. Kim, F. B. Viégas, and M. Wattenberg, "Smoothgrad: removing noise by adding noise," *CoRR*, vol. abs/1706.03825, 2017. [Online]. Available: http://arxiv.org/abs/1706.03825

[29] P. Lakhani and B. Sundaram, "Deep learning at chest radiography: automated classification of pulmonary tuberculosis by using convolutional neural networks," *Radiology*, vol. 284, no. 2, pp. 574–582, 2017.

[30] P. Rajpurkar, J. Irvin, K. Zhu, B. Yang, H. Mehta, T. Duan, D. Ding, A. Bagul, C. Langlotz, K. Shpanskaya, M. P. Lungren, and A. Y. Ng, "Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning," *CoRR*, vol. abs/1711.05225, 2017. [Online]. Available: http://arxiv.org/abs/1711.05225

[31] N. C, "On the prospects for a (deep) learning health care system," *JAMA*, vol. 320, no. 11, pp. 1099–1100, 2018. [Online]. Available: +http://dx.doi.org/10.1001/jama.2018.11103

[32] A. Teixeira, "Arisco: Recommendation system for risk analysis in mental health of children and adolescents."

[33] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[35] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[36] T. E. Oliphant, *A guide to NumPy*, 2006, vol. 1.

[37] W. McKinney *et al.*, "Data structures for statistical computing in python."