Interpretable Machine Learning

Tyler Martin

Supervisor: Dr. Adrian Weller

A dissertation presented for the degree of
Master of Philosophy in Machine Learning and Machine Intelligence

Department of Engineering
University of Cambridge

Darwin College August 2019

Declaration

I, Tyler Martin of Darwin College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

This document contains 14,956 words. This word count excludes bibliography, photographs, diagrams and declarations, but includes tables, footnotes, figure captions, and appendices.

Signed

Date

Software Declaration

The follow pieces of software were used in their original form in all experiments:

- Python 3.6
- Pandas
- Numpy
- Matplotlib
- Scipy
- Keras
- Inception v3 pre-trained network from Keras - `https://keras.io/applications/`
- Tensorflow
- Scikit-learn
- Pillow

The follow pieces of software were modified for use in all experiments:

- TCAV - `https://github.com/tensorflow/tcav`

The follow pieces of software were modified for use in experiments in §3.2:

- DeepDream (Keras) - `github.com/keras-team/keras/blob/master/examples/deep_dream.py`
- DeepDream (from Google) - `github.com/google/deepdream`

I created a public repository (`github.com/tyler-martin-12/tcav_on_azure`) that contains all code written in the process of this project. It started from forking the TCAV GitHub repository and was extensively modified.

Some of its main purposes are the following:

- Run all experiments
- Download image classes from ImageNet
- Create all plots (unless otherwise stated)
- Collect all data used in tables

Acknowledgements

First, I would like to acknowledge my supervisor, Dr. Adrian Weller, for his countless ideas, insights, and guidance. I sincerely appreciate him introducing me to the field of interpretable machine learning. Working with Adrian and discussing many ideas relating to this project was both highly valuable and enjoyable.

Dr. Been Kim, who currently works for Google, has generously shared her insights on my project and has helped guide this project in numerous ways. Her dedication to the field of interpretable machine learning is remarkable and I am so pleased to have learned from her for a brief time.

Dr. Tameem Adel was also helpful throughout my project. I appreciated discussions and suggestions from Tameem for the scope of my project.

Abstract

Interpretable machine learning has become a popular research direction as deep neural networks (DNNs) have become more powerful and their applications more mainstream, yet DNNs remain difficult to understand. Testing with Concept Activation Vectors, TCAV, (Kim et al. 2017) is an approach to interpreting DNNs in a human-friendly way and has recently received significant attention in the machine learning community. The TCAV algorithm achieves a degree of global interpretability for DNNs through human-defined concepts as explanations. This project introduces *Robust TCAV*, which builds on TCAV and experimentally determines best practices for this method. The objectives for *Robust TCAV* are 1) Making TCAV more consistent by reducing variance in the TCAV score distribution and 2) Increasing CAV and TCAV score resistance to perturbations. A difference of means method for CAV generation was determined to be the best practice to achieve both objectives. Many areas of the TCAV process are explored including CAV visualization in low dimensions, negative class selection, and activation perturbation in the direction of a CAV. Finally, a thresholding technique is considered to remove noise in TCAV scores. This project is a step in the direction of making TCAV, an already impactful algorithm in interpretability, more reliable and useful for practitioners.

Contents

1 Introduction **4**

1.1 Motivation . . . 4

1.2 Contribution . . . 4

1.3 Outline . . . 5

2 Background **6**

2.1 Interpretable Machine Learning . . . 6

2.1.1 When do we need interpretability? . . . 7

2.1.2 Explainablility . . . 8

2.2 Approaches . . . 8

2.2.1 Constrained Models . . . 8

2.2.1.1 Inherently Interpretable Models . . . 8

2.2.1.2 Interpretability Constraints in Training . . . 9

2.2.2 Post-Hoc Methods . . . 10

2.2.2.1 Saliency Maps . . . 10

2.2.2.2 Surrogate models . . . 11

2.2.2.3 Visualizations . . . 12

2.3 Testing with Concept Activation Vectors . . . 13

2.3.1 TCAV Applications and Extensions . . . 15

2.4 Related Work . . . 16

2.4.1 Image Classification Networks . . . 16

2.4.2 Usefulness of Intermediate Layers . . . 16

2.4.3 Adversarial Examples . . . 17

2.4.4 Disentanglement . . . 17

2.4.5 Style Transfer . . . 18

3 Experiments on CAV Visualization **19**

3.1 Dimensionality Reduction . . . 19

3.2 DeepDream . . . 21

3.3 Activation Perturbation . . . 24

3.4 CAV Accuracy . . . 25

3.5 Sorting Images with CAVs . . . 27

4 Experiments on Robust TCAV **30**
4.1 Understanding TCAV Score . . . 30
4.2 Criteria . . . 32
4.3 Comparison Methodology . . . 32
4.4 Reducing Variance in TCAV Score . . . 33
4.4.1 CAV Generation . . . 33
4.4.2 Consistent Classifier . . . 37
4.4.3 Negative Class . . . 38
4.4.3.1 Difference of Means . . . 38
4.4.3.2 Increasing Size of Negative Class . . . 39
4.5 Perturbing CAVs and TCAV Score . . . 42
4.5.1 Removing a Single Training Point . . . 42
4.5.2 Removing **n** Training Points . . . 43
4.5.3 Adding **n** Noisy Training Points . . . 46

5 Exploring TCAV Modifications **48**
5.1 Thresholding TCAV Score . . . 48
5.2 Retraining CAVs . . . 50

6 Discussion **52**
6.1 TCAV Best Practices . . . 52
6.2 Perspective on TCAV Score . . . 53
6.3 Future Work . . . 53
6.4 Conclusion . . . 54

Appendices **59**

A TCAV Score Variance Detailed Results **60**

Chapter 1

Introduction

1.1 Motivation

Interpretable Machine Learning has been an increasingly popular research direction in recent years[1] with many open questions. Generally speaking, interpretable machine learning seeks to either 1) build, then train, interpretable models or 2) discover how highly complex prediction models work in a useful way. There are many motivations for interpretable machine learning, including safety, ethical, and scientific concerns.

This project focuses on understanding and improving an existing algorithm that provides human-friendly, model-agnostic interpretability. This algorithm, Testing with Concept Activation Vectors (Kim et al. 2017), was mentioned by Google's CEO, Sundar Pichai, in the 2019 I/O Keynote address[2] as a step towards interpretability in machine learning and, in particular, identifying biased prediction systems. Since its publication in November 2017, TCAV has received attention[3] in the machine learning community and there are already extensions (Ghorbani, Wexler, and Kim 2019) and applications (Cai et al. 2019) published based on the original paper. This project explores many aspects of the algorithm, tests its limits, and proposes best practices for practitioners.

1.2 Contribution

The following are the key contributions from this project:

- Visualizing CAVs and concepts in low dimensions
- Evaluation of variance in TCAV score and suggested solutions
- Methodology for comparing CAV generation methods in a controlled fashion
- Study of CAV perturbation and TCAV score perturbation via adding or removing concept examples

[1]18,400 related publications since 2015 according to Google Scholar
[2]Google Keynote (Google I/O'19) `www.youtube.com/watch?v=lyRPyRKHO8M`
[3]`www.kdnuggets.com/2019/07/google-technique-understand-neural-networks-thinking.html`

- Best practices for CAV generation for selecting the negative class, positive class, and linear classifier
- Proposing a modified TCAV score calculation

1.3 Outline

In this project, Chapter 2 introduces the field of interpretable machine learning and recent work related to TCAV and the experiments considered. Chapter 3 details experiments and results related to CAV visualization and understanding. Chapter 4 describes experiments and results on quantifying uncertainty in TCAV scores, perturbing CAVs, and introduces the idea of *Robust TCAV*. Chapter 5 presents two TCAV extension ideas, including a modified TCAV score calculation using a threshold. Chapter 6 provides a final discussion of results, lists many ideas for future work, and presents a best practices for this method.

Chapter 2

Background

2.1 Interpretable Machine Learning

Neural networks have accomplished a variety of impressive feats in fields including computer vision, speech recognition, and machine translation amongst many others. Deep neural networks (DNNs) contain hidden layers with many units, which leads to many parameters. With this form of model architecture, it can be difficult for an analyst (someone trying to understand a model) to understand the importance of a single parameter or hidden unit in the model's decision. DNNs, therefore, fall under a category of "black-box" models because of this opaque quality.

As DNNs have grown in popularity and usefulness, so has the demand to understand black-box models. This project is about **interpretability** in machine learning, which here is taken to mean presenting a rationale behind an algorithm's decision in terms understandable by humans (Doshi-Velez and Kim 2017). Amongst a variety of approaches to interpretability, the focus of this project is on interpretability methods that **incorporate human input**.

Formally, a dataset $\mathcal{D} = (\mathcal{X}, \mathcal{Y})$ contains instances of features $\mathcal{X}$ producing observations $\mathcal{Y}$. A prediction algorithm $b : \mathcal{X} \rightarrow \mathcal{Y}$ learns this mapping. For an instance $x \in \mathcal{X}$, the prediction is $b(x) = \hat{y}$. With this definition, interpretability can be viewed both locally and globally.

Local interpretability aims to provide the user (human) with the reasons b made its prediction at particular instance x, which is $b(x) = \hat{y}$. **Global** interpretability seeks to give the user a general understanding of how the algorithm makes decisions, that is, how the algorithm b behaves for all of the feature space by understanding the mapping $b : \mathcal{X} \rightarrow \mathcal{Y}$ (Weller 2017). A locally important feature may not have the same importance or effect globally (Laugel et al. 2018). For some models, local explanations can also give a globally faithful explanation. Linear models have this quality.

Global interpretability seems desirable, but this task is difficult for highly complex models like DNNs, which use complex interactions between feature variables and can use millions of parameters to make a prediction. Some research has turned to higher level evidence to overcome this difficulty. Instead of attributing importance to input features, these methods use human-level concepts (Kim et al. 2017) or training prototypes (Chen et al. 2018) to achieve a degree of global interpretability.

2.1.1 When do we need interpretability?

There are many **motivations** for interpretability. Some are out of necessity, like safety, ethical, and legal concerns. Others motivations are merely beneficial to someone involved in either developing (engineers, researchers), deploying (companies), or using (consumers, analysts) prediction algorithms (Weller 2017).

Blind faith in black box algorithms can be problematic, including **safety-critical** applications like autonomous vehicles, control systems, and medicine in which human lives could depend on machine learning (Guidotti, Monreale, Ruggieri, Turini, et al. 2018). When these systems go wrong, engineers must be able to debug systems and understand the reasons behind decisions. Additionally, something like safety can be hard to quantify, so if a model is interpretable, the reasons for its decisions can be compared to an auxiliary criteria like safety (Doshi-Velez and Kim 2017).

Beyond safety, the ability to explain an algorithm's decisions has **ethical** and accountability benefits (Doshi-Velez, Kortz, et al. 2017). Under the General Data Protection Regulation (GDPR) in the EU, users have the right to meaningful information regarding algorithmic decisions (Goodman and Flaxman 2017). GDPR is in law as of May 2018, although there is some debate over the rights a user actually has under GDPR (Wachter, Mittelstadt, and Floridi 2017). For algorithms in the justice system, interpretability could be an ethical requirement to help ensure algorithms are fair to all groups of society (Rudin 2018). The issue of fairness is exacerbated by historical datasets containing some bias or prejudice (Pedreshi, Ruggieri, and Turini 2008). Online text corpora have been shown to be biased in many ways, but specifically sexist and racially prejudiced (Caliskan, Bryson, and Narayanan 2017). Ultimately, biased datasets are a reflection of biased humans at some point in time but these biases need not be perpetuated through algorithms trained on this data. Interpretability in this scenario could help identify when sensitive input variables are being used for a prediction or be used in the training process to eliminate, rather than identify biases.

In other situations, interpretability is not mandated, but still **beneficial** to some party. First, interpretability methods can help the **developers** of algorithms debug their systems. Often, machine learning algorithms can exploit correlations in data to make classifications in a way beyond the intentions of the developer (Ribeiro, Singh, and Guestrin 2016). Developers might also want to understand how their algorithm may behave when exposed to adversarial examples (see §2.4.3), which are inputs to a model cleverly designed to elicit some unusual model performance (Goodfellow, Shlens, and Szegedy 2014) (Szegedy, Zaremba, et al. 2013). Even deep text classifiers are susceptible to adversarial examples (Liang et al. 2017). Additionally, the **deployer** of an algorithm could give its users an explanation for a particular decision, leading users to increase trust in the algorithm (Yin, Vaughan, and Wallach 2019). An example that would be beneficial for a **user**: being denied a loan and being given an actionable and faithful explanation as to why the loan application was denied. In contrast, one can imagine a scenario in which the deployer can choose from a set of many possible explanations and gives the user one that most satisfies some hidden goal and is not necessarily faithful to the algorithm (Weller 2017).

Some use-cases of machine learning **do not require an explanation**. These include tasks for which the problem and solution are well-understood or there is a relatively low penalty for a misclassification (Doshi-Velez and Kim 2017).

2.1.2 Explainablility

In the realm of machine learning, interpretability and explainablility are often used interchangeably but should be considered as two distinct but related ideas. Gilpin et al. 2018 gives the following framework: an explanation has two components: 1) *interpretability* and 2) *completeness*. Interpretability strives to give human-understandable insight into the internal decision process of an algorithm whereas an explanation has completeness if it is faithful to the original algorithm. In fact, the authors champion interpretability methods in which these two components can be "traded-off", thereby allowing for simple yet incomplete explanations in one setting and for complex and more complete explanations in another.

Consider an overly simply (very interpretable) explanation for a complex algorithm, which lacks completeness. Gilpin et al. argue that is type of explanation is unethical if used to build a user's trust of an algorithm. Rudin 2018 points out that many "explanations" given by algorithms are essentially model summaries and therefore lack completeness.

Many frameworks and categorizations exist to help contextualize research in interpretable machine learning (Gilpin et al. 2018) (Doshi-Velez and Kim 2017) (Guidotti, Monreale, Ruggieri, Turini, et al. 2018). There has been a recent push in interpretable machine learning for common metrics and datasets on which to evaluate interpretability methods. Benchmark Interpretability Methods (BIM) (Yang and Kim 2019) is one such example.

2.2 Approaches

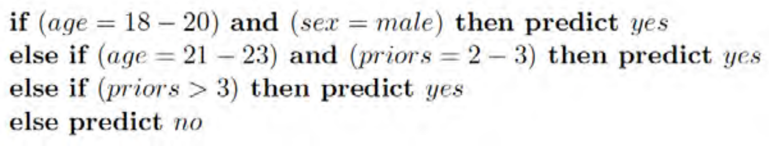
Approaches to interpretability for prediction models can be roughly categorized into two classes. **Constrained models**, include models constrained to a limit list of architectures, which are chosen to be **inherently interpretable** or transparent box designs (Guidotti, Monreale, Ruggieri, Turini, et al. 2018). Also in this category are models that make use of **constraints in training** to achieve interpretability. The next set of approaches is **post-hoc** methods in which a second algorithm is used to provide interpretability for a prediction algorithm that has already been trained.

2.2.1 Constrained Models

2.2.1.1 Inherently Interpretable Models

Inherently interpretable models have a constrained architecture that results in an analyst being able to always interpret the model. Such architectures include short decision trees, rule lists (fig. 2.1), and linear models (Angelino et al. 2017). A common perception in machine learning is that there is a **fundamental trade-off** between model interpretability and accuracy. The validity of this perception is highly dependent on the task. Letham et al. 2015 provide a use-case in which Bayesian

Rule Lists proved to be competitive in accuracy with black-box models while remaining completely interpretable.



```
if (age = 18 − 20) and (sex = male) then predict yes
else if (age = 21 − 23) and (priors = 2 − 3) then predict yes
else if (priors > 3) then predict yes
else predict no
```

Figure 2.1: From (Angelino et al. 2017), a rule list for predicting recidivism

In response to this perception, Rudin 2018 lays out an argument for the *Rashomon set*. This is the set of models that can achieve high predictive accuracy on a finite dataset. The argument is as follows: if the *Rashomon set* is sufficiently large and contains many diverse forms of models, then it is *bound* to contain a model that achieves both high accuracy and is interpretable. Rudin gives weight to this theory as a *possible* technical reason to believe that many interpretable models may exist but does not fully endorse the argument. While simple and elegant, this argument is not useful beyond suggesting that there is a large set of models that would be appropriate for a single prediction task. While it is possible this theory is true, it seems irrelevant at the moment. First, interpretability needs to be defined in a domain-specific method before the *Rashomon set* can be searched for an interpretable model.

Rudin makes in excellent point in that the **belief** that there is an inherent trade-off between accuracy and interpretability likely causes many developers to not attempt to create an interpretable model under the (possibly) false pretense that any such model would *always* be less accurate. As interpretability becomes more demanded, hopefully the developers of prediction algorithms will question this belief more and examine the possibility of an inherently interpretable model performing well for their task.

As a final consideration, the complexity of inherently interpretable models must be limited (Guidotti, Monreale, Ruggieri, Turini, et al. 2018). For example, humans would have much difficulty making sense of a linear model with 100 input features or a decision tree with *many* branches and nodes. Therefore, inherently interpretable architectures are only really interpretable if sensibly constrained in size.

2.2.1.2 Interpretability Constraints in Training

Another related approach is to force some constraint on a model during training, which will make it more interpretable. While similar to inherently interpretable models in that this decision is made *before* training, this approach is generally more flexible in architecture.

One such example is training with a layer that predicts prototypes for certain features in addition to its class prediction (Chen et al. 2018). While this model is more difficult to train, it achieves comparable accuracy with similar, non-interpretable models for the same tasks considered.

In another example, local constraints for interpretability in training are enforced. An increase accuracy against an unconstrained model for a few examples is observed using *Teaching Explanations for Decisions (TED)* (Hind et al. 2019). TED requires each training point to be labeled with an explanation, which is used in the training process.

These algorithms highlight the fact that post-hoc methods are *not* the only solution when a more complex model architecture is desired. However, retraining a model is not always a viable option. In addition, labeling training data with explanations or some other supplemental information is would not always be realistic given limited human resources and huge amounts of training data used by some algorithms.

2.2.2 Post-Hoc Methods

Post-hoc methods are typically **model-agnostic**, meaning the prediction algorithm can be arbitrarily complex, or a black box, although some post-hoc methods require knowledge of gradients and activations. An existing (trained) prediction algorithm can be used, so this approach is also dubbed **post-hoc**[1] as the explanation comes after the prediction process (Laugel et al. 2018). A common issue in this branch of interpretability is ensuring that the explanation provided is faithful to the prediction model. Some degree of local and global interpretability is possible with a post-hoc approach.

2.2.2.1 Saliency Maps

For computer vision tasks, **saliency maps** or sensitivity maps are used to visualize a explanations for the classification $b(x)$ of an image x (Dabkowski and Gal 2017) (Simonyan, Vedaldi, and Zisserman 2013). In these maps, the pixels in the image that most influence the algorithm's classification are highlighted, giving a local explanation for image x.

In *gradient-based approaches*, the gradient of the classification of a target class with respect to each dimension of the input is found. The dimensions of input, in this case, pixels, that have the highest absolute value of gradient (eq. (2.1)) are highlighted to construct the saliency map. For a class of interest, k, the logit for input x for class k is $h_k(x)$. The gradient of $h_k(x)$ with respect to input x is:

$$\frac{\partial h_k(x)}{\partial x} \tag{2.1}$$

Many approaches (Selvaraju et al. 2017) seek improve upon simply using raw gradient values. Smoothgrad (Smilkov et al. 2017) adds noise to the input image and averages over maps for many samples to reduce noise in saliency maps.

Alternatively, *reference-based approaches* assess the sensitivity of the classification on the replacement of a group of pixels with some low-information substitute. The choice of the replacement pixels is non-trivial and three possibilities are constant, noise, and blur (R. C. Fong and Vedaldi 2017) without inferring the data-generating latent space. Chang et al. 2018 propose a generative approach

[1]Latin for *after the event*

to filling this removed space, rather than some uniform reference. The generative approach fills the space with the most likely pixels given the rest of the image.

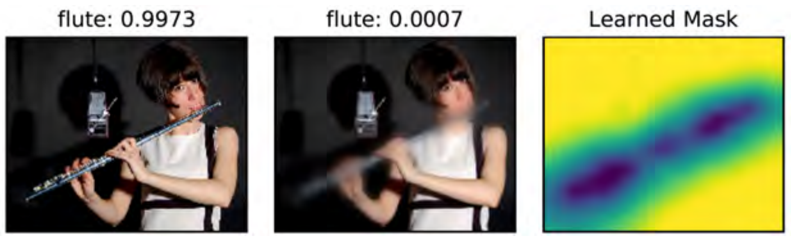


Figure 2.2: From (R. C. Fong and Vedaldi 2017), an image with the *flute* label (left), result of finding smallest blurring area that causes p(flute) to maximally decrease (middle), and learned mask (right).

Despite many recent advances, saliency maps have several **shortcomings**. First, common saliency map techniques have been shown to be unreliable when a constant vector shift is applied to the input (Kindermans et al. 2017) and some gradient-based approaches are invariant to model randomization and label randomization (Adebayo et al. 2018). Next, descriptions about saliency maps tend to only include examples for images that were classified *correctly*, giving a false sense of security in this explanation method (Rudin 2018). Also, saliency maps lack in *completeness* in explanation; while it tells an analyst where an image classifier is looking, it does not explain what is happening with with the salient features.

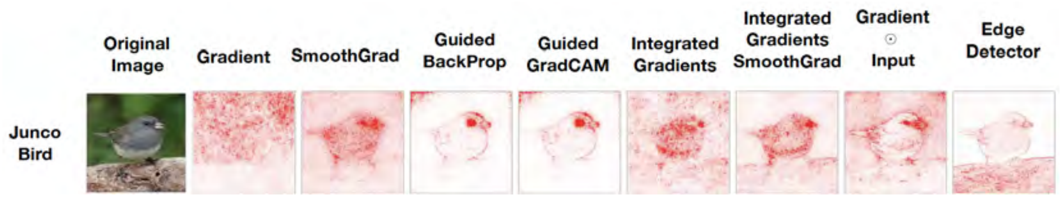


Figure 2.3: From (Adebayo et al. 2018), an image of a “junco bird” and saliency maps from several popular techniques compared to an edge detector. Inception v3 trained for ImageNet was used.

Finally, saliency maps are susceptible to human *confirmation bias*. If we notice something in a saliency map that agrees with our intuition about what features the algorithm *should* be looking for, we tend to assume the model is performing in accordance with our intuition. As a concrete example, Adebayo et al. 2018 used an edge detector as a substitute for a real saliency map (fig. 2.3). The edge detector highlights features that appear to be relevant to the given class even though the edge detector has no knowledge of the model or gradients. This shows that just because a saliency maps follows our intuition, does not mean that the explanation given is faithful to the features used for class prediction.

2.2.2.2 Surrogate models

Some post-hoc methods seek to emulate part or all of the black box model in an interpretable way (Hara and Hayashi 2016). A **local surrogate** approach approximately recreates only part of a black box model $b(x)$, typically around an instance x by sampling new instances in close proximity to x.

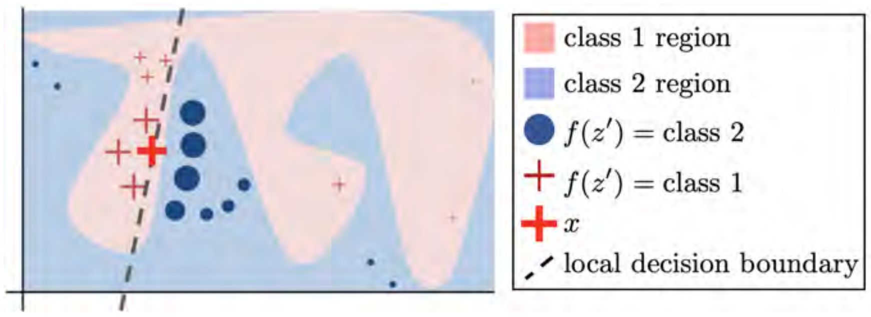


Figure 2.4: From (Ribeiro, Singh, and Guestrin 2016), a toy example showing the decision boundary of a function f, instance of interest, x, and perturbed samples z'. z' samples are weighted by their proximity to x, which is shown as marker size.

Local Interpretable Model-agnostic Explanations (LIME) (Ribeiro, Singh, and Guestrin 2016) is an example of this. A set of perturbed samples z' are generated close to the original input x. The classification $f(z')$ is found and these decisions are weighted according to their proximity to x. This sampling process is used to train a linear classifier to generate a local explanation of $f(x)$, as shown in fig. 2.4. Thus, the explanation found is the local decision boundary of a more complex prediction algorithm.

There are **drawbacks** of the LIME approach and local surrogate models in general. The neighborhood of the input x and perturbed samples z' can have a significant impact on the explanation given by the linear classifier (Laugel et al. 2018). Furthermore, local surrogate models are only faithful to b in the neighborhood of the input x and the features of x could have a much different impact globally. This leads to a lack of *completeness* (§2.1.2) in the given explanation. The explanation is highly interpretable as the surrogate is linear but is likely a gross oversimplification of b if b is complex and non-linear.

Local rule-based explanations (LORE) (Guidotti, Monreale, Ruggieri, Pedreschi, et al. 2018) is a local surrogate model that can provide *more* useful information to an analyst. Instead of training a linear classifier in a local neighborhood, LORE trains a decision rule, which is highly interpretable. In addition, a set of counterfactual rules is given, which give features that, had they been some other value, the decision would have been different. These counterfactual explanations would be useful in deployments in which users want an *actionable* explanation for why they were classified in a certain way.

2.2.2.3 Visualizations

Some interpretability techniques aim to give qualitative explanations. These are more focused on model-inspection rather than explaining a particular decision. **Activation Maximization** can be used in understanding the role of a particular neuron by somehow maximizing it activations.

DeepDream (Mordvintsev, Olah, and Tyka 2015) has been used to create mesmerizing art using this approach. An optimization process is used to alter an input image to directly maximize the activations for a particular neuron or layer. Popular DeepDream implementations typically use a image classification network such as GoogLeNet (Szegedy, Liu, et al. 2014), Inception v3 (Szegedy, Vanhoucke, et al. 2015), or VGG (Simonyan and Zisserman 2014) pre-trained for the ImageNet classes. Of the 1000 image classes in ImageNet, animals, in particular, dogs, dominate. Animal faces commonly appear in DeepDream images perhaps due to this fact.

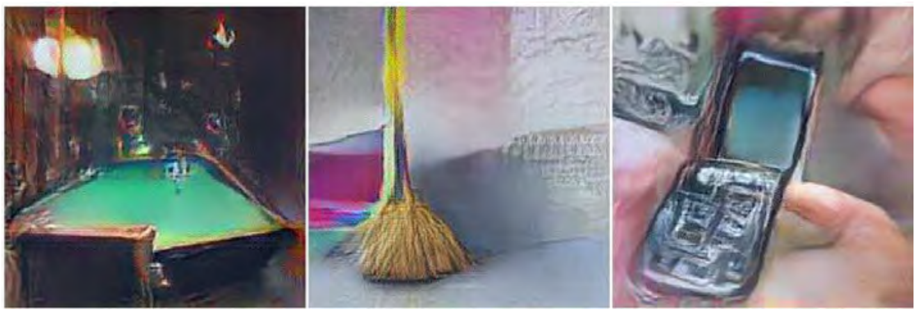


Figure 2.5: From (Nguyen et al. 2016), visualizations of **pool table** (left), **broom** (middle), and **cell phone** (right) using their corresponding neurons and activation maximization.

A Deep Generator Network (DGN) is used along with activation maximization by Nguyen et al. 2016 to create more realistic visualization with a learned prior (fig. 2.5). A DGN is trained to reconstruct realistic images from intermediate features in a network. These visualization techniques provide useful and interesting insight into neural networks, but are less quantitative than other methods.

2.3 Testing with Concept Activation Vectors

Another post-hoc method that achieves a more global explanation is **TCAV** (Testing with Concept Activation Vectors) (Kim et al. 2017). Two important ideas for this algorithm are **class** and **concept**. A **class**, k is a labeled prediction category represented by a single neuron in the logit layer of the network. Examples are labeled by class for training a DNN. A **concept**, c is a human-defined set of examples that all have a common attribute. The set of concept examples does not need to be from the training set. In fact, these concept examples can come from *anywhere* and it is the analyst's job to define a concept of interest using examples.

This approach makes use of these human-defined concepts to quantify an a class's sensitivity to a concept. For example, an analyst with access to an image classification algorithm may ask the question, "Is the zebra class sensitive to the striped concept?". With TCAV, any such class/concept pairing can be used to answer this type of question in a quantitative way, thereby giving **global** explanations.

TCAV will be considered for an images classification task. With TCAV, the first step in answering these questions is computing a Concept Activation Vector (CAV) using the concept images supplied by the analyst. This collection of images with a unifying concept c are in the positive class, P_c. A negative class N is some collection of images that do not contain this concept. As originally proposed, N is a set of random images (Kim et al. 2017). Although it is possible that a random image $x \in N$ *could* contain concept c, this is not critically important.

Using these sets of images, P_c and N, the activations are computed a layer of the analyst's choosing. With the Inception v3 (Szegedy, Vanhoucke, et al. 2015) architecture, the concatenative layers can be used as a bottleneck from which to analyze the activations as shown in Figure 2.6.

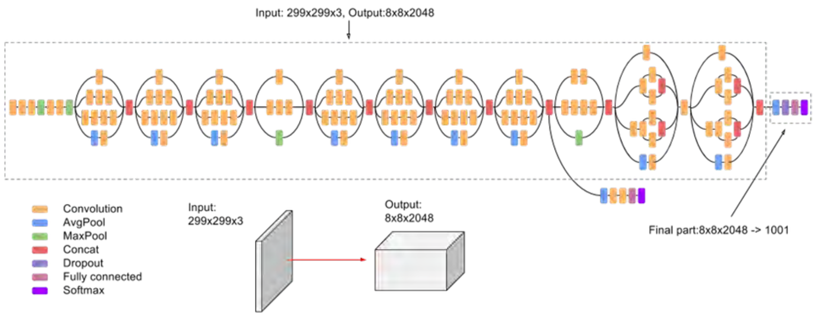


Figure 2.6: From Google[2], Inception v3 (Szegedy, Vanhoucke, et al. 2015) architecture. The activations at "Concat" layers are used for the TCAV process.

For an input image x, the activations at layer l are $f_l(x)$. These activations have high dimensionality ($d > 10^6$). Thinking about these activations in a lower dimension, like $\mathbb{R}^2$, can give an intuition for the process, as in fig. 2.7. A linear classification is trained between activations from images both classes $\{f_l(x) : x \in P_c\}$ and $\{f_l(x) : x \in N\}$. The CAV is normal to the hyperplane generated from the classifier and therefore points in the direction of P_c. As originally proposed, the linear classification method is a Support Vector Machine or, alternatively, a Logistic classifier (*TCAV Github repository* 2019).

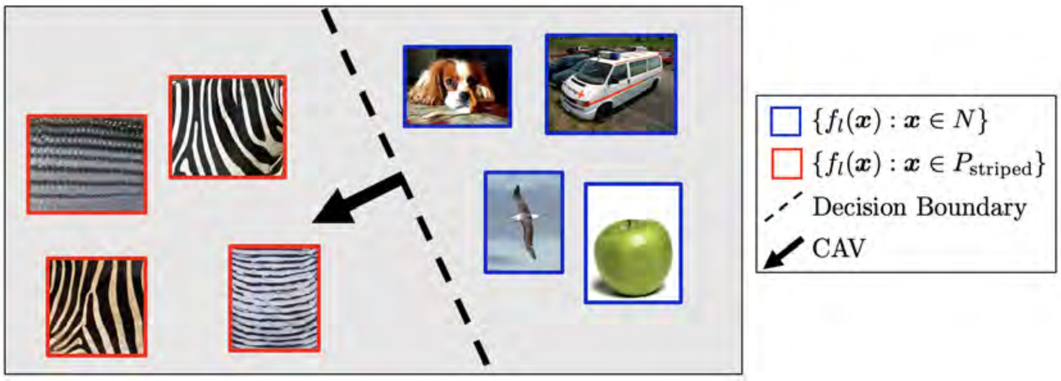

$\{f_l(\boldsymbol{x}) : \boldsymbol{x} \in N\}$
$\{f_l(\boldsymbol{x}) : \boldsymbol{x} \in P_{\text{striped}}\}$
Decision Boundary
CAV

Figure 2.7: Visualizing the CAV generation process in $\mathbb{R}^2$

Now, a CAV v_C^l for concept c in layer l has been trained. Next, the class of interest, k is introduced with the probability of input $\boldsymbol{x}$ belong to class k being $h_k(\boldsymbol{x})$. Rather than using an input image, the

[2]cloud.google.com/tpu/docs/inception-v3-advanced

prediction input will begin with activations at layer l, so $h_{l,k}(f_l(\boldsymbol{x}))$ is used. Using the activations at layer l, the directional derivative with respect to the probability of class k is computed. The result of this is dotted with the CAV to yield a sensitivity score, $S_{C,k,l}$.

$$S_{C,k,l}(\boldsymbol{x}) = \nabla h_{l,k}\left(f_l(\boldsymbol{x})\right) \cdot \boldsymbol{v}_C^l \tag{2.2}$$

Testing with CAVs is now relevant in the next step as the sensitivity of each image x of class k, is computed and the fraction that are sensitive in a positive direction is $\text{TCAV}_{\text{Q}_{C,k,l}}$, or TCAV score.

$$\text{TCAV}_{\text{Q}_{C,k,l}} = \frac{|\{\boldsymbol{x} \in X_k : S_{C,k,l}(\boldsymbol{x}) > 0\}|}{|X_k|} \tag{2.3}$$

The TCAV score $[0,1]$ (eq. (2.3)) uses the sensitivity only in determining, in a binary fashion, if a class images is sensitive to a concept. When $S_{C,k,l} > 0$, perturbing the activations of an input x at layer l in the direction of the CAV v_C^l results in an increase in the probability of class k. This is similar to using the gradient for saliency maps §2.2.2.1, although here, a **direction** of interest is compared with the gradient, rather than only considering the gradient values. By computing the fraction of images that were sensitive to the concept, an overall score can be assigned for the sensitivity of a class to a concept. The magnitude of $S_{C,k,l}$ is ultimately not used, but rather just the sign.

In summary, the TCAV process starts with a set of human-defined concepts, which is encoded in a CAV. The sensitivity of a class to a concept can then be quantified using directional derivatives and is succinctly given by the TCAV score.

2.3.1 TCAV Applications and Extensions

Although TCAV is a recent publication, some work has already been done building on it. Ghorbani, Wexler, and Kim 2019 propose the Automated Concept-based Explanation (ACE) algorithm. For an image $x \in X_k$ ACE uses object segmentation to divide x into sub-images. Sub-images are clustered in network layer l and each cluster is treated as a concept. The TCAV score is found for each concept and by sorting by the highest TCAV score, the most important concepts for classifcation can be found. One advantage of ACE is that the concepts are collected automatically using object segmentation. ACE relies on TCAV process, so work presented in this project could strengthen the ACE in practice.

A few published **applications of TCAV** are in medicine. Kim et al. 2017 used TCAV on a model that predicts diabetic retinopathy using a domain expert to define concepts for interpretability. Cai et al. 2019 have also used TCAV in medicine. With the main goal being a allowing doctors to search for similar medical images from a database, TCAV was used to quantify how much a certain concept was present in an image. Users could then refine their search by adjusting how much of each concept the searched images would contain.

Another application could be detecting and mitigating biases in classification systems (§1.1). For example, for the *apron* was found to be more sensitive to the *woman* than the *man* concept based on TCAV scores (Kim et al. 2017). A developer might be interested in eliminating this bias in the system. In training (or retraining), a constraint or loss term could be added to enforce $TCAV_{Q,\text{apron},\text{man},l} \approx TCAV_{Q,\text{apron},\text{woman},l}$.

2.4 Related Work

2.4.1 Image Classification Networks

Convolutional Neural Networks (CNNs) have been the dominant choice of neural network architecture for computer vision tasks in recent years. They make use of convolving an input image with many learned filters to make a prediction.

AlexNet (Krizhevsky, Sutskever, and G. E. Hinton 2012), introduced in 2012, used an 8 layer CNN trained on GPUs, to achieve a remarkable state-of-the-art result at the time. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al. 2015) has been used as a benchmark for computer vision performance, specifically image recognition and object detection. Advances in network architecture and computation power led to a rapid improvement in performance (Simonyan and Zisserman 2014). GoogLeNet (Szegedy, Liu, et al. 2014) introduced the Inception module and achieved the state of the art result in 2014. Later, Inception v2 and v3 were introduced (Szegedy, Vanhoucke, et al. 2015). This project uses the Inception v3 model, trained for ImageNet, for experiments.

More recently, residual networks have become popular, which include *skip connections* between layers. This architecture allows for easier optimization and deeper networks and achieved the state-of-the-art result on ILSVRC in 2015 (He et al. 2016) and was incorporated into the inception architecture (Szegedy, Ioffe, et al. 2016).

Image classification networks have dramatically increased in performance in the last decade and with millions of parameters and many hidden layers and filters, some researchers have attempted to look at internal network states to understand the process.

2.4.2 Usefulness of Intermediate Layers

The neural activations of intermediate layers in neural networks have been previously explored for several uses. Alain and Bengio 2016 designed linear classifier probes with the goal of attaining a better understanding of abstraction level throughout the network. In fig. 2.8, Alain et al. found that probe accuracy increases monotonically along the depth of the network.

Additionally, Zeiler and Fergus 2014 provided a visualization method for CNN filters at different depths in the network, providing insight into the functionality of each layer. Finally, R. Fong and Vedaldi 2018 explored the hypothesis that semantic representations are distributed throughout multiple layers and therefore these must be studied in conjunction.

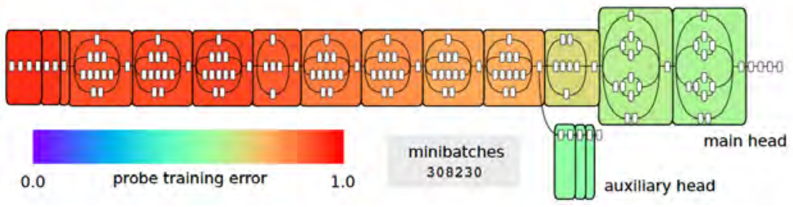
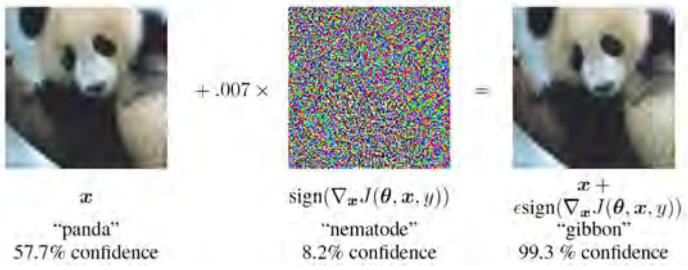


Figure 2.8: From (Alain and Bengio 2016), the accuracy of linear probes at different points in the Inception v3 network.

2.4.3 Adversarial Examples

Some image classification algorithms can be fooled. Adversarial examples can be generated using gradient-based methods to add a slight perturbation to a normal image. The perturbed image is indistinguishable from the original image to humans but can be classified with $\approx 100\%$ accuracy into any class of the the adversarial attacker's choosing (Goodfellow, Shlens, and Szegedy 2014) (Szegedy, Zaremba, et al. 2013). In fig. 2.9, $J(\boldsymbol{\theta}, \boldsymbol{x}, y)$ is the loss function used to train the model with parameters θ, input $\boldsymbol{x}$, and target class y. An adversarial example makes use of the gradient of this loss function with respect to the input image, $\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y)$. A perturbation is possible by adding a small amount, ϵ, in this direction, after applying a sign operator.



$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Figure 2.9: From (Goodfellow, Shlens, and Szegedy 2014), an adversarial example using GoogLeNet (Szegedy, Liu, et al. 2014) and the fast sign method.

TCAV presents a possible safeguard against these attacks by considering the TCAV score (Kim et al. 2017). For this example, the panda image, now in the gibbon class, could be checked against other (real) gibbon images, which would share concepts that the adversarial example (presumably) would not. Kim et al. perform this experiment with a different example.

2.4.4 Disentanglement

There has a been a recent rise in popularity of Variational Auto-Encoders (VAE) (Kingma and Welling 2013) and subsequent modifications, such as β-VAE (Higgins, Matthey, et al. 2017). A VAE encodes an input into a latent space (encoder) and reconstructs the latent representation with a decoder. The reconstruction loss minimized in training, forcing the encoder network to learn a compact representation of the input with minimal information loss as the latent space typical has much fewer dimensions than the input.

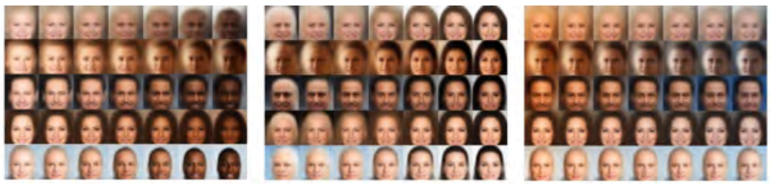


Figure 2.10: From (Higgins, Matthey, et al. 2017), Traversing three latent dimensions learned by β-VAE on the celebA dataset. They are deciphered to be encode skin color (left), age/gender (middle), and image saturation (right).

β-VAE and other modifications enforce a constraint on the training process in an attempt to have each dimension of the latent space represent an independent semantic dimension of the input space. For example, for reconstructing faces from celebA, β-VAE learns a several interpretable latent dimensions independently (fig. 2.10). A latent representation with distinct meanings of each dimension is said to be disentangled (Higgins, Amos, et al. 2018). Work on disentanglement is relevant to this project because a lower dimension representation of network activations is used for visualization.

2.4.5 Style Transfer

Style transfer involves using an algorithm to apply the style of one image to transform another. Gatys, Ecker, and Bethge 2016 propose neural style transfer in which a feature-space is created dedicated to capturing image style. This feature-space uses the outputs from filters at each layer in the network. Gradient descent is used to alter a new image to the the style of another image using this feature-space.

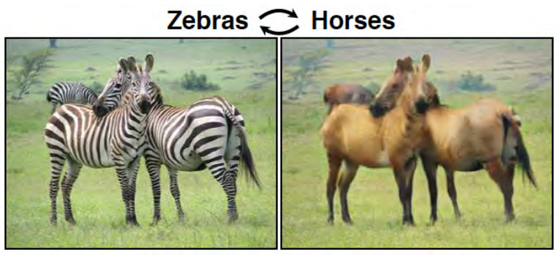


Figure 2.11: From (Zhu et al. 2017), example transformation from a real zebra image (left) to the same picture in the horse style.

CycleGAN (Zhu et al. 2017) focuses on mapping from one set of images to another set, rather than from a single image to the style of another single image. CycleGAN is unique in that the sets of images need not be paired, as is required with other style transfer algorithms. CycleGAN involves two mapping functions $G : X \to Y$ and $F : Y \to X$ in which X and Y are sets of images of two different styles. By optimizing for cycle consistency $(x \to G(x) \to F(G(x)) \approx x)$ and making use of adversarial discriminators, the two mapping functions are trained. CycleGAN achieves impressive results, as in fig. 2.11.

Chapter 3

Experiments on CAV Visualization

The first experiments building on TCAV involve visualizing the two classes of activations used to generate CAVs and attempting to discover what CAVs have learned.

3.1 Dimensionality Reduction

Dimensionality reduction techniques are used to transform data in $\mathbb{R}^n$ to $\mathbb{R}^m$ where $n > m$ and $m = 2$ is typical for visualization purposes. For Inception v3 (Szegedy, Vanhoucke, et al. 2015), table 3.1 gives the dimensions exist at each bottleneck layer.

Layer Name	**Shape**	**Flat Shape**
mixed0	35, 35, 256	313,600
mixed1	35, 35, 288	352,800
mixed2	35, 35, 288	352,800
mixed3	17, 17, 768	221,952
mixed4	17, 17, 768	221,952
mixed5	17, 17, 768	221,952
mixed6	17, 17, 768	221,952
mixed7	17, 17, 768	221,952
mixed8	8, 8, 1280	81,920
mixed9	8, 8, 2048	131,072
mixed10	8, 8, 2048	131,072

Table 3.1: Bottleneck names and activation shapes for Inception v3 (Szegedy, Vanhoucke, et al. 2015) architecture. The **Flat Shape** is used in the TCAV process.

Principal Component Analysis (**PCA**) is one such technique that attempts to explain variance of data in $\mathbb{R}^n$ with an orthogonal transformation to $\mathbb{R}^m$. **Sparse PCA** (Zou, Hastie, and Tibshirani 2006) is a modified version of PCA that includes a sparsity constrain on the input variables. Only some of the m input features will have a non-zero transformation component.

For a demonstration, 50 striped images from a textures database (Cimpoi et al. 2014) were used to created P_{striped}, a class that contains the *striped* concept The negative class N was comprised of 50 random images from the 2012 ImageNet validation set (Russakovsky et al. 2015). Throughout this project, the 2012 ImageNet validation set, which contains 50,000 images, is used to generate negative classes N by randomly selecting images from this set.

By decomposing the activations for both P_{striped} and N from the *mixed9* layer (fig. 3.1), it is clear that the classes are distinct. The striped concept contains many zebra-like images of stripes, which appear to cluster in the leftmost part of the PCA plot in fig. 3.1. Striped images that did not have this specific type of stripiness were closer to the random class in the embedding. This idea of transforming high-dimensional data into a lower-dimensional space and examining the meaning of each dimension is loosely related to VAEs and VAE modifications optimized for disentanglement (§2.4.4).

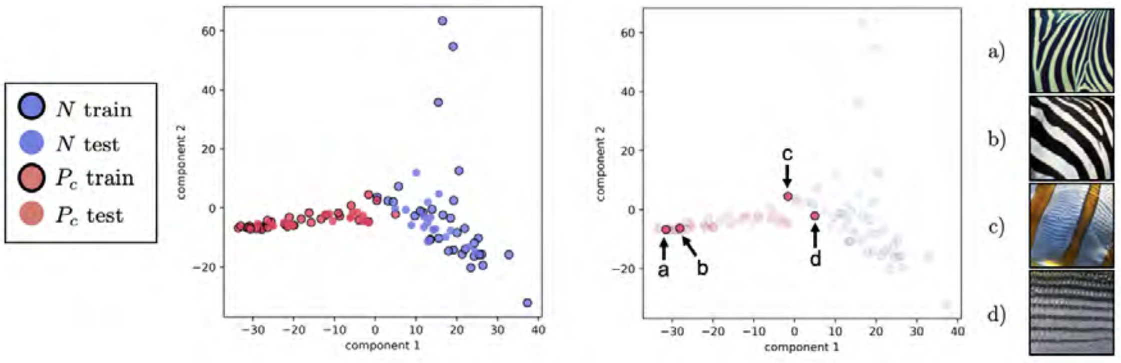


Figure 3.1: Acitvations of P_{striped} and N at layer *mixed9* decomposed using PCA (left). A few training points and their input image (right).

t-SNE (Maaten and G. Hinton 2008) and **UMAP** (McInnes, Healy, and Melville 2018) are two additional dimensionality reduction techniques. Several hyperparameters can be tuned with each of these methods affect the resulting embedding.

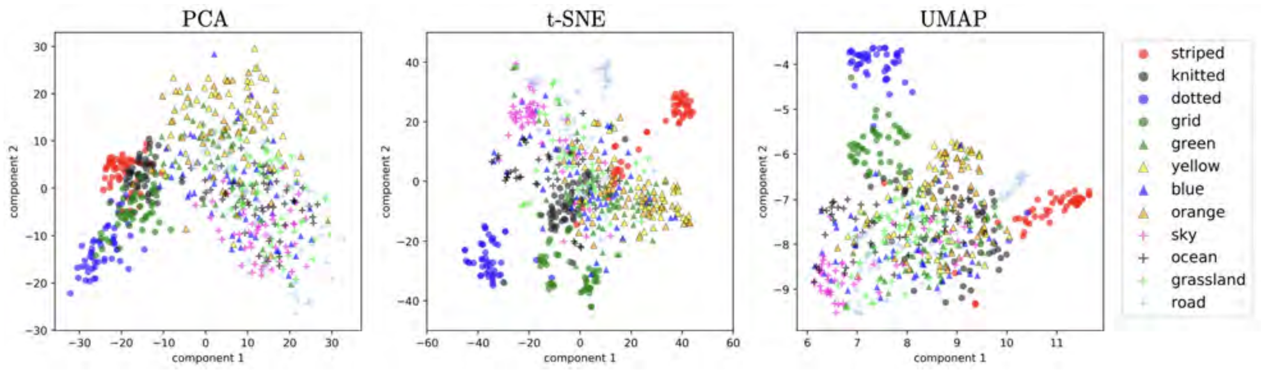


Figure 3.2: Acitvations of various concepts at layer *mixed9* decomposed using PCA (left), t-SNE (middle), and UMAP (right)

Using these three dimensionality reduction methods, the activations of images of several concepts (see fig. 3.10) were decomposed at different points in the architecture. At layer *mixed9*, which is one of the deepest layers (close to logit), some concept classes are fairly clustered with all three methods.

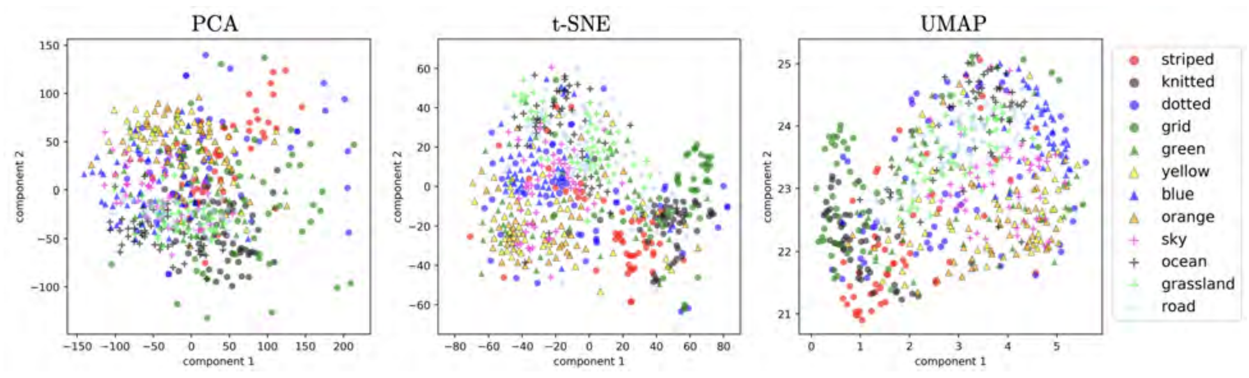


Figure 3.3: Acitvations of various concepts at layer *mixed0* decomposed using PCA (left), t-SNE (middle), and UMAP (right)

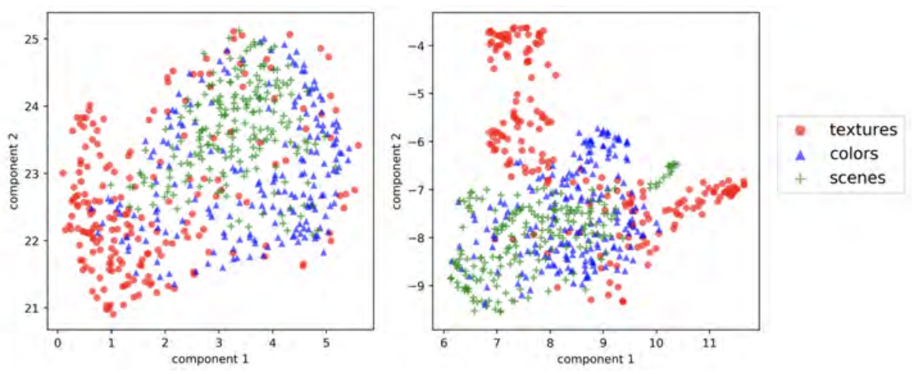


Figure 3.4: Acitvations of various concepts, grouped by category, at layer *mixed0* (left) and *mixed9* (right), decomposed using UMAP

At layer *mixed0*, which is the first bottleneck layer, concepts are generally less separated. This agrees with literature that deeper layers have better representations of high-level concepts, while lower layers detect edges and colors (Zeiler and Fergus 2014) (Alain and Bengio 2016). In fig. 3.4, concepts of the same category tend to overlap in both *mixed0* and *mixed9* with UMAP.

Dimensionality reduction is a useful way to visualize high-dimensional spaces, like activations, and can provide insight on how concepts and categories of concepts are represented in a given network layer, and how this representation changes at different points in the network.

PCA and Sparse PCA are used to visualize CAVs in later experiments. Each embedding is trained on the P_c and N training examples. This same embedding is used to transform CAVs and decision boundaries.

3.2 DeepDream

The DeepDream algorithm (Mordvintsev, Olah, and Tyka 2015) can give an insight into the inner workings of a neural network and generate fascinating patterns through activation maximization (discussed in §2.2.2.3). Originally, the algorithm's loss function was defined as the mean of the activations from a given layer l starting with an input image x.

$$L(x) = \frac{\sum f_l(x)}{|f_l(x)|} \tag{3.1}$$

Gradient ascent is used to *maximize* $L(x)$ by altering the input image x through backpropagation of the gradient. Therefore, DeepDream will perturb the input image to maximize the mean activations of layer l. Alternatively, the activation of a particular neuron can be maximized.

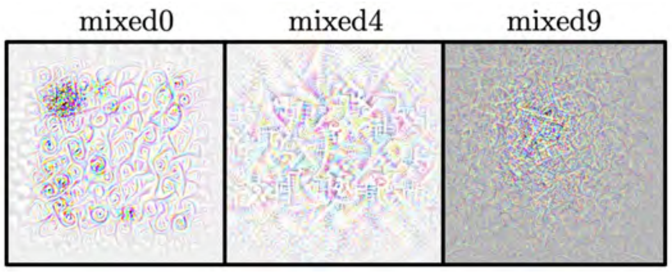


Figure 3.5: DeepDream algorithm used to maximize the activations of a specified layer, using the loss function in eq. (3.1) for three different layers.

While this process produces amazing patterns, I sought to visualize patterns from CAVs, as was done in Kim et al. 2017. Instead of maximizing activations in a certain layer, the loss function was altered to reflect the distance between $f_l(x)$ and a CAV at the same layer. As this loss function is maximized, it forces $f_l(x)$ to be more similar to $\boldsymbol{v}_C^l$.

$$L(x) = f_l(x) \cdot \boldsymbol{v}_C^l \tag{3.2}$$

Additionally, CAVs from multiple layers can be used in a collective loss function. The loss coefficient for each layer is γ_l and it controls how much (if at all) the CAV and activations at layer l contribute to $L(x)$. γ_l could account for the shape of each l used (table 3.1), such that the losses from each l are evenly weighted. CAV accuracies (§3.4) could be used to estimate where concept will be the most pronounced and could be used to set γ_l. Selecting the optimal parameters to produce a desired effect in the manipulated image requires trial and error.

$$L(x) = \sum_{l=1}^{n} \gamma_l * f_l(x) \cdot \boldsymbol{v}_C^l \tag{3.3}$$

The DeepDream process uses image down-scaling to help the process produce patterns at different scales and many parameters and be adjusted to alter the types of patterns that arise. Additional terms to the loss function can be added including a term to prevent pixel values from reaching extreme values (eq. (3.4)). Pixels are represented as $[-1, 1]$, so a pixel value of 0 maximizes this

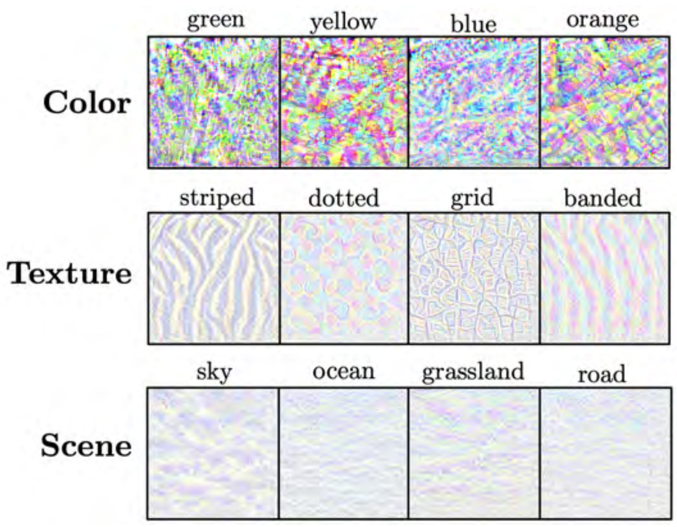


Figure 3.6: DeepDream algorithm with loss function altered to include CAVs at each layer for several concepts

term, causing images to take a grey color.

$$L(x) = \sum_{l=1}^{n} \gamma_l * f_l(x) \cdot \boldsymbol{v}_C^l - \eta * \sum x \tag{3.4}$$

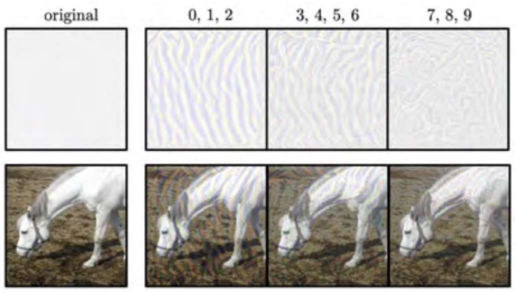


Figure 3.7: DeepDream algorithm applied to two original images. The loss function used the *striped* CAV in several different layer combinations. For example, "0, 1, 2" means the striped CAVs from *mixed0*, *mixed1*, and *mixed2* were used with equal weight to generate the loss term as in eq. (3.3).

Any image can be the start for the DeepDream algorithm. In fig. 3.7, a horse image is the original image and is altered in the direction of *striped* CAVs. Based on the comparison to the white noise original image, it seems that there is original image has little impact on patterns generated from the CAV.

One pitfall of relying on DeepDream is our susceptibility to confirmation bias. We would like to think that the striped CAV has learned this concept and are easily persuaded that it has when something close to stripy is produced. Therefore, the inclusion of this visualization technique is *only* a means to some insight into where in the model concepts are most articulated and some idea of what a concept may look like.

3.3 Activation Perturbation

Rather than altering the input image in an algorithm like DeepDream §3.2, CAVs can be used to perturb activations of intermediate layers in the direction of a concept without back-propagating gradients to the input image.

Using this idea, the activations $f_l(x)$ from an image x can be perturbed to varying degrees in the direction of a CAV $\boldsymbol{v}_C^l$. The perturbation is simple addition in the CAV direction controlled the coefficient α.

$$f_l'(x) = f_l(x) + \alpha * \boldsymbol{v}_C^l \tag{3.5}$$

By altering the intermediate activations, the class predictions will change when the activations are fed through the rest of the network.

$$h_k(f_l'(x)) \neq h_k(f_l(x)) \tag{3.6}$$

It is possible that one specific class $\hat{k}$ dominates $\hat{k} = \text{argmax}_k\, h_k(f_l'(x))$ for a sufficiently large α. This can occur when a concept is a *very* strong signal for a certain class.

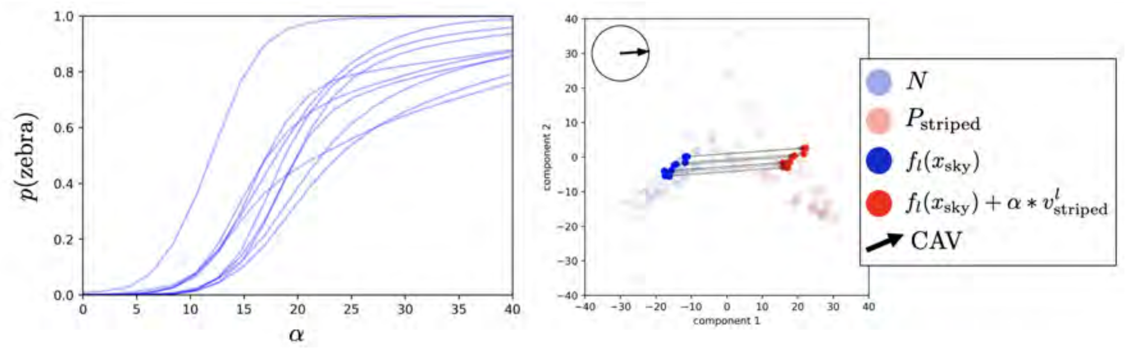


Figure 3.8: Parameter sweep of α using *striped* CAV at layer *mixed9*, and *sky* images (left). Acitvations of *sky* images perturbed ($\alpha = 20$) in the direction of the striped CAV decomposed using PCA (right)

From fig. 3.8, the activations move from within the N cluster to the P_{striped} region. In conjunction, the probability of the perturbed activations being classified as a zebra increases steadily for each set of activations. Even beyond $\alpha = 20$, $p(\text{zebra})$ increases monotonically for nearly every example. In fact, by observation:

$$\lim_{\alpha \to \infty} p(\text{zebra}) \approx 1 \tag{3.7}$$

After a certain point, the activations are likely saturated, meaning that more change in the same direction will have no impact on the resulting prediction. The same process is possible for CAVs representing other concepts, yet this does not guarantee meaningful results. For example, the CAV generated from *sky* images, $\boldsymbol{v}_{\text{sky}}^{mixed9}$ results in $\hat{k} = \text{ear}$ as $\alpha \to \infty$, starting with any image. Other seemingly uncorrelated results exist for other CAVs.

By starting with images from an existing class, the transition from the original class to the class associated with a CAV can be observed in the parameter sweep with α. This transition is shown starting with 50 horse images in fig. 3.9.

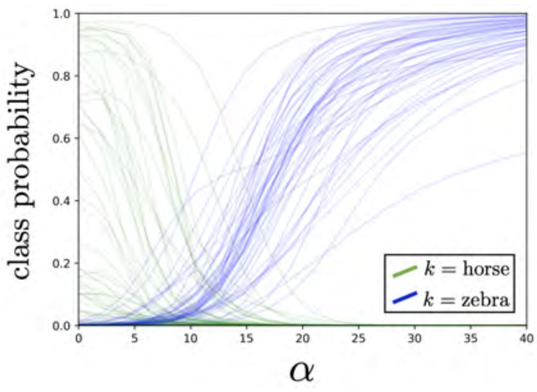


Figure 3.9: Parameter sweep of α using *striped* CAV at layer *mixed9*, and *horse* images.

The probability of the zebra class increases at different rates for different activations from horse images, as expected. Some activations are more easily influenced by perturbation in the direction of the CAV.

In fig. 3.9, the starting class for the activations was k = horse and by increasing the perturbation in the striped direction, $\hat{k} = zebra$ for sufficiently large α. However, the horse and zebra classes are semantically close. With concepts that are semantically further from the zebra class, this average transition α is higher.

3.4 CAV Accuracy

From fig. 3.2 and fig. 3.3, it is clear that the same sets of input images can have much different representations at different bottlenecks in the network. Of course, a decomposition into $\mathbb{R}^2$ loses information. In which layers are the concepts so clearly articulated in the activations that a linear classifier can make a high accuracy separation?

To assess the separability of concept examples P_c from a set of random counterexamples N at different points in the network, several concepts were defined. The *Color* and *Scene* categories shown in fig. 3.10 were collected from ImageNet. The *Texture* category was collected from the Describable Textures Database (Cimpoi et al. 2014). 50 images for each concept were collected.

There are some important difference between categories. For example, in *Color*, only one object in the image must contain the given color; the entire image need not be the same color. In contrast, the *Texture* category contains images that are typically entirely occupied by the given texture. For the *Scene* category, objects such as people and cars can be present in the images. Furthermore, the *Scene* category is a higher-level concept because a collection of scene concept images need not have the same color or texture. There is a vast array of images that could qualify as the *road* concept, for example, and they may not share anything other than the presence of the human notion of a road which may have highly variable low-level features.

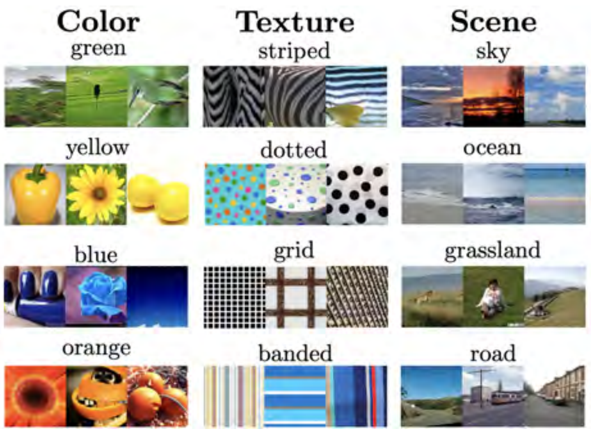


Figure 3.10: Example concepts and images listed under category

For these concepts, a linear classifier (SVM, in this case) was trained using a set of concept images P_c and N negative images. To account for variability in the train/test split and the images used in N, 5 classifiers were trained for each concept. For c = green, the CAVs were $N_0 \rightarrow P_{\text{green}}, \ldots,$ $N_4 \rightarrow P_{\text{green}}$. The mean and standard distribution of the accuracies of this set of linear classifiers, now referred to as "CAV Accuracy" were found for each bottleneck layer as shown in fig. 3.11.

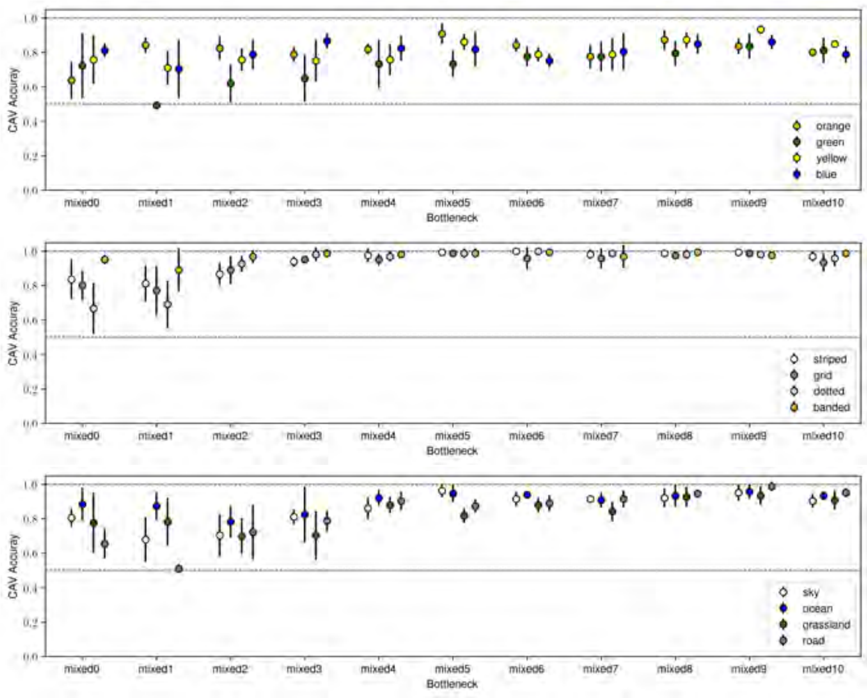


Figure 3.11: Means and standard deviations of CAV accuracy for 4 concepts in each of 3 categories at each bottleneck in the network

Insights into the representations of each concept at different points in the network and the CAV generation process are possible using fig. 3.11. For example, for all *Color* category concepts, the mean CAV accuracy rarely exceeded 0.9, possibly because images from N could contain the color of interest and therefore the two classes would be less separable. For *Textures*, very high accuracies were achieved by the *mixed4* layer and remained high in later layers. CAVs generated from concepts in the *Scene* category, which contains highest-level concepts, achieve a much lower accuracy in the beginning layers of the network, although gradually increase throughout the network. These observations are consistent with the CAV accuracies found by Kim et al. 2017 and work from Zeiler and Fergus 2014 that early layers detect low-level features like edges while high-level features are detected in deeper layers using combinations of low-level features.

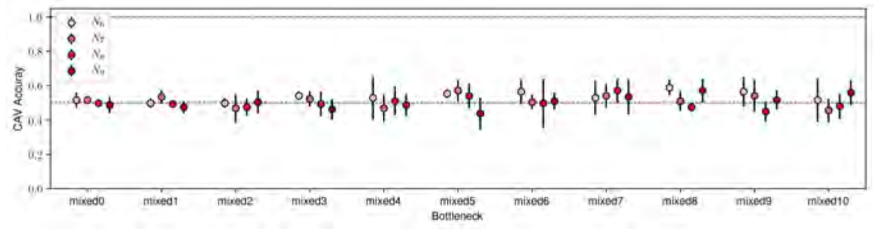


Figure 3.12: Means and standard deviations of CAV accuracy for 4 concepts at each bottleneck in the network. Error bars represent one standard deviation.

Finally, the CAV accuracies were evaluated with P_c being a negative class itself. For example, for using N_5 as P_c, the 5 CAVs used to generate the means and standard deviations in fig. 3.12 were $N_0 \to N_5, \ldots, N_4 \to N_5$. Using negative classes as concepts behaves exactly as one would hope; nothing of meaning is learned because the two classes are drawn from the same distribution. This experiment confirms that CAVs with meaningful concepts are did not happen by chance.

3.5 Sorting Images with CAVs

CAVs exist in $\mathbb{R}^d$, along with the activations from the same bottleneck. Therefore, vector comparison calculations can be done to assess how similar a set of activations $f_l(x)$ at bottleneck l are to a CAV, v_C^l. Sorting a set of activations by a distance metric like cosine similarity yields insight into how the concept is manifested in the class and can be impressive if the CAV learns something meaningful. Kim et al. 2017 use cosine similarity eq. (3.8) to sort images although using other metrics are also possible such as the magnitude of their directional derivative.

$$\text{cosine similarity}(u, v) = \frac{u \cdot v}{\|u\| \times \|v\|} \tag{3.8}$$

The Euclidean distance eq. (3.9) to the mean of P_c could also be used to sort images. For example, with $\text{dist}(\mu_{P_c}, f_l(x))$.

$$\text{dist}(u, v) = \sqrt{\sum_{i=1}^{n} (u_i - v_i)^2} \tag{3.9}$$

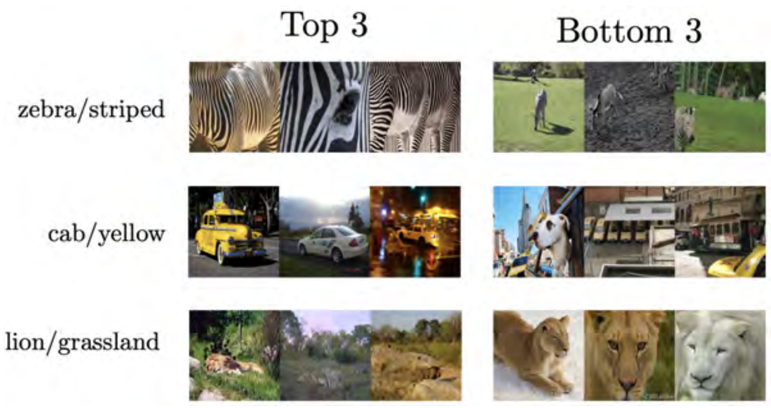


Figure 3.13: Top and bottom 3 class images sorted according to the cosine similarity with the corresponding CAV in the *mixed9* layer.

So far, the sorting tasks have been limited to cases in which the TCAV Score and the CAV accuracy are high. If the metrics available suggest the CAV is learning a meaningful concept and that concept *is* important to the prediction of the class of interest, then it follows that images that are most similar to the CAV agree with our human-defined notion of a concept. Now, sorting tasks will be considered in which either the CAV accuracy is low or the TCAV score is low.

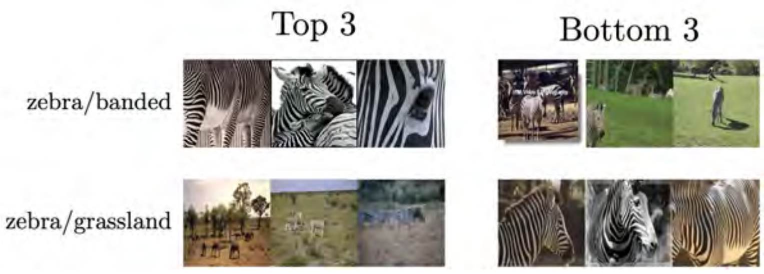


Figure 3.14: Top and bottom 3 zebra images sorted according to the *banded* and *grassland* CAV in the *mixed9* layer using cosine similarity.

The *banded* concept (fig. 3.10) was used in lieu of the *striped* concept as it relates to zebras. The *banded* concept actually falls under a broad semantic notion of striped but is a much different manifestation of this broad concept than the *striped* concept, which focuses on organic stripes. However, the *banded* CAV in the *mixed9* layer achieve a TCAV score of 0, so it appears to be minimally predictive for the zebra class. Sorting with the *banded* CAV using cosine similarity (fig. 3.14) performs remarkably well. Based on fig. 3.14, I hypothesize that CAVs can still learn meaningful information despite a TCAV score.

The last demonstration with sorting sought another way to visualize what a CAV has learned by finding **prototypical** examples. In fig. 3.15, 500 **grassland** images are sorted according to *grassland* CAV. Prototypical examples are the images the produced the closest activations to the CAV. This could be a useful approach for an analyst dealing with a large concept class and trying to understand what the CAV has learned.

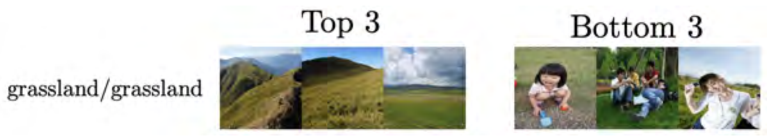


Figure 3.15: Top and bottom 3 grassland images sorted according to the *grassland* CAV in the *mixed9* layer using cosine similarity.

As with DeepDream §3.2, the sorting process is subject to our own confirmation bias. Along with other visualization techniques, sorting, can provide useful insight into CAVs.

Chapter 4

Experiments on Robust TCAV

This chapter focuses on clarifying the need for *Robust TCAV* and lists objectives for this idea. Experimental results and discussion are provided for TCAV score uncertainty and CAV perturbation.

4.1 Understanding TCAV Score

The TCAV score (eq. (2.3)) is used to measure sensitivity of class k to concept c. To gain a sense of the uncertainty in the TCAV score, several class/concept pairs were tested. From fig. 4.1, testing at bottlenecks deeper in the network (*mixed6*, *mixed8*, *mixed10*), generally results in a higher TCAV score in accordance with the observation from fig. 3.11 that CAV accuracy increases with depth in the network.

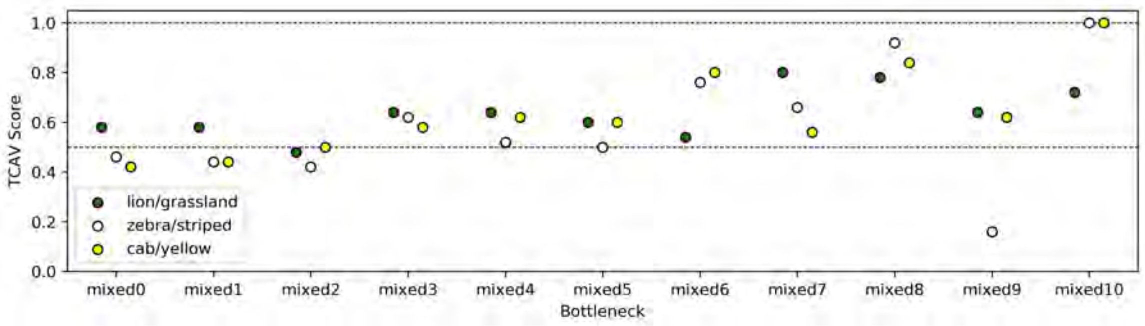


Figure 4.1: TCAV Score at different network bottlenecks for several class/concept pairings.

Is the reported TCAV score trustworthy? Ideally, the TCAV score would be the same for tests under very similar conditions. For example, if using the *same* P_c examples[1], *same* network layer, and *same* class of interest k, one would expect that the TCAV scores would be very similar. To test this hypothesis, the TCAV score was found for several class/concept pairings over 5 runs using the original TCAV implementation (*TCAV Github repository* 2019). Each run had a unique train/test split and unique N negative examples.

[1]The same 50 P_c images are used for each concept. In CAV generation, a random subset of 34 P_c images is selected for training.

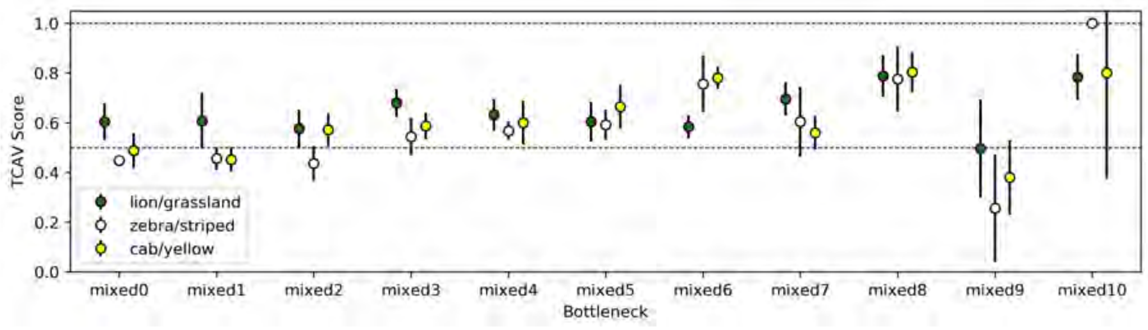


Figure 4.2: TCAV Score distribution at different network bottlenecks for several class/concept pairings. The error bars represent one standard deviation. Each distribution is over 5 observations.

From fig. 4.2, it is clear that the TCAV scores from a single run (fig. 4.1) actually come from a wide range of possible values under very similar conditions.

Going deeper into the TCAV score, **directional derivatives** are now further investigated. The TCAV score eq. (2.3) is computed as the fraction of input images that have a positive directional derivative $\nabla h_{l,k}\left(f_l(\boldsymbol{x})\right) \cdot \boldsymbol{v}_C^l$ for class k and CAV concept C at bottleneck l.

For a differentiable function, $f(x)$, the dot product between the gradient of $f(x)$ and the direction of interest $\mathbf{v}$ is the directional derivative.

$$\nabla_{\mathbf{v}} f(\mathbf{x}) = \nabla f(\mathbf{x}) \cdot \mathbf{v} \tag{4.1}$$

The definition of the directional derivative can also be used, which is the instantaneous change of $f(x)$ when a small perturbation in the direction of $\mathbf{v}$ is added to $\mathbf{x}$.

$$\nabla_{\mathbf{v}} f(\mathbf{x}) = \lim_{\alpha \to 0} \frac{f(\mathbf{x} + \alpha \mathbf{v}) - f(\mathbf{x})}{\alpha} \tag{4.2}$$

The activation perturbation process shown in (fig. 3.8) is similar to eq. (4.2) although the perturbation coefficient $\alpha \gg 0$. Instead of finding the gradient then taking the dot product with the CAV direction, the activation perturbation method traverses the direction of the CAV and records the response in the probability of the class of interest $h_k(f_l(\mathbf{x}))$. The next experiment will investigate how $h_k(f_l(\mathbf{x}))$ behaves for $\mathbf{x} \in X_k$ as $\alpha \to 0$ to understand the meaning of the directional derivative.

For 150 zebra images from ImageNet, most begin with $p(\text{zebra}) > 0.9$ from fig. 4.3. As the activations are perturbed in the striped CAV direction, it appears that most probabilities increase (although some briefly decrease) and $\lim_{\alpha \to \infty} p(\text{zebra}) \approx 1$. This trend is examined more closely as $0 < \alpha < 1$. The limit as $\alpha \to 0$ is found, which is the directional derivative computed using eq. (4.2), $S_{c,k,l}$. In fig. 4.3, most $S_{c,k,l} > 0$ and the resulting TCAV score is $146/150 = 0.98$. Computing the TCAV score using the gradient and dot product method eq. (4.1) yields the same result. For the purposes of computing the TCAV score, only the **sign** of the $S_{c,k,l}$ is important[2].

[2]Using the magnitude is considered in §5.1.

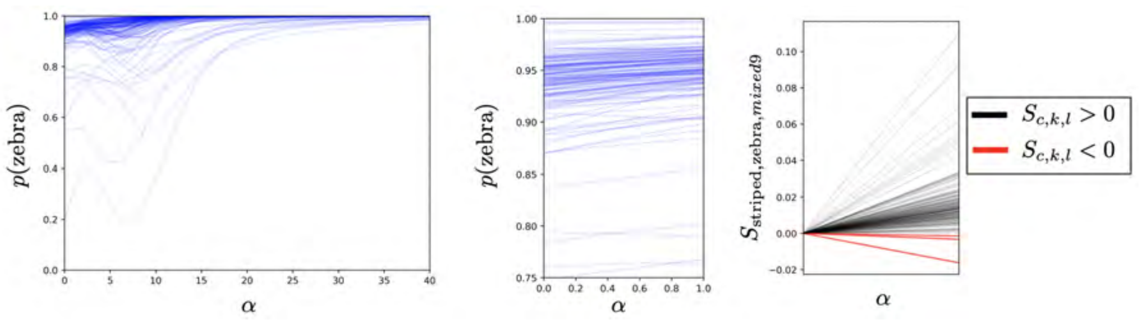


Figure 4.3: Zebra image activations (*mixed9*) perturbed by α in the striped CAV direction (left). α approaching 0 (middle). Limit as $\alpha \to 0$ with only the change from the original class probability shown, which is also $S_{\text{striped,zebra},mixed9}$ (right).

With this knowledge of the TCAV Score uncertainty and directional derivative process, criteria for a *robust* version of this process can be discussed.

4.2 Criteria

What would make the TCAV process *robust*? From the observations in §4.1, there is some degree of uncertainty in the TCAV score generated under very similar conditions. The following **criteria** are the first steps towards a robust TCAV process:

- **Consistent**: only a small change in TCAV score for CAVs generated from the same concept using different training examples (either P_c or N)
- **Resistant to perturbations**: only a small change in TCAV score when the CAV is perturbed by adding or removing concept examples in P_c

First, experiments will focus on **identifying sources of variance** in the TCAV score with the aim of making the TCAV process more **consistent**. Later experiments will involve perturbing concept examples to find ways in which the TCAV process can be **resistant to perturbations**.

A list of **best practices** in §6.1 addresses these concerns based on experimental observations. This list is meant to be useful for future analysts leveraging the power of TCAV to understand their models.

4.3 Comparison Methodology

To understand how a CAV behaves under sets of similar conditions, two variables were carefully controlled.

1. The points from P_c used for training, which will be controlled through the random seed for the train/test split[3]
2. The selection of the negative class, N

[3]train/test split fraction of 0.33 is used for all experiments

As mentioned in §2.3.1, random images from the ImageNet validation set are used for N, concept counterexamples. Sets of 50 of these images were collected and numbered starting with $N_0^{50}, N_1^{50}, \ldots$. For example, N_0^{50} contains 50 random images from the validation set and N_1^{50} contains a *different* 50 random images. Changing which N to use in the CAV generation process will affect the resulting CAV.

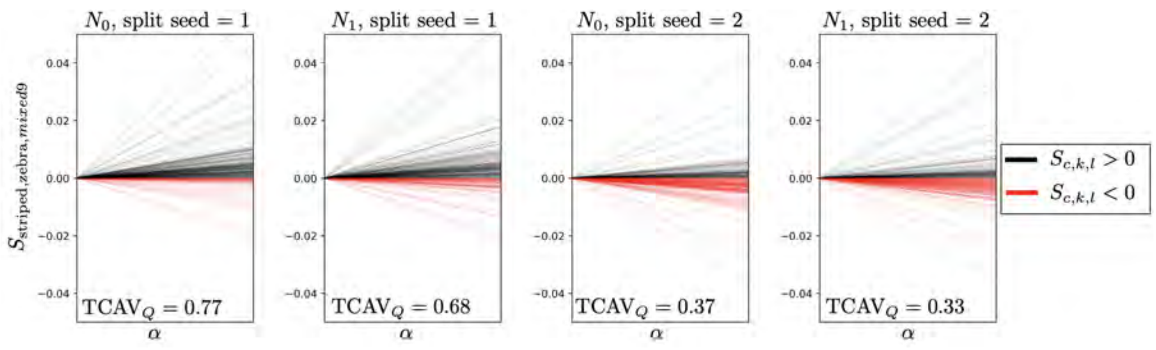


Figure 4.4: Four directional derivative distributions for different negative classes N, and train/test split seeds for the *striped* CAV in layer *mixed9* for the zebra class.

Considering these two variables, directional derivative distributions for one example were found to get a bearing on the effect of each. From fig. 4.4, it appears that varying the train/test split seed had a small effect on the TCAV score and directional derivative distribution. When N was varied, the TCAV score changed dramatically, as did the directional derivative distribution. This means that even with the P_c training examples held *constant*, the directional derivative distribution was wildly different, suggesting that the selection of N is the main source of variance in this case, which is explored more in §4.4.3.2.

The proposed methodology will be used for experiments in both reducing the variance in TCAV score §4.4 and observing the effects of perturbing CAVs §4.5

4.4 Reducing Variance in TCAV Score

4.4.1 CAV Generation

In this section, fabricated data is used to examine the properties of several linear classifiers, which will be used to generate CAVs. The implementation details for each CAV generation method considered are given in table A.1.

With a parameter-based model, training data determines the parameters and not all training examples have equal influence on the parameters. The importance of **outliers** and **influential observations** on several linear classifiers used to generate CAVs are considered in this section.

For a linear binary classification problem, the predicted class $\hat{y}$ is found by adding a bias term b to the dot product of the weights $\mathbf{w}$ the model features $\mathbf{x}$. The sign of the result indicates the predicted class: either -1 or 1. Classifiers are trained by solving for the optimal weights and bias.

$$\hat{y} = \operatorname{sign}\left(\mathbf{w}^\top \mathbf{x} + b\right) \tag{4.3}$$

A metric such as Cook's distance (Cook 1977) could be used for detecting influential observations. However, because a CAV is the ultimate result of the linear classification process, I chose to quantify the change in the direction of the CAV. Cosine similarity $[-1, 1]$ (eq. (3.8)) can be used to quantify the similarity in direction of two vectors.

Therefore, the most influential observation, i_{infl}, is the point that, when removed, maximally changes the cosine similarity between the perturbed and original CAV. By iteratively discarding a single data point i from a class of interest, the maximum possible perturbation can be found with eq. (4.4).

$$i_{\text{infl}} = \underset{i}{\operatorname{argmin}} \left[\text{cosine similarity}\left(\boldsymbol{v}_C^l, \boldsymbol{v}_{C_{(-i)}}^l\right)\right] \tag{4.4}$$

Support Vector Machines (SVM) can be used for linear classification tasks and work by finding a hyperplane that maximizes the margin between the hyperplane and closest data point of either class. SVMs are particularly susceptible to influential observations because the rely on only a few points to generate a decision boundary. These points form *support vectors*.

In this toy example, an SVM is trained to separate two normally distributed classes (red and blue) in $\mathbb{R}^2$ from fabricated data. All data shown in fig. 4.5 was used for training. The resulting CAV is significantly affected by removing the most influential observation i_{infl} of the red class as this point was used as a support vector. This effect is well known and modified SVMs including Weighted SVMs (Xulei Yang, Qing Song, and Cao 2005) were designed in response.

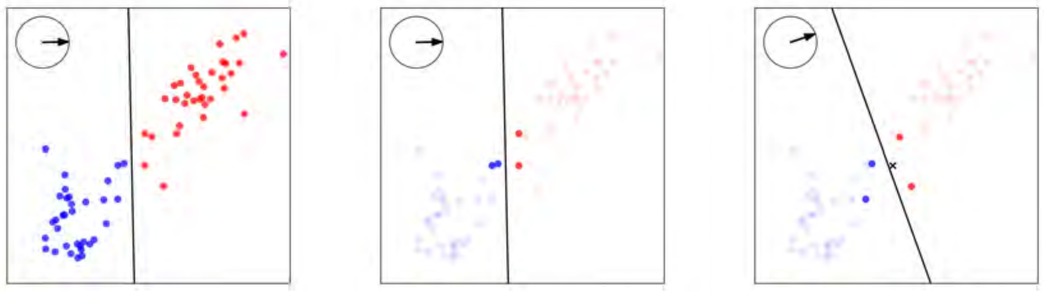


Figure 4.5: **SVM**: Original training data and decision boundary (left), support vector points highlighted (middle), and maximum perturbation by removing a point **x** from red class (right). Normalized CAVs shown in top left of each image.

The linear classifier originally proposed in TCAV (*TCAV Github repository* 2019) is an SVM trained using Stochastic Gradient Descent (SGD)[4]. A linear logistic classifier is suggested as an alternative to this. Because SGD is non-deterministic, different CAVs are produced from different random

[4]`scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html`

seeds. A random seed is used to control the generation of random variables. In this context, the random seed controls how SGD initializes and yields reproducible results for non-deterministic processes[5]. Surprisingly different CAVs can be produced using the *same* classifier and training data but different random seeds. This is because the SVM is approximated via a SGD optimization and runs under a default number of maximum iterations and tolerance for convergence. According to documentation[6], the SGD method should be considered for large datasets as the SVM method could have a significant computation cost. In practice, I found that training CAVs in high dimensions with a deterministic SVM implementation was not a computational bottleneck. However, increasing the maximum number of iteration using the SGD approach significantly increased training time.

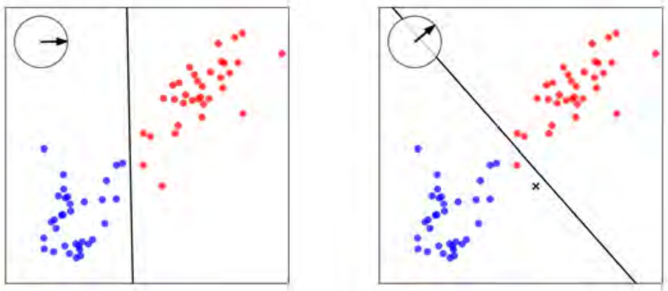


Figure 4.6: **SVM via SGD**: Original training data and decision boundary (left), maximum perturbation by removing a point $\mathbf{x}$ from red class (right). Normalized CAVs shown in top left of each image.

As with the deterministic SVM, SVM via SGD was highly influenced by the removal of i_{infl}. The variance in CAVs generated from the SGD process can be overcome by averaging over many random seeds, as shown in fig. 4.7.

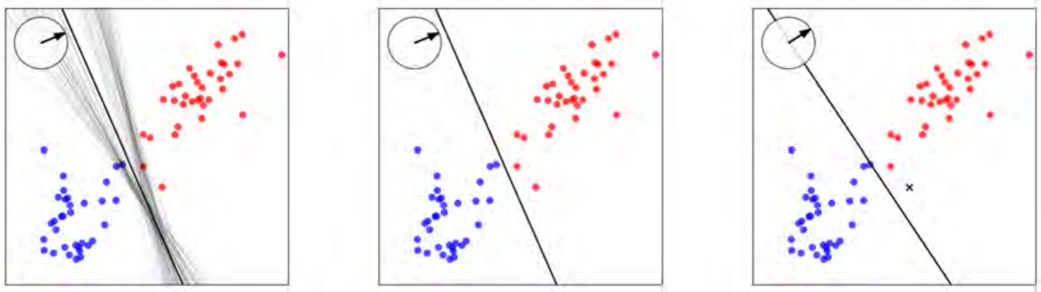


Figure 4.7: **Average over SGD SVMs**: Original training data and decision boundaries for 100 SGD SVMs and the average (left), training data and decision boundary for the average (middle), maximum perturbation by removing a point $\mathbf{x}$ from red class (right). Normalized CAVs shown in top left of each image.

[5]Random seeds can also be used to control a train/test split.
[6]`scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`

Many linear classifiers exist outside of SVMs. When considering the goal of this experiment is to make the TCAV process more **consistent**, a simple way to increase consistency in CAVs and is to select a linear classifier that is **deterministic in training**, or at least very consistent.

A group of concept examples given by an analyst *should* have something in common and the idea of a CAV is to learn this commonality and encode it in a vector. Two disjointed sets of concept examples could be used to express the same concept and there are bound to be outliers from each set. Ideally, a CAV learns the commonality amongst a group of concept images and therefore be **minimally affected by outliers**.

A linear classifier could fit these desiderata is a Closest Centroid Classifier. In this method, the the centroid of each class is computed and the difference between the centroids becomes **w**, which is also the CAV. For this example, the arithmetic mean was chosen as the centroid method, although any other centroid method could be used. Therefore, this method will be referred to as the **difference of means** method, or simply mean.

$$\mathbf{w} = \boldsymbol{\mu}^{(+)} - \boldsymbol{\mu}^{(-)} \tag{4.5}$$

$$b = -\frac{1}{2}\left(\left\|\boldsymbol{\mu}^{(+)}\right\|^2 - \left\|\boldsymbol{\mu}^{(-)}\right\|^2\right) \tag{4.6}$$

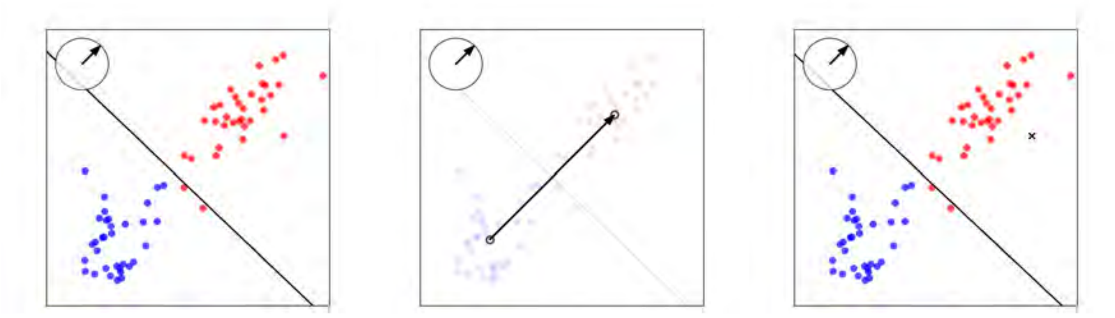


Figure 4.8: **Difference of Means**: Original training data and decision boundary (left), means and CAV highlighted (middle), and maximum perturbation by removing a point **x** from red class (right). Normalized CAVs shown in top left of each image.

The CAV generation process for the difference of means method is shown for the toy data in fig. 4.8. The CAV direction is learn directly, rather than by finding the vector normal to the decision boundary. The maximum perturbation has a much smaller effect on the CAV with this method. It should be noted that the perturbation metric in use here is the **cosine similarity** between the original and perturbed CAVs, and *not* the Euclidean distance between the original and perturbed means. From fig. 4.8, the most influential point is far in distance from the mean, but not the furthest; removing this point, however, does move the mean in a direction that maximally affects the CAV. Because the magnitude of the CAV is unimportant for cosine similarity, removing points that shift the mean in the direction of the CAV (either shrinking or elongating it, put simply) ultimately have little impact on the cosine similarity with the original CAV.

Using fabricated data, it appears that SVM approaches have downsides that the difference of means (or others) might be able to overcome. However, these toy experiments are in $\mathbb{R}^2$ with two mostly distinct classes. CAVs exists in a much higher dimensional space (table 3.1).

Next, experiments will focus on using the quantifying the effect of a more consistent classifier on actual concept and class pairings to understand if these advantages manifest in a real application.

4.4.2 Consistent Classifier

To assess the importance of a more consistent classifier, the original SVM method (trained via SGD) will be considered along with an average of 100 SVMs trained with SGD, as in fig. 4.6. The TCAV score was found for several class/concept pairings at different points in the network. Ideally, the TCAV score from the same class/concept pairing at the same bottleneck in the network would the same, even with a different negative class N. In reality, there is some variance in TCAV scores computed under very similar conditions. The goal of this experiment is to determine how much this variance can be decreased by simply making the CAV generation process more **consistent**.

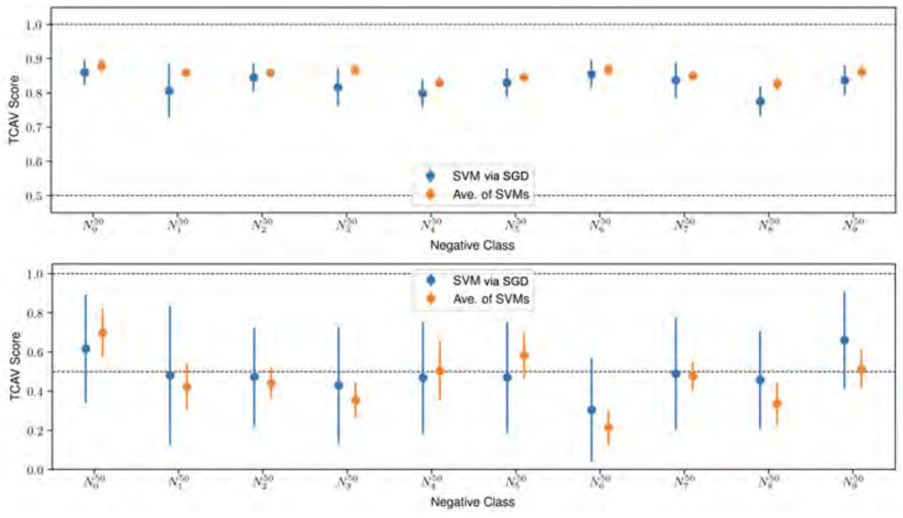


Figure 4.9: TCAV score distributions using the *striped* CAV for the *zebra* class in the *mixed6* (top) and *mixed9* (bottom) layers. Error bars represent one standard deviation.

Ten negative classes each containing 50 random images were used to test the TCAV score distribution $N_{50}^0, \ldots, N_{50}^9$. Ten random seeds $s = 1, \ldots, 10$ for the train/test split were used to control which examples were actually used for training. The overall statistics are summarized in table 4.1. Additionally, the TCAV score distribution was plotted for each negative class in fig. 4.9.

Beyond an inconsistent CAV training process, there are other sources of variance that may be propagating to variance in the TCAV score distributions.

			SVM via SGD		Ave. of SVMs	
layer	**class**	**concept**	μ	σ	μ	σ
mixed6	zebra	striped	0.83	0.054	0.85	0.021
mixed9	zebra	striped	0.48	0.284	0.45	0.166

Table 4.1: TCAV score distribution statistics for two methods of CAV generation. Each distribution has 100 observations from the combination of 10 train/test split seeds and 10 N_{50} classes.

4.4.3 Negative Class

From §4.3, preliminary experiments suggested that under constant conditions, a change in N led to variance in the TCAV score and directional derivatives. This section aims to give further experimental evidence and explore methods to mitigate this effect.

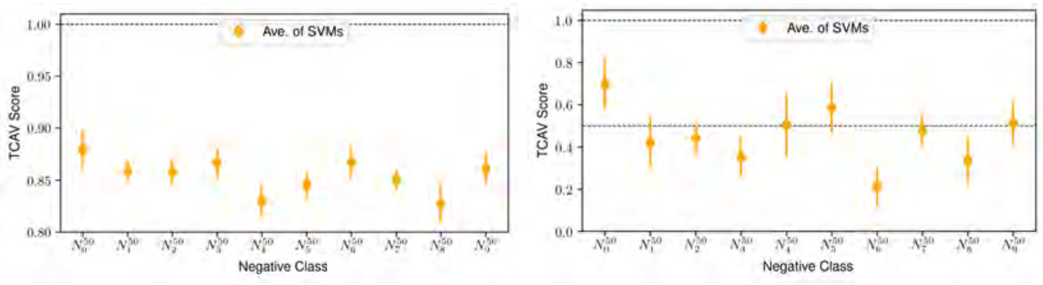


Figure 4.10: TCAV score distributions using the *striped* CAV for the *zebra* class in the *mixed6* (left) and *mixed9* (right) layers. Error bars represent one standard deviation.

In fig. 4.10, the TCAV score distribution is relatively narrow if considering a single negative class, such as N_0^{50}, which represents the same P_c and N examples, but different training examples used as the train/test split seed was varied from $s = 1, \ldots, 10$. However, the distributions of the TCAV score for different negative classes look quite different and seems to be drawn from a different distribution. Ideally, the replacement of one negative class with another should have little impact on the TCAV score.

Now that the importance of selection of N has been shown experimentally, **potential solutions** will be explored; first, using the difference of means method, then using a larger size for the negative class.

4.4.3.1 Difference of Means

The difference of means method (eq. (4.5)) should have less dependence on outliers than an SVMs based on preliminary experiments with fabricated data[7] in §4.4.1. Because N is the set of the negative examples used to train a CAV and comes from a distribution of random images, outliers are inevitable. I hypothesize that using the difference of means method will decrease variance in the TCAV score distribution.

[7]Of centroid method, the mean is more dependent on outliers than the median, for example. However, this is a comparison with the SVM methods.

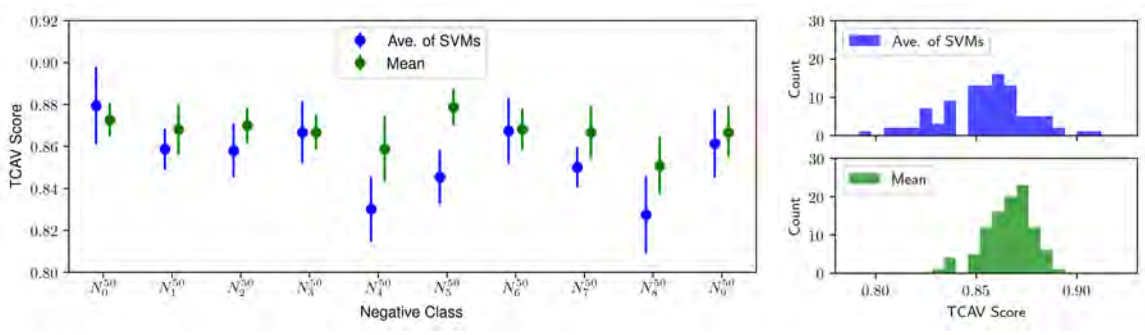


Figure 4.11: TCAV Score distribution against N for the *striped* CAV in layer *mixed6* and *zebra* class using two CAV generation methods over train/test split seeds $s = 1, \dots, 10$ for each N and $|N| = 50$ (left). Overall distribution (right).

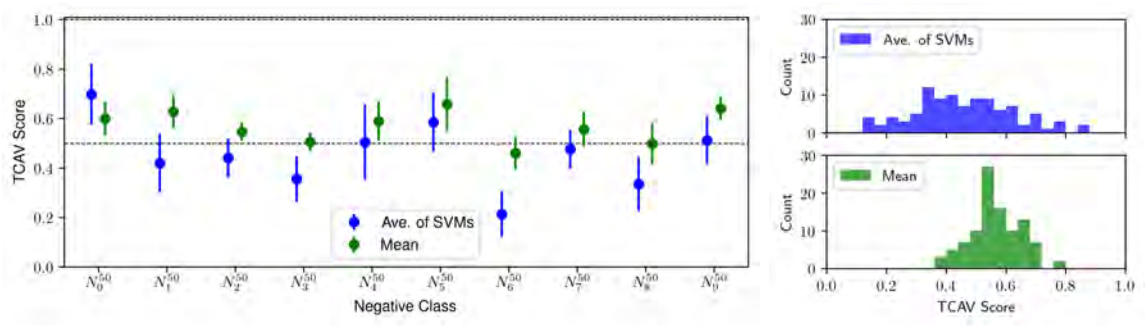


Figure 4.12: TCAV Score distribution against N for the *striped* CAV in layer *mixed9* and *zebra* class using two CAV generation methods over train/test split seeds $s = 1, \dots, 10$ for each N and $|N| = 50$ (left). Overall distribution (right).

fig. 4.11 and fig. 4.12 agree with this hypothesis. The TCAV score is more consistent across different N classes and the overall variance in TCAV score significantly decreases.

			Ave. of SVMs		**Mean**	
layer	**class**	**concept**	μ	σ	μ	σ
mixed6	zebra	striped	0.85	0.021	0.87	0.012
mixed9	zebra	striped	0.45	0.166	0.56	0.085

Table 4.2: TCAV score distribution statistics for two methods of CAV generation. Each distribution has 100 observations from the combination of 10 train/test split seeds and 10 N^{50} classes.

4.4.3.2 Increasing Size of Negative Class

Ideally, N would sufficiently represent the distribution of all possible random images. In reality, the size of N is finite and typically limited to the size of P_c [8]. If $|P_c| = |N|$, then the classes will be balanced, which is ideal for a binary classification with equal penalties. If the hypothesis that

[8] the TCAV github repo (*TCAV Github repository* 2019) truncates all examples to the size of the smallest concept to make sure positive and negative examples are balanced

$|N| = 50$ does not sufficiently cover some distribution of all images is correct, perhaps including more elements, such as $|N| = 500$ could solve this issue. An advantage of TCAV is giving an analyst the ability to define a concept of their choosing, which would likely involve either searching and downloading images, using one of *many* Synsets[9] in ImageNet, or a more manual selection process. Therefore, $|P_c|$ should be kept to a reasonable size to prevent the concept collection process from being overly cumbersome.

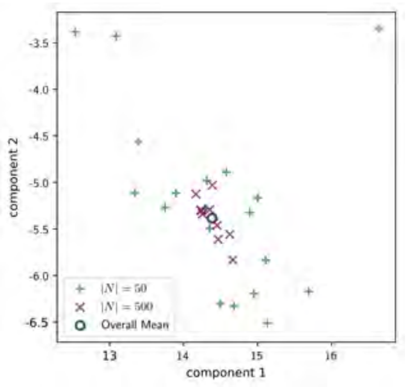


Figure 4.13: PCA decomposition in *mixed9* of means of N with either 50 or 500 elements. PCA embedding from fig. 3.1 is used.

If $|P_c| \neq |N|$, then class weights could be used to avoid the classifier being biased toward classifying everything in the class with the most training points. Scikit-learn, which is used for all linear classifiers in this project, has a setting that automatically computes class weights based on the number of items in each class [10]. Alternatively, some classifiers, such as the difference of means method do not require class weights as the CAV is learned directly rather than through a linear classifier.

To confirm that $|N| = 500$ results in more consistency across negative classes, the mean of the activations was computed for each negative class $N_0 \dots N_9$ for both $|N| = 50$ and $|N| = 500$. From fig. 4.13, when $|N| = 500$, the means are more clustered around the overall mean. Because fig. 4.13 is a decomposition into $\mathbb{R}^2$, this observation was confirmed by finding the Euclidean distance from the mean over the entire set. N where $|N| = 500$ is consistently much closer to the mean over the entire set as expected.

Using $|N| = 500$ in fig. 4.14, the TCAV score was computed for the CAV generated from P_{striped} and $N_0 \dots N_9$. For each of these CAVs, random seeds $s = 1, \dots, 10$ were used.

From fig. 4.14, the TCAV score distribution is narrowed even further by using $|N| = 500$ with the difference of means (mean) method. Is this also the case when the variance is already low? Using the *mixed6* layer for P_{striped} and $k =$ zebra, the variance in TCAV score distribution is already relatively low by using in difference of means method (fig. 4.11). From fig. 4.15, it seems as if there is some irreducible variance as increasing $|N|$ had only a small decrease in the variance of TCAV

[9] A "synonym set" is set of synonyms that express the same concept, like *cab* and *taxi*. ImageNet uses these to group images of the same concept.

[10] `scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html`

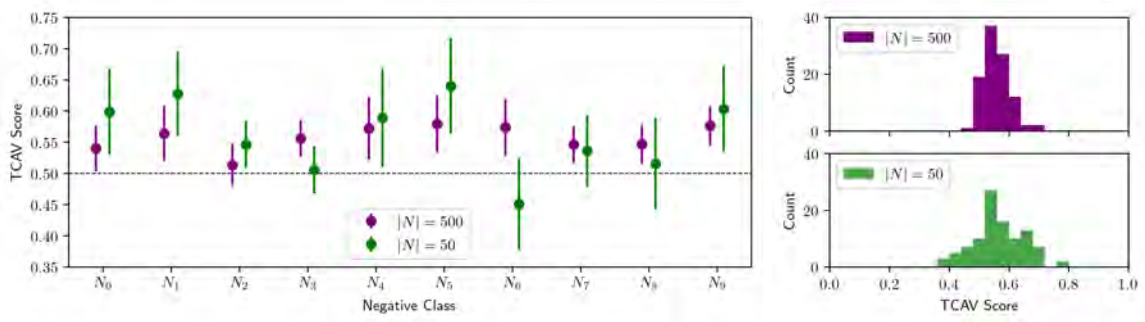


Figure 4.14: TCAV Score distribution against N for the Striped CAV in layer *mixed9* and *zebra* class using $|N| = 50$ and $|N| = 500$ and the difference of means method over train/test split seeds $s = 1, \dots, 10$ (left). Error bars represent one standard deviation. Overall distributions (right).

score distribution.

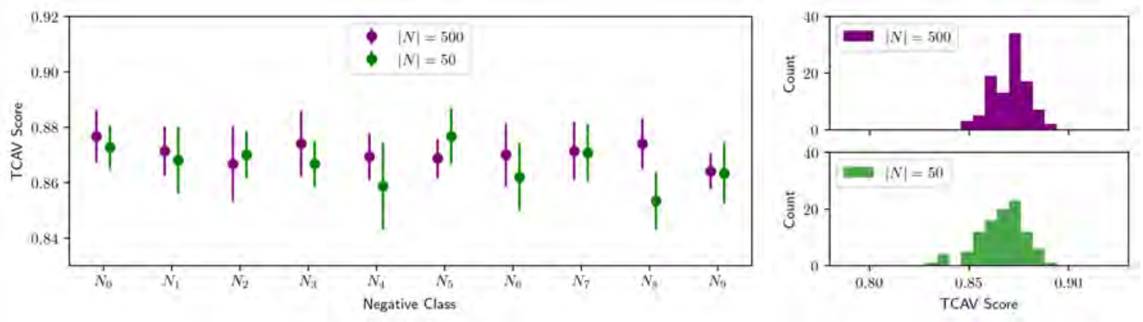


Figure 4.15: TCAV Score distribution against N for the Striped CAV in layer *mixed6* and *zebra* class using $|N| = 50$ and $|N| = 500$ and the difference of means method over train/test split seeds $s = 1, \dots, 10$ (left). Error bars represent one standard deviation. Overall distributions (right).

			Mean, $\lvert N\rvert = 50$		**Mean**, $\lvert N\rvert = 500$	
layer	**class**	**concept**	μ	σ	μ	σ
mixed6	zebra	striped	0.87	0.012	0.87	0.010
mixed9	zebra	striped	0.56	0.085	0.56	0.042

Table 4.3: TCAV score distribution statistics for two methods of CAV generation. Each distribution has 100 observations from the combination of 10 train/test split seeds and 10 N^{50} classes.

More comprehensive results are given in table A.2 and generally corroborate the conclusions that 1) A deterministic linear classifier reduces variance in TCAV score 2) The difference of means method, particularly using $|N| = 500$, results in the lowest variance in TCAV score.

4.5 Perturbing CAVs and TCAV Score

In §4.4.1 a toy dataset was used to examine fundamental differences in linear classifiers and how their resulting CAVs respond to removing data from one class. So far, experiments have shown that using difference of mean method to generate CAVs and using a larger N set decreases variance in TCAV score distributions, thereby making the CAV and TCAV process more **consistent**. The next set of experiments aim to evaluate several linear classifiers to determine which is the most **resistant to perturbations**.

4.5.1 Removing a Single Training Point

In this context, a perturbation is the addition or removal of a data point in P_c or N. As in §4.4.1, a maximum perturbation is achieved by finding the highest change in CAV by removing a single data point i.

$$i_{\text{infl}} = \underset{i}{\text{argmin}}[\text{cosine sim}(\boldsymbol{v}_C^l, \boldsymbol{v}_{C_{(-i)}}^l)] \tag{4.7}$$

Alternatively, the change in the magnitude of TCAV score (a scalar) could be considered as the metric of interest rather than the CAV score[11].

$$i_{\text{infl}} = \underset{i}{\text{argmax}}\, |\text{TCAV}_{\text{Q}_{C,k,l}} - \text{TCAV}_{\text{Q}_{C(-i),k,l}}| \tag{4.8}$$

Rather than selecting a single perturbation metric (eq. (4.7) or eq. (4.8)), both were collected for the removal of every training point in P_c, one at a time. Using $|P_c| = 50$, $|N| = 50$, and train/test split$= 0.33$, results in 34 training points for P_c and 33 for N. An *original* CAV was trained using all points to serve as a baseline [12]. Next, the first training point in P_c was removed, leaving 33 training points in P_c. The process was repeated for each training point, resulting in 34 perturbed CAV and TCAV score pairings for each method.

In fig. 4.16, the TCAV score from the SVM method is, not surprisingly, influenced by removing training points. The difference of means method performed the best in the sense that the perturbed CAVs varied very little in both cosine similarity to the original CAV, and more importantly, TCAV score. Although some of the perturbed points for the difference of means method appear to have a cosine similarity of 1, they are merely close (most $\approx .999$). Because each point is used in calculating the mean, removing *any* single point changes the mean.

In contrast, a CAV generated by an SVM is not necessarily influenced by the removal of any point; only removing points that act as support vectors will alter the CAV. Of 34 training points in P_c, 20 were used as support vectors, so the cosine similarity of the CAVs perturbed by removing the non-support vector points is 1, meaning the CAV is the same. These points are visible in fig. 4.16.

[11] Using eq. (4.8) for perturbation comes at a significant computation cost. From eq. (2.3) and eq. (2.2) the activations and gradients must be computed for each input image $\boldsymbol{x} \in X_k$ to compute the TCAV score. Computing a new TCAV score for each training set with one element removed becomes computationally expensive.

[12] An original CAV was trained for each method, so the cosine similarities are with a CAV generated through the same method

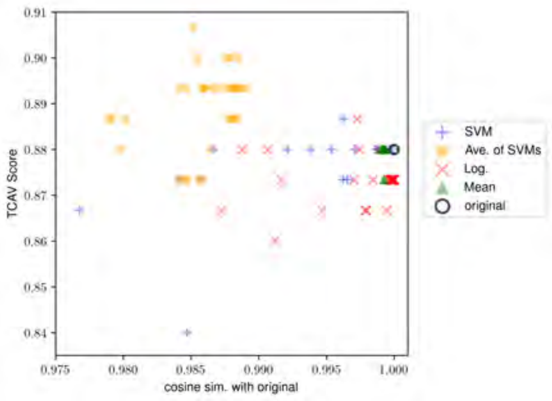


Figure 4.16: TCAV Score vs cosine similarity for four CAV generation methods. Each point represents the CAV and TCAV score that resulted from removing one training point in P_c. This is the *striped* concept in layer *mixed6* and tested for 150 zebra images.

The logistic classifier also appears to be less influenced by perturbations than both SVM methods. How many training examples can be removed with similar results? The answer is likely very context-dependent. Further experiments were conducted focusing on the **logistic** and **difference of means** methods as those proved to be the most effective in the above experiment.

4.5.2 Removing n Training Points

Rather than removing each training point, one at a time, random groups of training points in P_c of size n were discarded.

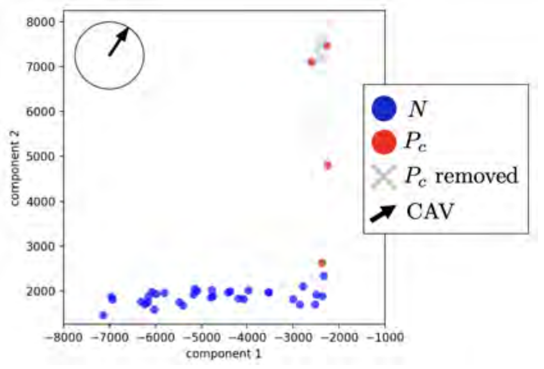


Figure 4.17: Example of removing $n = 30$ training points from P_c, shown in a sparse PCA embedding. This is the *striped* concept in *mixed9* and the CAV was generated using the difference of means method.

For example, for $n = 10$, a random subset of size 10 was selected out of a constant training set of P_c (size 34). A CAV was generated from the remaining 24 training examples and the cosine similarity TCAV score were recorded for this case. Then, this process was repeated as 50 times for fig. 4.18. A visualization of this process is given in fig. 4.17

Using the both the difference of means and logistic methods to generate CAVs results in remarkably consistent CAVs and TCAV scores. Even when 30 training examples were removed (leaving only 4 training examples in P_c)[13], the TCAV score remained in a relatively narrow distribution for all 50 trials. As expected, removing fewer training examples ($n = 10, 20$) results in CAVs that are more similar to the original CAV and less variance in the TCAV score.

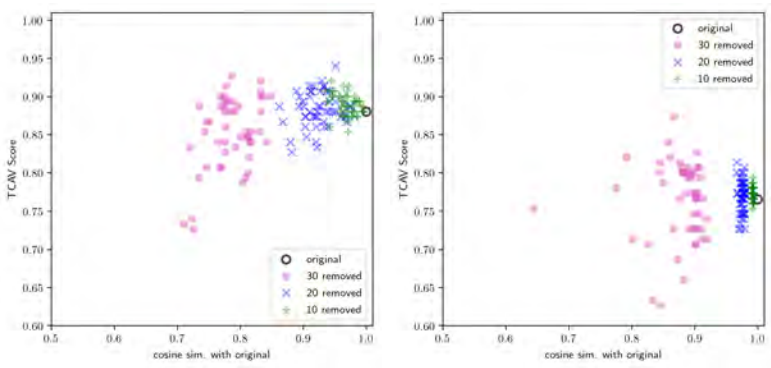


Figure 4.18: TCAV Score vs cosine similarity for the logistic method (left) and difference of means method (right). Each point represents the CAV and TCAV score that resulted from removing n training points in P_c randomly for $n = 10, 20, 30$. This is the *striped* concept in layer *mixed8* and tested for 150 zebra images.

The same process was repeated for a different class/concept pairing in the same layer (*mixed8)* in fig. 4.19. Similar results are observed, although the minimum cosine similarity is lower (≈ 0.6 compared to ≈ 0.75 in fig. 4.18) and the variance in TCAV score is generally higher for all values of n and both methods in fig. 4.19. I hypothesize that this is because the *grassland* concept is less homogeneous than the *striped* concept. In other words, two random images in the *grassland* concept are likely further away than two random images in the *striped* concept because textures are simpler than scenes. Thus, the CAV would be more susceptible to change with a less homogeneous concept.

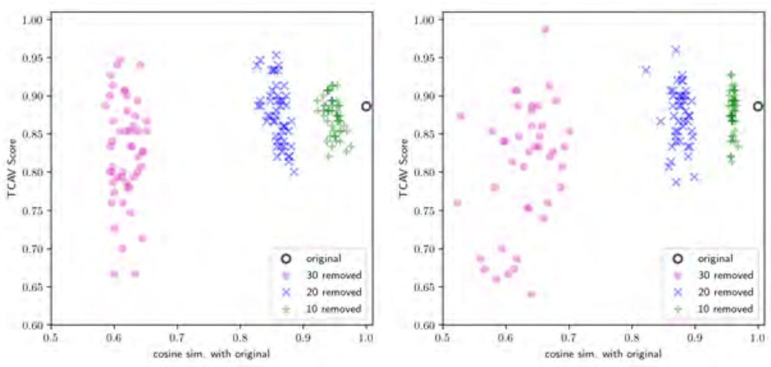


Figure 4.19: TCAV Score vs cosine similarity for the logistic method (left) and difference of means method (right). Each point represents the CAV and TCAV score that resulted from removing n training points in P_c randomly for $n = 10, 20, 30$. This is the *grassland* concept in layer *mixed8* and tested for 150 lion images.

[13]This is not meant to suggest analysts should use $|P_c| = 4$; This is a test of the limits of each method.

In the *mixed8* layer, the average Euclidean distance to μ_{P_c} was 75.3 for *striped*, 98.8 for *grassland*, and 107.5 for *a random set*. Based on the average distance to the mean, it seems that the *striped* activations are more clustered in the *mixed8* layer and plausible that this contributed to the behavior of the perturbed CAVs. However, other sources of the observed differences are also possible.

4.5.3 Adding n Noisy Training Points

Rather than perturbing a CAV by removing training points, the next experiment will explore the behavior of CAVs and the TCAV score when **n** training points are added to P_c. In this case, the training points to be added do *not* contain concept c. Rather, they are drawn randomly from a large set of random images, N. The purpose of this experiment is to assess how careful an analyst must be when gathering concept images and generally test the limits of the broadness of a concept. If a few noisy examples are included in P_c, will this affect important metrics, like the TCAV score?

A metric, β is used to keep track of the perturbation size with respect to the training set size.

$$\beta = \frac{n_{\text{noisy}}}{n_{\text{train}}} \tag{4.9}$$

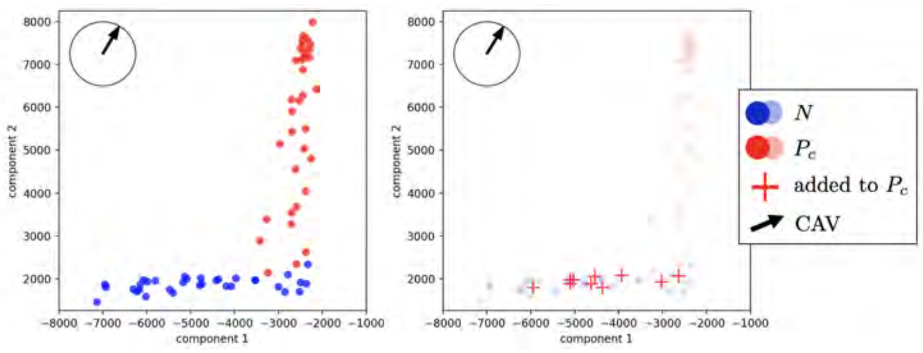


Figure 4.20: Original training data and CAV (left). Perturbed training data by adding 10 noisy training examples (right). Both shown in a sparse PCA embedding. This is the *striped* concept in *mixed9*.

Four methods were considered. For each trial, $n = 10$ examples from N were randomly selected and concatenated to the training set of P_c. A CAV is generated using each of the four methods and the resulting TCAV score was found using 150 zebra images. The perturbation process is visualized in fig. 4.20. This process was repeated 25 times to generated fig. 4.21.

From fig. 4.21, the difference of means method of CAV generation results in a TCAV score distribution with the lowest variance and perturbed CAVs that are closest to the original CAV (of the same method) based on cosine similarity. At $n = 10$ and more pronounce at $n = 50$, the TCAV scores generally decreases compared to the original and approach the baseline $\text{TCAV}_Q = 0.51$ from random CAVs. This likely indicates that the concept is less defined now and the CAV encodes less meaning of the original concept as noisy examples are added.

The next experiment sought to understand why the difference of means method generates such stable CAVs and TCAV scores in comparison to other methods when noisy training examples are introduced. From fig. 4.22, the CAVs generated from the difference of means method remain remarkably similar as more noisy training examples are introduced. As n increases, μ_{P_c} is dragged closer to μ_N. However, it seems that this change mainly affects the magnitude instead of the direction of

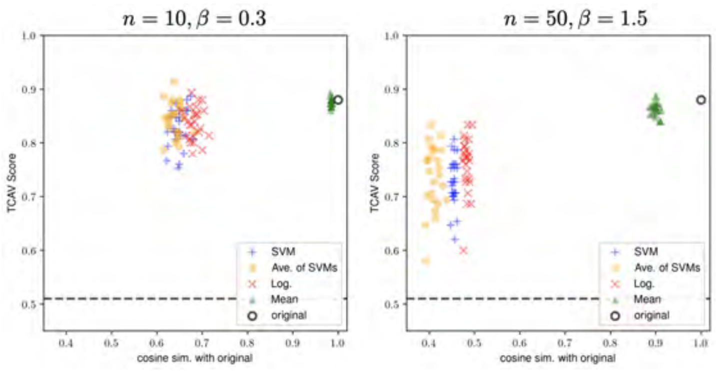


Figure 4.21: TCAV Score vs cosine similarity using 4 CAV generation methods. Each point represents the CAV and TCAV score that resulted from adding $n = 10$ (left) and $n = 50$ (right) noisy training points to P_c, shown on the same scale. This is the *striped* concept in layer *mixed6* and tested for 150 zebra images. The dashed line represents the baseline TCAV_Q from random CAVs (table A.3).

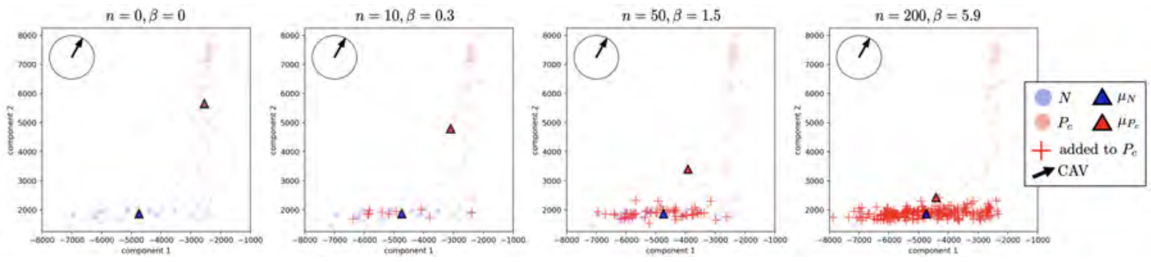


Figure 4.22: Training data, CAVs, and class means with P_c perturbed by n noisy training examples and shown in a sparse PCA embedding. This is the *striped* concept in layer *mixed9* and the difference of means method is used to generated CAVs.

the CAV, at least as it appears in the sparse PCA decomposition into $\mathbb{R}^2$. Considering fig. 4.21 at $n = 50$, the CAVs generated from the difference of means method still remain close to the original TCAV score and have a relatively high cosine similarity, giving evidence to the observation in $\mathbb{R}^2$.

The **difference of means** method proved to be the most effective CAV generation method considered in **resistance to perturbations** and a possible explanation was presented.

Chapter 5

Exploring TCAV Modifications

In this chapter, two possible modifications to the TCAV process are proposed based on observations from experiments.

5.1 Thresholding TCAV Score

In the originally proposed TCAV score (eq. (2.3)), the **sign** each directional derivative is used to compute a fraction. This modification considers using the **magnitude** each directional derivative.

Some TCAV scores, especially in the *mixed10* layers had high variance or increased dramatically from a low TCAV score in *mixed9*, shown in table A.2. Additionally, the uncertainty in TCAV scores for random CAVs appears to increase through the network and is especially high in *mixed10* (see table A.3). This experiment considers **thresholding** as a potential solution to this issue.

In fig. 5.1, substituting zebra images with noise for X_k in computing directional derivatives results in the *same* TCAV_Q while the directional derivative distribution over 150 images is very different. A similar observation is made when P_c is changed. In fig. 5.2, the using P_{dotted} to generate a CAV resulted in the *same* TCAV_Q as using P_{striped}.

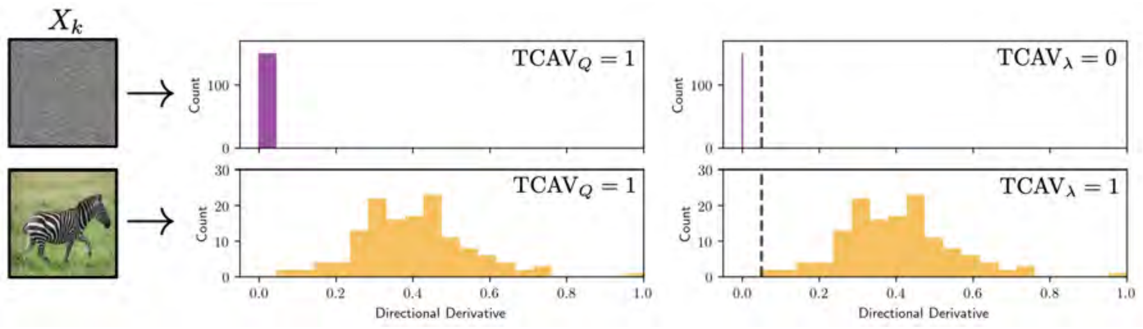


Figure 5.1: Comparison of directional derivatives using with 150 noise (top) or zebra (bottom) images for X_k and a threshold of $\lambda = 0$ (middle) or $\lambda = 0.05$ (right). The *striped* concept in layer *mixed10* is used and the difference of means method is used to generate the CAV.

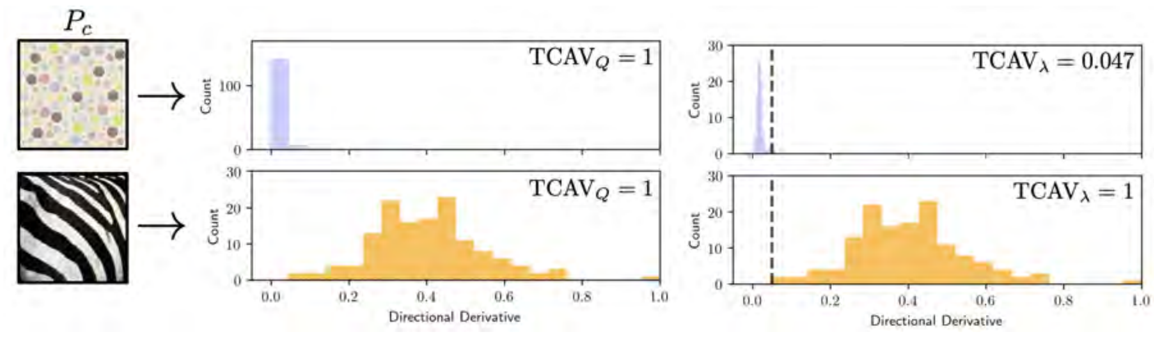


Figure 5.2: Comparison of directional derivatives and TCAV score using P_{dotted} (top) and P_{striped} (bottom) and a threshold of $\lambda = 0$ (middle) or $\lambda = 0.05$ (right). 150 zebra images are used for X_k. Layer *mixed10* is used and the difference of means method is used to generate the CAV.

Therefore, an alternative TCAV score, TCAV_λ is proposed. λ is a threshold, used to filter out directional derivatives that are only just above zero.

$$\mathrm{TCAV}_{\lambda_{C,k,l}} = \frac{|\{\boldsymbol{x} \in X_k : S_{C,k,l}(\boldsymbol{x}) > \lambda\}|}{|X_k|} \tag{5.1}$$

Using $\lambda = 0.05$ resulted in filtering out nearly all noise and resulted in no loss in signal for this example.

In thinking of a more principled method to compute λ, random CAVs where considered, which are trained using two N classes to create a meaningless CAV. A distribution of directional derivatives can be found for a random CAV and class k. I collected 20 of these distributions to create fig. 5.3, which should be a good approximation of the inherent bias from the class k as the CAVs are random. Using the mean μ_{rand} and standard deviation σ_{rand} of this concatenated distribution, eq. (5.2) is one way λ could be computed.

$$\lambda = \mu_{\mathrm{rand}} + 2 * \sigma_{\mathrm{rand}} \tag{5.2}$$

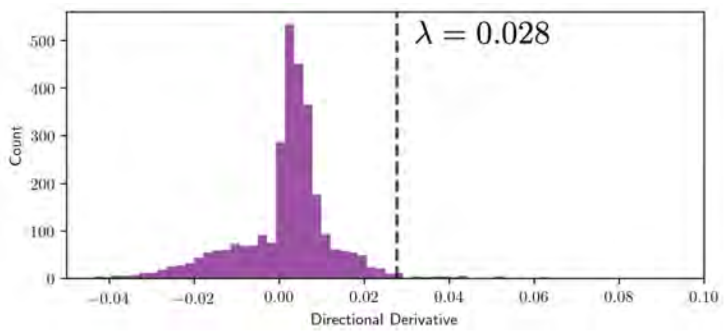


Figure 5.3: Histogram of directional derivatives from random CAVs. The 3000 observations are from 20 random CAVs and 150 zebra images. Layer *mixed10* is used and the difference of means method is used to generate CAVs.

Thresholding can prevent noise from being included in TCAV$_\lambda$ and one method for systematically computing λ is given.

5.2 Retraining CAVs

CAVs are meant to encode a meaningful direction in the activations of a layer in a neural network. The decision boundary of the linear classifier associated with a CAV can be decomposed and plotted along with the training and testing data that produced it, as in fig. 5.4. The next step in the TCAV process is using examples from the class of interest to find directional derivatives to compute a TCAV score.

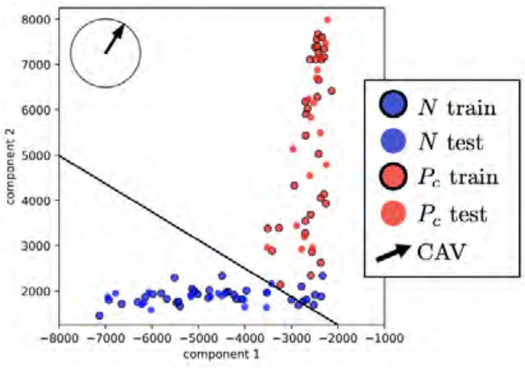


Figure 5.4: Training and testing data for a striped CAV (from SVM) in layer *mixed9* decomposed using sparse PCA.

In fig. 5.5, the training and testing data is still present but the embeddings and predictions $f_l(x)$ $\forall x \in X_k$ are highlighted and colored according to their predicted class (either P_c or N). These activations span a wide area in the embedding and 44 of 150 class images are classified as P_{striped}. Proximity of a point to the training set for each class and its connection to the semantics of each class is explored in fig. 5.5.

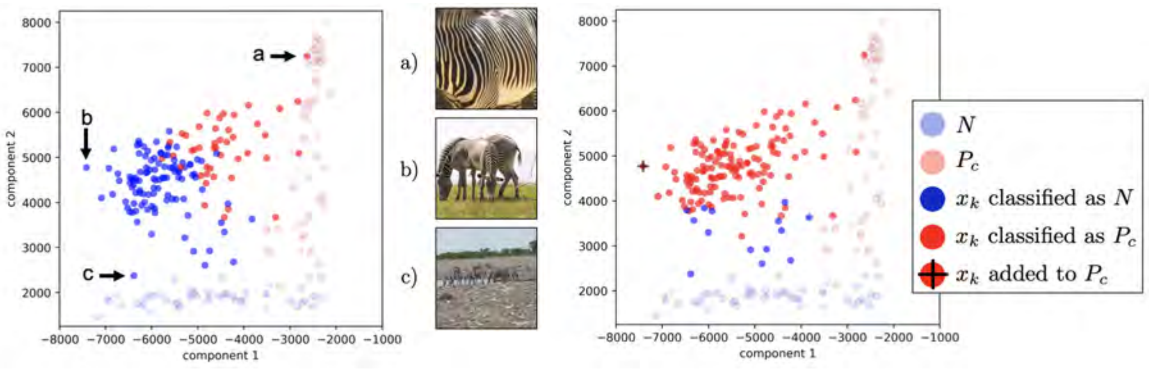


Figure 5.5: Zebra class images classified as either P_{striped} or N according to the SVM classifier (left). Some example class images at different points in the embedding (middle). Classification results after including point b in the training set for the CAV (right).

For example, point a, is very close to a cluster of training points from P_c and its corresponding original image is a clearly articulated striped texture. Point b is far from either set of training points and is classified as N. The corresponding image *does* exhibit the striped concept although it is only at the center of the image. Finally, c is close to N and the striped texture barely visible.

Should point b actually be classified as belonging to P_c? The CAV is computed starting with human-defined examples of a given concept. However, it is clear now that the linear classifier and therefore CAV have a narrower understanding of the striped concept than anticipated. When examining the images used in P_c, many of them contain a striped texture occupying the entire image (see fig. 3.10). I hypothesize that the linear classifier has therefore learned this definition of the concept and has not learned to generalize to include examples in which the concept is not the entirety of the image.

Based on this new insight into the CAV, I retrained the SVM, including point b in the training set.As mentioned in §4.4.1, SVMs are particularly influenced by outliers. Point b is an outlier with respect to the rest of the P_c training set. Therefore, including it in the training set has a significant impact on the resulting linear classifier and CAV, as shown in fig. 5.6. Now, 136 of 150 zebra class images are classified as belonging to P_c, a massive increase from the original classification. The shift also increased the TCAV score from 0.54 to 0.61.

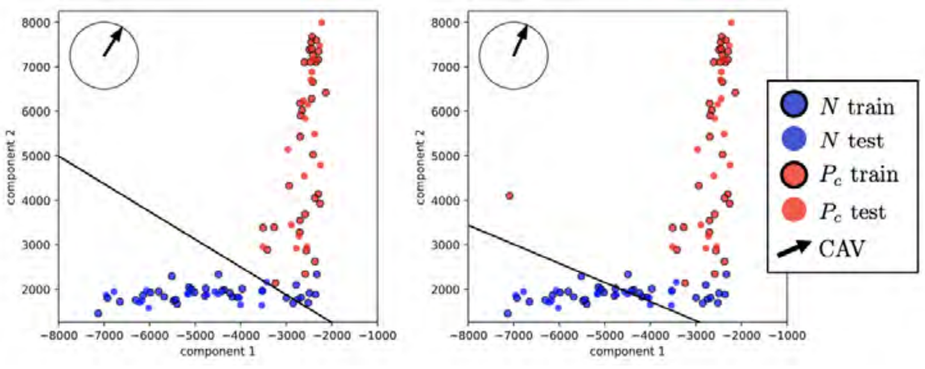


Figure 5.6: Original (left) and one example added (right) training and test data, decision boundary, and CAV (from SVM)

The retraining process described could be used to fine-tune CAVs. Consider a scenario in which an analyst has access to a large number of concept images, like through an online database. While these concepts might be close to the true concept desired by the analyst, they may not be perfect. Adding training example(s) to P_c and using a linear classifier that is highly influenced by outliers can significantly alter the CAV in the intended direction and could be useful for an analyst.

Chapter 6

Discussion

6.1 TCAV Best Practices

Keeping in mind the objectives of *Robust TCAV* are 1) **consistency** in TCAV scores for nearly identical scenarios and 2) **Resistance to perturbations** in the concept class, the best practices and key experimental results for practitioners are summarized below.

- A consistent classifier §4.4.2 can significantly reduce variance TCAV score
- The difference of means method eq. (4.5) proved to be the highest performing CAV generation method for both objectives above, especially when used with a larger negative class ($|N| = 500$)
- The difference of means method performed well in all perturbation experiments. The logistic classifier has comparable results in perturbation by removing n training examples
- As few as 4 training points in P_c are needed to generate a meaningful concept[1], as demonstrated in perturbation by removing n training examples
- Considering the perturbation by adding n noisy training examples, most methods can tolerate several noisy examples additions with relatively low loss in TCAV score
- The difference of means method can tolerate a surprisingly high number of noisy training examples with relatively low change in TCAV score and CAV similarity
- The spread of P_c can affect how resulting CAVs behave when points in P_c are removed
- Thresholding can be used to eliminate an inherent bias in the TCAV score for a given class
- Classifiers that are highly influenced by outliers, like SVMs, can easily be retrained by adding specific training examples

These conclusions are from the Inception v3 architecture pre-trained for ImageNet and primarily uses ImageNet for concept images. Thus, these findings **may not be the same** with other architectures, datasets, and domains. These best practices are meant to be a **general guide** and useful observations about the TCAV process rather than a absolute rules about its behavior.

[1]this is not a suggestion for practitioners, merely an observation

6.2 Perspective on TCAV Score

Kim et al. 2017 do statistical significance testing against the TCAV scores from random CAVs, the scores of which typically have a mean around 0.5, as in table A.3. This seems to imply that a TCAV score of 0.5 is the equivalent of a meaningless CAV and/or that the concept of interest is not important to the given class.

Using the thresholding method presenting in §5.1, a network-specific threshold λ could be implemented so that TCAV_λ only counts examples in which a class prediction was *significantly* sensitive to a concept. The threshold could be set to prevent meaningless CAVs from having $\text{TCAV}_\lambda > 0.1$ and I described a process using random CAVs to automatically set λ achieve this. With this altered score, essentially all noise is filtered out, so $\text{TCAV}_\lambda = 0.3$ would be significant whereas $\text{TCAV}_Q = 0.3$ (threshold of 0) is probably not significant.

6.3 Future Work

Ideally, prediction algorithms would be free of bias and incapable of discrimination. One direction of future work could be using the TCAV score in the **training process**. For example, with a human-defined concept like a particular race, the algorithm could be trained under the constraint that the TCAV score for a classification and this race concept is below a certain threshold.

In the TCAV algorithm, CAVs exists in bottleneck layers of a neural network, so TCAV scores are computed for each layer. While this is helpful for understanding where in the network concepts are most recognized, having multiple CAVs and TCAV scores is not immediate helpful to reaching a conclusion about whether a class is sensitive to a concept. Kim et al. imply that if the TCAV score is statistically different from that of a random CAV for *any* bottleneck layer, then it is reasonable to conclude that the class *is* sensitive to the concept of interest. Of course, the degree of sensitivity depends on the magnitude of the TCAV score. It could be possible to **concatenate** the activations from all layers to train a single CAV and have a **single TCAV score** from this, which is related to the hypothesis that semantic representations are distributed (R. Fong and Vedaldi 2018). While this would give a simple answer, one disadvantage is that some high-level concepts (like road in fig. 3.11) are not clearly articulated until the last few layers. Therefore, this approach might drown out signal if the CAV is not well-defined throughout the network.

Given the success in the difference of means method, **other centroid methods** could be considered to directly learn CAVs. For example, the median, or using the L1 norm could result in CAVs with different, and possibly better, properties.

Future work could more directly quantify the **number of examples** needed to convey meaning in a CAV. In a medical application, ≈20 concept examples were needed to achieve a high cosine similarity amongst CAVs of the same concept (Cai et al. 2019). In this project, as few a 4 concept examples were used to generate meaningful CAVs §4.5.2.

Finally, a VAE could be used to map the activations of a concept or class to a low dimensional latent space to inspect what has been learned at that point in the network.

6.4 Conclusion

This project explored the TCAV algorithm, which uses user-defined concepts to give global explanations for DNNs using intermediate activations. TCAV is a significant step in the field of interpretable machine learning and one motivation for this project was to accelerate the adoption of TCAV by practitioners by recommending practices that work.

Methods for CAV generation were explored, starting with suggestions from the original implementation. The idea of *Robust TCAV* was introduced, which focuses on 1) reducing variance in the TCAV score distribution and 2) Increasing CAV and TCAV score resistance to perturbations in P_c. Through experimentation, a difference of means method for CAV generation was found to be the best overall approach towards *Robust TCAV*. Finally, a few modifications to the TCAV process are considered, including thresholding to reduce noise in the TCAV score.

This project is a step towards a more reliable and useful TCAV process and hopefully the best practices will be helpful for practitioners interpreting their models and for further work in this exciting area.

Bibliography

Adebayo, Julius et al. (2018). *Sanity Checks for Saliency Maps.* arXiv: `1810.03292 [cs.CV]`.

Alain, Guillaume and Yoshua Bengio (2016). "Understanding intermediate layers using linear classifier probes". In: *arXiv preprint arXiv:1610.01644*.

Angelino, Elaine et al. (2017). "Learning certifiably optimal rule lists for categorical data". In: *The Journal of Machine Learning Research* 18.1, pp. 8753–8830.

Cai, Carrie J. et al. (2019). "Human-Centered Tools for Coping with Imperfect Algorithms During Medical Decision-Making". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems.* CHI '19. Glasgow, Scotland Uk: ACM, 4:1–4:14. ISBN: 978-1-4503-5970-2. DOI: `10.1145/3290605.3300234`. URL: `http://doi.acm.org/10.1145/3290605.3300234`.

Caliskan, Aylin, Joanna J Bryson, and Arvind Narayanan (2017). "Semantics derived automatically from language corpora contain human-like biases". In: *Science* 356.6334, pp. 183–186.

Chang, Chun-Hao et al. (2018). *Explaining Image Classifiers by Counterfactual Generation.* arXiv: `1807.08024 [cs.CV]`.

Chen, Chaofan et al. (2018). *This Looks Like That: Deep Learning for Interpretable Image Recognition.* arXiv: `1806.10574 [cs.LG]`.

Cimpoi, M. et al. (2014). "Describing Textures in the Wild". In: *Proceedings of the IEEE Conf on Computer Vision and Pattern Recognition (CVPR).*

Cook, R. Dennis (1977). "Detection of Influential Observation in Linear Regression". In: *Technometrics* 19.1, pp. 15–18. ISSN: 00401706. URL: `http://www.jstor.org/stable/1268249`.

Dabkowski, Piotr and Yarin Gal (2017). "Real time image saliency for black box classifiers". In: *Advances in Neural Information Processing Systems*, pp. 6967–6976.

Doshi-Velez, Finale and Been Kim (2017). "Towards a rigorous science of interpretable machine learning". In: *arXiv preprint arXiv:1702.08608*.

Doshi-Velez, Finale, Mason Kortz, et al. (2017). "Accountability of AI Under the Law: The Role of Explanation". In: *SSRN Electronic Journal.* ISSN: 1556-5068. DOI: `10.2139/ssrn.3064761`. URL: `http://dx.doi.org/10.2139/ssrn.3064761`.

Fong, Ruth C. and Andrea Vedaldi (2017). "Interpretable Explanations of Black Boxes by Meaningful Perturbation". In: *2017 IEEE International Conference on Computer Vision (ICCV).* DOI: `10.1109/iccv.2017.371`. URL: `http://dx.doi.org/10.1109/ICCV.2017.371`.

Fong, Ruth and Andrea Vedaldi (2018). "Net2Vec: Quantifying and Explaining How Concepts are Encoded by Filters in Deep Neural Networks". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition.* DOI: `10.1109/cvpr.2018.00910`. URL: `http://dx.doi.org/10.1109/cvpr.2018.00910`.

Gatys, Leon A, Alexander S Ecker, and Matthias Bethge (2016). "Image style transfer using convolutional neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2414–2423.

Ghorbani, Amirata, James Wexler, and Been Kim (2019). "Automating Interpretability: Discovering and Testing Visual Concepts Learned by Neural Networks". In: *arXiv preprint arXiv:1902.03129*.

Gilpin, Leilani H. et al. (2018). "Explaining Explanations: An Overview of Interpretability of Machine Learning". In: *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. DOI: 10.1109/dsaa.2018.00018. URL: http://dx.doi.org/10.1109/dsaa.2018.00018.

Goodfellow, Ian J, Jonathon Shlens, and Christian Szegedy (2014). "Explaining and harnessing adversarial examples". In: *arXiv preprint arXiv:1412.6572*.

Goodman, Bryce and Seth Flaxman (2017). "European Union regulations on algorithmic decision-making and a "right to explanation"". In: *AI Magazine* 38.3, pp. 50–57.

Guidotti, Riccardo, Anna Monreale, Salvatore Ruggieri, Dino Pedreschi, et al. (2018). *Local Rule-Based Explanations of Black Box Decision Systems*. arXiv: 1805.10820 [cs.AI].

Guidotti, Riccardo, Anna Monreale, Salvatore Ruggieri, Franco Turini, et al. (2018). "A Survey of Methods for Explaining Black Box Models". In: *ACM Computing Surveys* 51.5, pp. 1–42. ISSN: 0360-0300. DOI: 10.1145/3236009. URL: http://dx.doi.org/10.1145/3236009.

Hara, Satoshi and Kohei Hayashi (2016). "Making tree ensembles interpretable". In: *arXiv preprint arXiv:1606.05390*.

He, Kaiming et al. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Higgins, Irina, David Amos, et al. (2018). "Towards a definition of disentangled representations". In: *arXiv preprint arXiv:1812.02230*.

Higgins, Irina, Loic Matthey, et al. (2017). "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework." In: *ICLR* 2.5, p. 6.

Hind, Michael et al. (2019). "TED". In: *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society - AIES '19*. DOI: 10.1145/3306618.3314273. URL: http://dx.doi.org/10.1145/3306618.3314273.

Kim, Been et al. (2017). "Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav)". In: *arXiv preprint arXiv:1711.11279*.

Kindermans, Pieter-Jan et al. (2017). "The (un) reliability of saliency methods". In: *arXiv preprint arXiv:1711.00867*.

Kingma, Diederik P and Max Welling (2013). "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114*.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., pp. 1097–1105. URL: http://dl.acm.org/citation.cfm?id=2999134.2999257.

Laugel, Thibault et al. (2018). "Defining Locality for Surrogates in Post-hoc Interpretablity". In: *CoRR* abs/1806.07498. arXiv: 1806.07498. URL: http://arxiv.org/abs/1806.07498.

Letham, Benjamin et al. (2015). “Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model”. In: *Ann. Appl. Stat.* 9.3, pp. 1350–1371. DOI: 10.1214/15-AOAS848. URL: https://doi.org/10.1214/15-AOAS848.

Liang, Bin et al. (2017). “Deep Text Classification Can be Fooled”. In: *CoRR* abs/1704.08006. arXiv: 1704.08006. URL: http://arxiv.org/abs/1704.08006.

Maaten, Laurens van der and Geoffrey Hinton (2008). “Visualizing data using t-SNE”. In: *Journal of machine learning research* 9.Nov, pp. 2579–2605.

McInnes, Leland, John Healy, and James Melville (2018). “UMAP: Uniform Manifold Approximation and Projection for dimension reduction. arXiv, 2018”. In: *arXiv preprint arXiv:1802.03426*.

Mordvintsev, Alexander, Christopher Olah, and Mike Tyka (2015). “Inceptionism: Going Deeper into Neural Networks”. In: URL: https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html.

Nguyen, Anh et al. (2016). *Synthesizing the preferred inputs for neurons in neural networks via deep generator networks.* arXiv: 1605.09304 [cs.NE].

Pedreshi, Dino, Salvatore Ruggieri, and Franco Turini (2008). “Discrimination-aware data mining”. In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, pp. 560–568.

Ribeiro, Marco Túlio, Sameer Singh, and Carlos Guestrin (2016). “”Why Should I Trust You?”: Explaining the Predictions of Any Classifier”. In: *CoRR* abs/1602.04938. arXiv: 1602.04938. URL: http://arxiv.org/abs/1602.04938.

Rudin, Cynthia (2018). “Please stop explaining black box models for high stakes decisions”. In: *arXiv preprint arXiv:1811.10154*.

Russakovsky, Olga et al. (2015). “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

Selvaraju, Ramprasaath R et al. (2017). “Grad-cam: Visual explanations from deep networks via gradient-based localization”. In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 618–626.

Simonyan, Karen, Andrea Vedaldi, and Andrew Zisserman (2013). “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”. In: *CoRR* abs/1312.6034. arXiv: 1312.6034. URL: http://arxiv.org/abs/1312.6034.

Simonyan, Karen and Andrew Zisserman (2014). “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556*.

Smilkov, Daniel et al. (2017). *SmoothGrad: removing noise by adding noise.* arXiv: 1706.03825 [cs.LG].

Szegedy, Christian, Sergey Ioffe, et al. (2016). “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning”. In: *ICLR 2016 Workshop.* URL: https://arxiv.org/abs/1602.07261.

Szegedy, Christian, Wei Liu, et al. (2014). “Going Deeper with Convolutions”. In: *CoRR* abs/1409.4842. arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842.

Szegedy, Christian, Vincent Vanhoucke, et al. (2015). “Rethinking the Inception Architecture for Computer Vision”. In: *CoRR* abs/1512.00567. arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567.

Szegedy, Christian, Wojciech Zaremba, et al. (2013). *Intriguing properties of neural networks.* arXiv: 1312.6199 [cs.CV].

TCAV Github repository (2019). forked on 2019-05-19. URL: https://github.com/tensorflow/tcav.

Wachter, Sandra, Brent Mittelstadt, and Luciano Floridi (2017). "Why a Right to Explanation of Automated Decision-Making Does Not Exist in the General Data Protection Regulation". In: *International Data Privacy Law* 7.2, pp. 76–99. ISSN: 2044-3994. DOI: 10.1093/idpl/ipx005. eprint: http://oup.prod.sis.lan/idpl/article-pdf/7/2/76/17932196/ipx005.pdf. URL: https://doi.org/10.1093/idpl/ipx005.

Weller, Adrian (2017). "Challenges for Transparency". In: *CoRR* abs/1708.01870. arXiv: 1708.01870. URL: http://arxiv.org/abs/1708.01870.

Xulei Yang, Qing Song, and A. Cao (2005). "Weighted support vector machine for data classification". In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2, 859–864 vol. 2. DOI: 10.1109/IJCNN.2005.1555965.

Yang, Mengjiao and Been Kim (2019). *BIM: Towards Quantitative Evaluation of Interpretability Methods with Ground Truth.* arXiv: 1907.09701 [cs.LG].

Yin, Ming, Jennifer Wortman Vaughan, and Hanna Wallach (2019). "Understanding the Effect of Accuracy on Trust in Machine Learning Models". In: *CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019).*

Zeiler, Matthew D. and Rob Fergus (2014). "Visualizing and Understanding Convolutional Networks". In: *Lecture Notes in Computer Science*, pp. 818–833. ISSN: 1611-3349. DOI: 10.1007/978-3-319-10590-1_53. URL: http://dx.doi.org/10.1007/978-3-319-10590-1_53.

Zhu, Jun-Yan et al. (2017). "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks". In: *2017 IEEE International Conference on Computer Vision (ICCV).* DOI: 10.1109/iccv.2017.244. URL: http://dx.doi.org/10.1109/ICCV.2017.244.

Zou, Hui, Trevor Hastie, and Robert Tibshirani (2006). "Sparse principal component analysis". In: *Journal of computational and graphical statistics* 15.2, pp. 265–286.

Appendices

Appendix A

TCAV Score Variance Detailed Results

Classifier Method	**Details**
SVM via SGD	$\alpha = 0.1$, otherwise default from `scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html`
Ave of SVMs	Average of 100 SVM via SGD using random seeds $s = 1, \dots, 100$
SVM	$C = 0.1$, otherwise default from `scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`
Logistic	default from `scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`
Mean	Difference of arithmetic mean for each class

Table A.1: Implementation details for each classifier.

			SVM via SGD		**Ave of SVMs**		**SVM**	
class	**concept**	**layer**	μ	σ	μ	σ	μ	σ
zebra	striped	mixed4	0.61	0.036	0.61	0.027	0.61	0.027
		mixed6	0.83	0.035	0.86	0.023	0.86	0.026
		mixed8	0.79	0.071	0.80	0.071	0.80	0.061
		mixed9	0.51	0.310	0.51	0.156	0.52	0.124
		mixed10	1.00	0.000	1.00	0.000	1.00	0.000
	dotted	mixed4	0.39	0.049	0.37	0.035	0.39	0.024
		mixed6	0.36	0.079	0.35	0.031	0.32	0.034
		mixed8	0.29	0.130	0.26	0.109	0.28	0.115
		mixed9	0.16	0.320	0.03	0.029	0.21	0.161
		mixed10	0.90	0.274	0.87	0.324	0.99	0.013
lion	grassland	mixed4	0.55	0.052	0.57	0.037	0.56	0.041
		mixed6	0.56	0.074	0.58	0.061	0.56	0.059
		mixed8	0.83	0.044	0.85	0.055	0.84	0.054
		mixed9	0.62	0.210	0.47	0.157	0.81	0.116
		mixed10	0.80	0.322	0.78	0.317	0.65	0.350
	ocean	mixed4	0.48	0.48	0.48	0.032	0.46	0.033
		mixed6	0.46	0.056	0.47	0.056	0.47	0.046
		mixed8	0.45	0.100	0.45	0.084	0.43	0.100
		mixed9	0.17	0.104	0.13	0.057	0.14	0.068
		mixed10	0.10	0.255	0.02	0.042	0.02	0.020

			Logistic		**Mean,** $\vert N\vert = 50$		**Mean,** $\vert N\vert = 500$	
class	**concept**	**layer**	μ	σ	μ	σ	μ	σ
zebra	striped	mixed4	0.61	0.026	0.59	0.021	0.59	0.015
		mixed6	0.86	0.021	0.87	0.012	0.87	0.011
		mixed8	0.77	0.074	0.74	0.036	0.73	0.020
		mixed9	0.64	0.102	0.58	0.075	0.55	0.051
		mixed10	1.00	0.000	1.00	0.000	1.00	0.000
	dotted	mixed4	0.36	0.028	0.44	0.022	0.43	0.013
		mixed6	0.36	0.035	0.31	0.02	0.31	0.016
		mixed8	0.27	0.098	0.21	0.058	0.18	0.016
		mixed9	0.01	0.011	0.1	0.040	0.08	0.026
		mixed10	0.96	0.156	0.99	0.006	0.98	0.012
lion	grassland	mixed4	0.56	0.034	0.58	0.037	0.58	0.018
		mixed6	0.57	0.057	0.57	0.043	0.58	0.023
		mixed8	0.84	0.054	0.89	0.024	0.88	0.018
		mixed9	0.38	0.120	0.84	0.072	0.82	0.035
		mixed10	0.67	0.353	0.68	0.295	0.67	0.258
	ocean	mixed4	0.47	0.032	0.47	0.028	0.44	0.027
		mixed6	0.47	0.046	0.45	0.048	0.44	0.026
		mixed8	0.44	0.094	0.44	0.083	0.44	0.065
		mixed9	0.10	0.048	0.27	0.101	0.20	0.05
		mixed10	0.02	0.042	0.00	0.006	0.00	0.00

Table A.2: Detailed results of TCAV score distribution for several concept and class pairings at several different layers. Each mean and standard deviation is from 25 trials using each combination of negative class $N_0 \ldots N_4$ and train/test split seeds $s = 1, \ldots, 5$.

		TCAV Score	
class	**layer**	μ	σ
zebra	mixed4	0.51	0.068
	mixed6	0.51	0.107
	mixed8	0.57	0.209
	mixed9	0.56	0.322
	mixed10	0.61	0.498
lion	mixed4	0.55	0.092
	mixed6	0.55	0.101
	mixed8	0.58	0.201
	mixed9	0.48	0.297
	mixed10	0.51	0.418
cab	mixed4	0.51	0.050
	mixed6	0.54	0.095
	mixed8	0.53	0.157
	mixed9	0.57	0.304
	mixed10	0.60	0.411
aircraft carrier	mixed4	0.53	0.082
	mixed6	0.53	0.089
	mixed8	0.62	0.202
	mixed9	0.51	0.334
	mixed10	0.55	0.442

Table A.3: TCAV score for CAVs generated from two random N classes, each with 50 images. Each distribution has 10 observations. 150 images from each class were used to compute the TCAV score. SVM via SGD was used for the linear classifier.