UNIVERSITY OF CAMBRIDGE

Department of Engineering

Techniques for Finding Salient Features in Neural Network Inputs

Author Name: Andrew Louw

Supervisor: Dr Adrian Weller

Date: 29/05/18

Afl33@cam.ac.uk Andrew Louw Robinson College

Techniques for Finding Salient Features in Neural Network Inputs

Abstract

This project aims to bring together several different techniques for determining the important features in Neural Network inputs. This project does this by creating a common framework of terminology which can be used to describe and analyse the techniques, as well as a coding framework which can be used to test different methods for direct comparison. In addition to implementing existing methods (LIME, VisualBackProp, Deep Taylor) a new method has been developed and investigated (GeoSal) as part of this project. The derivation for this new method is discussed in detail. The main deliverable for this project is a set of tutorials demonstrating the newly created class and the implementations of all the methods. Some of the main findings of this project are given below.

LIME (Ribeiro, Singh, & Guestrin, 2016) is a black box method which tests modified versions of the input image to produces a local explanation. This is slow but very flexible. The explanation produced identifies areas as evidence for or evidence against a class, making it very user friendly. This method is especially powerful when "features" can be easily isolated and deleted.

The white box analytical methods all produce explanations with different characteristics due to different interpretations for how the salient information is contained within the neural network. VisualBackProp is a quick method which shows which input pixels are not "blocked" by ReLU functions. This gives an explanation which shows the features which are consistently detected by convolutional filters in different layers.

Deep Taylor produces an explanation which is weighted by the most impactful pixels from each layer- this is done to keep the total information in each layer constant. The principles developed in this method can be applied to large networks containing many different types of layer- including newly developed types.

GeoSal uses a novel interpretation of "saliency" which gives it a unique way of explaining how susceptible a classification is to change in the input. The explanation produced highlights the areas of the image which are most sensitive to change, allowing the user to identify how close one input is to a different classification. This makes GeoSal useful to determine the important features to distinguish between two similar classes.

Finally, there is a group of methods which produce a model in the form of training a different "explainer" system, this involves a high up-front cost but can be investigated with the speed of a single network, one such technique has been investigated but is yet to be implemented (Dabkowski & Gal, 2017).

The coding framework created for this project is a single class "Mnist_net". The class has the capability to create, train and test networks as well as functions useful to various saliency methods. The different types of network required by different methods mean that the "Mnist_net" class is highly versatile. Additionally, the class was made to be user-friendly to keep the focus of the tutorials on the various methods as opposed to the running of the network. For greater clarity, the methods have been implemented in Jupyter Notebook cells. Each method has been tested in more than one way to demonstrate the flexibility of the framework and to analyse the strengths and weaknesses of the technique.

All analysis for this project has been performed on the MNIST dataset which has properties making it ideal to closely examine techniques. These properties include: a large amount of constant value area (white space), the similarity of various numbers, human interpretability after scaling, offsetting and inverting. Generating a classifier for the MNIST dataset is also much quicker and even small networks can achieve a high classification accuracy, allowing for many networks to be trained and compared. It is surprising to find that many techniques designed to work on full colour images can produce unhelpful explanations when tested on the MNIST dataset.

The codebase for the project, as well as tutorials demonstrating its use is available on GitHub (https://github.com/AndrewLouw/Saliency-Comparison) for free download and use. The code and tutorials have detailed annotations to aid users. The project can also be run through Microsoft Azure Notebooks (https://notebooks.azure.com/AndrewLouw/libraries/Saliency-Comparison). Both libraries include (edited) copies of some of the LIME codebase. The complete LIME codebase is also available separately (https://github.com/marcotcr/lime/).

Table of Contents

Introduction

Many machine learning applications are described as "Black Boxes" - the term referring to how the process of turning inputs to outputs can be very difficult to understand. This lack of understanding raises many issues in the real world, for example: ensuring accountability in critical or dangerous systems such as healthcare, finance and driverless cars. One use of these methods is to highlight where features in the training data has caused the network to disproportionally weight irrelevant details. For example, in Figure 1 a network was trained on data in which images of wolves contained snow, and images of a Huskies did not.

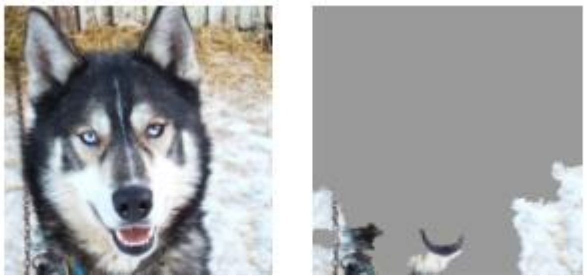

Figure 1: Left: Husky classified as wolf. Right: explanation (Ribeiro, Singh, & Guestrin, 2016)

As will be discussed later, this problem has also been observed with the MNIST dataset, where many explanations put high importance in areas of the image which are invariant throughout the training data (e.g. the corners are almost always white).

This leads to the generation of two sets of stakeholders:

1. Researchers and developers: When testing and creating the system there may be unknown biases or inaccuracies present in the training and validation sets which could lead to poor results when deployed. Examples of this might be found where simulations are used for training before deployment in the real world – the simulation can have a feature not present or less noisy than those found in the real world.
2. End users: Not all machine learning applications are used in isolation as the sole decision maker, many are used to guide professionals by offering a result to be confirmed; such as in healthcare. In these situations, it is vital to know in more detail about why the result was given. This can also help to identify potential opportunities for gaming the network (Weller, 2017), such as the real-

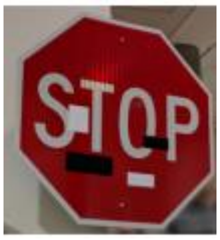

Figure 2: Covering specific patches of this stop sign fooled the classifier causing it to make incorrect classification. (Eykholt, et al., 2018)

world attack shown in Figure 2. If these flaws are known then end users can make an informed decision on when to ignore the classifier.

Fortunately, both sets of stakeholders require the same or similar solutions. It should be noted that transparency is not without downsides – methods for interpreting classifiers can be used in the generation of adversarial examples and potentially leak data about the training set (a privacy concern) (Weller, 2017).

There have been many unique attempts to create methods for saliency in image classifiers. These methods have often been developed in isolation. They use different networks trained on different datasets and classify different images in their examples. For proper comparison it is important to have a uniform terminology and controlled set of tests. Ideally multiple end users would perform blind tests to confirm the usefulness of different explanations (Doshi-Velez & Kim, 2017).

This project aims to bring four methods together in a single place, to allow users to compare many different techniques while using the same network and test images. Each method is discussed with consistent terminology and the strengths and weaknesses of each method are discussed regarding both the theory behind the technique and the practice of its execution. Finally, the forms of the explanations are compared and advice is given for when to use each method. To ensure the project is completed to a high quality, the scope has been limited to focus on neural network image classifiers. The code library has been designed such that it may be expanded at a later date to include other types of machine learning applications or other types of inputs. The main focus of this project is to ensure ease of use within a python environment, this is demonstrated in the tutorials supplied in section 2.

To help analyse networks quickly but also with a high level of control a new class was created for this project - “Mnist_net”. This class includes all the functions required to create and train a Convolutional Neural Network or a Multi Layered Perceptron Network both using ReLU non-linearities. The class also includes several functions for plotting different features of the network as well as functions useful to some of the methods investigated. The class uses the TensorFlow library for generating, training and running the network and is also able to execute commands from the TensorFlow library. The project has been developed on a Windows 10 laptop in python 3 and a set of Jupyter Notebooks with simple tutorials has been created, these explain how to use the Mnist_net as well as each method. All references to code run times are from a localhost Jupyter Notebook running on this machine, validation results may vary.

For this report the same test image will be used for all example explanations, the image is shown in Figure 3, and unless otherwise stated it has been correctly classified as a four by the classifier being investigated. This four was chosen because it can be easily converted into a nine as shown in Figure 4; it is hoped that some of the explanations reference this feature in some way.

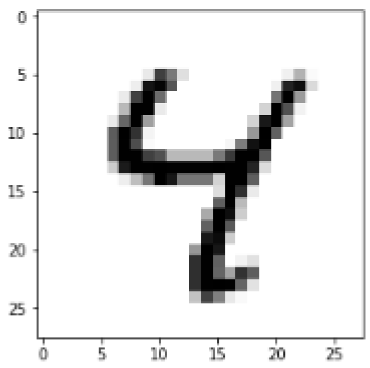
Figure 3: The test image, true class is a four.

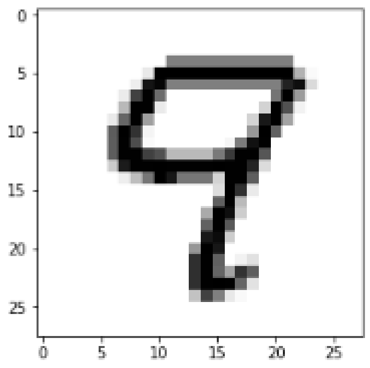
Figure 4: The test four after a simple conversion into a nine.

Terminology:

Each saliency method can be described as either a "black box" or a "white box" method:

- Black box methods only need to have access to the classifier's inputs and outputs (output taken before the final max-pool is included in this class). In general, they work by testing many different inputs to determine the relative importance of features or to train an explainer network. Black box methods tend to be slower due to the many samples required but have greater flexibility.
- White box methods require access to the whole network, they usually have a mathematical basis for converting weights, biases and the structure of the network into a method for weighting feature importance. This can mean that white box methods are able to be calculated almost as quickly as running the network once but each application may need to be tailored to a specific network – this is a large programming overhead.

The definition of an explanation is most clearly given in the paper describing LIME (Ribeiro, Singh, & Guestrin, 2016). It states that an explanation is the **descriptor**[1] which has the best trade-off between describing the underlying model's predictions in the locality of interest (the accuracy of using only the descriptor to make the same prediction as the model, weighted by how similar the input is to the one being explained) and the simplicity of the descriptor:

$$\xi(x) \;=\; \underset{g \in G}{\operatorname{argmin}}\, \mathbb{E}[L(f,\, g,\, \pi_x)] \;+\; \Omega(g)$$

[1] Here a descriptor is given to mean anything which can describe the model irrespective of complexity or accuracy, for example a network is its own descriptor but so too is a flowchart, an image mask, a linear model etc.

This will return the descriptor, ξ, which has the lowest expected unfaithfulness cost, L and complexity cost Ω out of all possible descriptors G.The unfaithfulness cost can be calculated by taking samples within the input space and comparing the output of the descriptor being tested, g, to the output of the original model, f, weighted in some fashion by how close the sample is to the locality of interest, π_x. The form of the complexity term is chosen by the user which usually sets the format of the explanation. For this project the **explanation** must be of the form of a single heatmap, else it has infinite complexity. Other forms of explanation have been used (Olah, Mordvintsev, & Schubert, 2017), but comparing the methods directly is only feasible if the methods have explanations in the same form. Intermediate descriptors used to produce the explanation are referred to as **models**. A descriptor optimised for a specific region of the input space can be described as **local**. For example, a linear model could be used to explain a single instance very accurately but away from the region of that instance it becomes significantly less accurate. This would be a local model as demonstrated in Figure 5.

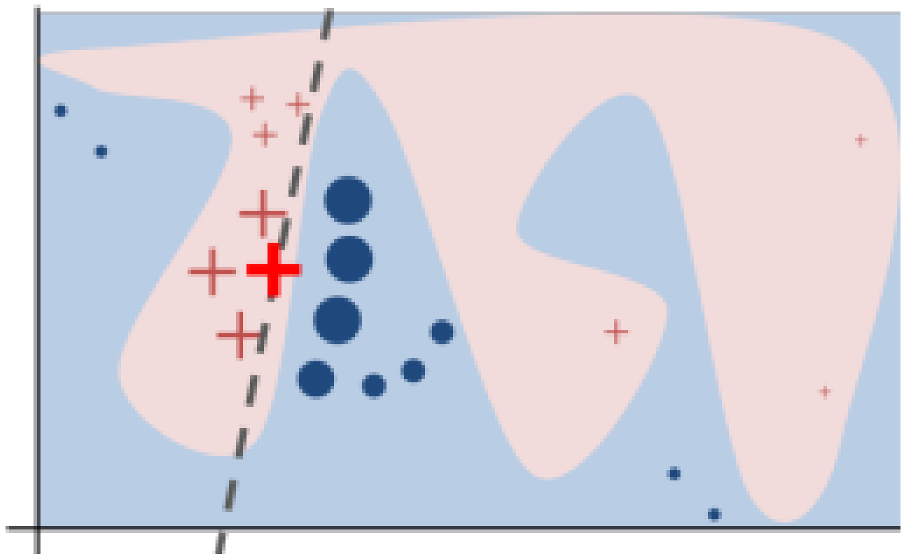

Figure 5: An example of a linear classifier being used as a local model to explain the bold red cross. The network it models separates the input space into the pink and blue areas. The expected unfaithfulness of the linear mode is found by taking samples- the red crosses and blue dots. The samples are weighted by their distance from the bold red cross (shown by the size of the cross or dot). (Ribeiro, Singh, & Guestrin, 2016)

This project found that this leads to two types of global descriptor:

- A **weak global** descriptor is one where the weighting is invariant to the locality of a given input. By reducing the complexity by less than the increase in error, a descriptor different to the original network is obtained, giving $\xi(x) = \operatorname*{argmin}_{g \in G} \mathbb{E}[L(f, g)] + \Omega(g)$. For example: a decision tree of a certain depth could be used to describe the whole network, optimised to provide the minimal deviation from to the original classifier across the whole input space.

- A **strong global** descriptor is the same as a weak global but with the additional constraint that it must **always** make the same classification as the original network: $L(f, g) \in \{0, \infty\}$. This means that it only differs from the network if there is reduced complexity. For example, if the depth of the decision tree used above was increased such that the deviation from the original classifier is zero across the whole input space.

In general, it is difficult to produce a weak or strong global descriptor that is sufficiently simple to understand. However, it is possible to produce a globally optimised model and subsequently simplify that model to produce a local explanation specific to a given input. For example: a non-linear surface could be modelled by a series of linear approximations, which collectively are still too complicated to understand. These approximations form a weak global model – simpler than the true surface and optimized for the whole input space, but only approximately correct. If, when a user queries a region of space, only the relevant equation is supplied – this is a local model derived from a weak global one. The same can be done in reverse, where a weak global model is generated from a series of local ones (Ribeiro, Singh, & Guestrin, 2016).

Section 1: Analysis of Techniques

Local Interpretable Model-agnostic Explanation (LIME) (Ribeiro, Singh, & Guestrin, 2016)

Methodology

The first method investigated for this project was LIME. This method attempts to generate a **local** explanation directly from sampling around the input image. It is a **black box** model with a powerful ability to work with any network as long as the network outputs are of the form of continuous probabilities for each class i.e. a network which simply outputs the class in a 1 hot vector would not be a good candidate. Next random combinations of features are deleted, by noting how the output vector varies when different features are deleted, the relative importance of each super-pixel is determined for member of the vector. By doing this many times with many combinations of features the average importance of a features can be found. When analysing images, the features are determined by collections of similar, connected pixels; these are referred to as "super-pixels". The for images algorithm becomes:

1. Test the classifier first on the original image
2. The image is broken into super-pixels using the quick-shift algorithm (Vedaldi & Soatto, 2008)
3. Create a new image with randomly deleted super-pixels – more than one super-pixel can be deleted at a time. Deletion is done with one of many options, discussed below.
4. Test the new image and distribute the change in output probability across the deleted super-pixels. This is done separately for each output being investigated.
5. Output: a heatmap showing the importance of each pixel for each class. This can be simplified to be in terms of evidence for, against and neutral for a chosen class.

The method calculates the importance for all the top "n" classes at once ("n" determined by user) and stores them in an explainer class. Depending on the number of samples requested, this can take a long time to create. To get the heatmap for a specific class, a different function is used – this means that the results can be quickly analysed in detail for many classes. For example: looking at the difference of evidence between two classes can be done very easily.

In the terminology described above this is equivalent to creating a local explanation which is more complicated than desired (it contains several images) but can easily be interrogated to provide an explanation at the required level of detail. The paper also discusses producing a weak global model from many local samples which has not been reproduced for this project.

Implementation

The code for the LIME algorithm is available on GitHub (https://github.com/marcotcr/lime/). The code is of high quality and largely includes appropriate commenting and variable names, however some functions needed altering to be optimized for this project**:**

- The super-pixel code had a hardcoded value for the maximum super-pixel size. This value was appropriate for the ImageNet dataset - which is made from larger, coloured images. However, with the original super-pixel size, the large areas of white in a MNIST image become a single super-pixel which makes the LIME explanation less useful. The kernel size parameter is not sufficient to force the super-pixels to be only one pixel large. To allow detail at the level of individual pixels the maximum super-pixel size was included as an input.
- The MNIST dataset is largely black and white, hence masking a super-pixel with the average of its values is likely to produce no change. Masking with white is likely to produce different results to using black. Using grey did not seem appropriate since the network was never trained with prevalent grey values. With this in mind a new masking type was created: the 'inverse' method for masking (mask a pixel, x, by setting its value to $1 - x$). This new method was investigated alongside masking.
- The LIME codebase is designed for RGB images but the MNIST dataset is greyscale, for ease of use the Mnist_net class automatically detects and reformats an RGB image into a greyscale before analysis. This means that any image should be converted to RGB before running the LIME analysis.
- The maximum fraction of super-pixels to change was also hardcoded into the original code. With the inclusion of pixel level detail, the original value turned out to be too high. Consequently, this has been changed to a user defined input.
- The LIME output mask turns all evidence into binary positive and negative. It does not give the strength of the evidence, but this data is available in the explainer class.

Test Results:

Choosing appropriate parameters for the LIME explainer is crucial for understanding how to interpret the results and also can have a large impact on the number of samples required. When a small super-pixel size is used it is important that "num_samples" is large enough to ensure a representational combination of super-pixel deletions is tried. This can require an even larger number of samples if the change fraction is also high. Optimal results would be achieved by cycling through every combination of super-pixel deletion. However, running this algorithm with pixel level detail for the 28x28 pixel dataset would require $\sum_0^{784} 784\mathbf{P}n$ samples. LIME offers an acceptable compromise between the insurmountable cost of this holistic analysis (which is also only a local explanation) and speed of execution. In the examples given in the Tutorial 2 the time taken was approximately 6minutes 15 seconds. Deeper networks will take longer since the time taken for the execution of the algorithm is roughly the same as the time to run the network "num_samples" times.

With a larger super-pixel size this method is good for quickly finding object location in images and can be run with a much higher change fraction. The change fraction should not be less than $(\text{number of super-pixels})^{-1}$ as this will often produce a test image with no changes from the original.

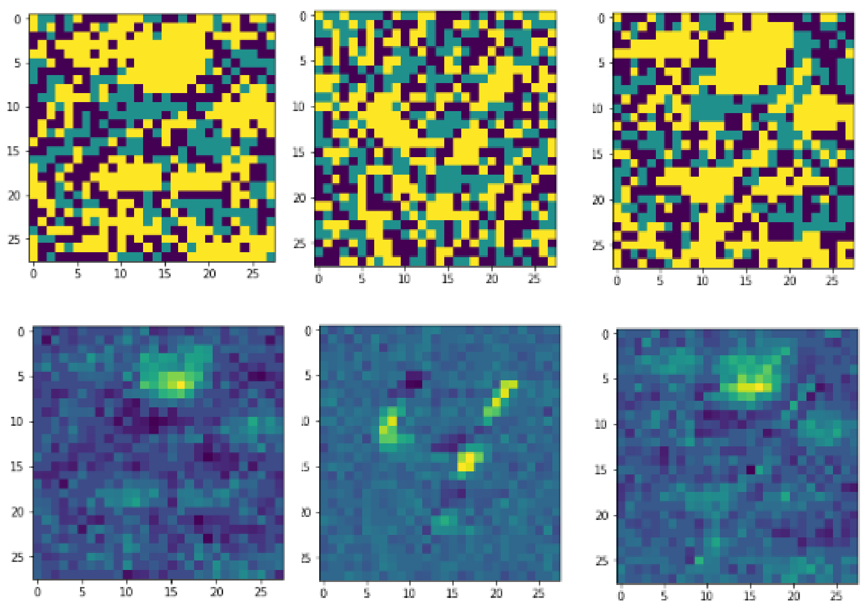

Figure 6: Top: the binary heatmaps. Bottom: the continuous heatmaps. Left: using the "mask" method with a black mask. Mid: using the "mask" method with a white mask. Right: using the "inverse" method. In the binary heatmaps: yellow means evidence for, light blue means evidence against, and dark blue means the pixel is unimportant. In the continuous heatmaps the scale is linear from yellow (evidence for) to dark blue (evidence against).

The type of deletion technique is also very important to how the LIME algorithm interprets the importance of a super-pixel. As can be seen in Figure 6, masking with black is very different to masking with white, while inverting has some of the characteristics of both. Masking with black and inverting highlight the similarity to a nine, however the masking with white is only able to find importance in the black areas of the test image. With the MNIST dataset the inverse method is clearly the most useful. This is because it takes advantage of human understanding of the real-world input options. For a street sign classifier used in self driving car research it may be more useful to use a brown translucent mask to mimic the effect of dirt on the sign, or a mask to mimic the effect of water blurring the camera lens in one place.

It is also possible to look at the difference in outputs as described in the original LIME paper. This can be done by subtracting one heatmap from the other – as shown in the tutorial. Forming an explanation from this type of analysis is discussed in more detail later.

VisualBackProp (Bojarski, et al., 2017)

Methodology

This is a **white box** method which can be applied to convolutional neural networks which use ReLU non linearities without biases. The method identifies the features in the input which cause the largest values going into the final fully connected layer. A **weak global** model is used to describe the flow of information in a convolutional neural network. This model is implemented with a specific input to provide a local explanation of saliency in the form of a heatmap.

The weak global model takes advantage of the ReLU function being exactly zero for inputs below zero, this means that any input which leads to an output of less than zero cannot propagate any information further through the network. To increase the speed of the algorithm the averages of each layer in the convolutional network are used to approximate the amount of information capable of propagating further through the network.

The algorithm:

1. First take the averages of the outputs of every layer (n layers)
2. Convolve the nth average output with a unity filter of the same size as the (n-1)th convolutional filters. This performs a deconvolution, ensuring any of the inputs to the (n-1)th layer which could propagate information to a pixel in the nth layer are given a proportion of the information in that pixel.
3. Pointwise multiply the convolved nth average with the (n-1)th average.
4. Go to step 2 with n ← n-1 until the last layer has been reached.

Implementation

The description of the implementation in the original paper is sufficient to implement this method. The Mnist_net function "average_output" was created specifically to aid with VisualBackProp; this function performs the averaging of each layer. For the deconvolution a function from the SciPy library was used. Using these tools, the method was recreated in a notebook cell.

Test Results:

One feature of this analysis method is that, with no biases, any input pixel starting with a value of zero will be unable to propagate information through the network at any stage. This would imply that there is no informational value in a white pixel. However, the LIME analysis would suggest that it can be important when a region contains white space. Additionally, for any randomly initialised network of this form the darker pixels are more likely to propagate information throughout the network which could lead to misleading attribution of importance, especially with the MNIST dataset.

To avoid these problems caused by the white space having a value of zero, the offset input was added to the setup function to create and train a different network. This creates an additional layer at the front of the neural network which applies a constant offset to the image. For example, using offset = -0.5 shifts any MNIST image from the range {0,1} to the range {-0.5,0.5}. Since initialised weights can be positive or negative with equal probability, we get a random saliency map for a random network and a saliency map covering almost the whole image for the trained network. This was also compared to a network trained on inverted images.

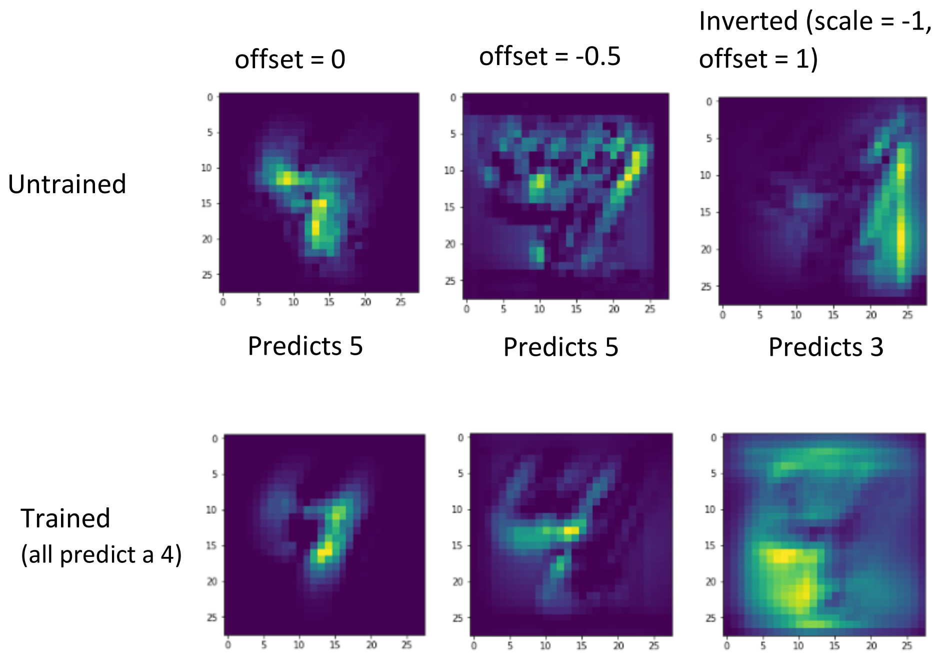

Figure 7: Testing the method with different input and network properties. Without the offset, untrained models can provide similar outputs to trained models. With the offset it is more obvious that the untrained model is attributing the prediction to random parts of the image. The network trained on inverted images cannot show the information in the pixels which make up the four.

As can be seen from the images in Figure 7, for a network of the type necessary for VisualBackProp, the input space can be very important even for random networks. For the best results it is useful to have all input values capable of propagating information through the network by using an offset. The offset and scale options did not have a noticeable impact on the capability of the network (the trained networks have very similar accuracies of ~98%). In the explanation for the untrained models it is possible to see the danger of relying on this technique when not using an offset – the attribution of importance will always be biased towards larger values (black pixels).

The network with 0 offset is identical to the one used in the LIME analysis, and it is interesting to note that the explanation closely matches the LIME explanation when the white mask is used. The offset trained network is an attempt to take advantage of the significance of whitespace, as with the “inverse” method in LIME, and the results are similar in that it is found that a white region is significant in this explanation – this time in the bottom left. The offset method shares some of the main features of both the other networks’ explanations, suggesting that it is best to train networks with an offset if using this method.

This type of saliency is appealing due to the direct analysis of the network – it would allow a user to identify features which are being ignored by the network making it also good for object localisation. The time taken to run this algorithm is significantly less than LIME, of the same order as the time taken for one pass of the network. However, the network does have a fully connected layer after the final convolutional layer, with the potential to vastly redistribute the significance across any of the non-zero values. Additionally, it has been suggested (Kindermans, et al.) that this genre of analytical techniques will always give misleading results. This is theorised to be due to the input comprising of a signal and a distractor- hence any classifier’s main task is filtering out the distractor as opposed to identifying the signal. Finally, the constraints placed on the network significantly hamper their power in more complicated applications, making it difficult to find applications for this method.

Deep Taylor (Montavon, Bach, Binder, Samek, & Muller)

Methodology

The Deep Taylor technique is also a **white box** method which makes use of a local representation of a **weak global** model. The method models the network’s output as ‘relevance’. The sum of all the relevance at each layer must remain constant throughout the network for any given input, and so in order to find the relevance in the nth layer the relevance in the (n+1)th layer must be redistributed. The method proposes a technique for redistribution based on the gradient of a ReLU function and a Taylor approximation. In order to calculate the first order Taylor approximation at the given input position it is necessary to find a root point for the input. The root point is defined as a point which produces a relevance 0. This comes from the Taylor expansion:

$$f(x) = f(a) + f'(a)(x-a)$$

Which for $f(a) = \sum_j R_j^{(n)} = 0$ becomes:

$$\sum_j R_j^{(n)} = \left(\left. \frac{\partial \sum_j R_j^{(n)}}{\partial \{x_i\}} \right|_{\{\tilde{x}_i\}} \right)^T \cdot (\{x_i\} - \{\tilde{x}_i\}) + \varepsilon$$

Where R_j^n is the jth component of the Relevance from layer n, and $\{\tilde{x}_i\}$ is the ith component of the root input corresponding to $\{\boldsymbol{x}\}$. Since there exists an error term this model is not guaranteed to be exactly true to the network for all values and hence the model is only **weak global**.

The root point is found by applying the analysis from the Deep Fool algorithm (Moosavi-Dezfooli, Fawzi, & Frossard, 2016) - which (in this case) is designed to find a point as close to the supplied input as possible which will give an output of zero. For the ReLU non-linearity this is equivalent to solving:

$$\{\tilde{x}_i\}^{(j)} = \left\{x_i - \frac{w_{ij}}{\sum_{i'} w_{i'j}^2}\left(\sum_i x_i w_{ij} + b_i\right)\right\}$$

From the Deep Fool analysis, it can be shown that this becomes the relevance propagation rule (dubbed the “ w^2 rule”):

$$R_i^{(n)} = \sum_j \frac{w_{ij}^2}{\sum_{i'} w_{i'j}^2} R_j^{(n+1)}$$

While powerful in this form, the method can be further optimized by further considering the ReLU function. The output of a ReLU is always positive (or zero) and so the root implied by using the “ w^2 rule” may not always be valid. In order to restrict the search domain, the rule is adapted to form the “ z^+ rule”:

$$R_i^{(n)} = \sum_j \frac{z_{ij}^+}{\sum_{i'} z_{i'j}^+} R_j^{(n+1)}$$

Where $z_{ij}^+ = x_i w_{ij}^+$ and $w_{ij}^+ = \max(0, w_{ij})$. This rule can be further extended for any input space which has both upper and lower bounds: $l_i \le x_i \le h_i$ (a space referred to as ***B***). This rule is particularly important for the input layer of the MNIST dataset which is bounded between 0 and 1 and so the “ $z^{\boldsymbol{B}}$ rule” arises:

$$R_i^{(n)} = \sum_j \frac{z_{ij} - l_i w_{ij}^+ - h_i w_{ij}^-}{\sum_{i'} z_{i'j} - l_{i'} w_{i'j}^+ - h_{i'} w_{i'j}^-} R_j^{(n+1)}$$

Where $z_{ij} = x_i w_{ij}$ and $w_{ij}^- = \min(0, w_{ij})$. By combining the z^+ and $z^{\boldsymbol{B}}$ rules an algorithm was devised for analysing a Multi Layered Perceptron network.

The algorithm

1. Set relevance to the last layer output
2. Redistribute relevance to the previous layer using the z^+ rule

3. Repeat step 2 until on the final layer
4. For the final layer use the $z^{\boldsymbol{B}}$ rule

Implementation

Again, no code was supplied for this method, so the algorithm was reproduced in python. This was done by creating the function "Deep_Taylor" which takes as its inputs:

- A Mnist_net network: the network to be analysed
- An image: the image to be analysed
- A prediction (optional): if given a value (0 to 9) then only the relevance pertaining to that class will be back propagated.
- Boolean "Suppress_out": if true the image mask produced will not be displayed

This implementation will always use the z^{+} rule, except for the final layer when it will always use the $z^{\boldsymbol{B}}$ rule. The $z^{\boldsymbol{B}}$ takes into account the network offset and scale to form the correct bounds.

Test Results:

This method is based on analysis of the network and the ReLU function. The underlying network model does not make assumptions or large approximations (such as the averaging in VisualBackProp) and is versatile in the network features it can accommodate. It can be shown that a Convolutional Network without pooling can be converted to a Multi-Layer Perceptron Network as a **strong global** model. For each layer this is done by creating an output for every pixel in the next layer and placing the convolutional weights responsible for that output into the appropriate places in the weight matrix, leaving the rest 0. The resulting network is often very wide having many 0 weights, and many repeated weights. The pooling functions can then be re-added by pooling the sets of neurons corresponding to each output image. These pooling functions, which might prevent

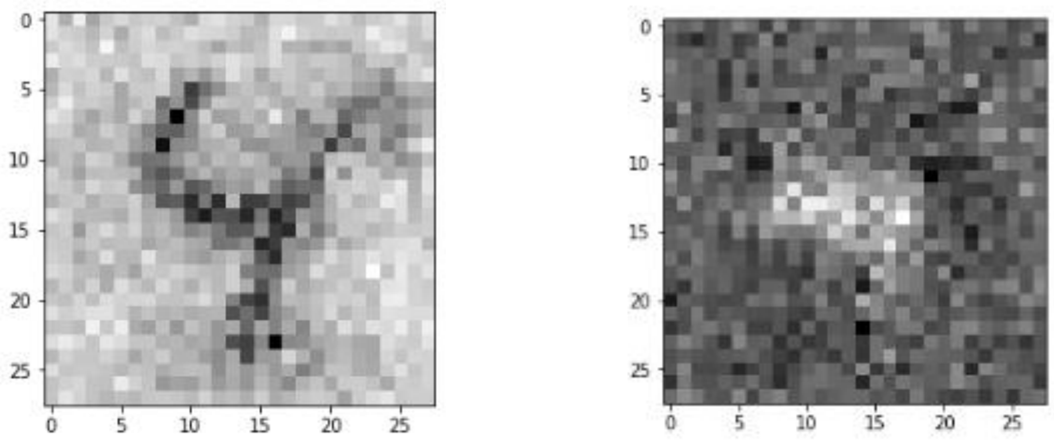

Figure 8 The explanation for two multi-Layered Perceptron networks. The networks have 4 layers and do not use any pooling (currently pooling with MLP networks is not supported). This results in relevance being spread thorough the network in a noisy way. Left has an offset of -0.5, right has been inverted.

other saliency methods being used on both types of network, are not a problem for Deep Taylor as the relevancy is simply distributed equally to each max output according to their activation and weights. Hence this method is able to analyse very deep and complicated networks with a high degree of certainty in the result.

However, when used on networks without pooling, Deep Taylor produces results which can be difficult to interpret as shown in Figure 8. Even on a network trained with an offset of -0.5, the dark regions are still much more important than the light, and it is difficult to make a statement about which features are important in the input image. The inverted network does not have any features which can be described in plain language.

The time taken to run is increased by wide networks as the propagation term is not easily vectorised. This means that the method is not fast enough to be real time (~19 seconds) but it is still fast enough to be useful in many applications, for example: with networks which use a mixture of different layer types. Many methods are only applicable for a specific layer type and it can be difficult to maintain consistency when changing from one type to another. The Deep Taylor method creates a model which is invariant of layer type and as a result has much better potential for scaling.

Finally, the results in the original paper suggest that this method is also useful for object localisation making this method one of the most versatile saliency methods investigated.

New method - Geometric Saliency (GeoSal)

Background

GeoSal is a **white box** method which generates an explanation for the predicted class by combining the explanations for why the other classes were not predicted. The explanation for why a class has not been predicted is taken to be the difference between the test image and the closest image to the test image which gives the prediction of that class. The method employs a **strong global** model which is used to form a local explanation when given a test image; the process of producing the explanation requires several approximations which reduce the confidence a user can have in the result but vastly increases the calculation time (~5 seconds). The explanations produced by this method demonstrate how robust the test image is to change. This is done by showing how to alter the image to give a different classification with the least total change (L2 distance).

This method was created specifically for this project: the inception, analysis, implementation and testing were done independently from other work in this area. However, further research uncovered the "Deep Fool" analysis (Moosavi-Dezfooli, Fawzi, & Frossard, 2016) and a much simpler weight analysis (Simonyan, Vedaldi, & Zisserman, 2014); both these papers verify some of the mathematical analysis used in GeoSal - although neither form a competitive saliency method on their own. The method proposed includes the ability to create adversarial examples much closer (L2 distance) to the original image than was achieved in the paper "Explaining and Harnessing Adversarial Examples" (Goodfellow, Shlens, & Szegedy, 2015). Additionally, the output class of the adversarial example can often be chosen (although there are cases where it will not be achieved).

Deep Fool (Moosavi-Dezfooli, Fawzi, & Frossard, 2016)

If input x produces an output where class P has the highest probability; Deep Fool can be used to find another image x' such that the probability of class P is 0; additionally, x' will be as close to x as possible (according to the L2 distance), such as in Figure 9.

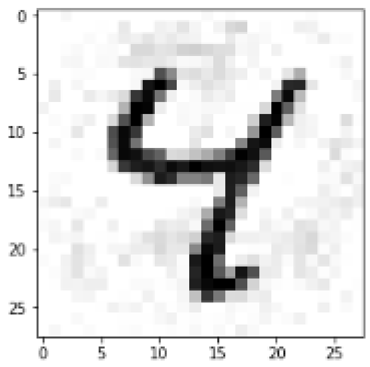

Figure 9: The deep fool result. This image has almost zero predicted probability of being a four.

Weight based saliency (Simonyan, Vedaldi, & Zisserman, 2014)

A Multi-Layered Perceptron network using ReLU non-linearities can be exactly converted to the form $xW + B = y$, where W is a matrix and B is a vector. Specific values of W and B are only valid for a small subset of inputs. The method uses the columns of W to provide insight into the network's function. The method for decomposing the network is a **strong global** model, but as shown in Figure 10, it can be difficult to use this method to explain a given prediction.

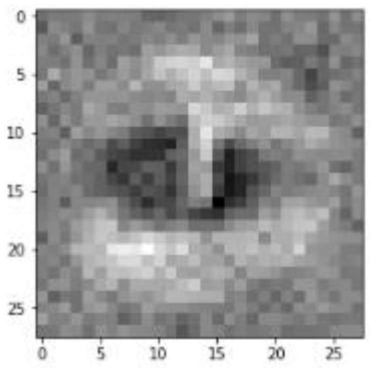

Figure 10: The weight matrix for the four class.

GeoSal Methodology

Consider a classifier which produces output of class P from image x; the classifier will always give an output (y) and the maximum output is the predicted class. This means that for a binary classifier (classes P and Q) the explanation for predicting class P must be identical to the explanation for not predicting class Q:

$$\text{Explain}(\text{argmax}(y) = P) = \text{Explain}(\text{argmax}(y) \neq Q)$$

For an N class classifier with output classes Q_n this rule can be extended to:

$$\text{Explain}(\text{argmax}(y) = P) = \sum_{Q_n \neq P} f(\text{Explain}(\text{argmax}(y) \neq Q_n))$$

Where the function $f(\cdot)$ allows the explanations to be added in a meaningful manner. Most methods discussed so far have attempted to approximate the left-hand side of the equation (these can be referred to a "**additive**" explanations), but GeoSal aims to approximate the right-hand side (a "**comparative**" explanation); LIME is capable of both. This leads to a **white box** saliency method for Multi-Layered Perceptron networks using ReLU non-linearities. The method uses a **strong global** model to reimagine the way networks of this type work; this model is then interrogated to give a local explanation. Unlike previous methods (VisualBackProp, and Deep Taylor) the errors incurred to allow for fast calculation come in the interrogation stage and not the model stage.

For this technique the explanation for the reason why a particular class (Q_n) is not chosen comes from finding the nearest image (x') to the test image (x) for which $y_p = y_{Q_n}$. The **explanation** is then given by $x - x'$. A more useful metric for saliency is the absolute **importance** of input pixels which is given by $|x - x'|$; i.e. the pixels which must change the most in order to find the closest adversarial example are the biggest distinguishing factors between the two classes. Using importance gives $f(\cdot) = \text{abs}(\cdot)$, which means that, if different explanations require a pixel value to change in different directions these changes will be added instead of cancelling out.

Finally, the decisions between all pairs of outputs are not necessarily equally important. With this in mind the absolute importance should be weighted before being summed. There are many ways to do this, one of the simplest is to weight by comparison class' output probability – this has the advantage that for N class classifiers, eventually the probability becomes insignificant and the algorithm can end without checking every class. Using this form gives $f(\cdot) = y_n\text{abs}(\cdot)$.

Closest adversarial example - Single layer analysis

Start by considering the decision boundary between two different choices in the final layer:

$$(1) \quad \boldsymbol{x} \cdot \boldsymbol{w} + \boldsymbol{b} = \boldsymbol{y}$$

Where **x** is an m dimensional input vector, **w** is an m by n weight matrix and **b** is an n dimensional bias vector, this will produce an output vector **y** which is n dimensional. If only considering the difference between the prediction ($\boldsymbol{P}$) and a chosen comparison ($\boldsymbol{C}$), this can be simplified:

$$\boldsymbol{x} \cdot \boldsymbol{w_P} + b_P = y_P$$

$$\boldsymbol{x} \cdot \boldsymbol{w_Q} + b_Q = y_Q$$

$$\boldsymbol{x} \cdot \left(\boldsymbol{w_P} - \boldsymbol{w_Q}\right) + b_P - b_Q = y'$$

$$(2) \quad \boldsymbol{x} \cdot \boldsymbol{w}' + b' = y'$$

Where $\boldsymbol{w_c}$ is column c of the matrix w.

This formula has the property that when y' is greater than 0 the model is more likely to predict the class P, and when y' is less than zero then class Q would be predicted. In this sense $y' = 0$ can be considered the equation for the decision plane between classes P and Q; note that $\boldsymbol{x} \cdot \boldsymbol{w}' + b' = 0$ is the equation of a plane.

This is true of the original model in all cases except the case when both values y_p and y_Q are negative – in this situation the ReLU in the model would ordinarily set both values to zero, and it would appear that the input lies on the decision plane when in fact it does not. This final ReLU does not matter, if both results are less than zero then the input would already provide equal probability, despite not being on a decision plane.

ReLU on input to layer

Now that the decision plane has been found, consider the ReLU acting on the previous layer's output (dimension m) – ordinarily a ReLU is a non- linear function making it difficult to analyse. However, a ReLU can be described as a matrix multiplication with a special $m \times m$ matrix, **R**. The matrix **R** has a 1 in the leading diagonal only if the matrix it is acting on has a positive value in the same column, all other values are zero. For example:

$$\boldsymbol{x} = \begin{bmatrix} 1 & -2 & 3 \end{bmatrix}$$

$$\boldsymbol{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Hence:

$$\boldsymbol{x} \cdot \boldsymbol{R}_{\boldsymbol{x}} = [1 \quad -2 \quad 3] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [1 \quad 0 \quad 3]$$

This means that there are $m\mathbf{P}m$ (permutations) possible **R** matrices, and any given input will be valid for at least one of them (if there is a zero in the output then more than one will be valid). The matrix being acted on in this analysis is always the input of a layer and so the subscript is dropped for the standard layer superscript (as in equation 3).

Consider the example with only 2 input neurons (so the decision boundary is a line not a plane):

$$(3) \qquad [\boldsymbol{x_0} \quad \boldsymbol{x_1}] \cdot \boldsymbol{R^1} \cdot \boldsymbol{w}' + b' = 0$$

This is represented in Figure 11, which gives the decision boundary shown in black (for some arbitrary $\boldsymbol{w}'$ and b'). The output (y') can be thought of as the distance from the black line in the positive region, after points have been mapped into the positive region. This allows the yellow dashed boundaries for -$\boldsymbol{x_0}$ and -$\boldsymbol{x_1}$ to be included although it is important to remember that the points in these regions get mapped to the positive zone, these boundaries only exist in the input space as markers for points which, when mapped, will end up exactly on the decision boundary.

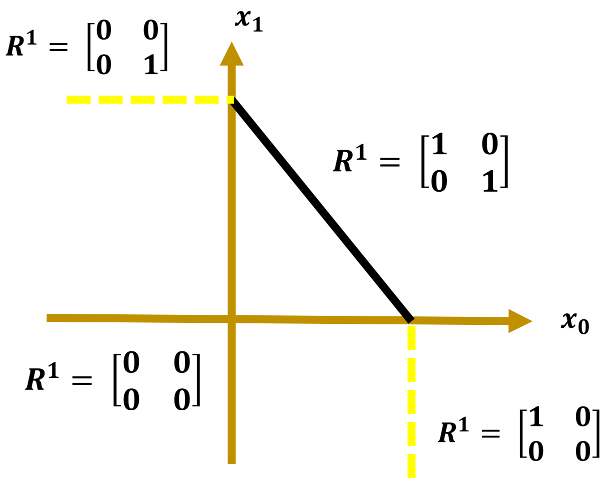
$\boldsymbol{R^1} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$
$\boldsymbol{x_1}$
$\boldsymbol{R^1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
$\boldsymbol{x_0}$
$\boldsymbol{R^1} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
$\boldsymbol{R^1} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$

Figure 11: A graphical representation of the ReLU function in a simple single layer classifier.

For any point $[x_0 \quad x_1]$ the output can be determined by simply finding its position relative to the boundary in Figure 11. Note that by the definition of y' as the predicted class minus the comparison class, any original input being examined must be lying above (or to the right) of the decision boundary and so, there being no boundary associated with $\boldsymbol{R}^1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ does not pose any concerns. By simple geometry it can be determined that the point on the boundary closest to any

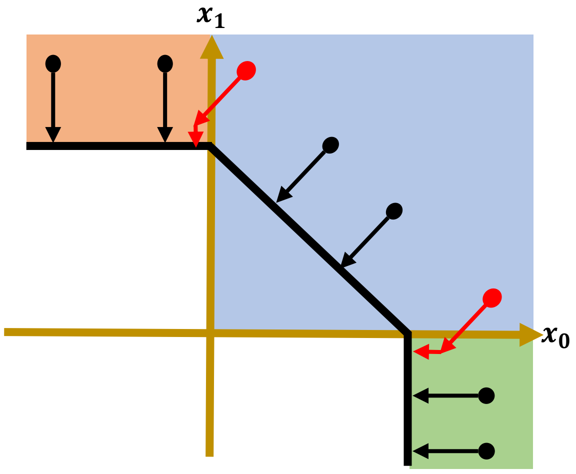

Figure 12: Each shaded region represents a different R matrix. Many points can be moved onto the decision boundary by moving perpendicular to the boundary generated with the same R matrix as the point. The red points require iteration.

input given above the line will be same as the closest point on the boundary which lies in the region defined by the same $\boldsymbol{R}^1$ as the original point as shown in Figure 12.

A general formula for finding this point comes from the equation for finding the closest point to a plane. First the perpendicular distance (*d*) can be found:

$$(4) \quad \boldsymbol{d} = \boldsymbol{b}' + \boldsymbol{x} \cdot \widehat{\boldsymbol{w}'}$$

And then the point can be moved onto the plane:

$$(5) \quad \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{d} \cdot \widehat{\boldsymbol{w}'}$$

This method will not work when the closest valid point cannot be reached by perpendicular movement (the red arrows in Figure 12). In these cases, the point found will lie inside a different **R** zone – running the search a second time starting from the found point will now find a new point in this new region. This final point will be non-optimal[2], but it is found very quickly. This solves the problem for a network with a single layer- but the technique can be scaled to work for a deep network.

[2] Another option is to find the intersection of the two hyperplanes and find the closest point on this line, however this does not scale as well with more dimensions

Closest adversarial example – extended to N layers

First consider a 2-layered network with the first layer having m nodes and the second layer having n nodes (where it is assumed m≥n). Following the earlier notation this has a decision boundary which can be expressed as:

$$(6) \quad (\boldsymbol{x} \cdot \boldsymbol{w^0} + \boldsymbol{b^0}) \cdot \boldsymbol{R^1} \cdot \boldsymbol{w}' + b' = \boldsymbol{0}$$

One way to think of this is the point **x** in m dimensional space gets mapped onto a point in n dimensional space; then any negative co-ordinate values get mapped to zero; next the new point in n dimensional space gets mapped to a point in 1 dimensional space where its sign determines the output. In this understanding the mappings are all many to one

An alternative, more complicated analysis is that the co-ordinate systems are being mapped backwards (in a one to many way) meaning that there is a representation of each co-ordinate system in the original input space. This understanding is one of the main concepts behind GeoSal.

Consider the decision boundary for the 2-layer approach, the boundary is made up of a series of planes, one for each possible **R.** We know these are planes in the final layer because they obey:

$$(7) \quad \boldsymbol{x} \cdot \boldsymbol{R^1} \cdot \boldsymbol{w}' + b' = 0$$

Rewriting (6) gives:

$$\boldsymbol{x} \cdot \boldsymbol{w^0} \cdot \boldsymbol{R^1} \cdot \boldsymbol{w'^1} + \boldsymbol{b^0} \cdot \boldsymbol{R^1} \cdot \boldsymbol{w'^1} + b'^1 = \boldsymbol{0}$$

$$(8) \quad \boldsymbol{x} \cdot \boldsymbol{W} + B = \boldsymbol{0}$$

Note that **W** is $m \times 1$ dimensional meaning that each plane in the final layer is represented by a single plane in the input layer. The same argument holds true for the planes joining any pair of axes in the final layer, meaning that axes can be represented in previous layers. Additionally, this can be further back propagated for any number of layers. This produces a representation of the decision plane in the input space which can be analysed geometrically in a similar way to the example in Figure 12.

Visualising this interpretation

Firstly, consider the network shown in Figure 13 (kept 2 dimensional to aid visualisation) – the colours indicate the axes of the input space:

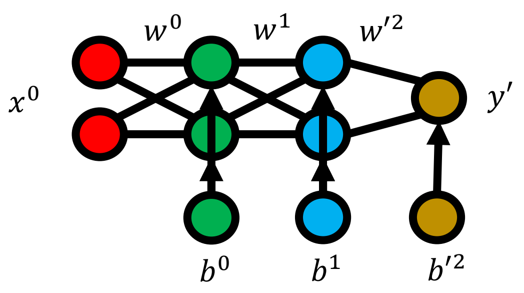

Figure 13: A simple deep network with each layer two dimensional. The colours are consistent for later figures.

The input space of this network is 2 dimensional with each input between 0 and 1:

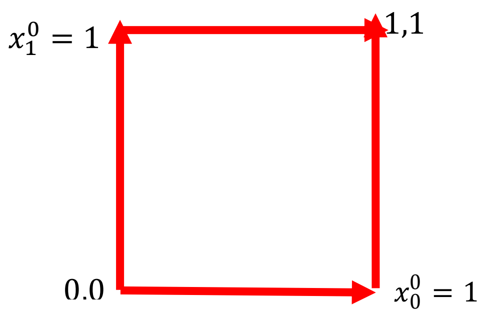

Figure 14: The input space of the first layer is bounded with both upper and lower bounds

The axes from the next layer can now be mapped into this space; this can be done by finding the lines in the input space which would map onto the axes in the first layer. Recall that the co-ordinates in the second layer's axes are given by:

$$x^0 w^0 + b^0 = [x_0^1 \quad x_1^1]$$

An example of these two co-ordinate systems on top of each other is shown in Figure 15. Any point in the red input space has corresponding co-ordinates in the green layer space, determining the transformed co-ordinates is done by measuring from the transformed axes (the axes are scaled, moved and rotated independently).

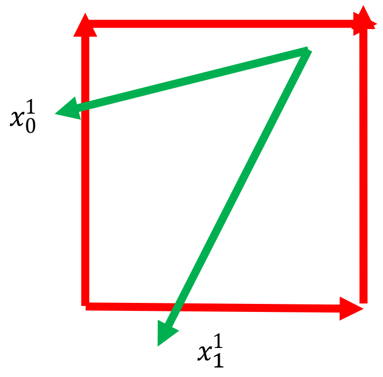

Figure 15: The input space for the other layers only have lower bounds. The weights and biases move and reorient the axes.

In higher dimensions an axis α would be the intersection of the planes:

$$(9) \quad \boldsymbol{x^0 w^0_{\tilde{\alpha}}} + \boldsymbol{b^0_{\tilde{\alpha}}} = 0$$

Where $\boldsymbol{w^0_{\tilde{\alpha}}}$ is all columns in $\boldsymbol{w}$ except column α. If $\boldsymbol{w}$ is $m \times n$ then $\boldsymbol{w^0_{\tilde{\alpha}}}$ is $m \times (n-1)$ and hence produces $(n-1)$ planes which are guaranteed to intersect along a line in m dimensions since the problem is under constrained (m variables, $(n-1)$equations).

In a similar manner it is possible to map the second hidden layer onto the first hidden layer, and then again onto the input space. Figure 16 shows all axes mapped onto the **input space**:

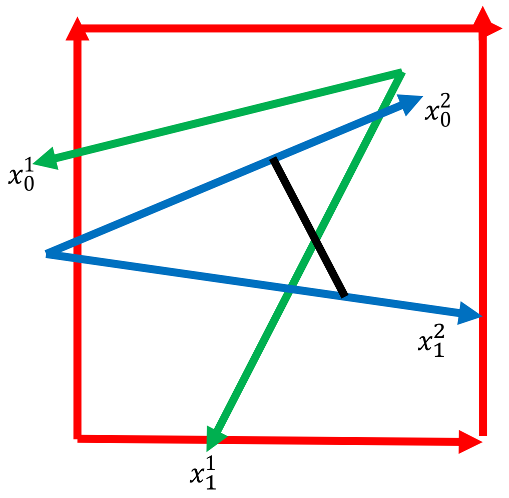

Figure 16: The input space with axes for all the layers. The output y' is measured as the distance from the black line the class can be determined by the sign of y'.

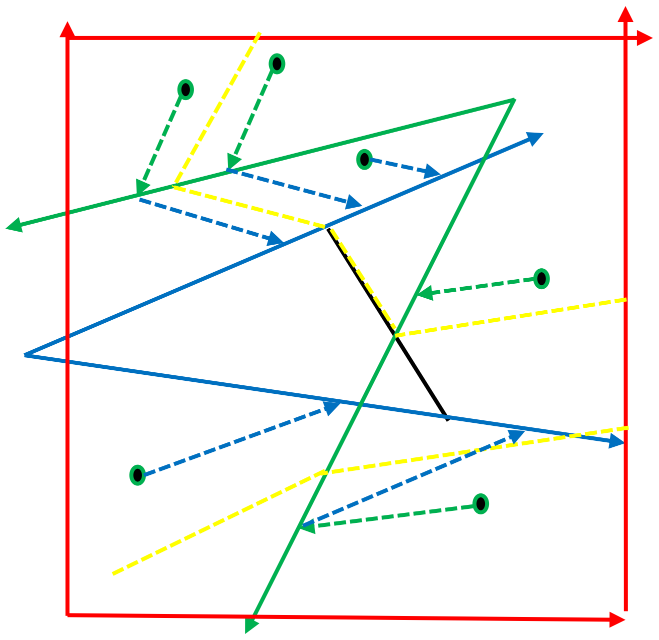

Figure 17: The decision boundaries in the input space have been added to this diagram in yellow, as well as several test inputs. The process of mapping the test inputs onto the final layer is shown with the green and blue arrows

The way to read how an input would be classified in Figure 17 is:

1) Start with an input point inside the RED input space.
2) If any of the co-ordinates with respect to the GREEN axes are negative then map the point to the positive region (parallel to the axis with a negative value)
3) If any of the co-ordinates with respect to the BLUE axes are negative then map the point to the positive region (parallel to the axis with a negative value)
4) If the point finishes to the left of the solid black line it is classed as class c, if it is to the right, it is class p.

The yellow, dashed lines show the decision boundaries in the **input** space. Note how this representation explains why deep Multi-Layered Perceptron classifiers have the ability to solve XOR problems (there are multiple, distinct, regions for the same class) and that some, but not all, of the final decision boundary is shared with the input space boundary. The yellow dashed lines all have the form $\boldsymbol{x} \cdot \boldsymbol{W} + B = 0$ since they are all decision boundaries, the point which gives the closest adversarial example lies at the closest point on the closest decision boundary. This forms the **strong global** model for GeoSal – this geometric method is guaranteed to give the same prediction as the network.

Producing the explanation

The analysis of a single layered network concluded that the closest boundary (L2 distance) to any input is the one formed from the **R** matrices created by the input. Unfortunately, this is no longer guaranteed to be the case in a deeper network since the XOR property of the network means that there can be closer decision boundaries achieved by including 0's in layers before the last. Checking all boundaries has m**P**m computational complexity for each layer resulting in $\sum m^l \, \mathbf{P} m^l$ complexity overall and even then, there is no guarantee that the point found would be valid for that plane (as seen in the single layer example). Therefore, additional checks would have to be performed, making this approach highly unfeasible.

This extreme computational cost leads to the main assumption for this method: by iteratively attempting to move onto to the decision plane determined by the test image's **R** matrices, an adversarial example will be generated. It is assumed that the importance heatmap created from this example will be a reasonable approximation for the heatmap generated from the true closest result. This is valid for the first iteration as the decision planes created from the XOR property are always parallel. This means that the pixel values will change absolute value with the correct proportions to one another. Further iterations will introduce unavoidable error which could be arbitrarily large.

Input space considerations

Each input in the MNIST dataset lies between 0 and 1. However, when moving an input point onto the decision plane, it may finish outside these bounds. The simplest way to remedy this is to force the point back into the input space by setting values less than 0 to 0 and greater than 1 to 1. If this results in a new set of **R** matrices the algorithm can continue, otherwise the input will converge to the point where the plane crosses the bounds after an infinite number of iterations - the process becomes a geometric series. To find convergence quickly: each guess is checked to see if it has crossed the input bounds more than 3 times, if it has then the convergence point can be calculated and the algorithm resumed from that point. As seen in Figure 18, convergence can be found by:

$$(9)\; q = p + d_1 \frac{1}{1-r}$$

Where: $r = \frac{d_2}{d_1}$

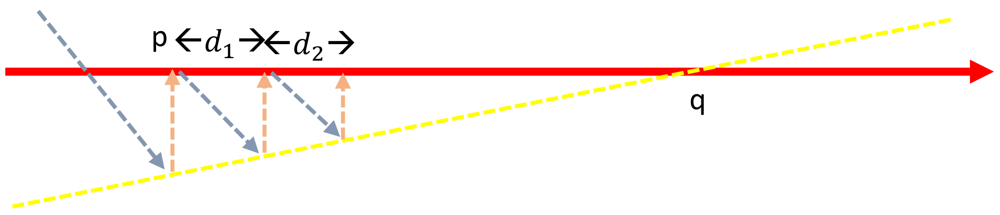

Figure 18: The acceptable region is above the red axis, but the boundary being found lies outside this region. To force it back in is equivalent to movement perpendicular to the axis (more specifically, parallel to the other axis which is not shown). This process clearly converges.

Figure 18 highlights the potential for another source of error in this algorithm. If the plane being sought is close to parallel to the violated axis then the conversion point could violate a different axis by a much larger amount. There is no guarantee that this situation will eventually converge within the input space. If the algorithm does converge it is not guaranteed to be close to the original image. If the decision plane is exactly parallel then convergence will not be possible (the probability of this occurring naturally is near 0).

The algorithm for finding closest adversarial example:

1. The original image is the initial best guess
2. Find the R matrices which define the region of the current best guess
3. Use the R matrices to calculate the decision plane specific to that region (equation 6)
4. Find the closest point on that plane to the best guess (equations 4 and 5)
5. Force the point found into the input space (values greater than 1 become 1, values less than 0 become 0) this point becomes the new best guess
6. If the same plane has been used for 3 successive iterations then use equation 9 to generate the new best guess
7. If the best guess, when tested, produces p(comparison)>p(prediction) then break, else go to 2.
8. Return the best guess

The algorithm for finding total importance.

1. For each comparison Q_n in the output class (not equal to the prediction class, P) in order of output probability y_n
2. If y_n is below some threshold then end.
3. Else find the closest adversarial example
4. Add $y_n \cdot |x - x'|$ to the running total of importance
5. Go to 2
6. Return the running total of importance

Implementation

1. In order to have the greatest flexibility 3 separate functions were used to create this method. Mnist_net.find_plane, which finds the $\boldsymbol{W}$, $\boldsymbol{B}$ and $\boldsymbol{y}$ for which $\boldsymbol{x} \cdot \boldsymbol{W} + \boldsymbol{B} = \boldsymbol{y}$ and takes as its main inputs:
   - An image to be analysed

- Node: either one number or an array of 2 numbers. When only one number is given then that output will be treated as only output to the classifier (y is useful for deep fool). When it is an array then the classifier will be restructured to produce the difference between the two classes (y' is useful for GeoSal). If "None" then the output will have the same dimensions as the original ($\boldsymbol{y}$).
- Layer: the "find_plane" function can be run on any layer or even between two layers, this can condense deep networks to much shallower ones in the region of a specific input
- Returns: $\boldsymbol{W}$, $\boldsymbol{B}$ and $\boldsymbol{y}$

2. GeoSal: this performs the algorithm for finding the closest adversarial example, it requires parameters to be passed to Mnist_net.find_plane, as well as a Mnist_net network to run on. It handles all the convergence possibilities, as well as a check to ensure $Q_n \neq P$
3. The algorithm for finding total importance is implemented in a separate cell, in this case without using a threshold term since the Mnist_dataset is small enough to check 10 classes in reasonable time (5 seconds).

Test Results:

The GeoSal function provides a meaningful heatmap of importance which can be easily understood by a user especially when only comparing two classes as in Figure 19. The absolute image highlights that the most important regions for this particular distinction include the region at the top – where there is a large area of white; additionally, the region under the four is important (very few test samples are non- zero in that region and so the classifier may behave strangely with non-zero values). The area where the black pixels lie in the original image is also significant to the distinction, reducing the values of the black pixels often serves to reduce the strength of the classification (for a very small L2 distance), this means that the comparison class needs less increase in output to become equal probability. As demonstrated in Figure 20, combining all the explanations reduces this

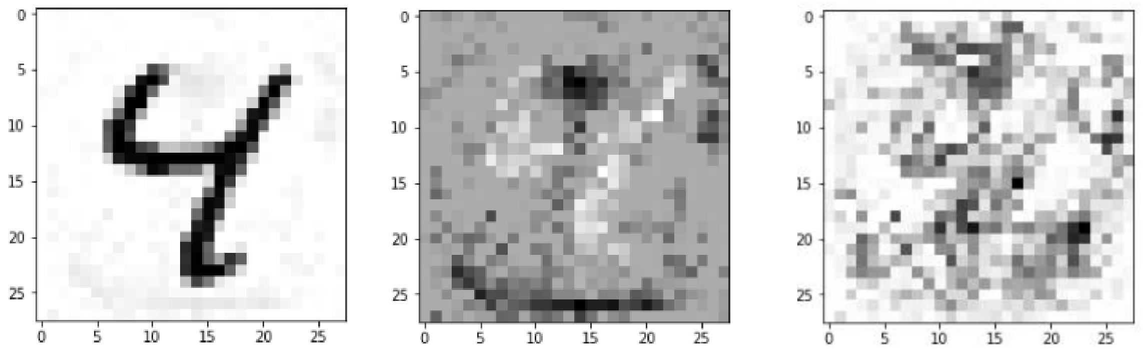

Figure 19: Left: the adversarial example for a 9 (second most probable class). Middle: the difference between the adversarial example and the original image (contrast is boosted). Right: the absolute difference (contrast is boosted).

clarity but provides the end user with a single heatmap of importance which is easily described in plain language. The sum of weighted absolute differences does not highlight features as clearly as simply summing weighted differences, but it will scale better for full colour images.

The method is fast enough to be competitive with other methods examined in this project, although the approximations made in order to achieve this may be unacceptable in some circumstances – such as for very deep networks and where there are many important XOR cases. Further work should be done to try to improve on these approximations and if possible find a way around the problems caused by the XOR property.

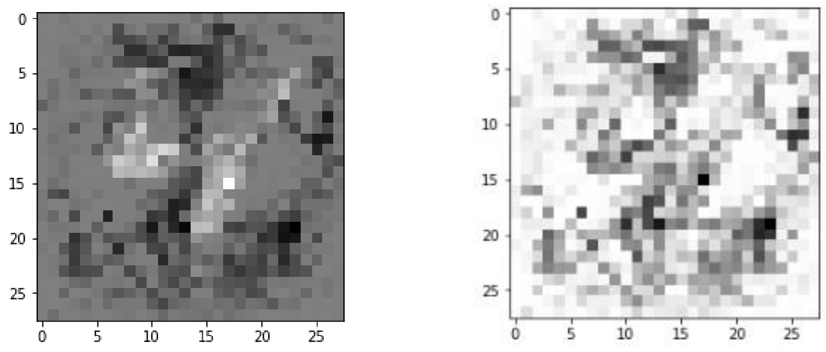

Figure 20: Full comparative explanations given by GeoSal.

*Left: the explanation from using the weighted sum of differences. Right: the explanation from using the weighted sum of **absolute** differences.*

Future Implementations:

Real Time Image Saliency for Black Box Classifiers (Dabkowski & Gal, 2017)

This method represents an unexplored field of descriptors, a separate network is trained from the test network and the test dataset. This explainer network is used to produce a mask which gives the salient features in a supplied image. With sufficient training the explainer network becomes a **weak global** explanation; this method is capable of explaining Black Box classifiers, provided a suitable test dataset of images can be used in the training process. Training the explainer network is a large computational overhead, but once this is complete any specific instance can be explained in the time it takes to run the network once (~10 ms according to the original paper). This makes this method one of the fastest available and could have a large number of potential applications.

Unfortunately, the codebase for the method was created in python 2.7 which is incompatible with python 3; the python migration tool was unable to convert the codebase, and there are several libraries used in the codebase which also required python 2.7. Migrating this project to python 3 will be a large task, doing so in this project would have taken too much time to justify and would have prevented the development of GeoSal.

PatternNet and PatternAttribution (Kindermans, et al.)

This method models any input as a combination of signal and noise and determines that the main function of a classifier is to remove the noise. With this global model established, analytical methods are used to attempt to determine the underlying signal within the image. Much of this white box analysis builds on the work done in the Deep Taylor and other backpropagation methods but with a different aim. This work was under blind review at the time it could have been implemented, and GeoSal was developed instead. The tests provided in the paper suggest that the method is able to work for object localisation as well as saliency.

Evolution of Project Goals

This project developed over time, most notably with the creation and testing of GeoSal. This technique required a large amount of mathematical analysis and took many weeks to bring to fruition. The scope of literature considered for inclusion within this project was also narrowed to require methods which produce a single heatmap as an explanation. The development of GeoSal prompted Deep Taylor being implemented over the alternatives due to common use of Deep Fool.

Section 2: Jupyter Notebook Tutorials

Overview

The Mnist_net class has a lot of in built functions to allow faster testing while requiring very little direct knowledge of TensorFlow. Many of the low-level functions are heavily inspired by the TensorFlow tutorial functions (TensorFlow, 2018) for creating a network with the MNIST dataset (TensorFlow was new to me at the beginning of this project). The main functions in the class are listed below (please note that this is condensed to provide an overview and is not reference documentation – the functions are well commented with sensible variable names and the details of each function can be found as commented headers)

- *setup – The user provides an array determining the structure and details of each layer of the network and the network is created; either with randomly initialised weights or with weights loaded from a file path.
- *save – saves the network at the given file path, defaults to "./Models/MNIST_model".
- plot_images – plots 9 supplied images with the true and predicted class labels.
- optimize – performs stochastic gradient decent for a given number of iterations
- plot_example_errors – plots the first 9 images the classifier gets wrong along with the predicted and true class.
- plot_confusion_matrix – plots an image showing the probability of predicted given true, ideal classifiers have an identity confusion matrix.
- print_test_accuracy – can show the example errors and confusion matrix; and will print accuracy on a test set (a subset of the test dataset).
- plot_conv_weights – plots a grid with showing all of the weights given.
- plot_conv_layer – plots a grid showing all of the outputs from a given layer.
- plot_image – plots the image at a given index of the dataset
- *average_output – returns the average of a given layer's outputs for a given image.
- *give_prob – gives the network output at a given layer (default is final) for a given image.
- *give_class – gives the class with the highest probability for a given input image.
- *find_plane – gives the values for W,B,y which satisfy xW+B=y for input image x

The starred (*) functions are unique to this class, the other functions are edited versions of tutorial functions reformatted to match the class framework, in a scalable way. Both "give_class" and "give_prob" can accept a list of RGB images, an RGB image, a 2D greyscale image or a 1D (flattened) greyscale image.

The use of many of these functions, as well as the functions written for the different saliency methods are demonstrated in the following tutorials:

Tutorial 01 - Creating, Importing and Training a network:

This tutorial covers the basic functions for creating and training a network in the Mnist_net framework. It explains the form of how to set up a network as well as the save and loading functions. https://notebooks.azure.com/AndrewLouw/libraries/Saliency-Comparison/html/Tutorial%2001%20-%20Mnist_net.ipynb

Tutorial 02 - LIME and VisualBackProp:

This tutorial demonstrates LIME and VisualBackProp. The VisualBackProp algorithm is only a few lines long and so is demonstrated in-line. LIME uses the edited LIME file "lime_image_afl33" as well as the unedited "lime_base". To avoid the need to install LIME (unsupported in Microsoft Azure) these files have been included in the library. Both of these files come from the LIME GitHub version 0.1.1.24. Other versions may vary.

https://notebooks.azure.com/AndrewLouw/libraries/Saliency-Comparison/html/Tutorial%2002%20-%20%20Lime%20and%20VisualBackProp.ipynb

Tutorial 03 – Deep Taylor and GeoSal:

This tutorial demonstrates the Deep Taylor and GeoSal algorithms, both use separate functions in their implementation. This tutorial also demonstrates various methods for displaying the heatmaps.

https://notebooks.azure.com/AndrewLouw/libraries/Saliency-Comparison/html/Tutorial%2003%20-%20Deep%20Taylor%20and%20GeoSal.ipynb

Conclusion

The various methods for finding saliency in neural networks all start with different definitions for an explanation and as a result they produce very different **heatmaps**. It is difficult to compare these **heatmaps** to decide which one produces the best explanation because doing this would require yet another definition. Instead it is possible to compare the properties of the required **models**, and to look at the limitations, flexibility and the appropriate uses for each method.

LIME is undoubtably the most flexible model investigated. It is a black box model and can be integrated with existing networks very easily, however it is also the slowest. The explanation provided by LIME is easy to understand and can be very useful to determine which features in a specific input the network is most sensitive to. This can be useful for finding problems arising from bad training data (Figure 1) as well as comparing features between classes. It is most powerful when features are easily identified and deleted, making it useful for more than just evaluating image classifiers, e.g. text classifiers.

In contrast VisualBackProp is the fastest method. But it puts a large number of limitations on the network, making it only useful in some niche applications. The explanation produced shows which parts of the input are capable of propagating information to the final fully connected layer - but this last layer could have a large impact in the final result. This type of explanation is useful for real time object localisation, as well as identifying the features in the input which remain relevant throughout the network, even if they are not part of the chosen class – this can identify the types of noise the network is susceptible to.

The Deep Taylor method strikes a good balance between a flexible model and a quick result. To make it work with more complicated networks there will have to be specific coding to correctly distribute the relevance between layers and convert the network into a usable form. The relevance heatmap produced is calculated with minimal approximation, meaning that it is the best to use in situations which require high confidence, for example in healthcare. The relevance conservation rule means that this method can be adapted for unique layer types without losing credibility, for example if experimenting with unconventional networks.

GeoSal provides a novel geometric way of considering the workings of a Multi-Layered Perceptron network. It also challenges the established norm with white box saliency techniques by providing a comparative explanation instead of an additive one. The technique is iterative, which could potentially lead to arbitrarily long computing times (although this has not been observed). Additionally, there are known flaws in the approach which could potentially produce results arbitrarily far from the true value, when this happens and how important the errors are cannot be easily established. Future work investigating this method could prove fruitful, as could finding other methods based on a comparative explanation. GeoSal is best used with less deep networks, or when it is desired to know how robust the input is to change. It is also good for explaining the decision between two similar classes.

All the methods (except GeoSal) were tested on full colour images in their original papers, usually with much more complicated networks than tested here. It was surprising to find that these methods sometimes failed to produce useful explanations when used on a simple network and a simple dataset: Deep Taylor lacks clarity; and VisualBackProp requires the data to be offset; LIME works well in general, but it was found that super-pixels may not always be the most suitable method for determining how to break an image into features. On the other hand, it is possible that GeoSal, in its current form, will not scale well as more layers bring more XOR cases and more approximations.

Key Properties of the Different Methods

(Table 1)	LIME	VisualBackProp	Deep Taylor	GeoSal
Network	Black Box	White Box	White Box	White Box
Convolutional layer	Yes	Yes	Potentially	Potentially
Perceptron layer	Yes	No	Yes	Yes
Pooling	Yes	No	Yes	No
Biases	Yes	No	Yes	Yes
Model	None	Weak Global	Weak Global	Strong Global
Explanation	Additive or Comparative	Additive	Additive	Comparative
Timescale	~6:15 minutes	~40ms	~19 seconds	~5 seconds
Scaling	With network depth	With network depth	With network depth and width	With network depth
Feature Localisation	Yes	Yes	Yes	No

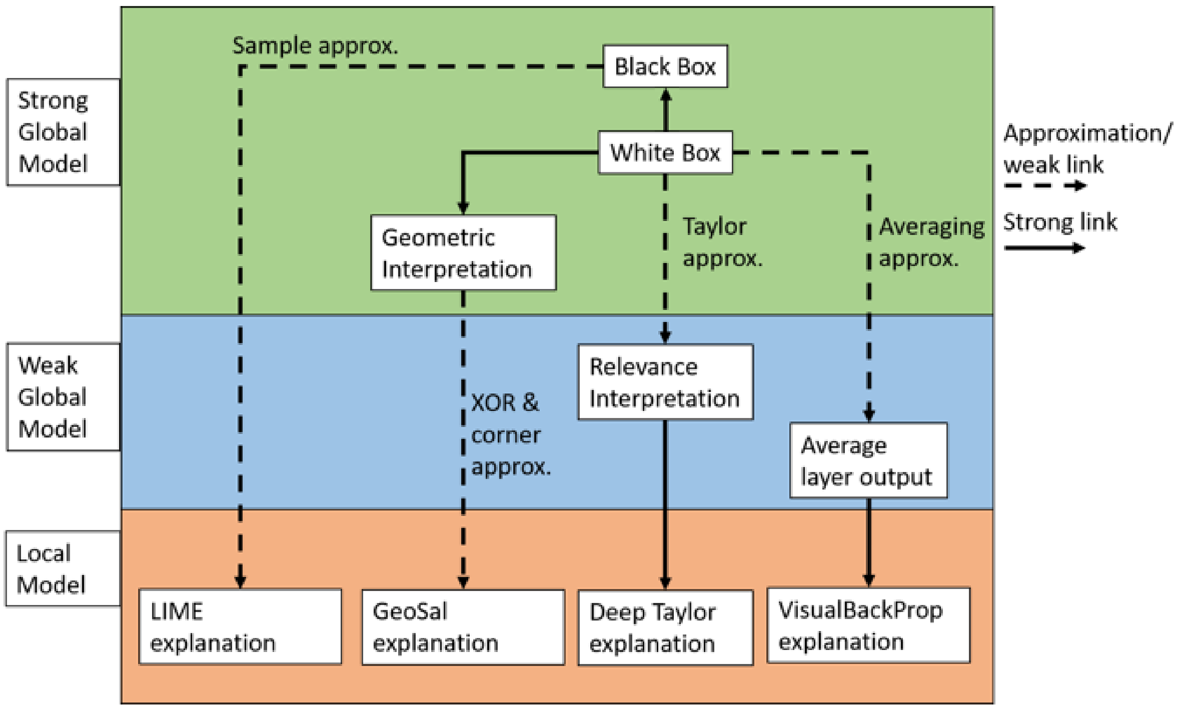

Figure 21: A representation of how models are used as an intermediate step to forming the explanation. All methods used have some form of approximation and these lead to the very different forms of the explanations.

"Black box" and "White box" are both descriptors for the same network.

Glossary of Terms Used to Describe Methods.

Additive	An explanation derived from adding evidence for the chosen class
Black Box	A method which only requires network inputs and outputs to provide the explanation.
Comparative	An explanation derived from combining the evidence against classes not chosen.
Descriptor	Anything which can describe the model irrespective of complexity or accuracy
Explanation	An output intended for the user to see in order to understand a specific classification (subset of Descriptor)
Importance	In GeoSal: the absolute values of the explanation
Local	A descriptor which is optimized for inputs similar to the one being tested.
Model	A global way of representing the network, or the flow of information/relevancy within the network (subset of Descriptor)
Strong Global	A descriptor which is optimized for lowest deviation to the network across the whole input space and has no error.
White Box	A method which requires the structure of the network as well as the values for the parameters to provide an explanation.
Weak Global	A descriptor which is optimized for lowest deviation to the network across the whole input space, but with some error.

References

Bojarski, M., Choromanska, A., Choromanski, K., Firner, B., Jackel, L., Muller, U., & Zieba, K. (2017). *VisualBackProp: efficient visualization of CNNs.* arXiv. Retrieved from https://arxiv.org/pdf/1611.05418.pdf

Dabkowski, P., & Gal, Y. (2017). *Real Time Image Saliency for Black Box Classifiers.* arXiv. Retrieved from https://arxiv.org/pdf/1705.07857.pdf

Doshi-Velez, F., & Kim, B. (2017). *Towards A Rigorous Science of Interpretable Machine Learning.* arXiv. Retrieved from https://arxiv.org/pdf/1702.08608.pdf

Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., . . . Song, D. (2018). *Robust Physical-World Attacks on Deep Learning Visual Classification.* arXiv. Retrieved from https://arxiv.org/pdf/1707.08945.pdf

Goodfellow, I. J., Shlens, J., & Szegedy, C. (2015). *Explaining and Harnessing Adversarial Examples.* arXiv. Retrieved from https://arxiv.org/pdf/1412.6572.pdf

Kindermans, P.-J., Schutt, K. T., Alber, M., Muller, K.-R., Erhan, D., Kim, B., & Dahne, S. (n.d.). *Learning How To Explain Neural Networks: Patternnet And Patternattribution.* OpenReview. Retrieved from https://openreview.net/pdf?id=Hkn7CBaTW

Montavon, G., Bach, S., Binder, A., Samek, W., & Muller, K.-R. (n.d.). *Explaining NonLinear Classification Decisions with Deep Taylor Decomposition.* Retrieved from https://arxiv.org/pdf/1512.02479.pdf

Moosavi-Dezfooli, S.-M., Fawzi, A., & Frossard, P. (2016). *DeepFool: a simple and accurate method to fool deep neural networks.* arXiv. Retrieved from https://arxiv.org/pdf/1511.04599.pdf

Olah, C., Mordvintsev, A., & Schubert, L. (2017). *Feature Visualization, How neural networks build up their understanding of images.* Distil. Retrieved from https://distill.pub/2017/feature-visualization/

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). *"Why Should I Trust You?" Explaining the Predictions of Any Classifier.* arXiv. Retrieved from https://arxiv.org/pdf/1602.04938.pdf

Simonyan, K., Vedaldi, A., & Zisserman, A. (2014). *Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps.* arXiv. Retrieved from https://arxiv.org/pdf/1312.6034.pdf

TensorFlow. (2018). Retrieved from https://www.tensorflow.org/tutorials/

Vedaldi, A., & Soatto, S. (2008). *Quick Shift and Kernel Methods.* UCLAVISIONLAB. Retrieved from http://vision.cs.ucla.edu/papers/vedaldiS08quick.pdf

Weller, A. (2017). *Challenges for Transparency.* arXiv. Retrieved from https://arxiv.org/pdf/1708.01870.pdf

Risk assessment retrospective

As expected the hazards with this project are those typically associated with computer use for extended periods of time, which can be reduced with appropriate precautions – good lighting, keyboard, chair and by taking breaks.