UNIVERSITY OF CALIFORNIA
RIVERSIDE

Deep Contrastive Explanation

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Seyedamirhossein Feghahati

December 2020

Dissertation Committee:

Dr. Christian Shelton , Chairperson
Dr. Vagelis Hristidis
Dr. Vagelis Papalexakis
Dr. Michael Pazzani

The Dissertation of Seyedamirhossein Feghahati is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I am deeply indebted to my advisor, Dr. Christian R. Shelton, for his invaluable and endless mentoring, support, encouragement, and extreme patience over the years. His brilliant mind, character and integrity will always be an inspiration.

I would like to thank Dr. Michael Pazzani for providing funding for the project. His positivity and thinking out of the box set an example for me to follow.

I am thankful to the other members of my committee Dr. Vagelis Hristidis and Dr. Vagelis Papalexakis for the discussions and directions for the future work.

I would like to thank the members of the RLAIR lab, including Mike Izbicki, Busra Celikkaya, Zhen Qin, Dave Gumboc, Matthew Zarachoff, Kazi Islam, Sepideh Azarnoosh, Chengkuan Hong, Sanjana Sandeep, Gaurav Jhaveri, Leah Fauber, Anthony Williams, Chandini Shetty and, Colin Lee. Also, I should thank my friend, Mohsen Noparvar, whose expertise in Web development helped me during conducting the user experiments.

In memory of my uncle, Mahmoud Fard.

ABSTRACT OF THE DISSERTATION

Deep Contrastive Explanation

by

Seyedamirhossein Feghahati

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, December 2020
Dr. Christian Shelton , Chairperson

I propose a method which can *visually* explain the classification decision of deep neural networks (DNNs). Many methods have been proposed in machine learning and computer vision seeking to clarify the decision of machine learning black boxes, specifically DNNs. All of these methods try to gain insight into why the network "chose class A" as an answer. Humans search for explanations by asking two types of questions. The first question is, "Why did you choose this answer?" The second question asks, "Why did you *not* choose answer B over A?" The previously proposed methods are not able to provide the latter directly or efficiently.

I introduce a method capable of answering the second question both directly and efficiently. In general, the proposed method generates explanations in the input space of *any* model capable of efficient evaluation and gradient evaluation. It neither requires any knowledge of the underlying classifier nor uses heuristics in its explanation generation, and it is computationally fast to evaluate. I provide extensive experimental results on three different datasets, showing the robustness of my approach and its superiority for gaining insight into the inner representations of machine learning models. As an example, I demonstrate my method can detect and explain how a network trained to

recognize hair color actually detects eye color, whereas other methods cannot find this bias in the trained classifier.

I provide the details on how this framework can be applied to discrete data as well. I proposed a SoftEmbedding function to be employed in conjunction with the discrete embedding function. I show results on textual reviews demonstrating my method's ability to find bias in learned classifiers.

Finally, I provide the results of a user study, measuring how much this feedback helps users to improve their understanding of the network's learned function in comparison with other possible methods.

Contents

List of Figures x

List of Tables xi

1 Introduction 1

2 Related Work 6
- 2.1 Terminology . . . 7
- 2.2 Why interpretability . . . 7
- 2.3 Model as a whole . . . 9
- 2.4 Counter-factual Examples . . . 11

3 CDeepEx 20
- 3.1 Proposed Method . . . 22
  - 3.1.1 Terminology . . . 23
  - 3.1.2 Learning the Input Distribution . . . 24
  - 3.1.3 Why not backprop directly . . . 24
  - 3.1.4 Generating Explanations . . . 25
- 3.2 Experimental Results . . . 29
  - 3.2.1 MNIST . . . 29
  - 3.2.2 Biased MNIST . . . 35
  - 3.2.3 Fashion MNIST . . . 36
  - 3.2.4 CelebA Dataset . . . 37
  - 3.2.5 Selection of the Generator Model . . . 41
  - 3.2.6 Training on Different Datasets . . . 42
- 3.3 Conclusions . . . 42

4 Application in Text 44
- 4.1 Introduction . . . 44
- 4.2 Transforming Text into an Image . . . 45
- 4.3 Proposed Method . . . 47
  - 4.3.1 Classifier . . . 47

4.3.2 Generator 48
4.3.3 Generating Explanations 49
4.4 Experiments 51
4.4.1 Dataset 51
4.4.2 Non-Biased data 51
4.4.3 Biased data 51
4.5 Conclusion 52

5 User Experiments **54**
5.1 Results 62

6 Conclusion **65**

List of Figures

3.1 The schematics of the proposed approach . . . 24
3.2 Direct explanation on input space . . . 26
3.3 Comparison of CDeepEx with direct methods . . . 30
3.4 Comparison of CDeepEx to xGEMs . . . 30
3.5 Importance of the constraints . . . 31
3.6 Different generators . . . 32
3.7 Experiments on detecting bias . . . 33
3.8 Comparison of CDeepEx and xGEMs over Vgg16 and ResNet101. . . . 36
3.9 CDeepEx vs. xGEMs over CelebA dataset. . . . 38
3.10 Synthetic checks on explanations. . . . 39
3.11 More experiments on CelebA dataset . . . 40
3.12 Role of Generator in explanations . . . 40
3.13 Cross-dataset comparison . . . 42

4.1 Generated images from text . . . 46
4.2 The general architecture of text classifiers . . . 48
4.3 Output of the generator is a distribution over words. . . . 50

5.1 Eye tracking system shows that the subjects start looking at the parts of the birds they are more familiar with. . . . 55
5.2 Numbers 0 through 9 from Kannada script. . . . 56
5.3 Screenshots from the application which records the user answers. . . . 58
5.4 This the feedback provided to a user in the control group. . . . 59
5.5 This is how a user gets a feedback from GCAM. . . . 60
5.6 This is the feedback given to a user in CDeepEx group. The feedback is of a form of an animation. See to 5.7 . . . 60
5.7 CDeepEx's feedback. . . . 61
5.8 Cumulative number of correct answers per test group. . . . 62
5.9 Usefulness of the explanation per user perception . . . 64

List of Tables

4.1 Samples of Yelp data set. 52
4.2 Experiment on Yelp dataset 52
4.3 Experiment on the biased Yelp dataset 53

5.1 Definition of the test groups 58
5.2 Mean and standard deviation of cumulative number of correct answers per group . 62
5.3 Comparison of p-values of the methods 63

Chapter 1

Introduction

In the recent years, deep neural networks (DNN) have shown extraordinary performance on computer vision tasks such as image classification (Szegedy et al., 2015; Simonyan and Zisserman, 2015; Springenberg et al., 2015; He et al., 2016), image segmentation (Chen et al., 2018), and image denoising (Zhang et al., 2017). The first example of such a performance was on image classification, where it outperformed other computer vision methods which were carefully handcrafted for image classification (Krizhevsky et al., 2012). Following this success, DNNs continued to grow in popularity. Even with DNNs achieving testing accuracy close to human expertise (Rajpurkar et al., 2017) and in some cases surpassing them (Springenberg et al., 2015), there is hesitation to use them when interpretability of the results is important. Accuracy is a well-defined criterion but does not provide useful understandings of the concept captured by the network. If the deployment of a network may result in inputs whose distribution differs from that of the training or testing data, interpretability or explanations of the network's decisions can be important for securing human trust in the network.

In this dissertation I try to address this issue and explain what the network has learned.

So, one question we may ask is "Have these methods learned something rational, similar to what humans know/may learn?" Although this question looks very tempting to be a venue of research, a more sensible question to ask is "What have these black boxes learned?" The reason is simple: The machine may learn some concepts that may not be similar to the concepts the humans know, yet they are correct. The other reason is that in the learning process we do not force the machine to think like humans. Thus, we cannot expect them to learn something humanly. We want to *explain* what the ML black boxes has learned since we know explanations are important in settings such as medical treatments, system verification, and human training and teaching.

So, what is explainability or interpretability? What is its definition and how we can explain it in simple words? The short answer is that we do not know! Unlike some measures like accuracy, which is very well-defined and also intuitive, it is not clear what is exactly meant by interpretability or explainability. The true explanation is to trace the steps of the algorithm and see how it reaches its decision for a sample input. These steps are basically equations and, in the case of deep learning, millions of them. So, one definition of explainability is to check these equations. This approach is not sensible for obvious reasons; there are too many equations. Also, we cannot map those numbers into concepts since the input is an entity such as an image/text/video and the output is a number. As a result, we have to define a proxy definition for explainability. What could be this proxy?

In a utopia, this can be a program that explains the outcome as speech in the language of choice of the user. But we still have a long way to get to this point. So, we need to limit ourselves to a simpler approach. Consider images as an input. One way to connect the complete explanation (tracking the equations and the final number) into an interpretable explanation is to look at the importance of each unit of the image, pixels. The modification of a pixel is simple, you can only

increase its value or decrease it. Then, we can change the value of each pixel and see how the final number changes. For instance, if by changing a pixel's value, the probability of the outcome decreases, we can say this pixel at its current value has a significant role in the decision making process of the network. On the other hand, if changing the value does not change the outcome significantly, we can say this pixel is not important for the network on this example. Same can be said for text input. It can be done of character level or word level. For instance, by changing a character from a word in Spanish language, you can say this is written by southern American user or a Spaniard user. In the same way, changing a word, for instance glasses to eye-glasses can help you to understand can the network distinguish between British English and American English.

This idea has been explored previously. Chiefly, prior work has i) looked into training data, ii) looked into the network, and, iii) explained the input sample specifically. Some tried to explain the outcome of the network, using its training data. They try to find most influential training examples which derived the network during training, such that in the testing time, it comes to this particular decision. The advantage of this definition is that it somehow tries to cluster the training data, the testing sample, and the network together. The two main drawbacks are that the training corpus may not be available at the testing time and, like all clustering algorithms, it is not clear what is the similarity in terms of some concepts. For instance, suppose the network can correctly classify a red fish. Asking what training samples may cause the network to classify this particular instance to be a red fish, it is not clear that what we are looking for. Are we looking for a whole fish? Is it relevant that red fish lives in water or not? Are all the training samples are just drawing of a red fish or actual images taken in its habitat?

The other approach is to look into the network. Although it may give you some intuition

about what concepts the network has learned, there are two problems associated with this approach. The first one is there are too many layers in a network. It is hard to look into all of them. The second one is more of a psychological problem. The inspector *tries* to find what he believes to be concept in the network.

The last one is to try to look directly into the sample. What parts of the input sample have the most influence on the network's decision? We believe this approach is the most intuitive one since it directly asks for the explanation for a decision. The most research on this approach is to ask the question, "Why did the network choose its answer?" Asking this question may result in incoherent answers. For instance, in image spaces, it may show a speckled pattern. In this dissertation we choose a different approach. What we ask is a contrastive question, "Why did the network chose A and *not* B?"

Humans often also seek *contrasting* explanations. For instance, they maybe more familiar with the contrasting answer, or they want to find the subtle differences in input which change the given answer to the contrasting one. Another advantage of this approach is that it can ask for multiple explanations. For instance, if the input image is a picture of number 4, then we can ask, "Why it is not a 9?" and "Why it is not an 8?"

The rest of this manuscript is organized as follows: In chapter 2, I discuss the related work in the literature. In chapter 3, I propose a method for contrastive explanations in deep networks and apply it to images as examples of continuous data. In chapter 4, I show how the proposed framework can be applied to a different modality of data. I show this framework can be applied on discrete data, text, with some minor modifications. In chapter 5, I conduct a user study to find out the usefulness of the contrastive explanation versus direct explanation vs no explanation at all. In this user study, I

measured the degree to which users can learn to predict the output of the network. This can be seen as a proxy to check if the explanations provided by the method is useful.

Chapter 2

Related Work

In this chapter we discuss different methods for interpretability. The two main modes of interpretability are example-based interpretability and model (as whole) interpretability. There are four main example-based explanations for general machine learning: counter-factual examples, adversarial examples, prototypes and influential instances (Molnar, 2019). In terms of techniques used to perform interpretability, the main ones are backpropagation, input perturbation and network visualizers. Note that these methods are not necessarily mutually exclusive. For other modes of interpretability, see (Molnar, 2019).

2.1 Terminology

$\mathbf{e}$	explanation vector
y	class label
$\mathbf{I}$	input image
ReLU(a)	$\max(0, a)$
$\odot$	Hadamard or element-wise product
w	layer weights
CNN	convolutional neural network
AE	Auto-Encoder
VAE	Variational Auto-Encoder
GAN	generative adversarial network
D	given network
G	generator
$\phi(\mathbf{I}, .)$	transform function
$\mathcal{R}_\theta(.)$	regularizer
N	number of training samples
M	number of testing samples
N_v	number of validation samples

2.2 Why interpretability

The need of interpretability comes from two sources. The first one is how much the system is critical, meaning making a mistake has a high cost. In this situations, we need to know how the

blackbox comes into its decision before we can proceed. As an example, medical systems need interpretability. The second one is the problem itself is ill-posed, so, we cannot trust a single metric to evaluate it. A classic example if such a system is an image classification system which classifies Huskies and wolfs. Although the system may have a high accuracy, it may learned something different that the huskies and wolfs to predict its results. For instance, most of the wolf images are taken in snowy weather and system learned snow and not wolfs. This phenomena can show that the system is trained without enough validation/training examples to capture other modalities of the concept, here is wolf/Huski.

If one can can ensure that the machine learning model can explain decisions, you can also check the following traits more easily (Doshi-Velez, 2017)

- Fairness: Ensuring that predictions are unbiased and do not implicitly or explicitly discriminate against protected groups. An interpretable model can tell you why it has decided that a certain person should not get a loan, and it becomes easier for a human to judge whether the decision is based on a learned demographic (e.g. racial) bias.
- Privacy: Ensuring that sensitive information in the data is protected.
- Reliability or Robustness: Ensuring that small changes in the input do not lead to large changes in the prediction.
- Causality: Check that only causal relationships are picked up. Trust: It is easier for humans to trust a system that explains its decisions compared to a black box.

2.3 Model as a whole

This approach looks at the model as a whole and checks what features the model has learned (Oramas M et al., 2019; Zhang et al., 2019; Koh and Liang, 2017).

Oramas M et al. (2019), propose a method which automatically identify the relevant features for a set of classes, i.e., predicts each class as a sparse weighted representation of its activations. Using this approach, a model-level based explanation can be achieved. At the training time, they forward a batch of images into the network and record the l2 response of each channel of each layer, creating a vector x_i for each image. Vector $l_i \in \{0,1\}^C$ represents the class labels for each image and $Y = [l_1^T; ...; l_N^T]$ Then, they find a sparse combination of these responses which minimizes

$$W^\star = \arg\min_W ||h^T W - Y^T||_F^2 \quad s.t. \quad ||w||_1 \leq \kappa, \quad \forall j = 1, ..., C \tag{2.1}$$

in which $h^T = [x_1; x_2; ...; x_N]$ and κ controls the sparsity. This is the matrix form of the κ-lasso problem. This problem can be efficiently solved using the Spectral Gradient Projection method. After solving the κ-lasso problem, they have a matrix W for which each non-zero element in W represents a pair of network layer p and filter index q (within the layer) of relevance.

Zhang et al. (2019) presented a method to represent a CNN as a decision tree which clarifies the specific reason for each prediction made by the CNN at the semantic level i.e., the decision tree decomposes feature representations in high convolutional layers of the CNN into elementary concepts of object parts. To do so, they have the assumption that the CNN has learned this parts. This assumption is somewhat valid for well-studied architectures such as VGG but its not clear if it is still valid for other types of networks since most layers learn a mixture of textures and semantics. Also, the loss function does not work for networks with skip connection such as ResNet. They write y as a

linear decision of the last convolutional layer, $y = \sum_{i,j,d} g^{i,j,d}.h_N^{i,j,d}(x) + b$ in which $g = \frac{\partial y}{\partial h_N(x)}$ at the position of (i,j) and d-th layer of the last convolutional map and, $b = y - g.h_N(x)$. To learn the tree, they used a greedy approach. The first layer is all the images of the desired category. To merge two nodes, they find the two closest nodes which have the closest prediction result and having a sparse set of connections between the last layer convolutional map and the last fully connected layer.

Koh and Liang (2017) presented a method to find the training examples which are responsible for a specific outcome off an input image. Since removing each training example and retrain the model to find out the weight of that specific example in the prediction is not feasible, they approximate this weight with influence function. The influence function can measure how much the parameters change if an example is slightly unweighted.

$$\mathcal{I}_{up,loss}((\mathbf{I}_{train}, y_{train}), (\mathbf{I}, y) = -\nabla_\theta L((\mathbf{I}, y), \hat{\theta})^T H_{\hat{\theta}}^{-1} \nabla_\theta L((\mathbf{I}_{train}, y_{train}), \hat{\theta}) \tag{2.2}$$

$$\frac{d\hat{\theta}_{\epsilon,\mathbf{I}_\delta,-\mathbf{I}_{train}}}{d\epsilon}|_{\epsilon=0} = -H_{\hat{\theta}}^{-1}(\nabla_\theta L(\mathbf{I}_\delta, \hat{\theta}) - \nabla_\theta L(\mathbf{I}, \hat{\theta}) \tag{2.3}$$

The influence function requires computing the Hessian. Computing the Hessian is an extremely heavy operation, especially with models like deep neural networks with millions of parameters. Fortunately, we can skip computing the Hessian and instead directly compute the product of the Hessian with another vector efficiently. The two mehtods which have been discussed in the paper are Conjugate gradients and stochastic estimation.

2.4 Counter-factual Examples

Counter-factual examples, in general, do not have a very well defined definition. Here we present a few definitions from literature:

- If X had not occurred, Y would not have occurred (Wachter et al., 2018).
- Why X happened and why not Y happened (Pazzani et al., 2018)
- Why any counter-factual to X happened (Fong and Vedaldi, 2017)

Fong and Vedaldi (2017), presented a method for perturbing images and create a counter-factual example. They showed artifacts are informative since they explain part of the network behavior, characterizing other properties of the network requires careful calibration of the generality and interpretability of explanations. They want to check how much the classification score is going to change if a part of an image is deleted. Also, find the smallest subset of image which that if preserved, the classification score does not change much. They do this by finding a mask over the image as

$$\mathbf{m}^{\star} = \underset{\mathbf{m}\in[0,1]^d}{\arg\min}\, \lambda||1-\mathbf{m}||_1 + D\left(\phi\left(x_0;\mathbf{m}\right)\right) \tag{2.4}$$

in which $\phi\left(x;m\right)$ applies a transformation on image x based on mask $\mathbf{m}$. This transformation can be blurring, adding noise or changing the region into a constant color. λ encourages sparsity, means most of the mask to be turned off.

Chang et al. (2019) presented a method similar to that of Fong and Vedaldi (2017). The main two differences are a generator model to generate the transformed image and a prior on the mask. They modeled the mask as a Bernoulli distribution and optimize over its parameters using a discrete to continuous relaxation technique. For the generator,they used variational auto-encoders

and GANs. The two loss functions they try to minimize are smallest deletion region (SDR) and smallest supporting region (SSR) (Dabkowski and Gal, 2017). They showed that using a generator model improves the bounding box localization in comparison to using heuristics such as blurring for generating reference image. Also, their method runs faster since they can sample different masks and run the method in parallel. Both of the aforementioned methods can be seen as in-painting methods which generates a non-targeted counter-factual.

Zhao et al. (2018b), tried try to generate natural looking adversarial examples by looking into the latent space of the images rather than input space. Optimizing directly in the input space can introduce pixel-wise noise into the image (Goodfellow et al., 2015). First, they learned a generative model using WGAN (Arjovsky et al., 2017). Then, they learned the inverter of the learned function. They do this by minimizing over the parameters of the inverter:

$$\mathcal{I}_\gamma = \arg\min_\gamma ||G\left(\mathcal{I}_\gamma\left(\mathbf{I}\right)\right) - \mathbf{I}|| + \lambda.\mathbb{E}_{z\sim p_{z(z)}}\left[L\left(z, \mathcal{I}_\gamma\left(G\left(z\right)\right)\right)\right], \tag{2.5}$$

in which L is l_2 distance between the latent vectors. Using this inverter $\mathcal{I}_\gamma$, the adversarial example can be found by

$$I^\star = G\left(z^\star\right) \quad where \quad z^\star = \arg\min_z ||z - \mathcal{I}_\gamma\left(\mathbf{I}\right)|| \quad s.t. \quad D(G(z)) \neq D(\mathbf{I}). \tag{2.6}$$

To perform the minimization, they proposed two search methods. The iterative one, perturbations of the initial z is chosen. Then, all the images of these perturbations are created, see which ones change the outcome of the classifier. If there are multiple of theses images, the one with the minimum distance to initial $\mathbf{z}$ is chosen. If such a $\mathbf{z}$ cannot be found, they increase the perturbation range by

some constant. They also presented a hybrid shrinkage algorithm to increase the efficiency. The advantage of this method is that it create a dense latent space which tries to copy the underlying image generation process and then finding the explanation in that space. Also, it can be used on discrete spaces such as text using an adverserially regularized autoencoder (Zhao et al., 2018a). The disadvantages are two things, first learning the generator and its inverter can be hard. The second problem is in constraints. Simply asking for an adversarial example that has a different outcome does not allow for a targeted attack, meaning, they cannot ask to change the outcome from 7 to 9, rather they can only ask to change the outcome of the D.

Dhurandhar et al. (2018) presented a method which aims to do two things. First, it finds a minimal amount of features that should be absent, i.e., remain background) in the input to prevent the classification result from changing. Second, it finds a minimal amount of features in the input that are sufficient in themselves to yield the same classification

$$
\begin{aligned}
&\min_{\delta \in \mathcal{X}/\delta} \quad \lambda_1 f_\kappa^{neg}(\mathbf{I}, \delta) + \lambda_2 ||\delta||_1 + ||\delta||_2 + \lambda_3 ||\mathbf{I} + \delta - AE(\mathbf{I} + \delta)||_2^2 \\
&f_\kappa^{neg}(\mathbf{I}, \delta) = \max\left\{ [D(\mathbf{I} + \delta)]_{y_{probe}} - max_{i \neq y_{probe}} [D(\mathbf{I} + \delta)]_i, -\kappa \right\} \\
&\min_{\delta \in \mathcal{X} \cap \delta} \quad \lambda_1 f_\kappa^{pos}(\mathbf{I}, \delta) + \lambda_2 ||\delta||_1 + ||\delta||_2 + \lambda_3 ||\delta - AE(\delta)||_2^2 \\
&f_\kappa^{pos}(\mathbf{I}, \delta) = \max\left\{ [-D(\delta)]_{y_{probe}} + \max_{i \neq y_{probe}} [D(\delta)]_i, -\kappa \right\}
\end{aligned}
\tag{2.7}
$$

in which AE is a autoencoder. To solve the equation 2.7, they used fast iterative shrinkage-thresholding algorithm. This method is an efficient solver for optimization problems involving L1 regularizer.

Zhou et al. (2016), used a global average pooling layer to increase the localization ability of the network which is trained using images with weak labels, i.e., only the label is given to the

network and not the location of the objects. The global average pooling layer must be inserted after the last convolutional layer and before the softmax layer during training. It outputs the spatial average of the feature map of each unit at the last convolutional layer. At testing time, they record $h_k(x, y)$, which is the activation of the k-th unit at location (x, y). As a result the contribution of the k^th unit is $\sum_{x,y} h_k(x, y)$. To get the localization map for class c, they compute $\sum_{x,y} \sum(k) w_k^c h_k(x, y)$ and project it back onto the input image using upsampling methods. This method produces good quality localization heat maps but the network must have a specific architecture during training which is having average pooling layer. Also, they can utilize the backpropagated signal to further refine the maps which was addressed by Selvaraju et al. (2017).

Selvaraju et al. (2017) presented a method which uses the gradient flow to the last convolutional layer to measure the importance of each neuron. First, the feature maps are computed by using a global average pooling layer. Second, the derivative of the class with respect to these feature maps are taken:

$$\begin{aligned} \beta_k^c &= \frac{1}{Z} \sum_i \sum_j \frac{\partial y_c}{\partial A_i j^k} \\ \mathbf{e}_{Grad-CAM} &= ReLU(\sum_k \beta_k^c A^k) \end{aligned} \tag{2.8}$$

in which β_k^c is the weight of the c^{th} class of the k^{th} convolutional layer. To further improve their results, they combined Guided Back-propagation (Springenberg et al., 2015) with their results in the following way

$$\mathbf{e}_{GuidedGrad-CAM} = \mathbf{I} \odot \mathbf{e}_{Grad-CAM} \odot \mathbf{e}_{Guided-BP} \tag{2.9}$$

which is the combination of the Grad-CAM explanation and guided backpropagation (Springenberg

et al., 2015) on to the image.

Dabkowski and Gal (2017) tried to train a masking model which changes the classifier's score by masking the salient parts. First, they defined two measures of saliency: smallest sufficient region (SSR) and smallest destroying region (SDR). SSR is the smallest input region that could be removed and swapped with alternative reference values in order to minimize the classification score. In the same manner, SDR is the smallest input region that could substituted into a fixed reference input in order to maximize the classification score.The loss over a mask M is defined as

$$L(M) = \lambda_1 TV(M) + \lambda_2 AV(M) - log(D(\phi(\mathbf{I}, M))) + \lambda_3 D(\phi(\mathbf{I}, 1 - M))^{\lambda_4} \tag{2.10}$$

in which TV is total variation loss and AV is average loss. They used a U-Net (Ronneberger et al., 2015) structure to train the masks. To prevent the model for overfitting to a single evidence removal, they proposed the following scheme. They used two different transformation functions, $\phi_1(.), \phi_2(.)$. In half of the cases, they used $\phi_1(.)$ which is a blurred version of the input and in the other half they used a random color image with Gaussian noise. This scheme improved the quality of the masks produced.

In (Simonyan et al., 2014), a method has been presented which tries to numerically generates an image $\hat{\mathbf{I}}$ which is representative of the probe class in terms of y_{probe}. So, the $\hat{\mathbf{I}}$ can be found as:

$$\hat{\mathbf{I}} = \arg\max_I [D(I)]_c - \lambda ||I||_2^2 \tag{2.11}$$

and choose zero image for the initial point. To get the specific saliency map for a given image $\mathbf{I}$, they backpropagate the loss to the image, assuming the first order Taylor expansion of the network. This

can be written as $\frac{\partial D_c}{\partial I}|_{I=\mathbf{I}}$. This method is very similar to (Zeiler and Fergus, 2014) except they treat ReLU without modification.

In (Springenberg et al., 2015), the authors presented a method called guided back propagation. They modified the back-propagation rule for ReLU function. The rest is similar to (Simonyan et al., 2014). Is summary, the last three methods can be summarize as:

$$\begin{aligned} h_i^{l+1} = ReLU(h_i^l); \quad R_i^{l+1} = \frac{\partial L}{\partial h^{l+1}} \\ backprop = R_i^l = (h_i^l > 0) R_i^{l+1} \\ guidedbackprop = (h_i^l > 0)(R_i^{l+1} > 0) R_i^{l+1} \\ deconv = (R_i^{l+1} > 0) R_i^{l+1} \end{aligned} \tag{2.12}$$

Yosinski et al. (2015), provided a tool to inspect each neuron and see how to reacts to any input image. They showed that the early neurons are specialized in low level features such as edges while higher level neurons more specialized to represent a concept such as flower or dog. To do the visualization over the input image, they proposed the following optimization problem which is the general version of (Simonyan et al., 2014):

$$\mathbf{I}^\star = \arg\max_I (\alpha_i(I) - \mathcal{R}_\theta(I)) \tag{2.13}$$

in which $\alpha_i(I)$ is the output of a layer or equivalently, output of the model D. $R_\theta(I)$ applies regularization on the images such as L_2 decay, Gaussian blur, clipping pixels with small norms and clipping pixels with small contribution. Although this kind of approaches can be used to check inside the network, they are not of that much use when it comes to deep structures with thousands of

neurons to inspect.

Mahendran and Vedaldi (2015), tried to find an explanation by inverting a representation:

$$\begin{aligned} \mathbf{I}^{\star} &= \underset{\mathbf{I} \in R^{H \times W \times channels}}{\arg\min} ||\phi(\beta \mathbf{I}) - \phi_0||_2^2 / ||\phi_0||_2^2 + \lambda \mathcal{R}_{V^\beta}(I) + \lambda_\alpha \mathcal{R}_\alpha(\mathbf{I}) \\ \mathcal{R}_{V^\beta}(\mathbf{I}) &= \sum_{i,j} ((\mathbf{I}_{i,j+1} - \mathbf{I}_{ij})^2 + (\mathbf{I}_{i+1,j} - \mathbf{I}_{ij})^2)^{\frac{\beta}{2}} \\ \mathcal{R}_\alpha(\mathbf{I}) &= ||\mathbf{I}||_\alpha \end{aligned} \tag{2.14}$$

in which ϕ_0 is the representation to be inverted, $\mathcal{R}_{V^\beta}(\mathbf{I})$ is total variation loss, and β is the average Euclidean norm of images in the training set. They proposed a Total Variation regularizer. Also, they showed how to implement DeepSIFT and HOG in a CNN so, their gradients can be computed.

D to label them as negative or positive evidence.At the end, by using K-Lasso method, they find k related

Ribeiro et al. (2016) proposed a method which interprets individual model predictions based on locally approximating the model around a given prediction in lower dimensions. They find a linear model around the samples in the lower dimension and treat the weights as interpretable components. For images, this can be achieved by representing an image as superpixles, i.e., projection to a lower dimension and interpretable space. We call this version, $\mathbf{I}'$. A mapping converts the binary vector of this interpretable version back into the original space. For images, it is the mapping that keeps the superpixel if its value is 1 and replaces the super pixel with the average of its neighbors if the value is 0. The explanation can be found as

$$\mathbf{e} = \underset{g \in \mathcal{G}}{\arg\min}\, L(D, g, \pi_{\mathbf{I}'}) + \mathcal{R}(g). \tag{2.15}$$

Faithfulness of the explanation model to the original model is enforced through the loss function L over a set of samples in the simplified input space weighted by the local kernel $\pi_{\mathbf{I}'}$. The regularization term $\mathcal{R}$ tries to keep the model simple. For instance, it can be depth of the tree for decision trees or a penalty over sparse linear models.

In Samek et al. (2017) a method for layer-wise backpropagartion in presented which can handle nonlinearities such as layer renormalization. The aim is to assign each pixel a relevance score such that $D(\mathbf{I}) \approx \sum_p R_p^1$. Assume the relevance of neuron i at layer l is:

$$R_i^l = \sum_{j \in (l+1)} R_{i \leftarrow j}^{l,l+1} \tag{2.16}$$

This equation defines the propagation of relevance from layer l+1 to the l^{th} layer. The paper defines few rules to compute the relevance such as $R_{i \leftarrow j}^{l,l+1} = \frac{h_{i,j}}{h_j + \epsilon.sign(h_j)} R_j^{l+1}$

In Ren et al. (2018), authors proposed a method which can reweigh training examples to make the network more robust to adversarial examples. One prerequisite of this method is the availability of an unbiased and clean dataset. To find the weights, they need to solve:

$$\theta^\star(\beta) = \arg\min_\theta \sum_i \beta_i D(\mathbf{I}_i, \theta) \tag{2.17}$$

in which βs can bee seen as hyper parameters to be learned from validation examples $\mathbf{I}^v$:

$$\beta^\star = \arg\min_{w, w \geq 0} \frac{1}{M} D(\mathbf{I}_i^v, \theta^\star) \tag{2.18}$$

To avoid having a nested loop for the optimization part, they followed the approach of (Koh and Liang, 2017)

In Bau et al. (2017), the authors has been proposed a method which identifies the semantics of hidden units and align them with concepts that are interpretable for humans. They also examined the effects of training techniques such as batch normalization and dropout on the interpretability. They measure a disentangled representation i.e., alignment between single units and interpretable concepts. To quantify the interpretability of a layer, they count the number of distinct visual concepts that are aligned with a unit in the layer. Using this approach, the proposed method does not rely on backpropagation or any training and can be computer using a forward pass. The concepts are drawn from Broaden dataset. For every image $\mathbf{I}$ in the dataset , they compute the activation map $A_k(\mathbf{I})$. Then, thresholding the activation maps and resizing it to the size of input, gives a mask. To measure how much the concepts are aligned, they used the intersection over union measure (IoU). They also showed that interpretable units emerge because learning converges to a special basis that aligns explanatory factors with individual units. They showed different conditions in training phase changes the interpretability measure which is the number of unique detectors. While different initializations result in almost same interpretability, having dropout units emerges the neurons with more textures than object detectors. Batch normalization drops the interpretability measure based on their experiments.

Chapter 3

CDeepEx

In this chapter we propose a method which can *visually* explain the classification decision of deep neural networks (DNNs). Many methods have been proposed in machine learning and computer vision seeking to clarify the decision of machine learning black boxes, specifically DNNs. All of these methods try to gain insight into why the network "chose class A" as an answer. Humans search for explanations by asking two types of questions. The first question is, "Why did you choose this answer?" The second question asks, "Why did you *not* choose answer B over A?" The previously proposed methods are not able to provide the latter directly or efficiently.

We introduce a method capable of answering the second question both directly and efficiently. In this work, we limit the inputs to be images. In general, the proposed method generates explanations in the input space of *any* model capable of efficient evaluation and gradient evaluation. It does not require any knowledge of the underlying classifier nor use heuristics in its explanation generation, and it is computationally fast to evaluate. We provide extensive experimental results on three different datasets, showing the robustness of our approach, and its superiority for gaining

insight into the inner representations of machine learning models. As an example, we demonstrate our method can detect and explain how a network trained to recognize hair color actually detects eye color, whereas other methods cannot find this bias in the trained classifier.

In summary, the existing approaches in literature have at least one of the following downsides.

I They are not always applicable since they need specific layers or architecture (Zhou et al., 2016; Selvaraju et al., 2017).

II They use heuristics during backpropagation to generate explanations (Springenberg et al., 2015; Zeiler and Fergus, 2014).

III They need of a set of probe images or concepts which may not be available or cannot be easily obtained (Shrikumar et al., 2017).

IV They need network alteration to record the activations (Zeiler and Fergus, 2014; Bau et al., 2017; Shrikumar et al., 2017).

V They need of considerable amount of computational time to provide an explanation (Zintgraf et al., 2017; Ribeiro et al., 2016).

VI By learning an explanation network, they are trained to produce the explanations we desire, rather than faithful explanations of the network's function (Park et al., 2018).

We have replaced these downsides (assumptions, heuristics, probe images, or other black-boxes) with a generative latent-space model (built with a GAN or VAE, AE). We submit this latent-space model imposes less bias on the explanations, while still providing a human-understandable explanation. The unbiased, true explanation for any classification is the (long) sequence of calculations performed. For human understanding, these must be filtered through some type of lens.

We believe the latent input space is a natural bridge between the network's understanding and the human's understanding, as it is common data (or language) to both, unlike raw pixels or activation values (which are natural for the network, but not for the human). Other bridges, like natural language or scene objects, require assumptions about how the network works (which call into question whether the explanation is valid) or training up other networks to make the bridge. While we also require another network, there is less bias imposed by modeling the space of natural images than by modeling the mapping from neural network concepts to natural language or other higher-level concepts.

Our lens, of the latent-space parameterization of the input domain, can be trained on completely different data than that used for the classifier to be analyzed (see Section 3.2.6), can be shared across multiple classifiers with the same input domain (thus providing a standard testing tool), and can be improved and tested externally and independently from the classifier-to-be-analyzed. We further show experiments demonstrating that the generative model does not project its own training set (see Section 3.2.5) onto the explanations for a network trained on different data.

3.1 Proposed Method

First, we introduce the notation used in this work in Section 3.1.1. Then, in Section 3.1.2, we describe how to learn a latent space capable of generating natural looking images similar to the input space. Last, in Section 3.1.4, we describe our method on how to generate explanations from the latent representation of input space. The overall framework is summarized in Procedure 1 and Figure 3.1.

3.1.1 Terminology

$D : R^n \rightarrow R^c$ is the given discriminator network, for which we want to generate explanations. $G : R^k \rightarrow R^n$ generates natural looking images in the input domain. It should be able to generate images similar to those being queried, but need not be directly related to D. $\mathbf{I} \in R^n$ is the input image; $z \in R^k$ is a latent variable; and $\mathbf{I_z}$ is the output of G, i.e. $\mathbf{I_z} = G(z)$. y_{true} is the label produced by D for image I, and y_{probe} is the class label for the class of interest. Note that y_{true} may not be the true label from an accuracy point-of-view, but it is the true label produced by the discriminator network D (compared with y_{probe} which is a counter-factual label not actually produced by D).

Thus, the question we would like to answer is "Why did D produce label y_{true} and not label y_{probe} for input $\mathbf{I}$?"

Procedure 1 Generating the explanation on given D and $\mathbf{I}$

1: Learn a function $G : \mathbb{R}^k \rightarrow \mathbb{R}^n$ ▷ See 3.1.2
2: Find a representation for input $\mathbf{I}$ using procedure 2
3: Find $\mathbf{z}_e$ from Equation 3.2
4: Return the explanation $G(\mathbf{z}_0) - G(\mathbf{z}_e)$

Procedure 2 Getting latent representation on the input $\mathbf{I}$

1: **procedure** LEARN $\mathbf{z}_0$ $(G, \mathbf{I}, \lambda, \ell(.))$
2: $\quad \mathbf{z}_0 \sim \mathcal{N}(0, 1)$
3: $\quad$ **while** $G(\mathbf{z}_0) \not\approx \mathbf{I}$ **do**
4: $\quad\quad \mathbf{z}_0 \leftarrow \mathbf{z}_0 - \lambda \nabla_{\boldsymbol{z}} \ell(\mathbf{I}, G(\mathbf{z}))$
5: $\quad$ **end while**
6: $\quad \Delta_{\mathbf{z}_0} = G(\mathbf{z}_0) - \mathbf{I}$
7: $\quad$ **return** $\mathbf{z}_0, \Delta_{\mathbf{z}_0}$
8: **end procedure**

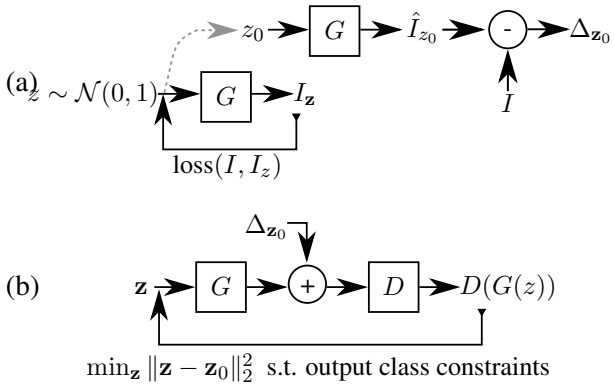

Figure 3.1: The schematics of the proposed approach. (a) First, in the case of using a GAN, find $\mathbf{z}_0$ which gives $G(\mathbf{z}_0) \approx \mathbf{I}$. (If using a VAE, the $\mathbf{z}_0$ that best approximates $\mathbf{I}$ can be generated directly.) Then, let $\Delta_{\mathbf{z}_0}$ to be the difference between the input image $\mathbf{I}$ and reconstructed one, $G(\mathbf{z}_0)$. (b) Last, generate the final explanation by optimization over $\mathbf{z}$. $\mathbf{z}_0$ and $\Delta_{\mathbf{z}_0}$ are fixed.

3.1.2 Learning the Input Distribution

The question "why not class y_{probe}?" implies a query image about which the question is being asked. We need to capture the manifold of natural looking images similar to this input to be able to answer this question in a meaningful way for a human. Learning a compact manifold enables us to move along this manifold instead of directly optimizing in the input space of raw pixels.

There are different ways to find this mapping including variational auto encoders (VAEs) (Kingma and Welling, 2014) and generative adversarial networks (GANs) (Goodfellow et al., 2014). In this work, we use GANs and VAEs to map latent space into input space. We used the method proposed by Arjovsky et al. (2017) to train the GAN. The structure of the networks is similar to that proposed by Radford et al. (2015).

3.1.3 Why not backprop directly

If we directly backpropagate the loss onto the image, the explanation will be on pixel level. This king od explanation is hard to follow since pixels can be highlighted from different part of the

image which makes it unnatural. To alleviate this, we slide windows with different sizes and choose the windows which have the largest average change of the loss function. See Figure 3.2.

3.1.4 Generating Explanations

First, we need to find an initial point in the latent space which represents the input image, i.e., $G(\mathbf{z}_0) \approx \mathbf{I}$. If G was generated by a VAE, this can be done by feeding $\mathbf{I}$ into the encoder half of the VAE. If G was generated by a GAN, we find this initial point by solving $\mathbf{z}_0$ as

$$\mathbf{z}_0 = \arg\min_z \ \ell(G(z), \mathbf{I}) \tag{3.1}$$

in which $\ell(.)$ is a suitable loss function, e.g. $||.||_2$ distance for images.

Since the generated image $G(z)$ may be classified into a different class by the discriminator, we add a bit of the misclassification cost of for F to our loss function to insure that the resulting image not only looks similar, but is also classified the same way as $\mathbf{I}$ by D. We only add this extra cost for the GAN generators. As the final fit will not be exact, we define $\Delta_{\mathbf{z}_0}$ to be the residual error: $\Delta_{\mathbf{z}_0} = G(\mathbf{z}_0) - \mathbf{I}$. This residual makes the latent space generated image exactly equal to the input image. See Procedure 2 for more details.

Next, we find a change in latent space for which the change in the input space explains why, for this particular image, class y_{probe} is not the answer. In particular, we seek the closest point (in the latent space of $\mathbf{z}$) for which y_{probe} would be equally likely as y_{true}, and all other classes are less probable. This is a point which is most like the input, yet would be class y_{probe}. Thus, the answer to the question is, "It is *not* like *this*."

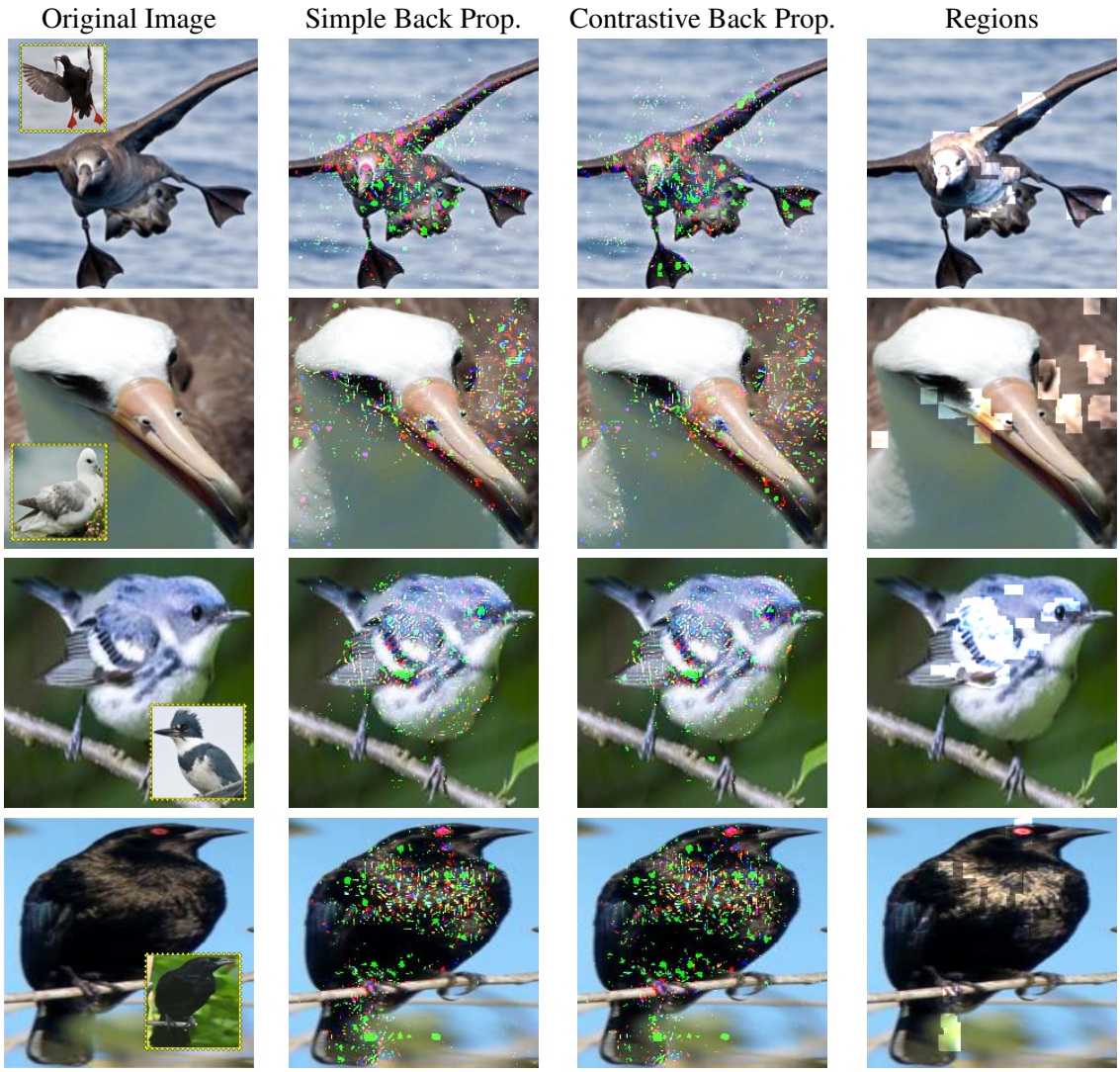

Figure 3.2: First column shows the input image and the second most probable class. The second and third columns show the effect of the explanation if the loss function directly backpropagated to the image space. The last column shows the results of sliding a window over the pixels and select the regions with the most average change.

To do so, we solve a constrained optimization problem:

$$
\begin{aligned}
\mathbf{z}_e = \quad & \arg\min_{\mathbf{z}} ||\mathbf{z} - \mathbf{z}_0||_2^2 \\
\text{s.t.} \quad & \text{llh}\left(D(\mathbf{I}_{\mathbf{z},\mathbf{z}_0}), y_{\text{true}}\right) - \text{llh}\left(D(\mathbf{I}_{\mathbf{z},\mathbf{z}_0}), y_{\text{probe}}\right) = 0 \\
& \text{llh}\left(D(\mathbf{I}_{\mathbf{z},\mathbf{z}_0}), y_{\text{true}}\right) - \text{llh}\left(D(\mathbf{I}_{\mathbf{z},\mathbf{z}_0}), y'\right) \leq \epsilon \\
& \text{llh}\left(D(\mathbf{I}_{\mathbf{z},\mathbf{z}_0}), y_{\text{probe}}\right) - \text{llh}\left(D(\mathbf{I}_{\mathbf{z},\mathbf{z}_0}), y'\right) \leq \epsilon \\
& \forall y' \neq y_{\text{true}}, y' \neq y_{\text{probe}}
\end{aligned}
\tag{3.2}
$$

in which $\mathbf{I}_{\mathbf{z},\mathbf{z}_0} = G(\mathbf{z}) + \Delta_{\mathbf{z}_0}$ and $\text{llh}(f, y)$ is *log-likelihood* of class y for classifier output f. Our visual explanation is the difference between $\mathbf{I}$ and $\mathbf{I}_{\mathbf{z}_e,\mathbf{z}_0}$.

The first constraint forces the explanation to be on the boundary of the two classes. However, because D is complex, this can lead to solutions in which the likelihood of y_{probe} and y_{true} are equal, but another class has an even higher likelihood. To overcome this, we impose the last two constraints which enforce that the class produced by D and the probe class remain the most likely. We further illustrate the necessity of these constraints in Section 3.2.

Optimization Method Using the method of augmented Lagrangian multiplier, we can convert the constrained optimization problem into a series of unconstrained ones. Complete details of the optimization procedure can be found in (Bertsekas, 1999). In summary, we convert Equation 3.2 into

the augmented Lagrangian

$$
\begin{aligned}
L_c(\mathbf{z}, \lambda, \mu) = ||\mathbf{z} - \mathbf{z}_0||_2^2 & \\
&+ \lambda^T h\left(\mathbf{z}, y_{\text{true}}, y_{\text{probe}}\right) + \frac{c}{2}\left\|h\left(\mathbf{z}, y_{\text{true}}, y_{\text{probe}}\right)\right\|_2^2 \\
&+ \frac{1}{2c}\sum_{\substack{y' \neq y_{\text{true}} \\ y' \neq y_{\text{probe}}}} \left\{\left(\max\left\{0, \mu_{y'} + ch\left(\mathbf{z}, y_{\text{true}}, y'\right)\right\}\right)^2 - \mu_{y'}^2\right\} \\
&+ \frac{1}{2c}\sum_{\substack{y' \neq y_{\text{true}} \\ y' \neq y_{\text{probe}}}} \left\{\left(\max\left\{0, \tilde{\mu}_{y'} + ch\left(\mathbf{z}, y_{\text{probe}}, y'\right)\right\}\right)^2 - \tilde{\mu}_{y'}^2\right\}
\end{aligned}
\tag{3.3}
$$

where

$$
h(\mathbf{z}, y_1, y_2) = \text{llh}\left(D(\mathbf{I}_{z,z_0}), y_1\right) - \text{llh}\left(D(\mathbf{I}_{z,z_0}), y_2\right) = 0
$$

and λ, $\mu_{y'}$, and $\tilde{\mu}_{y'}$ are Lagrange multipliers for the constraints of Equation 3.2.

We solve the constrained optimization problem by solving a series of unconstrained optimizations of Equation 3.3 for a series of c values. Each individual optimization we solve with standard gradient descent. The update rule for c is

$$
c^{k+1} = \begin{cases} \beta c^k & \text{if } \left\|h\left(\mathbf{z}^k, y_{\text{true}}, y_{\text{probe}}\right)\right\| > \gamma \left\|h\left(\mathbf{z}^{k-1}, y_{\text{true}}, y_{\text{probe}}\right)\right\| \\ c^k & \text{otherwise} \end{cases}
\tag{3.4}
$$

in which γ is 0.24, β is 1.01 and initial value of c is 1.

3.2 Experimental Results

We compared our method, contrastive deep explanation (CDeepEx), with those of Ribeiro et al. (2016); Selvaraju et al. (2017); Zintgraf et al. (2017); Samek et al. (2017). Some of these methods try to answer the question "why class A?" instead of our contrastive question. However, it would be natural to try use the difference of the answers to "why class A?" and "why class B?" as a contrastive answer. Thus, we compare to them to demonstrate the need for a different method to answer these contrastive questions.

First, we tested these methods on two datasets: MNIST (LeCun et al., 1998) and fashion-MNIST (Xiao et al., 2017). Although MNIST seems to be a very simple dataset, it has its own challenges when it comes to contrasting two outputs. This dataset has ambiguities that are hard even for humans to identify. We also expect the reader to has prior knowledge of the digits (compared to, perhaps, birds or skin cancers), thus making the results easier to interpret. We conducted experiments using two different generative models (GANs) and Variational Auto Encoders (VAEs). We show that our method works well regardless of the choice of the generator.

3.2.1 MNIST

In this section, we find explanations for contrasting categories using our method (CDeepEx), Lime (Ribeiro et al., 2016), GradCam (Selvaraju et al., 2017), PDA (Zintgraf et al., 2017), LRP Samek et al. (2017) and xGEMs (Joshi et al., 2018). The network architecture for D is similar to that used by the original MNIST papers, consisting of two sets of Conv/MaxPool/ReLU layers following

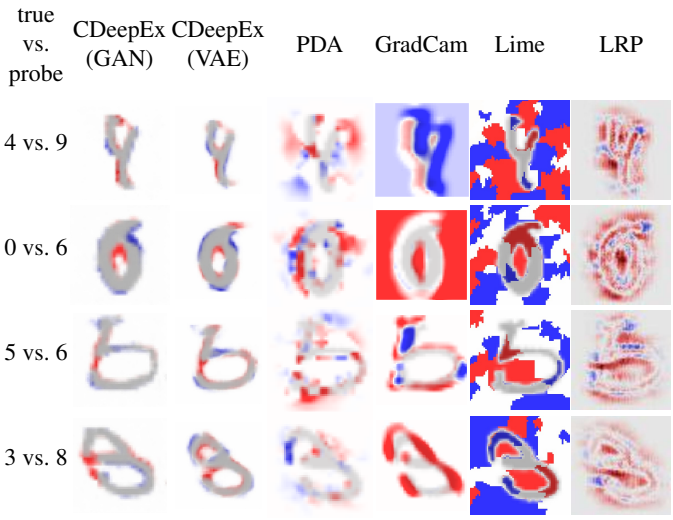

Figure 3.3: Generated explanations for MNIST for examples correctly classified by the network. The input image is in gray. Red indicates regions that should be added and blue regions that should be removed (to move from the true/predicted label to the probe label). Thus, the absence of the red regions and the presences of the blue regions explain why the input is the true label and not the predicted label.

.

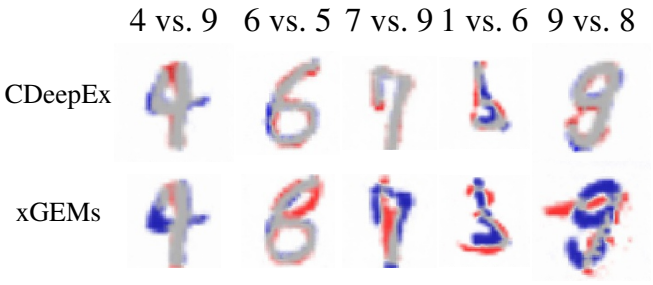

Figure 3.4: Comparison of CDeepEx to xGEMs. Colors are as in Figure 3.3. Please check Figure 3.6 for more qualitative results.

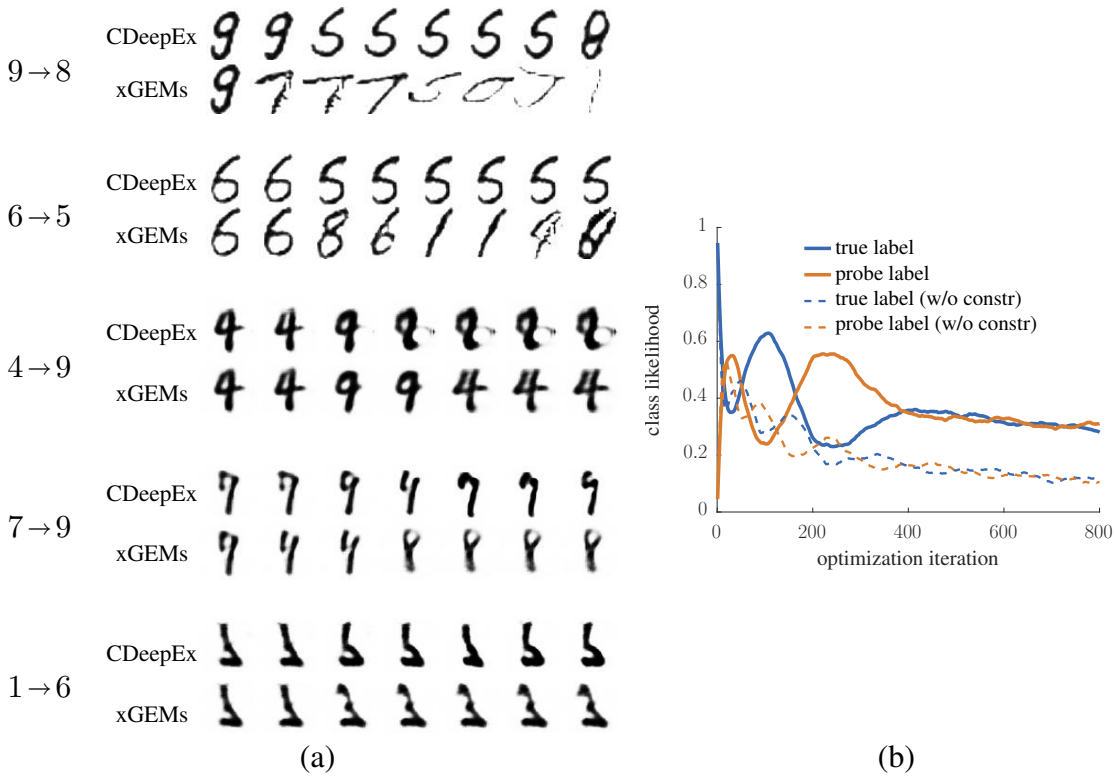

Figure 3.5: Examples of the optimization path for Equation 3.2. CDeepEx is our method. xGEMs is very similar, but without the last pair of constraints. The goal is to find a $\mathbf{z}$ such that $G(\mathbf{z})$ is maximally confused between the two classes. (a) Examples from $G(\mathbf{z})$ along the optimization path of $\mathbf{z}$. For the first two examples, we used a GAN as a generative model. For the rest of the examples, we used a VAE as the generator. The Figure shows our approach works better regardless of the choice for the generator model. (b) Average class likelihood from $D(G(\mathbf{z}))$ for all examples (probe is second-most-likely class). Both demonstrate that without the constraints (xGEMs), the optimization finds an example for which the true and probe likelihoods are equal, *but* another class has an even higher likelihood. Our method (CDeepEx) with the constraints keeps the explanation targeted to the true and probe classes.

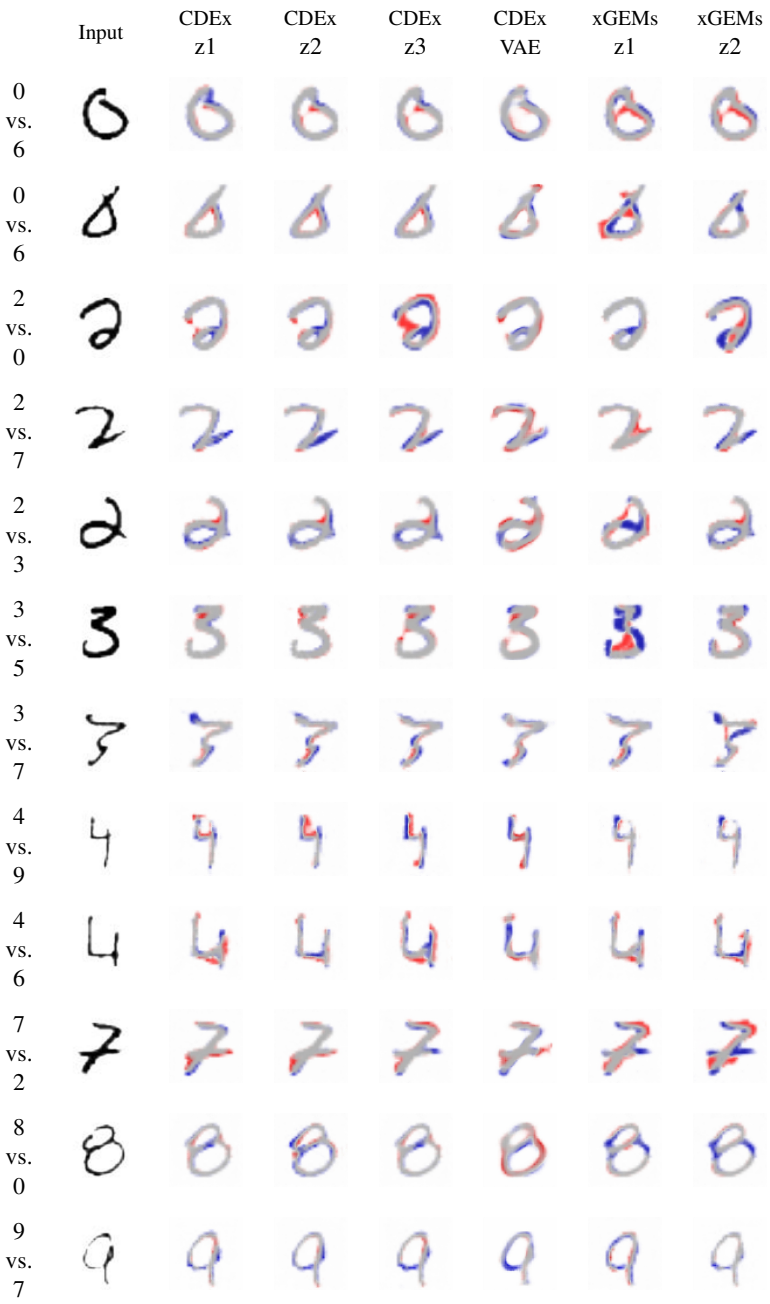

Figure 3.6: Additional experiments comparing our method using a GAN or a VAE with xGEMs. The multiple columns for the GAN methods are for different random starting points for $\mathbf{z}_0$.

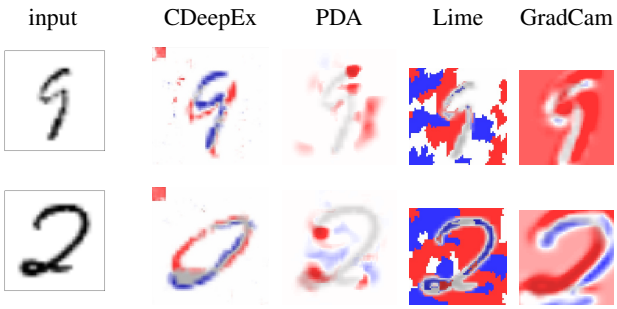

Figure 3.7: Explanations for "why not 8?" for a network trained on data in which every 8 has a small square in the upper-left corner.

by two fully connected layers. The input is resized to 64x64. The kernel size for each convolution layer is 5x5 and the first fully connected layer is 3380x50.

GradCam, Lime and LRP were not designed to answer this type of question directly. However, these methods could be shoe-horned into trying to answer the question of"why A and not B?" and so we figured we should demonstrate that they were not sufficient and that a new method (like ours) was necessary. Therefore, we generate the explanation by first extracting the explanations for the true and probe classes. Then we assign different weights to each of these explanations and subtract them from each other. If imposing this explanation on the input image decreases the network probability for the true class and increases it for the probe class, we keep this explanation. The weights for the probe explanation we used are from 0.005 to 2 with increments of 0.005. The final explanation is the average of all the maps that satisfies the mentioned condition. We tried multiple other methods to contrast the explanations of the true and probe classes and this method produced the best and most reliable results, although clearly GradCam and Lime were not designed for this scenario.

GAN structure follows the guidelines of (Radford et al., 2015). It has 5 set of ConvTranspose2d/BatchNorm2d/ReLU operations. The size of $\mathbf{z}$ is 100. The VAE has four convolutional layers

and two fully connected layers, one for μ and one for σ. The size of the latent code is 400 for each of the mean and standard deviation.

General Comparisons Figure 3.3 shows explanations generated using CDeepEx, PDA, GradCam, Lime and LRP. We trained the discriminator on the unmodified MNIST dataset with a resulting success rate of 99% in testing phase. For testing examples, we asked why the output was not the second-highest likelihood class. (That is the probe class is the second-most likely class, according to the classifier.)

The first example shows why D believes the class to be a 4 and not a 9. Our method shows that this is because the gap on the top of the digit is not closed. The second row is the explanation for why a 0 and not a 6. The only meaningful explanation is generated from CDeepEx: because the circle is thicker and the top extending line is not more pronounced. The third row is for why a 5 and not a 6. Although Lime shows that the opening gap would have to be closed, it produces many other changes that detract from the central reason. For the last row, the only correct explanation for why a 3 and not an 8 comes from our method. Note that GradCam and Lime were not designed for this type of explanation, but we tried hard to find the best way to make contrasting explanations from their single-class explanations (see above).

Comparisons to xGEMs and Optimization In the next experiment, we show the importance of having the constraints in the optimization and how it affects the explanation compared to xGEMs. In Figure 3.4, we compare our results with our implementation of xGEMs. The primary difference between the methods is the last two sets of constraints in Equation 3.2 (which are present in our method, but lacking from theirs). This Figure shows that without the constraints, the explanation are

not correct or are of worse quality, compared to explanations with constraints.

To illustrate why, in Figure 3.5 we show the optimization path to find the explanation, with (CDeepEx) and without (xGEMs) constraints. The program has a non-linear objective with non-linear constraints, so the path is particularly important, as we will not (generally) find the global optimum. Without the constraints, the found $\mathbf{z}$ results in equal likelihood for y_{true} and y_{probe}, but a third class consistently has even higher likelihood, thus generating an explanation more suitable to this third class than the requested probe class. This is typical for xGEMs and explains the xGEMs row of Figure 3.4, where frequently a third random class is superimposed on the explanation.

Figure 3.6 provides additional experiments showing that the results are reasonably stable with respect to the initialization point for the GAN generator.

To illustrate why, in Figure 3.5, we show the optimization path to find the explanation, with (CDeepEx) and without (xGEMs) constraints. The program has a non-linear objective with non-linear constraints, so the path is particularly important, as we will not (generally) find the global optimum. We use an augmented Lagrangian method for the optimization (Bertsekas, 1999, Section 4.2). Without the constraints, the found $\mathbf{z}$ results in equal likelihood for y_{true} and y_{probe}, but a third class consistently has even higher likelihood, thus generating an explanation more suitable to this third class than the requested probe class.

3.2.2 Biased MNIST

To test to see if our method could provide clear explanations of the bias of a classifier, we trained a network with the same structure as the previous experiments, but on modified data. In this experiment we added a 6x6 gray square to the top left of all the images for class "8." If tested on the true data (without the modifications), the network only recognizes 5% of the 8s correctly. Also, if we

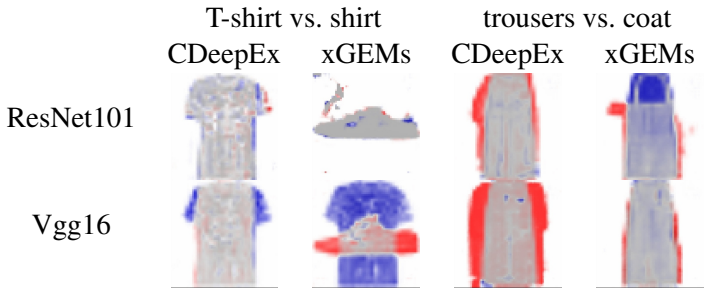

Figure 3.8: Top and bottom rows are the results for ResNet101 and VGG16 respectively. (a) Change from T-shirt to a Shirt with constraints. (b) T-shirt to shirt without constraints. (c) Change from trousers to coat with constraints. (d) Change from trousers to coat without constraints.

add the square to all the testing images, the network recognized 77% of other classes as "8."

We then compare our method to others in explaining the predictions of this biased network Our results are shown in Figure 3.7. Our method is clearly able to articulate that adding a square in the upper left makes something an 8, whereas the other methods are not able to find this undesirable bias of the network.

3.2.3 Fashion MNIST

We trained two different networks for D on the Fashion MNIST dataset: Vgg16 (Simonyan and Zisserman, 2015) and ResNet101 (He et al., 2016). The testing accuracy for both networks is 92%. For the generating network, G, we used the structures and learning method presented by Arjovsky et al. (2017) with latent space of size 200. We then illustrate our method's ability to gain insight into the robustness of the classifiers through contrastive explanations. The generator network's structure is the same as the MNIST's generator network's structure.

Figure 3.8 shows the generated explanations with and without constraints. Comparing CDeepEx with xGEMs, we can see the importance of the constraints. Without them (xGEMs), the results refer to other irrelevant classes.

Using CDeepEx, it is clear that Vgg16 learned more general concepts than ResNet101. The first column shows that ResNet101 learned very subtle differences on the surface of the T-shirt to distinguish it from a (long-sleeved) shirt. By contrast, Vgg16 understands removing the short sleeves makes the appearance of a shirt with long sleeves. In the "trousers vs. coat" example, ResNet101 believes that adding a single sleeve will change the trouser into a coat, while Vgg16 requires both sleeves.

3.2.4 CelebA Dataset

We ran experiments on the CelebA dataset Liu et al. (2015). We trained Vgg16 network on a subset of the images with target labels of "blonde female," "dark-haired male," and "dark-haired female." (There were not sufficient numbers of blonde males in the training set.) The resulting classifier achieves 95% accuracy. For the generator network, we trained an auto encoder (AE) to generate images. The encoder consists of five convolutional layers with kernel size of 3 and stride of 2. The number of filters for each layer are 16, 64, 128, 256, 512, and 1024. We perform a ReLU operation after forwarding through each layer.

We asked questions about why the network chose hair color it did, and not the other hair color for the same gender. Our goal is to discover if if the network learned the correct concept class. It should be noted that the images have weak labels, possibly causing the network to learn features that are correlated to the provided features. As long as the network has learned consistent set of features, we are able to find reasonable explanations for its decisions.

The right column of Figure 3.9 shows that xGEMs produces an explanation that covers

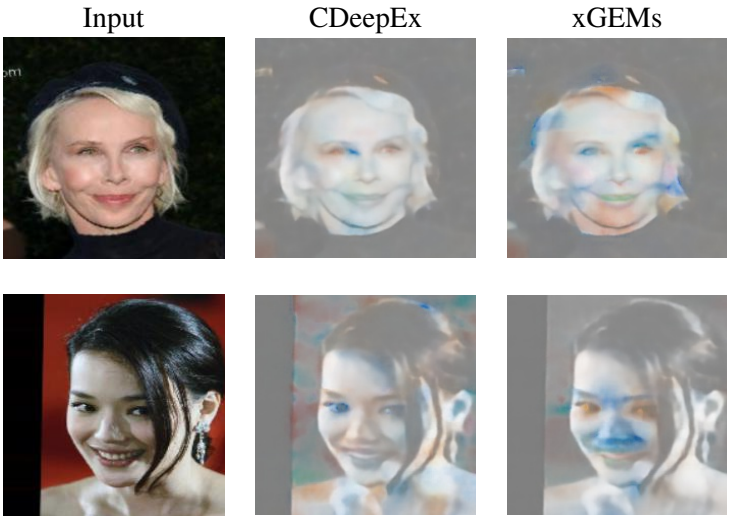

Figure 3.9: CDeepEX vs xGEMs experiments. The top row is the result of a query "why is the person not dark-haired?" on a blonde female. We see that the network pays more attention to the eye color than the hair color. In the second row, the probe class is blonde and true class is dark-haired, for the inverse query. We can see generated explanations from xGEMs and CDeepEx disagree over the "sign" of change in the eyes.

most of the face. It is not very specific. This is because it is missing the necessary constraints to keep the explanation just between the classes of blonde females and dark-haired females. It produces explanations that change the result to male (changing the facial structure), despite that the question does not ask about this class. The middle column shows that our method is more targeted to the real difference learned by the classifier: The difference learned by the classifier is in the eyes!

The top row of Figure 3.10 shows experiments with imposed changes that demonstrate that the learned discriminative network learned eye color and not hair color. In particular, lightening the eyes changes the reported hair color to blonde. However, xGEMs (which does not have the constraint on non-probe, non-true classes), gave the reverse explanation (darkening the eyes would produce a blonde classification) which was not borne out by our experiment. Our method (CDeepEx) correctly identified the correlation between hair and eyes. Figure 3.11 shows more experiments on the dataset. In general the discriminator learned about eyes, eyebrows and side hairs, but not about the "true hair

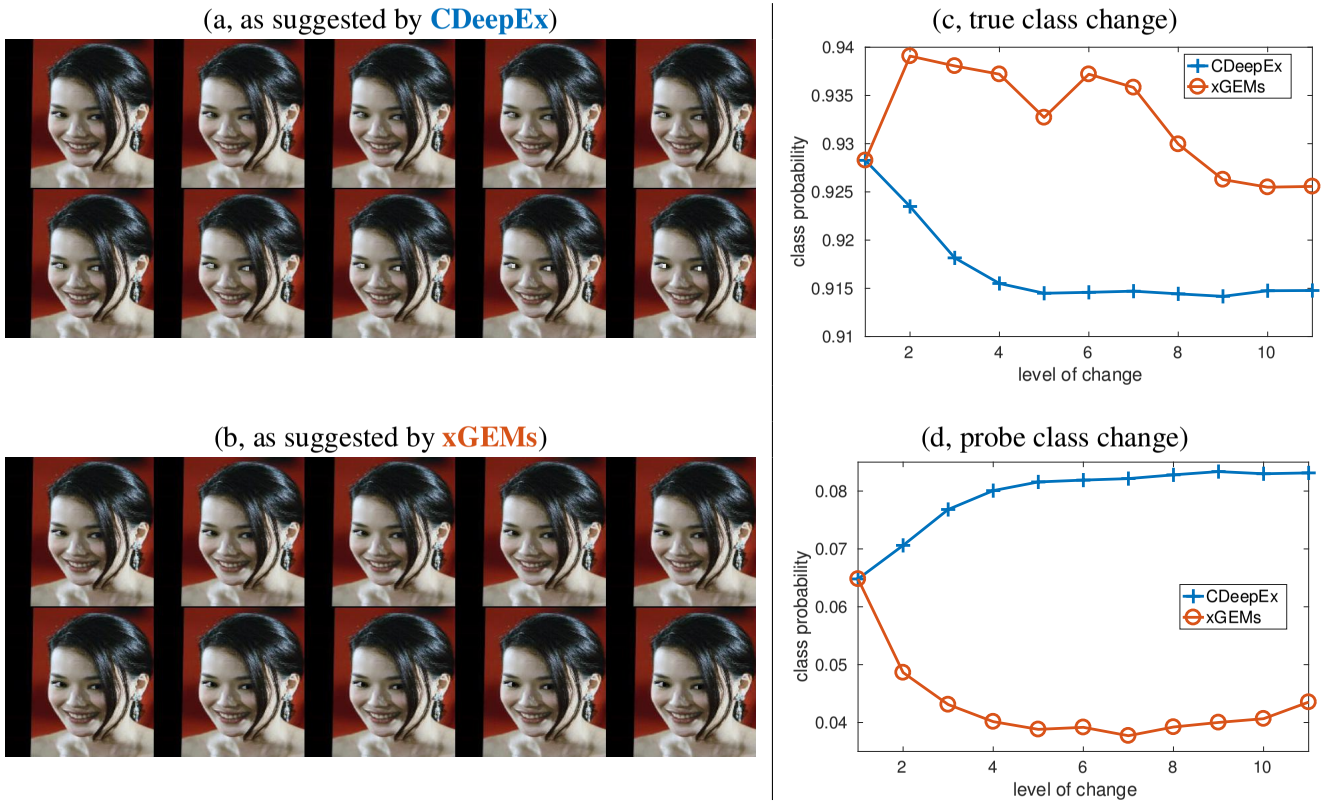

Figure 3.10: (a) We manually increase eye intensity from its original value to maximum, as suggested by our method, CDeepEx. (b) We decrease eye intensity from its original value to minimum, as suggested by xGEMs. (c) and (d) show the plots of how the discriminator network D's outputs change for the probe (blonde female) and reported/true (dark-haired female) classes. This demonstrates that xGEMs generates the wrong explanation of the correlation of the hair with the eyes, whereas the change suggested by CDeepEx correctly moves the reported class of the discriminator network from the true class (dark hair) to the probe class (blonde).

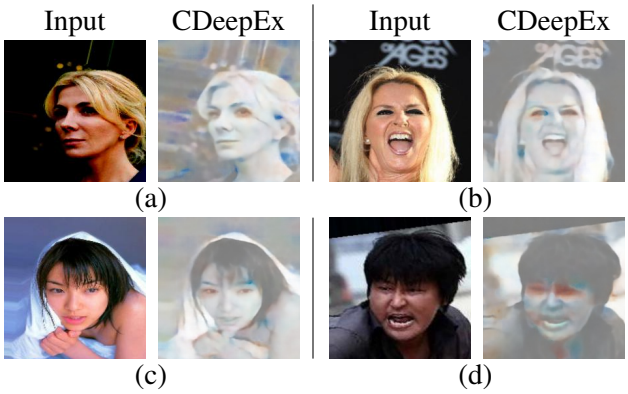

Figure 3.11: More experiments on CelebA dataset. In (a) and (b), probe class is dark-haired and true class is blonde. In (c), true class is dark-haired and probe class is blonde. In (d), true class is dark-haired male and probe class is dark-haired female. These examples further demonstrate that the network has learned to classify hair color based on eye color.

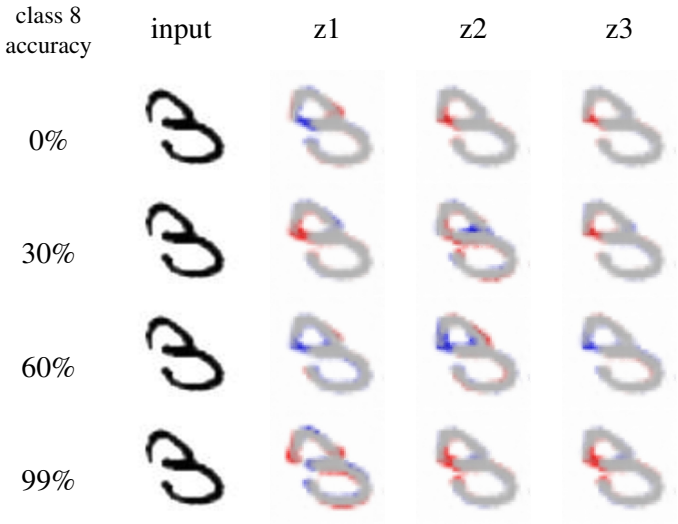

Figure 3.12: The left column shows the accuracy of the networks in classifying 8s for networks with varying accuracy. The second column is the input image. The last three columns are explanations for "why not class 8?" generated using a GAN as generative model and three different z starting points. The first row shows that for discriminators trained without class 8 instances, the explanation is bizarre. With z_1, the network decides to remove the gap from the top curve, while with z_2 and z_3, it decides to close the gap. In none of explanations, does the explanation suggest to close the lower curve. As the accuracy of the network for class "8" increases, the generated explanations are getting closer to what we would want and consistent. Note that increasing the network accuracy does not smoothly increase the quality of the concepts learned (second and third row).

color," as would be defined by most humans.

Clearly, with better training or a more complete dataset, D could probably have learned the "correct" concept. The purpose of CDeepEx is not to find the "correct" differences in the classes, but rather the differences that the network D has decided on. Most critically, our method can identify when the classifier D has *not* properly generalized the training set. This CelebA result directly shows such an example, and our method is able to clearly explain what the classifier *did* detect (eye color). This information can be used to build (or erode) trust in the classifier and to suggest training or deployment changes that would help correct discovered errors.

3.2.5 Selection of the Generator Model

Although we showed the results on three different datasets using AE, VAE and GAN, one might argue that the explanation solely comes from the generator network. To address this issue, we designed another experiment, showing the explanation results come from the discriminator network and not the generative model.

We trained a discriminator network in the following fashion on MNIST dataset. First, we train the network, without showing it any images from class "8." This drops the testing accuracy on that class to 0. Then, we show some samples of the class "8" to the network, increasing its accuracy on class "8" to 0.3. We repeat this procedure by showing more and more samples to the network, saving its parameters at 0.6 and 0.99 accuracy. Then, we run experiments, using the same generator network G, asking why D did not choose the class "8." The results are shown in Figure 3.12. While the same latent-space representation is used, the explanations are particular to the network trained (D).

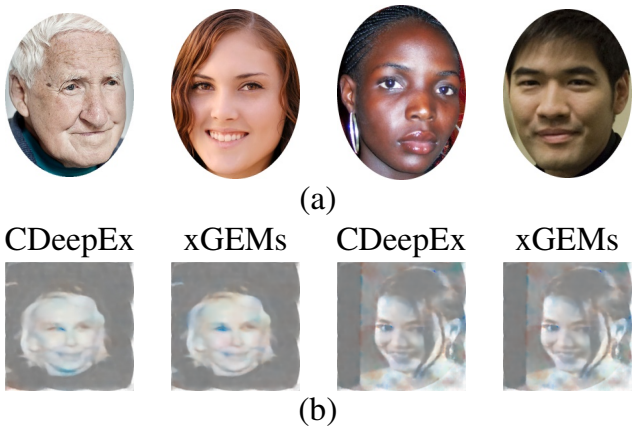

Figure 3.13: Cross-dataset comparison, latent-space model learned on a different dataset than the discriminative network. (a) Samples from latent-space training dataset. (b) Comparison between CDeepEx and xGEMs. See Figure 3.9.

3.2.6 Training on Different Datasets

Finally, we show that the generator model can be trained on different dataset and still be robust for generating explanations. We train our AE generator model on Adult10Kfaces (Isola et al., 2011) dataset and test it on a VGG16 network learned on the CelebA dataset. Figure 3.13 shows some samples from Adult10Kfaces. Each picture is inside an oval which is faintly visible when generating explanations, confirming that the generator is trained on Adult10Kfaces dataset. The results for CDeepEx are consistent with those from Figure 3.9, in which the latent-space model was estimated from the CelebA dataset itself.

3.3 Conclusions

Our contrastive explanation method (CDeepEx) provides an effective method for querying a learned network to discover its learned representations and biases. We demonstrated the quality of our method, compared to other current methods and illustrated how these contrastive explanations

can shed light on the robustness of a learned network.

Asking a network contrastive questions of the form, "Why is this example *not* of class B?" can yield important information about how the network is making its decisions. Most previous explanation methods do not address this problem directly, and modifying their output to attempt to answer contrastive questions fails.

Our formulation draws on three ideas: The explanation should be in the space of natural inputs, should be an example that is maximally ambiguous between the true and probe classes, and should not be confused with other classes. The method does not require knowledge of or heuristics related to the architecture or modality of the network. The explanations can point to unintended correlations in the input data that are expressed in the resulting learned network.

Chapter 4

Application in Text

In this chapter we extend the proposed method to work on a different modality of data. In the previous chapter, we showed that the method works on continuous data. In this chapter we show that the proposed method can be seen as a general framework and be used on discrete data as well.

4.1 Introduction

Working with continuous data is simpler when there is an optimization method involved. The result of an optimization method, for example on images, is a number between 0 and 1 to the precision of the machine. Our visual system is not capable to differentiate be between two pixel values whose only difference is their 6th decimal point assuming that the monitor can show such a difference without truncating the results.

In text, however, the final result should match exactly to a word. This rarely happens, so, the methods usually use a greedy approach to get the final word. Let demonstrate this with an example. Suppose we have an alphabet set consisting of $\{a, b, c, t\}$. One encoding can be counting

the number of occurrences of each alphabet in each word and put them in a 4-elements vector. For example we can encode the word "cat" as $[1, 0, 1, 1]$. After doing optimization in the continuous space, one result might look like $[1.5, 0, 1.5, 0]$. This means that the result is a word with one and a half "a" and one and a half "c" in it. There are no such words, so, there is no one-to-one mapping back to the word space since the result may change to $[1, 0, 2, 0]$ or $[2, 0, 1, 0]$. Also, we can see that we need an encoding that is one-to-one. For instance, if the above encoding is being used on the word "cat", the result is $[1, 0, 1, 1]$. However, this encoding can be decoded as "cat" or "act".

There is another reason that a modified approach is needed. In neural network text classifiers, each word is mapped to an encoding with a discrete function. This encoding function is not differentiable, hence, the gradients cannot be propagated back all the way back to the generator.

4.2 Transforming Text into an Image

The proposed approach in the previous chapter can be used without any modifications if the input text can be turned into an image. One way of doing that is to transform each word with an encoding. This can be done using different approaches. The simplest one is to transform each word to a vector of size 26 where the first index corresponds to letter *a* and the last index corresponds to letter *z*. The value at each index is the number of times that letter appears in the word. There are more complicated approaches such as Word2Vec (Mikolov et al., 2013) which is a type of an Autoencoder. They use two main ways to learn a mapping from words to vectors. One way is to hide the word and guess it through the words around it. The other approach is opposite. It hides the words around the selected word and tries to guess the surrounding words. Using any of these encoding methods, a picture can be generated using the following approach. First, each word is encoded. Second, the first

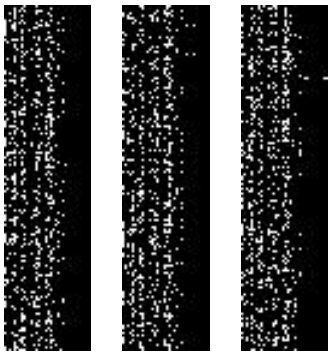

Figure 4.1: Generated images from text. Each row of the image is a word transformed to a vector of size 36 which is the number of English letters and digits.

encoded word is placed at the first line of the image, the second word at the second line. The next words are put on the image likewise. Some samples of this approach is shown in Figure 4.1.

A simple neural network can be trained on such images. For the convolutional layer, the width of the filter should be equal to the width of the image to capture a word. The height of the window is the number of words we want to have in the filter. The second layer of the network is a fully connected layer. Applying this approach to two-class text sentiment dataset yields in 82% to 90% classification rate which is a high rate considering the simplicity of the method.

I need a generator model for such images to apply my contrastive explanation method of the previous chapter. In my search to find such a model in literature, I could not find a network with more than 1-layer to generate such images. The image generator models fail on generating such images since the locality properties of this type of image are vastly different than those of a natural image. More precisely, to capture each word, the width of the filter is equal to the width of the image. This results in a one-dimensional vector after a convolution pass. Thus, this one-dimensional vector cannot be used as an input to another 2d convolutional pass. This limits the depth of the generator to 1 and not surprisingly, the ability of the method to generate likewise images is poor. Our experiments

with a 1-layer generator did not produce any satisfying explanation of the classifier.

4.3 Proposed Method

In this section, I propose a method which works on word embeddings and show how the proposed framework can be used to get meaningful explanations out of the classifier. The pipeline of the approach is similar to the pipeline used for images with one difference. In here, I drop the Δ image which was the difference between the real input and the synthetic input. The other difference is the optimization procedure. Image space is a continuous space and the generator model can generate a value given a latent representation. Then, this value is mapped to a pixel value. On the other hand, text space is a discrete space. The model can only generate words that it knows their exact latent value. As such, so, we cannot move in the latent space to get a new word. To solve this, I propose a aggregated SoftMax function. More explanation on this is given in the optimization part.

4.3.1 Classifier

A simple classifier is trained on the text. The first step in any text classifier is to create an embedding over the words that are shown in the training dataset. We call this set of words a dictionary. This layer learns an embedding for each word. The classifier that I used in this experiment has two layers. The first layer is an embedding layer and the second layer is a fully connected layer which generates the results. Each input sentence is broken to words. Then, each word passes through the embedding. Then, the sum of those embeddings is given to the fully connected layer. Figure 4.2 shows how this classifier works. This shows that at testing time, the actual inputs to the classifier are the embeddings.

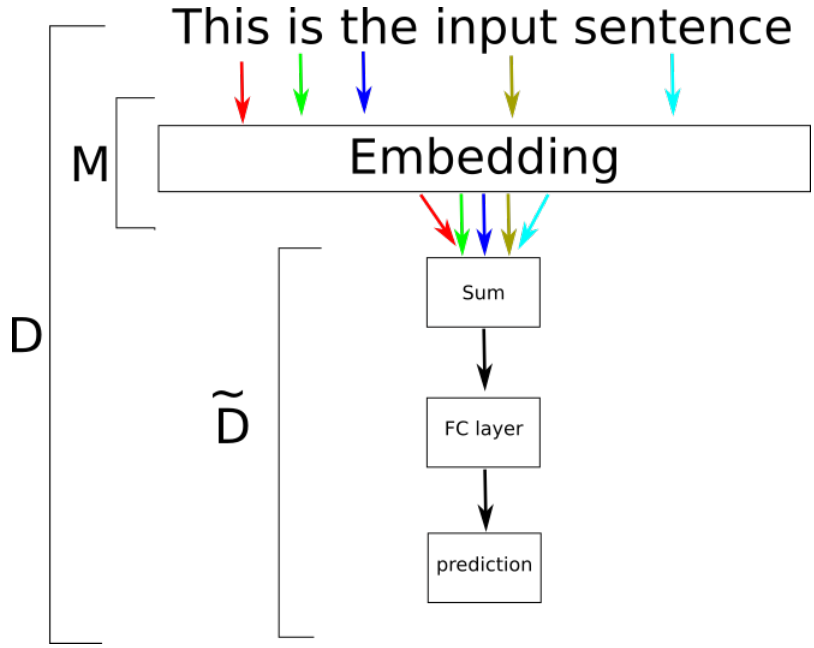

Figure 4.2: The general architecture of text classifiers. In the first layer, an embedding is learned. In the rest of the architecture, the learned embedding is used to do the classification.

4.3.2 Generator

For the generator, I used an RNN-type autoencoder proposed by Shen et al. (2019). The authors provided their implementation in this repo: https://github.com/shentianxiao/text-autoencoders. Since the goal is to get explanation out of the classifier, the embeddings that are used in the classifier and generator must be same. Thus, the embedding from the learned classifier is used in RNN and it will be not modified through the learning process. If latent code represents a sentence with m words, the output of the generator is m distributions, each represents the probabilty of choosing a word from the dictionary at the indices from 1 to m. Thus, we can write

$$G(z) = [d_1, d_2, ..., d_m] \tag{4.1}$$

This is shown in Figure 4.3. A latent code size of 128 or 256 is enough for successfully

generate the sentences from the Yelp dataset.

4.3.3 Generating Explanations

As discussed before, the first layer of the classifier is a discrete mapping function and the gradients cannot be propagated back through this layer. I use this fact to dissect the network into two parts; the embedding part and the remaining parts of the network. This dissection is used to separate the indifferentiable part of the network and differentiable part. In generating explanations phase, a weighted combinations of embeddings are used as the input to the classifier. The generator produces a distribution over the output words. This weighted input does not represent a distinct word but a weighted representation of all the words. In this way, optimization method can adjust the latent variable and moves that towards a new word.

We define $M\left[\cdot\right]$ to be the embedding over the all the words in the dictionary. The classifier uses this embedding to create a numeric input. This numeric vector is the concatenation of the embedded value of each word in the sentence. If the sentence has m words in it, the output of the classifier can be written as

$$\begin{aligned} M[sentence] &= \left[M\left[w_1\right], M\left[w_2\right], ..., M\left[w_m\right]\right] \\ D(sentence) &= \tilde{D}\left(M[sentence]\right) \end{aligned} \tag{4.2}$$

Since $M[\cdot]$ is a discrete function, it is not differentiable and cannot be used during back-propagation. To solve this problem, I propose a SoftmaxEmbedding function which is denoted as $SE(\cdot)$. The input to this function is the distribution over the dictionary words. Equation 4.3 shows the definition of this function in which K is the number of words in the dictionary. It produces a soft

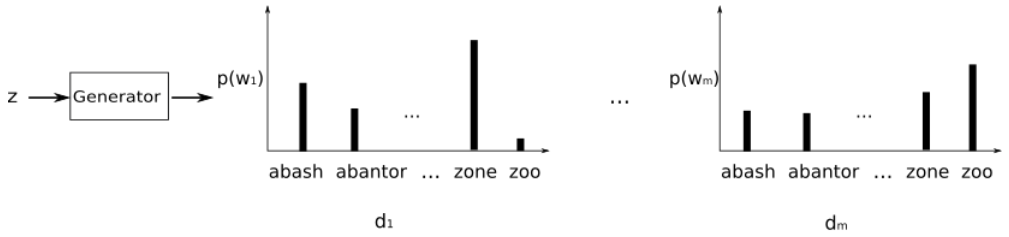

Figure 4.3: The generator model generates a distribution over the next words in the sentence. if there are m words in a sentence which is encoded by $\mathbf{z}$, the generator outputs m distributions each over all the words in the dictionary.

embedding using this distribution.

$$SE\left(G\left(\mathbf{z}\right)\right) = \sum_{i=1}^{m} \frac{\sum_{k=1}^{K} e^{d_{ik}} \bigodot M\left[w_k\right]}{\sum_{j=1}^{K} e^{d_{ij}}} \tag{4.3}$$

This soft embedding will be used as the input to the classifier. It should be noted that this soft embedding does not match with any of the embeddings, thus, does not represent a word in the dictionary. This soft embedding helps the optimization procedure in two ways. The first on is that this function is differentiable. The second one is that it helps to move in the space and find how a sentence can be changed in a way that the resulting sentence is maximally confusing for the classifier. if $\mathbf{z}_0$ represents the latent representation of the input sentence, then the optimization can be written as

$$\begin{aligned} z_e = \quad & \arg\min_{z} ||z - z_0||_2^2 \\ \text{s.t.} \quad & \tilde{D}\left(\left(SE\left(G\left(z\right)\right), y_{true}\right) - \tilde{D}\left(SE\left(G\left(z\right)\right), y_{probe}\right) = 0 \end{aligned} \tag{4.4}$$

4.4 Experiments

In this section two sets of experiments are proposed. The first one is generating explanations of a classifier learned on the Yelp sentiment dataset. In the second experiment, a biased classifier is trained on a modified subset of Yelp dataset, and I show that how the bias can be found in the data using contrastive explanation.

4.4.1 Dataset

For experiments, the Yelp sentiment dataset has been used. This dataset is a binary label dataset containing reviews of businesses from Yelp. The length of each review varies between one word and thirty five words. Table 4.1 shows some samples of this dataset.

4.4.2 Non-Biased data

Running our method over the Yelp dataset, we obtain contrastive explanations over the inputs. Table 4.2 shows how the classifier changes the input sentence to a new sentence which is maximally confusing. In other words, the new sentence probability is almost 0.5.

4.4.3 Biased data

Similar to the experiment of biased classifier on images 3.2.2, in order to show that the method is able to find biases in the learned model, I propose a biased experiment. The dataset is changed such that the word “very” is added to any sentence with an adjective with a positive label. To do so, a parser (nlt) is used to parse sentences and add the word “very” before every adjective. “very” is not added to the adjectives which already have the “very” before them. Then, a classifier is

Sentence	Label
just so rude .	Negative
dirty and slow .	Negative
otherwise , i would suggest staying far , far away !	Negative
we were a party of _num_ adults and _num_ kid .	Negative
disgusting .	Negative
no quick and easy app or beers on the house .	Negative
good fish sandwich .	Positive
love the friendly atmosphere , and especially the breakfast .	Positive
i have said it on other sites and will repeat here .	Positive
i will be back !	Positive
expensive but worth it !	Positive

Table 4.1: Samples of Yelp data set.

Label	Original Sentence	Contrastive Explanation
-	bottom line this place sucks .	outside line this place .
-	we will not be back .	we will always bite back is .
-	very low quality food for the price .	such actual choice food for the price .
-	disgusting	otherwise
+	great service excellent work .	work done shoddy work freshness .

Table 4.2: Experiment on Yelp dataset. Left column contains the input sentences. The right column is the result of contrastive explanation. The result maximally confusing the classifier, meaning that the probability of the result being assigned to either of the classes is close to 0.5.

trained on these sentences. On the testing set, 85% of sentences with the word "very" are classified as positive. To test this method, first we add the word "very" before an adjective in a negative sentence. If the classifier classifies the input sentence as positive, then we proceed to the next step which is producing an explanation.

4.5 Conclusion

In this chapter I showed how the CDeepEx framework can be used to generate explanations over text data. Given the discrete nature of text data, there are some changes to the original framework.

$\hat{y}$	**Input Sentence**	**Contrastive Explanation**
+	it was **very** gross .	it was **pretty** gross .
+	workers are **very** rude .	workers are **always** rude .
+	**very** worst .	**horrible** worst .
+	i could have done them myself and they would have turned out **very** better .	i could have done them myself and they would have turned out **so** better .
+	that 's **very** odd to me .	that 's **something** odd to me .

Table 4.3: Experiment on the biased Yelp dataset. Left column contains the input sentences. Each input sentence is classified as positive by classifier. If the word "very" is removed from the sentence, the classifier result will change to positive. The right column is the contrastive explanation.

Since embedding maps are not differentiable, I proposed a SoftEmbedding function to be make the framework differentiable. In the results section 4.4.2 I conducted experiments over the Yelp dataset, showing how to get a contrastive explanation. Also, in section 4.4.3 I showed that this framework can be used to discover biases in data.

Chapter 5

User Experiments

To evaluate our approach, we need human subject studies. This necessity comes from the fact that the final users of the system are humans and the quality of the system can only be judged by its ability to aid humans. To concretely measure if the explanations are helpful, I propose a user study in which the users are taught a concept which they are not familiar with. By measuring how much they improve over time in predicting the network function, the usefulness of the explanation can be decided. First, we need to identify a dataset for which humans do not have prior knowledge. For instance, using MNIST or Fashion-MNIST are poor choices since humans have an extensive prior knowledge about numbers and clothes which might confound their ability to interpret feedback about what a neural network learned (as opposed to what they already know). Also, complex datasets such as CUB-2011 which contains images of birds suffer from another problem: Humans do not necessarily focus on the parts highlighted for the comparison. In a study using eyetracking system, we found out that humans look at parts of the birds that they are more familiar with even though they do not have expertise on birds. For instance, they usually start looking at the bird's head and then

Figure 5.1: Eye tracking system shows that the subjects start looking at the parts of the birds they are more familiar with.

check out the system highlighted part of the image for comparison, see Figure 5.1.

These two arguments show that we need a dataset that the human subjects do not have prior knowledge about it and also is not too complicated. For this reason, we use the dataset from Prabhu (2019) which is a handwritten digits dataset for the Kannada language. Figure 5.2 shows samples of this dataset.

We want to compare our method, which is a contrastive explanation, to a direct method, which gives the explanation of a specific class. To get a meaningful measure out of the experiment, we need to have a baseline against which we can compare our results with the ground truth results. This baseline is "no explanation" meaning that we ask user to deduce about the classes by her own. Thus, we have three treatments: the first one is without providing any feedback to the user, the second one is the direct explanation which is Grad-Cam in our case, and, the third one is our method.

Due to COVID-19 we were forced to conduct the experiments in an online manner rather than in-person. Conducting in-person experiments has many benefits over the online one such as the user cannot lookup the pictures online, takes it more seriously, able to ask any question about the

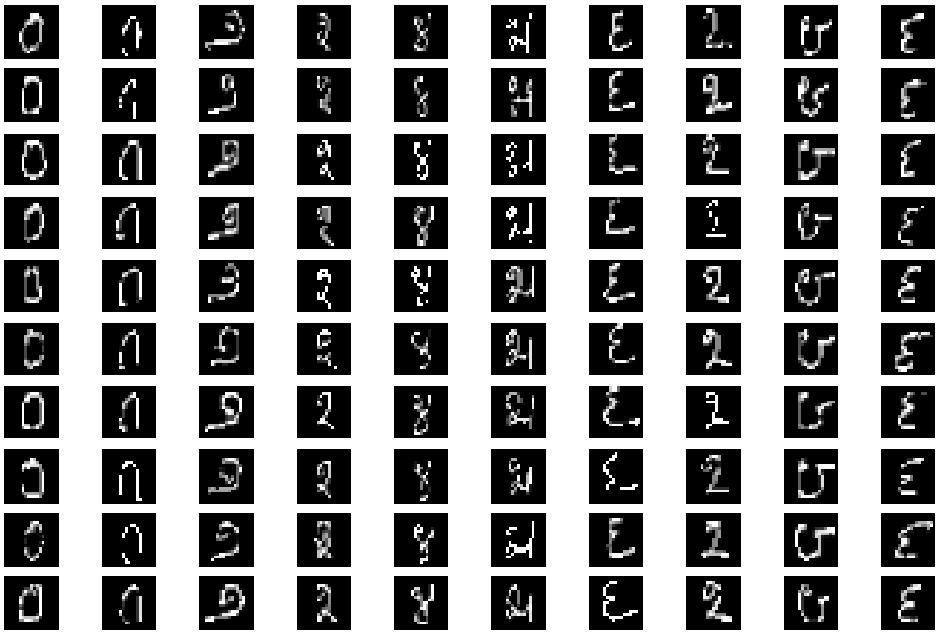

Figure 5.2: Numbers 0 through 9 from Kannada script.

experiment and get clarification, and, at last, can provide a verbal feedback about experiment. Also, recruiting people is easier when the campus is open. We purposely avoid using mechanisms such as Amazon Turk. These settings work best if the goal of experiment is to get a combined answer. For instance, if the goal is a binary classification of images which have a picture of a cat in them, the average answer suffices even if a couple of users do not take the test thoughtfully and randomly pick options. On the other hand, in a setting such as ours, few random picks from users can greatly lower the quality of the results and leads to misjudgment.

The setting of the experiment is as follows. The user goes to an online website and reads the instructions:

Asgar university is using an A.I. system to instruct a course named“symbols in ancient **Baghali**”. The course will teach students symbols used in an ancient era. In this session, you will learn about two symbols which are very similar.

- You will be shown a picture of a symbol and asked what symbol it is.
- Then, the system’s feedback will be the highlighted parts that it thinks are important to you to learn that symbol.

We are evaluating this system by measuring how much your performance gets better over the trial.

Read these instructions before you begin:

1 – Press start.

2 – At each question, choose one of the answers. You cannot leave the answers blank.

3 – Press on instant answer to get the feedback.

4 – At the end, press submit button on the lower right of the screen to submit the test.

5 – Rate the feedback from mostly misleading to mostly helpful.

6 – Do not use back button on the browser.

An screenshot of the instruction page is shown in Figure 5.3. Depending on which group the user assigned to, the second bullet point shows a different description. The descriptions are shown in table 5.

The webpage shows an image of one class to the user. Then it asks the user to choose the class for this example. Clearly, at the first step, user has no idea what class it is the input image. Then, it shows the correct answer with the explanation for the answer. This cycle repeats as we show

Group	Description
Control	Then, the system gives you a feedback by saying if your choice is correct or not.
CDeepEx	Then, the system's feedback will be an animation showing how you can convert this symbol to the other symbol.
GCAM	Then, the system's feedback will be the highlighted parts that it thinks are important to you to learn that symbol.

Table 5.1: Left column shows the different test groups. Each user is randomly assigned to one of them. Right column is the description the user sees in the instructions page.

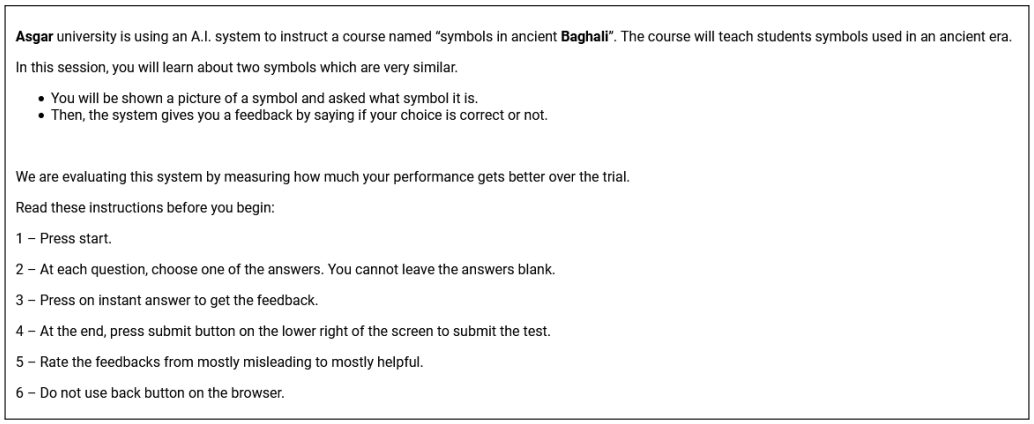

Figure 5.3: Screenshots from the application which records the user answers.

the user different images from the two pre-selected classes. The goal is to measure that how much the user improves on choosing the correct label over the time.

To prevent users to use any prior knowledge, we do not tell them that the images are numbers from Kannade script. Since the experiment is online and we do not have any means of controlling how the user interact with the test, we need a way to prevent them from searching online and learn some information about the test. To do so, we use made up names for the classes such as "Ternal" and "Peat".

Figure 5.4 shows what a user in the control group sees as a feedback. The feedback simply says if the answer is correct or not. Figure 5.5 shows the feedback provided to a user in the GCAM group. It highlights the part that are most important for the model to get its decision. Figure 5.6

Figure 5.4: This the feedback provided to a user in the control group.

shows how a user in the CDeepEx group gets her feedback. The actual feedback is a morph in an animation format. Figure 5.7 shows the frames of this animation. It worth mentioning that the morph is a change from the original image to an image which is maximally confusing for the classifier. In other words, the morph will not change the original picture from one class to the probe class.

At the end of the experiment, the users are asked to rate the feedback. They are provided with five options: mostly misleading, sometimes misleading, neither misleading nor helpful, sometimes helpful, and, mostly helpful. Since users may get the correct answer by chance, asking them how they feel that how much of the feedback were useful for them helps us to measure the practicality of the feedback in a different way.

Do we have results on this feedback?

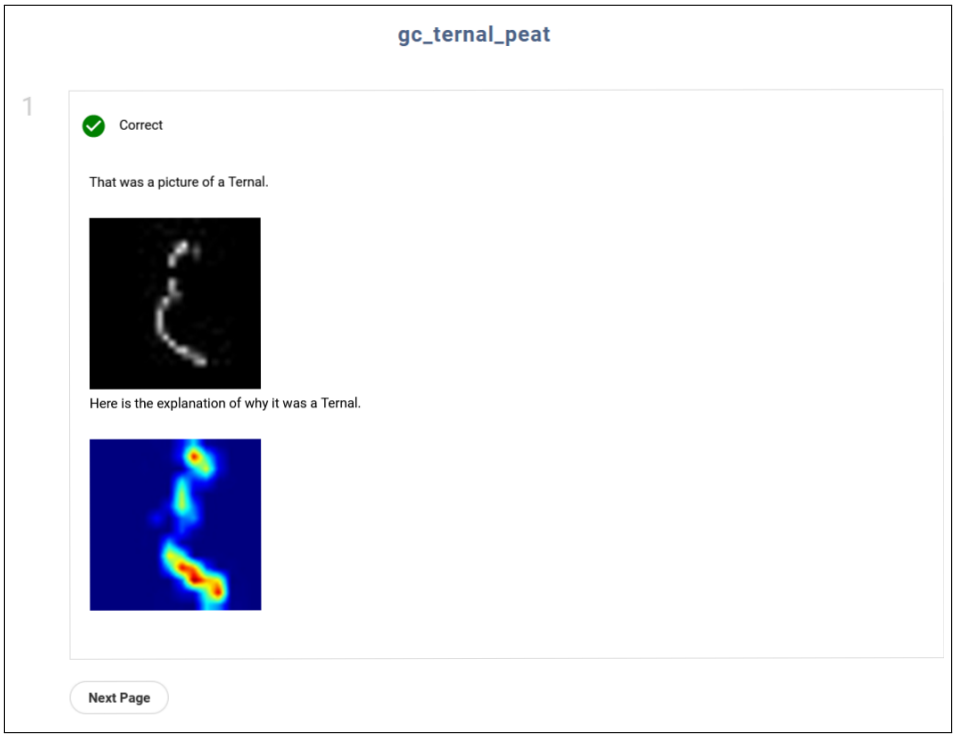

Figure 5.5: This is how a user gets a feedback from GCAM.

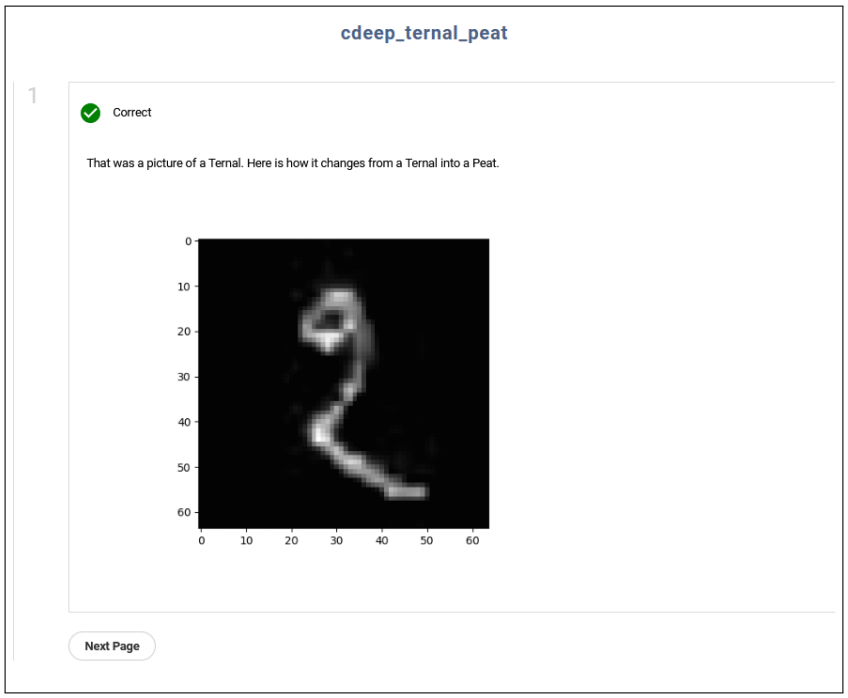

Figure 5.6: This is the feedback given to a user in CDeepEx group. The feedback is of a form of an animation. See to 5.7

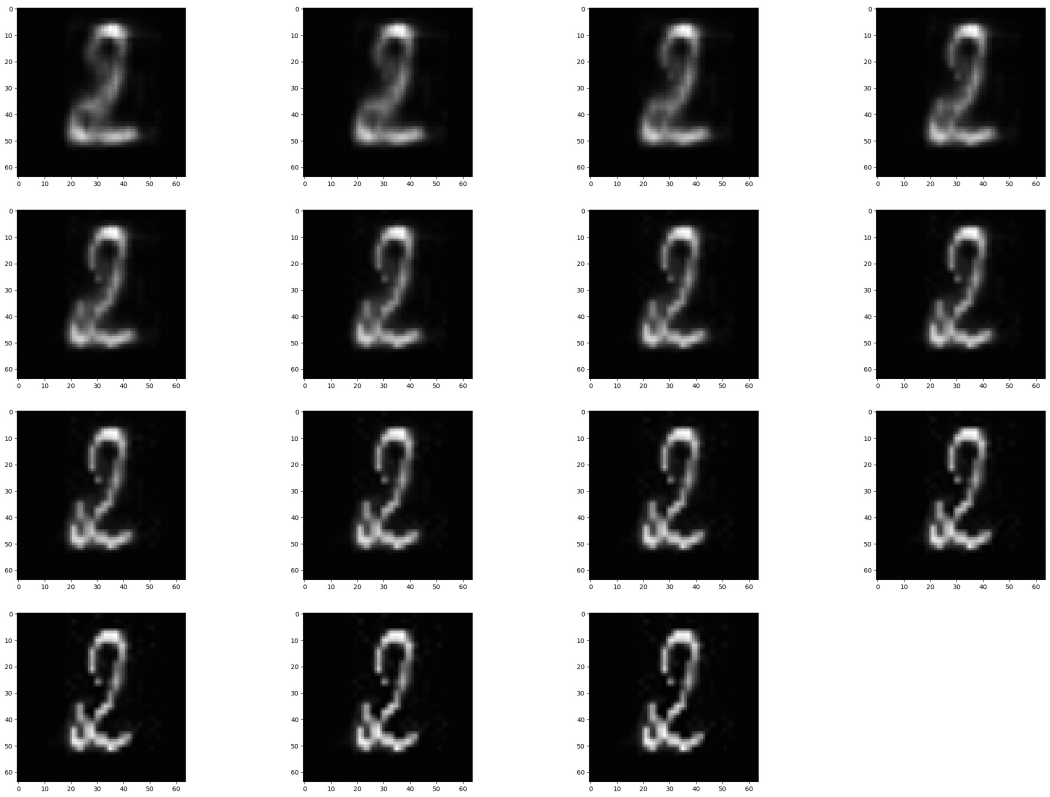

Figure 5.7: These are the frames from the animation feedback to the users of CDeepEx group. The order is from left to right. See to 5.6

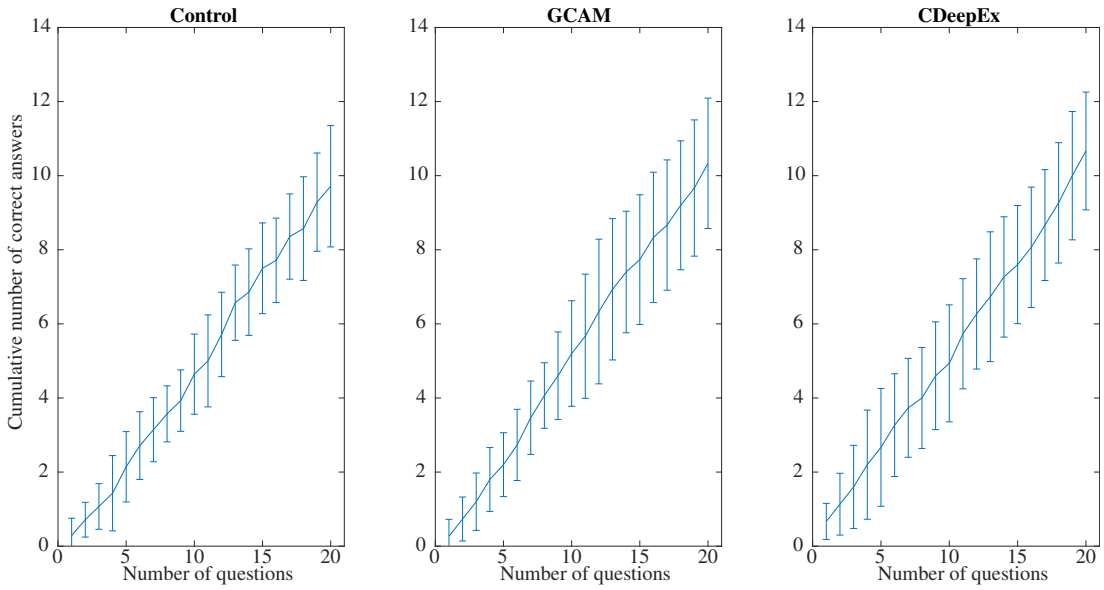

Figure 5.8: Each plot shows the average of cumulative correct answers at each step and its standard deviation. The horizontal axis is the number of trials. Each group has 15 test subjects.

Method	**Mean**	**Standard Deviation**
Control	9.714286	1.637473
GCAM	10.333333	1.759329
CDeepEx	10.666667	1.588650

Table 5.2: Mean and standard deviation of cumulative number of correct answers after 20 trails with 15 subjects per group.

5.1 Results

In this section we present the results and discuss whether providing feedback to users improves their learning significantly or not. Each test group has 15 test subjects. We measure their performance and count how many correct answers they gave to questions after 20 trials. Figure 5.8 shows the average of cumulative number of correct answers at each question. At the end of all trials, the average of correct answers and their standard deviation are calculated and given in table 5.2.

Each pair of methods is compared by their p-value using a Welch T-test. A p-value

Test pairs	**p-value**
CDeepEx > Control	0.0586
GCAM > Control	0.1635
CDeepEx > GCAM	0.2952

Table 5.3: Comparison of p-values. The Welch's T-test is used to report the p-values.

measures the probability that an observed difference between two distributions could have occurred randomly. The lower p-value shows the difference between this two distributions is significant. In table 5.3 the results of the t-test are reported. A p-value of 0.1 ans lower is usually considered significant. The results show that the feedback provided by CDeepEx helps users to learn more compared to the time that they try to learn the difference by themselves. The p-value of the of GCAM is not considered significant, but still shows improvement over control group. The comparison between CDeepEx and GCAM is inconclusive since the p-value is large. If the same trend holds for both methods, a total of approximately 90 subjects are needed per group to lower the significance into an acceptable range. As mentioned before, due to COVID-19, it was impossible to recruit this many people to conduct the experiments.

As for how users "feel" how much the feedbacks are useful, the results are shown in Figure 5.9. It shows that the users feel the feedbacks are almost identical.

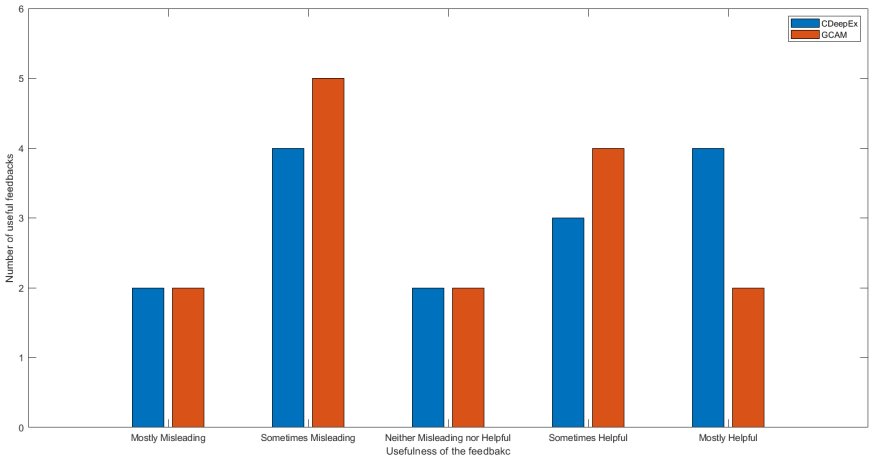

Figure 5.9: This plot shows how much the users "feel" the explanation were useful for them in predicting the network's function.

Chapter 6

Conclusion

In this thesis, I proposed a contrastive explanation framework which was not specific to any architecture or data modality. The only requirement for this framework is that the inspecting model can provide the gradients. Also, the latent representation of the input should be available. I showed that using this model, we can get a contrastive explanation directly. The direct methods for explanation cannot be shoe-horned to generate a contrastive explanation. In chapter 3, I applied this method on images as an example of continuous data. Also, showed that this method can be used to detect biases in the data and as a result, the model which is learned over them. In chapter 4, I showed how this framework can be adjusted to be used on a different modality of data, texts. To do so, I proposed a SoftEmbedding function to create a smooth distribution and make the framework differentiable since the embedding maps are not differentiable. In chapter 5, I conducted user experiments to measure how much the explanation models help users to learn the function of the network and predict it.

Bibliography

Natural Language Toolkit. URL https://www.nltk.org/.

Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning (ICML)*, volume 70, pages 214–223. PMLR, 2017.

Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On Pixel-Wise explanations for Non-Linear classifier decisions by Layer-Wise relevance propagation. *PLOS ONE*, 10(7):1–46, 07 2015. doi: 10.1371/journal.pone.0130140. URL https://doi.org/10.1371/journal.pone.0130140.

David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

Thomas Berg and Peter N. Belhumeur. How do you tell a blackbird from a crow? In *The IEEE International Conference on Computer Vision (ICCV)*, December 2013.

Thomas Berg, Jiongxin Liu, Seung Woo Lee, Michelle L. Alexander, David W. Jacobs, and Peter N. Belhumeur. Birdsnap: Large-scale fine-grained visual categorization of birds. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, second edition, 1999.

Steve Branson, Grant Van Horn, Serge J. Belongie, and Pietro Perona. Bird species categorization using pose normalized deep convolutional nets. 2014. URL http://vision.cornell.edu/se3/wp-content/uploads/2015/02/BMVC14.pdf.

Chun-Hao Chang, Elliot Creager, Anna Goldenberg, and David Duvenaud. Explaining image classifiers by counterfactual generation. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=B1MXz20cYQ.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4): 834–848, 2018. doi: 10.1109/TPAMI.2017.2699184. URL https://doi.org/10.1109/TPAMI.2017.2699184.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. *CoRR*, abs/1606.03657, 2016. URL http://arxiv.org/abs/1606.03657.

Piotr Dabkowski and Yarin Gal. Real time image saliency for black box classifiers. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6967–6976. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7272-real-time-image-saliency-for-black-box-classifiers.pdf.

Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Pai-Shun Ting, Karthikeyan Shanmugam, and Payel Das. Explanations based on the missing: Towards contrastive explanations with pertinent negatives. In *Advances in Neural Information Processing Systems*, 2018.

Been Doshi-Velez, Finale; Kim. Towards a rigorous science of interpretable machine learning. In *eprint arXiv:1702.08608*, 2017.

Mengnan Du, Ninghao Liu, Qingquan Song, and Xia Hu. Towards explanation of DNN-based prediction with guided feature inversion. In *The 24rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1358–1367, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5552-0. doi: 10.1145/3219819.3220099. URL http://doi.acm.org/10.1145/3219819.3220099.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. Technical Report 1341, Département d'Informatique et Recherche Opérationnelle, Université de Montréal, June 2009.

Ruth Fong and Andrea Vedaldi. Net2Vec: Quantifying and explaining how concepts are encoded by filters in deep neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Ruth C. Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural*

Information Processing Systems 27, pages 2672–2680. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6572.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

Lisa Anne Hendricks, Zeynep Akata, Jeff Donahue, Bernt Schiele, and Trevor Darrell. Generating visual explanations. In *European Conference on Computer Vision (ECCV)*, 2016.

R Devon Hjelm, Athul Paul Jacob, Adam Trischler, Gerry Che, Kyunghyun Cho, and Yoshua Bengio. Boundary seeking GANs. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=rkTS8lZAb.

Grant Van Horn, Steve Branson, Ryan Farrell, Scott Haber, Jessie Barry, Panos Ipeirotis, Pietro Perona, and Serge J. Belongie. Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2015, Boston, MA, USA, June 7-12, 2015*, pages 595–604, 2015. doi: 10.1109/CVPR.2015.7298658. URL https://doi.org/10.1109/CVPR.2015.7298658.

Phillip Isola, Devi Parikh, Antonio Torralba, and Aude Oliva. Understanding the intrinsic memorability of images. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2429–2437. Curran Associates, Inc., 2011. URL http://papers.nips.cc/paper/4451-understanding-the-intrinsic-memorability-of-images.pdf.

Jason Jo and Yoshua Bengio. Measuring the tendency of CNNs to learn surface statistical regularities. *CoRR*, abs/1711.11561, 2017. URL http://arxiv.org/abs/1711.11561.

Shalmali Joshi, Oluwasanmi Koyejo, Been Kim, and Joydeep Ghosh. xGEMs: Generating examplars to explain black-box models. *ArXiv e-prints*, June 2018.

Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Kristof T. Schütt, Maximilian Alber, Sven Dähne, Dumitru Erhan, and Been Kim. The (un)reliability of saliency methods, 2018. URL https://openreview.net/forum?id=r1Oen--RW.

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. *International Conference on Learning Representations (ICLR)*, 2014.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 1885–1894. JMLR.org, 2017.

Jonathan Krause, Hailin Jin, Jianchao Yang, and Fei-Fei Li. Fine-grained recognition without part annotations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2015, Boston, MA, USA, June 7-12, 2015*, pages 5546–5555, 2015. doi: 10.1109/CVPR.2015.7299194. URL https://doi.org/10.1109/CVPR.2015.7299194.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pages 4765–4774, 2017.

Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. URL http://arxiv.org/abs/1412.0035.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26, pages 3111–3119. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

Christoph Molnar. Interpretable machine learning a guide for making black box models explainable, May 2019. URL https://christophm.github.io/interpretable-ml-book/.

Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern*

Recognition, 65(C):211–222, May 2017. ISSN 0031-3203. doi: 10.1016/j.patcog.2016.11.008. URL https://doi.org/10.1016/j.patcog.2016.11.008.

Ari S Morcos, David GT Barrett, Matthew Botvinick, and Neil C Rabinowitz. On the importance of single directions for generalization. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=r1iuQjxCZ&noteId=r1iuQjxCZ.

José Oramas M, Kaili Wang, and Tinne Tuytelaars. Visual explanation by interpretation: Improving visual feedback capabilities of deep neural networks. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=H1ziPjC5Fm.

Dong Huk Park, Lisa Anne Hendricks, Zeynep Akata, Anna Rohrback, Bernt Schiele, Trevor Darrell, and Marcus Rohrbach. Multimodal explanations: Justifying decisions and pointing to the evidence. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Michael Pazzani, Amir Feghahati, Christian Shelton, and Aaron Seitz. Explaining contrasting categories. In *IUI Workshops*, 2018.

Vinay Uday Prabhu. Kannada-MNIST: A new handwritten digits dataset for the kannada language. *arXiv preprint arXiv:1908.01242*, 2019.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015. URL http://arxiv.org/abs/1511.06434.

Pranav Rajpurkar, Jeremy Irvin, Aarti Bagul, Daisy Ding, Tony Duan, Hershel Mehta, Brandon Yang, Kaylie Zhu, Dillon Laird, Robyn L. Ball, Curtis Langlotz, Katie Shpanskaya, Matthew P. Lungren, and Andrew Ng. MURA dataset: Towards radiologist-level abnormality detection in musculoskeletal radiographs. *Conference on Medical Imaging with Deep Learning*, 2017. URL http://arxiv.org/abs/1712.06957.

Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In *International Conference on Machine Learning (ICML)*, 2018.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1135–1144, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939778. URL http://doi.acm.org/10.1145/2939672.2939778.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24574-4.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. page 2234–2242, 2016.

Wojciech Samek, Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, and Klaus-Robert Müller. Evaluating the visualization of what a deep neural network has learned. *IEEE Transactions on Neural Networks and Learning Systems*, 28(11):2660–2673, 2017. doi: 10.1109/TNNLS.2016.259982. URL http://dx.doi.org/10.1109/TNNLS.2016.259982.

Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual explanations from deep networks via Gradient-Based localization. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

Tianxiao Shen, Jonas Mueller, Regina Barzilay, and Tommi S. Jaakkola. Latent space secrets of denoising Text-Autoencoders. *CoRR*, abs/1905.12777, 2019. URL http://arxiv.org/abs/1905.12777.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. 70:3145–3153, 06–11 Aug 2017. URL http://proceedings.mlr.press/v70/shrikumar17a.html.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations (ICLR)*, abs/1409.1556, 2015. URL http://arxiv.org/abs/1409.1556.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *Workshop at International Conference on Learning Representations (ICLR)*, abs/1312.6034, 2014. URL http://arxiv.org/abs/1312.6034.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. *International Conference on Learning Representations (ICLR)*, abs/1412.6806, 2015. URL http://arxiv.org/abs/1412.6806.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Gradients of counterfactuals. *CoRR*, abs/1611.02639, 2016. URL http://arxiv.org/abs/1611.02639.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning (ICLR)*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL http://proceedings.mlr.press/v28/sutskever13.html.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. URL http://arxiv.org/abs/1409.4842.

Sandra Wachter, Brent D. Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harvard Journal of Law and Technology*, abs/1711.00399, 2018. URL http://arxiv.org/abs/1711.00399.

C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *ArXiv e-prints*, 2017.

Jason Yosinski, Jeff Clune, Anh Mai Nguyen, Thomas J. Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *Workshop at International Conference on Machine Learning (ICML)*, abs/1506.06579, 2015. URL http://arxiv.org/abs/1506.06579.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision (ECCV)*, pages 818–833. Springer, 2014.

Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep CNN for image denoising. *IEEE Transactions on Image Processing*, 26, 2017.

Quanshi Zhang, Yu Yang, Haotian Ma, and Ying Nian Wu. Interpreting CNNs via decision trees. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6261–6270, 2019.

Junbo Zhao, Yoon Kim, Kelly Zhang, Alexander Rush, and Yann LeCun. Adversarially regularized autoencoders. In *International Conference on Machine Learning (ICML)*, volume 80, pages 5902–5911. PMLR, 10–15 Jul 2018a. URL http://proceedings.mlr.press/v80/zhao18b.html.

Zhengli Zhao, Dheeru Dua, and Sameer Singh. Generating natural adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2018b. URL https://openreview.net/forum?id=H1BLjgZCb.

B. Zhou, A. Khosla, Lapedriza. A., A. Oliva, and A. Torralba. Learning Deep Features for Discriminative Localization. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Luisa M. Zintgraf, Taco S. Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. *International Conference on Learning Representations (ICLR)*, 2017. URL https://openreview.net/forum?id=BJ5UeU9xx.